CASE Orientada a Objetos com Múltiplas Visões e Implementação Automática de Sistemas - MVCASE

Tathiana da Silva Barrére
Antonio Francisco do Prado
Vitor César Bonafe
E-mail: (tathiana,prado,bonafe)@dc.ufscar.br
Universidade Federal de São Carlos - Departamento de Computação
Rod. Washington Luiz, Km 235 – CEP 13565-905 - São Carlos - SP

Resumo

Este artigo apresenta uma ferramenta CASE para especificação de requisitos de sistemas com múltiplas visões e implementação automática orientada a objetos, denominada MVCASE. A MVCASE é composta por uma ferramenta gráfica, denominada JavaRC, e por um sistema transformacional orientado a domínios, denominado Draco. A JavaRC dispõe de uma interface para edição gráfica e textual dos requisitos do sistema e de recursos para obtenção de múltiplas visões destes requisitos. O sistema Draco é responsável pela geração automática de código em linguagem executável e pela geração automática de bases de dados do sistema, partindo das especificações dos requisitos. A MVCASE também está integrada com uma ferramenta, denominada Visual Café dbDE, com recursos para programação visual das interfaces gráficas dos sistemas usando Frames ou Applets em java.

***Palavras-chave**: Engenharia de Software, Ferramentas CASE, Representação Canônica de Requisitos, Sistemas Transformacionais, Interfaces Visuais.*

1. Introdução

Ferramentas que auxiliam o desenvolvedor nas diferentes fases do ciclo de vida do software, bem como na sua gerência e documentação, são conhecidas como ferramentas CASE (Computer-Aided Software Engineering). Estas ferramentas suportam desde a especificação e análise dos requisitos, até o projeto e implementação do sistema, com uso de bases de dados e interfaces gráficas e textuais. Elas têm se mostrado cada vez mais importantes e poderosas, auxiliando o desenvolvedor na maximização de suas habilidades intelectuais e criativas para a obtenção de software de mais alta qualidade, com maior produtividade.

Uma ferramenta CASE eficiente deve:

- permitir a modelagem de todas as técnicas de especificação de requisitos;
- possuir vários métodos possibilitando o intercâmbio entre eles;
- possuir interfaces gráficas; e
- gerar código.

A MVCASE, uma ferramenta CASE Orientada a Objetos (OO), foi desenvolvida com o propósito de investigar a automação do processo de desenvolvimento de sistemas de software, desde a análise e especificação de requisitos, utilizando técnicas de diferentes

métodos, até a implementação automática em linguagem executável e posterior manutenção. Ela é composta por uma ferramenta gráfica, denominada JavaRC, que suporta a especificação de requisitos do sistema usando técnicas de diferente métodos OO, e por um sistema transformacional, denominado Draco [12], que é responsável pela implementação automática do sistema.

A JavaRC persiste as especificações do sistema numa descrição em linguagens definidas no Draco. A linguagem utilizada na descrição das especificações dos requisitos do sistema baseia-se numa Representação Canônica (RC) para requisitos, proposta por Alan Davis [4]. Além da RC, é utilizada uma linguagem de pseudocódigo, denominada LBE [13], para completar a especificação dos modelos com a descrição do comportamento dos objetos, através das miniespecificações dos serviços em cada classe do sistema.

A descrição das especificações dos requisitos, gerada pela JavaRC para as técnicas de um determinado método, é usada para obter as múltiplas visões em técnicas correspondentes de outros métodos. Esta descrição também é analisada pelo Draco para gerar automaticamente o código do sistema em linguagem executável, como java [14]. Outro recurso da MVCASE suporta a geração automática de comandos SQL [6] para criação de bases de dados do sistema baseado nessa descrição.

Outro componente integrado com a MVCASE é a ferramenta Visual Café dbDE [15] na qual o desenvolvedor pode construir interfaces gráficas do sistema, para aplicações que podem ou não serem executadas na Internet.

Este artigo apresenta a ferramenta MVCASE da seguinte forma: Na seção 2 são apresentados os componentes responsáveis pela obtenção das múltiplas visões e pela implementação automática dos requisitos modelados. A seção 3 descreve a integração entre os componentes apresentados. Para ilustrar a utilização da MVCASE, é apresentado na seção 4 um estudo de caso sobre um sistema de Distribuidora de Produtos. Finalmente, na seção 5 são apresentadas as conclusões deste trabalho.

2. CASE OO com Múltiplas Visões e Implementação Automática

Os componentes da MVCASE são:

- a *Ferramenta JavaRC*, para especificação dos requisitos do sistema;
- o *Sistema transformacional Draco*, responsável pela geração automática de código; e
- a *Ferramenta Visual Café dbDE*, que suporta a construção de interfaces gráficas.

Segue a apresentação da ferramenta JavaRC que é o *front-end* da ferramenta MVCASE e responsável pela obtenção das múltiplas visões dos requisitos do sistema.

2.l. A Ferramenta JavaRC

As interfaces com o desenvolvedor são requisitos fundamentais de uma ferramenta CASE. A JavaRC provê interface gráfica e textual para a modelagem de sistemas orientados a objetos. O desenvolvedor escolhe um determinado método e então modela o sistema desejado segundo as técnicas deste método.

Vários conceitos da engenharia de software são representados em modelos gráficos, e a utilização destes modelos facilita o entendimento do sistema, reduzindo sua complexidade. Outros conceitos são melhor e mais rapidamente compreendidos através de representações textuais. Técnicas gráficas e textuais completam a modelagem do sistema.

A Figura 1 apresenta uma tela na qual o desenvolvedor faz a modelagem de um sistema. Esta tela disponibiliza, através de uma barra de ferramentas, os componentes visuais que o desenvolvedor usa na modelagem, segundo o método escolhido. Estes componentes visuais encapsulam as técnicas dos métodos de especificação de requisitos e são instanciados para compor os diagramas que representam graficamente os modelos do sistema em cada método.

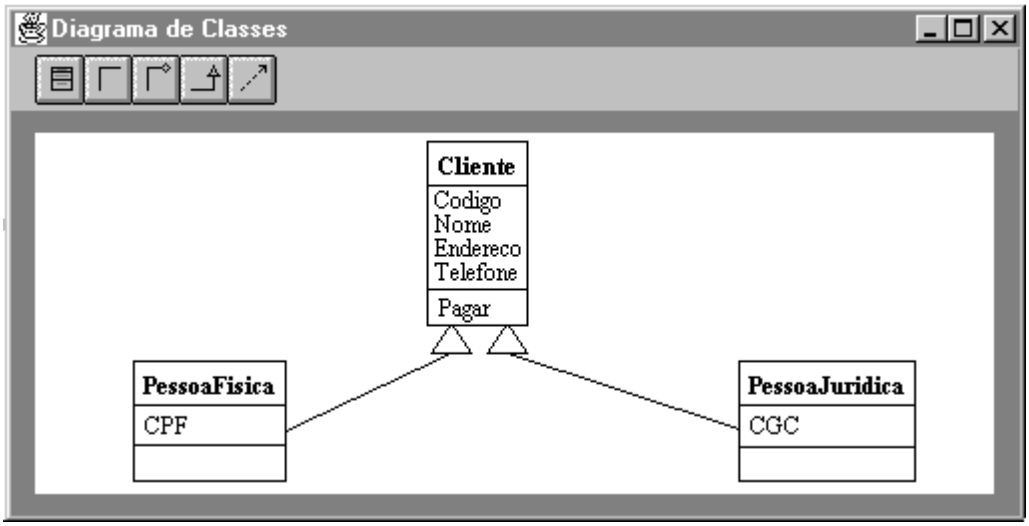

Figura 1 - Modelagem Usando a Ferramenta JavaRC

A JavaRC faz a persistência dos modelos que representam os requisitos de um determinado sistema nas linguagens RC e LBE. Para fazer esta persistência a ferramenta faz determinadas verificações de consistência. Por exemplo, é verificado se o desenvolvedor especificou as cardinalidades, que indicam as ocorrências de um objeto em relação a outros objetos em que está conectado.

A seguir são apresentadas as linguagens RC e LBE, utilizadas para armazenar as especificações de requisitos do sistema.

2.2. Linguagens RC e LBE

A especificação dos requisitos de um sistema é uma etapa que produz um documento que descreve o comportamento externo esperado do sistema de software, e deve servir como base para as atividades de teste e verificação dos resultados do sistema. Essa especificação deve ser clara, consistente e tão completa quanto possível, para evitar que erros introduzidos na fase inicial do desenvolvimento de um software se propaguem ao longo das demais fases e produzam um sistema que não atenda às necessidades do desenvolvedor.

Embora existam várias técnicas e modelos para a especificação de requisitos, nenhuma linguagem provê uma imagem completa dos requisitos de um sistema. Para tentar prover todas as possíveis visões de requisitos, foi proposta por Alan Davis [4] uma Representação Canônica (RC) para requisitos. Esta RC apresenta uma forma de analisar e armazenar requisitos independentemente do método de especificação utilizado.

Uma representação canônica $RC(E,R)$ é uma estrutura composta de um conjunto de elementos $E=\{e_1, e_2,...,e_n\}$ e de um conjunto de relações $R=\{r_1, r_2,...,r_n\}$, onde cada relação r conecta um par ordenado de elementos $e, f \in E$, que não são necessariamente distintos. Além

disso, cada $e_i \in E$ é um tupla: $e_i=(et_i, el_i)$ onde et_i é o tipo do elemento, $et_i \in ET \cup FT$, onde ET = {*entidade, processo, estado, mensagem, atributo, predicado, restrição, informação, transição*}, com $FT = 2^{ET}$. Por sua vez, el_i é uma identificação única para o elemento. Também, cada $r_i \in R$ é uma quádrupla: (rt_i , rl_i, se_i, te_i), onde rt_i é o tipo de relacionamento, $rt_i \in RT$, com RT = {*parte de, instanciação, tem valor, envia, recebe, estímulo, resposta, equivalência, associação, operando*}, rl_i é uma identificação única para a relação, se_i e te_i são o elemento inicial e o final do relacionamento, $se_i \in E$ e $te_i \in E$.

Essa representação é composta de um conjunto de 9 tipos de elementos de requisitos e de um conjunto de 10 tipos de relacionamentos possíveis entre os elementos. Os elementos e relacionamentos são descritos nas Tabelas 1 e 2 [7].

Tabela 1 - Elementos da Representação Canônica

Elemento	Descrição
Entidade	Algo existente no mundo real relevante para o problema em questão.
Processo	Ação, tarefa, função ou atividade a ser realizada.
Estado	O sistema se comporta de maneira a conservar algumas características.
Mensagem	Algo que se movimenta de um elemento para outro.
Atributo	Característica ou descrição de algum outro elemento do modelo.
Predicado	Preposição ou um operador booleano relacionados à aplicação tratada.
Restrição	Algo que deve obrigatoriamente existir entre um ou mais elementos.
Informação	Valor ou um conjunto de valores referente à aplicação considerada.
Transição	Agentes externos causadores de mudança no sistema e os resultados.

Tabela 2 - Relacionamentos da Representação Canônica

Relacionamento	Descrição
Parte de	Indica que um elemento é parte de outro no sistema considerado.
Instanciação	Indica que um determinado elemento é uma generalização de um outro.
Tem valor	Indica que um elemento tem valor de outro.
Envia	Capta os requisitos de um elemento e gera uma mensagem.
Recebe	Capta os requisitos de um elemento para aceitar uma mensagem.
Estímulo	Capta os elementos que causam a ocorrência de uma transição.
Resposta	Capta os elementos que serão modificados por uma transição.
Equivalência	Dois elementos são equivalentes se representam algo idêntico.
Operando	Representa um relacionamento entre um predicado e seus operandos.
Associação	Possui características distintas dos relacionamentos anteriores.

Baseada na RC foi definida uma linguagem, denominada linguagem RC, para armazenar as especificações dos requisitos. A linguagem RC é composta por uma combinação dos elementos e relacionamentos citados. A JavaRC faz a persistência das diferentes técnicas de especificação de requisitos em descrições na linguagem RC.

Para detalhar o comportamento dos objetos especificados pela RC, integrou-se também à MVCASE uma linguagem de pseudocódigo, denominada LBE, que é usada nas especificações dos serviços em cada classe definida pela RC.

O uso das linguagens RC e LBE fica mais claro com a apresentação do próximo componente da MVCASE que é o sistema transformacional Draco responsável pela geração automática do código Java, a partir das descrições em RC e LBE.

2.3. O Sistema Transformacional Draco

O sistema transformacional Draco tem sua versão inicial como resultado da tese de doutorado apresentada por Neighbors [11] e foi posteriormente reconstruído no Departamento de Informática da PUC-RJ [12]. Ele tem por objetivo desenvolver, colocar em prática e testar o paradigma transformacional para o desenvolvimento de software orientado a domínios. O Draco propõe um modelo para o processo de produção de software, no qual uma nova especificação pode ser obtida através da aplicação de regras de transformação, descritas formalmente, sobre uma especificação de entrada. As regras de transformação são responsáveis pela automatização do processo de construção de software [8][9]. Um domínio encapsulado no sistema Draco é representado por:

- Uma linguagem definida por um *parser* baseado em uma gramática livre de contexto. O *parser* é responsável por gerar a representação interna (DAST - *Draco Abstract Syntax Tree*) utilizada no Draco. Quando uma descrição escrita na linguagem de um domínio é analisada, o *parser* gera automaticamente a DAST correspondente;
- Um *prettyprinter* ou *unparser* responsável por mapear representações em sintaxe abstrata para a sintaxe concreta da linguagem, ou seja, a partir de uma DAST descreve novamente a representação na linguagem do domínio; e
- Conjuntos de transformações, compostos de regras de transformação que manipulam a DAST, e transformam descrições de um domínio em descrições do mesmo domínio, ou de outro domínio encapsulado no sistema Draco. Uma regra de transformação é essencialmente formada por dois padrões distintos: o padrão de reconhecimento (LHS - Left Hand Side) e o padrão de substituição (RHS - Right Hand Side). Para aplicar uma transformação, o sistema procura um padrão de reconhecimento definido, e o substitui pelo padrão de substituição desejado. As regras podem ainda possuir restrições semânticas, impondo condições que devem ser seguidas antes ou após a sua aplicação [1].

Sendo um sistema transformacional orientado a domínios, o primeiro passo para a utilização do Draco na MVCASE foi a construção dos domínios baseados na linguagem RC, usada para armazenar as descrições das especificações de requisitos na forma canônica, e na linguagem LBE, usada para especificar o comportamento mais detalhado dos objetos [10]. Estes domínios são compostos por:

- Um *parser* para a linguagem do domínio;
- Um *prettyprinter* para escrever a especificação na sintaxe concreta da linguagem, a partir da representação interna do domínio (DAST);
- Um conjunto de transformações que realiza a geração do código do sistema a partir da especificação persistida em RC e LBE.

A descrição gerada pela JavaRC é analisada pelos *parsers* dos domínios RC e LBE e sobre esta descrição são aplicadas as regras de transformação que mapeiam a descrição para código java ou especificações em SQL para criar as bases de dados do sistema. A Figura 2 mostra um exemplo de transformação do domínio LBE para o domínio java.

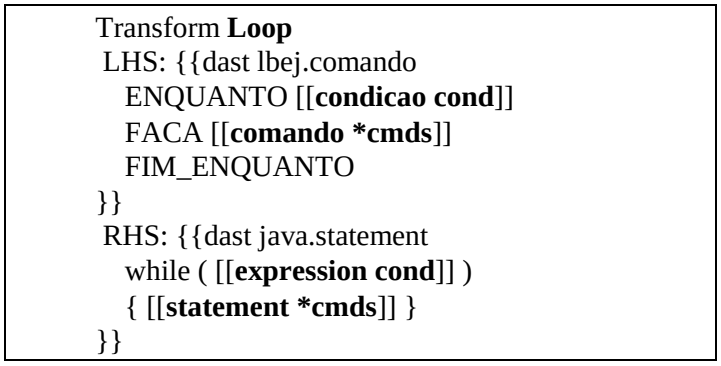

Figura 2 - Exemplo de Transformação da Linguagem LBE para Java

O exemplo destacado na figura mostra a transformação **Loop**, na qual o LHS identifica o comando ENQUANTO, escrito em LBE, e o substitui pelo padrão do RHS, equivalente em java. Os símbolos **cond** e **cmds** entre [[ ]] são metavariáveis. Antes de aplicar a transformação **Loop**, são aplicadas outras transformações que substituem essas metavariáveis por um padrão formatado da categoria sintática especificada pelos tipos **condicao**, **comando**, **expression** e **statement**. O sinal ***** antes de **cmds** representa uma repetição dessa metavariável. Da mesma forma outras transformações foram escritas para transformar de RC para java. Um conjunto com oitenta transformações completam o processo de das transformações das descrições RC e LBE para java.

A seguir é apresentado o ambiente da ferramenta MVCASE, que integra as ferramentas JavaRC e Visual Café dbDE com o sistema transformacional Draco.

3. Integração dos Componentes da Ferramenta MVCASE

A MVCASE suporta o desenvolvimento de sistemas de software orientados a objetos, desde a análise dos requisitos até a implementação e construção de interfaces. Desenvolvedores podem modelar um sistema na JavaRC e obter as múltiplas visões dos requisitos através da Representação Canônica. A JavaRC persiste os requisitos de cada sistema modelado em descrições textuais nas linguagens RC e LBE. Baseado nestas descrições, desenvolvedores podem gerar automaticamente o código do sistema em linguagem executável usando o sistema Draco. A integração com a ferramenta Visual Café dbDE possibilita a construção de interfaces gráficas para o sistema.

Utilizando a MVCASE, pode-se projetar o sistema como um todo, com poucos detalhes e mais tarde refiná-lo, ou se concentrar em partes do sistema, criando porções reusáveis, que juntas formarão o sistema. Em qualquer ponto do desenvolvimento, pode-se gerar o código do sistema em linguagem executável. A Figura 3 mostra como estão integrados os componentes da MVCASE, apresentados na seção anterior.

Uma das principais atividades suportada pela MVCASE é a modelagem de sistemas usando técnicas de diferentes métodos OO. Para modelar um sistema na MVCASE, desenvolvedores interagem com a ferramenta JavaRC para especificar os requisitos do sistema usando as técnicas de modelagem de um dos métodos disponíveis na JavaRC. A ferramenta JavaRC suporta as técnicas dos métodos OO Coad/Yourdon [2], Fusion [3] e UML [16].

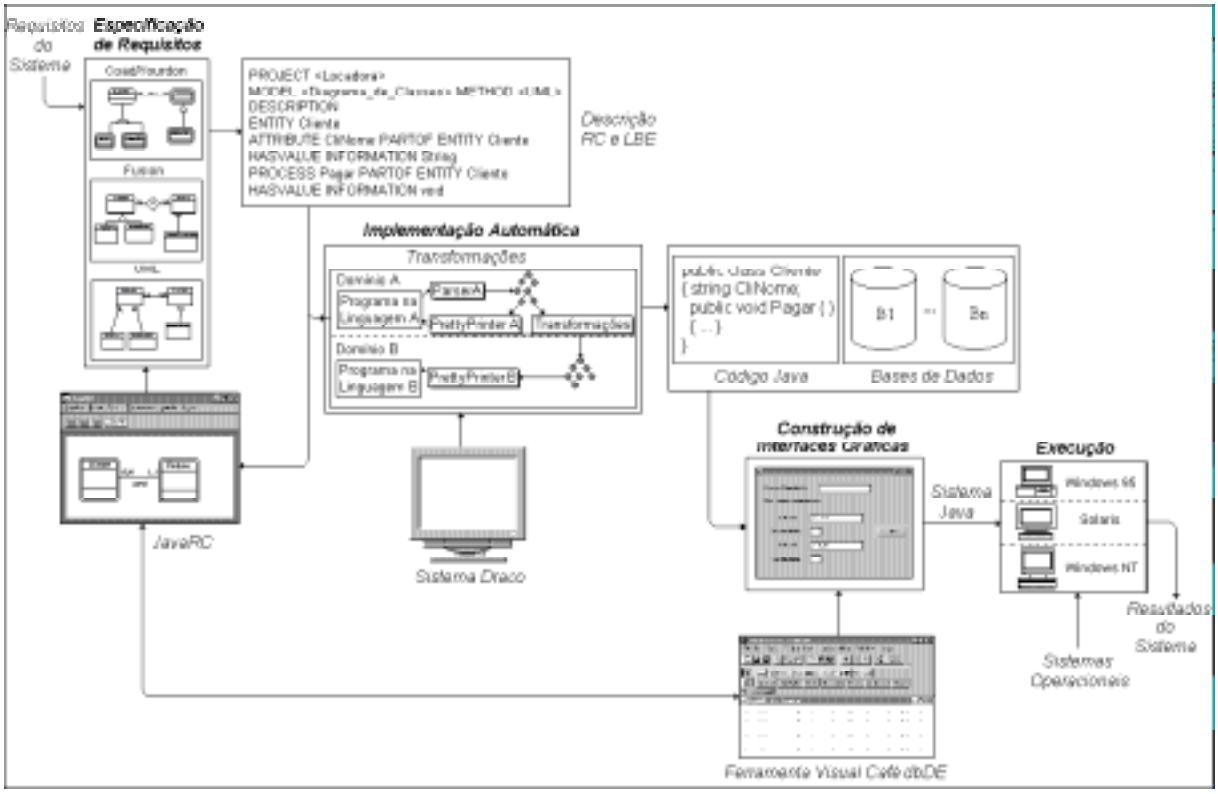

Figura 3 - Integração dos Componentes da MVCASE

A MVCASE suporta a modelagem do sistema em determinado método de desenvolvimento de software orientado a objetos, e persiste os modelos em uma descrição textual nas linguagens de modelagem de requisitos RC e LBE. Através desta descrição, pode-se obter uma nova visão do mesmo sistema segundo outro método OO disponível na JavaRC. Neste caso, a JavaRC recupera a descrição em RC e LBE do sistema e reprojeta o modelo armazenado usando técnicas similares de outro método, fornecendo assim ao desenvolvedor uma nova visão.

A descrição em RC e LBE do sistema também é usada pelo sistema Draco para geração de código do sistema em linguagem executável. O desenvolvedor pode também realizar a geração de bases de dados do sistema, definindo as classes e atributos que serão persistidos na base de dados. Atualmente a MVCASE suporta a geração de bases de dados para o Sistema de Gerenciamento de Bancos de Dados Relacionais (SGBDR) Sybase [6].

Durante a geração de código ou bases de dados, o sistema Draco analisa a descrição gerada pela JavaRC, transformando-a automaticamente para código executável ou especificações em SQL que poderão ser acessadas e executadas normalmente.

Para completar a implementação do sistema muitas vezes é necessária a construção de interfaces gráficas. A MVCASE está integrada com a ferramenta Visual Café dbDE que já dispõe de componentes necessários para a programação visual dessas interfaces.

Para criar as interfaces, além dos recursos visuais da ferramenta Visual Café dbDE, desenvolvedor também dispõe do código e das bases de dados geradas pelo Draco que podem ser integradas com o código gerado pela ferramenta Visual Café dbDE. O código java gerado pelo Draco contém as classes, com seus atributos e serviços, que podem ser acessadas pelo código da interface. Durante a programação da interface visual o desenvolvedor pode instanciar objetos das classes criadas e utilizar-se dos serviços disponíveis nessas classes.

Estando no Visual Café dbDE, o desenvolvedor pode visualizar as especificações do sistema para obter as informações, como protótipos dos serviços e atributos em cada classe, para construir a interface. Uma API [14] permite que o desenvolvedor visualize, na Visual Café dbDE, os modelos do sistema, criados na JavaRC. Com a execução dessa API a ferramenta JavaRC é ativada para exibir os modelos desejados do sistema, facilitando a integração dos códigos java gerados pela JavaRC e pela Visual Café dbDE.

O resultado da integração do código java e das bases de dados com o código da interface produz a implementação do sistema modelado. O código gerado pode ser executado nas diferentes plataformas de hardware e software, produzindo os resultados especificados para o sistema. Caso o código gerado não produza os resultados esperados o desenvolvedor pode corrigir as especificações, trabalhando em alto nível de abstração, e gerar novamente o código.

A seguir será apresentado um estudo de caso de um sistema de Distribuidora de Produtos, que mostra como o desenvolvedor pode utilizar a ferramenta MVCASE.

4. Estudo de Caso

Trata-se de um sistema de Distribuidora de Produtos, cuja descrição está resumida na Figura 4.

Uma distribuidora recebe pedidos de produtos pelo correio, telefone ou diretamente. Caso o cliente não esteja cadastrado, esta atividade é realizada por ocasião do pedido. O pedido contém uma lista de produtos, e é aceito se pelo menos um deles existir no cadastro. Caso contrário o pedido é rejeitado.

Semanalmente são emitidas requisições aos fornecedores, que ao serem atendidas são faturadas para os clientes

Figura 4 - Descrição Resumida do Sistema Distribuidora de Produtos

4.1. Especificação dos Requisitos do Sistema

Na MVCASE a fase de especificação dos requisitos é realizada com a JavaRC. Inicialmente o desenvolvedor seleciona o método orientado a objetos segundo o qual pretende modelar o sistema. Uma vez definido o método, tem-se disponíveis os recursos para a utilização das técnicas deste método. Por exemplo, uma vez escolhido o método UML tem-se disponível as técnicas do Diagrama de Classes, o Diagrama de Use Case, o Diagrama de Seqüência, o Diagrama de Colaboração, o Diagrama de Estados, o Diagrama de Componentes e o Diagrama de *Deployment*, conforme as visões da UML.

Selecionado o método UML e a técnica Diagrama de Classes da Visão Lógica, a edição das classes é feita como mostra a tela da Figura 5. Entra-se com o nome da classe, *Pedido* no caso, com os atributos e com os serviços encapsulados na classe. A edição dos atributos e serviços é feita como mostram as telas das figuras 6 e 7, respectivamente. Na edição dos atributos da classe, define-se o Tipo do atributo, sua Visibilidade, e outros modificadores opcionais que procuram cobrir as necessidades da programação orientada a objetos em java. São disponibilizados os tipos básicos da linguagem java, e tipos especiais, como Date e Vector, para que o desenvolvedor possa utilizar tipos correspondentes às classes de prateleira. Para cada serviço o desenvolvedor especifica seu Tipo de Retorno, seus Parâmetros (nome e tipo) e sua Visibilidade.

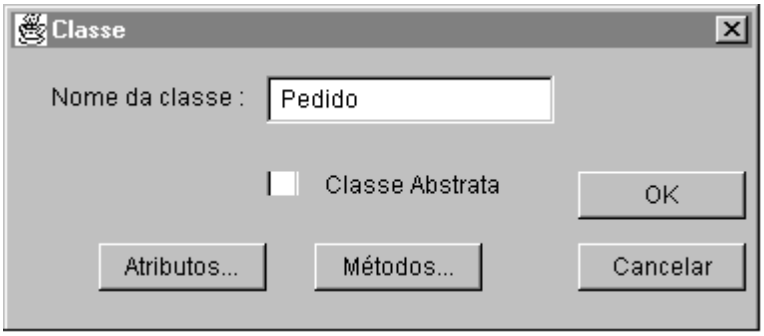

Figura 5 - Edição de Classes na JavaRC

Figura 6 - Edição de Atributos na JavaRC

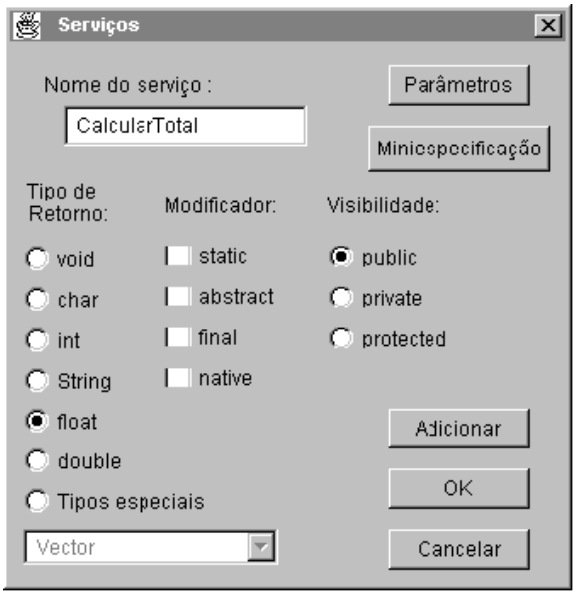

Figura 7 - Edição de Serviços na JavaRC

Para cada serviço editado o desenvolvedor pode definir seu comportamento através de miniespecificação na linguagem LBE. Um exemplo de miniespecificação para o serviço *CalcularTotal*, da classe *Pedido*, que calcula o valor total dos itens de um pedido, é mostrado na Figura 8.

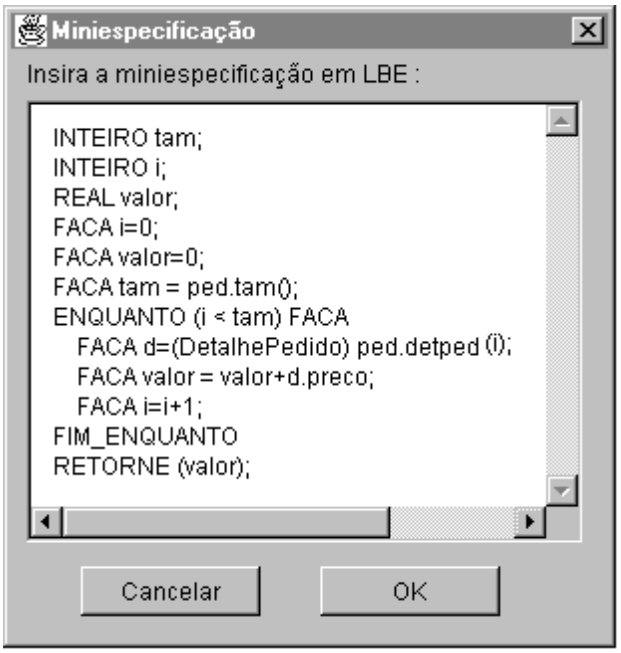

Figura 8 - Miniespecificação do Serviço *CalcularTotal* da Classe *Pedido*

A Figura 9 mostra uma tela para modelar as conexões entre as classes do sistema. As definições das ocorrências dos objetos de uma classe em relação a outros objetos, são representadas pelas cardinalidades, junto às estruturas que conectam as classes.

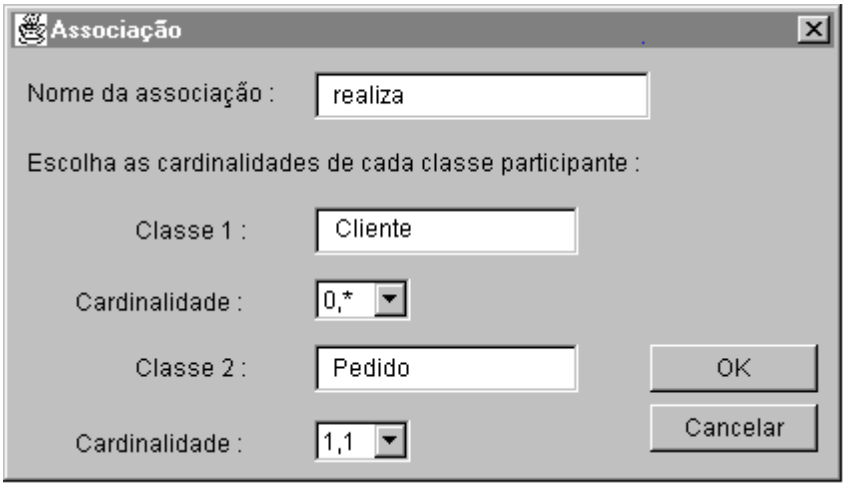

Figura 9 – Edição de uma Associação na JavaRC

A JavaRC dispõe dos componentes para especificação da técnica que está sendo utilizada através da sua seleção na respectiva barra de ferramentas. O modelo resultante, na técnica Diagrama de Classes do método UML, para o sistema de Distribuidora de Produtos é

apresentado na Figura 10. Baseado nesse diagrama é que a JavaRC gera os comandos em SQL para criação das bases de dados correspondentes.

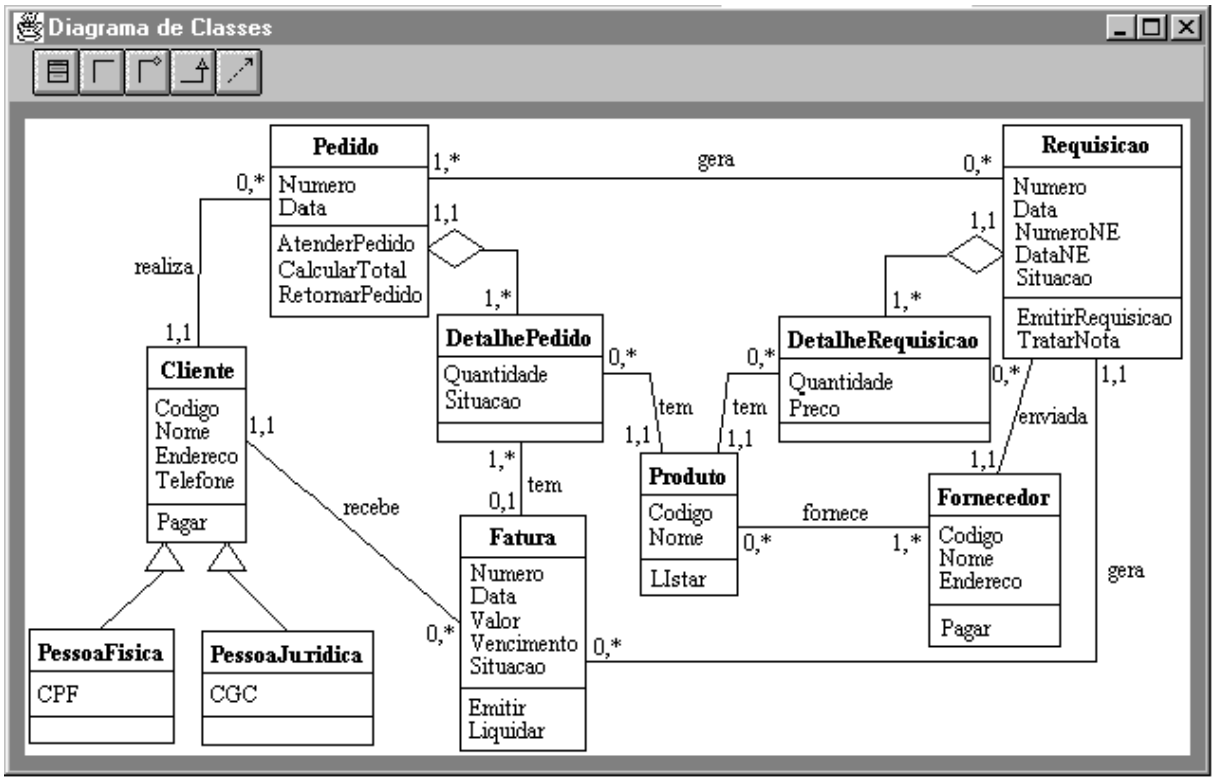

Figura 10 – Modelagem do Sistema Distribuidora de Produtos na JavaRC

A MVCASE dispõe das outras técnicas que completam a modelagem do sistema segundo o método escolhido. Por exemplo, a Figura 11 mostra a modelagem do Diagrama de UseCase para a atividade *FazerPedido*. Esse diagrama pode ser especificado na MVCASE da mesma forma que foi feito com o Diagrama de Classes.

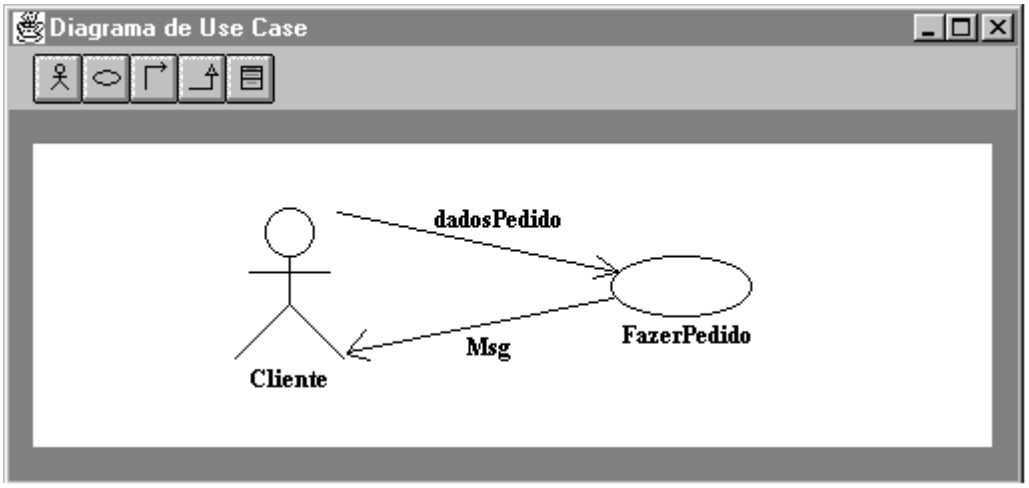

Figura 11 – Modelagem do Diagrama de UseCase para a Atividade *FazerPedido*

Os modelos das figuras 10 e 11 são persistidos em uma descrição nas linguagens RC e LBE, pela ferramenta JavaRC. Esta descrição será utilizada pelo Draco, para a geração do

código em linguagem executável, e pela ferramenta JavaRC, para obter novas visões em diferentes métodos orientados a objetos. As miniespecificações dos serviços das classes, descritas em LBE, são embutidas nas descrições RC. Quando se define o protótipo do serviço em cada classe na RC, embute-se na seqüência a respectiva miniespecificação em LBE. Na Figura 12 pode-se identificar a descrição RC das técnicas utilizadas durante a modelagem e a descrição LBE dos serviços das classes.

```
PROJECT <Distribuidora_de_Produtos>
MODEL <Diagrama_de_Classes> METHOD <UML>
DESCRIPTION
ENTITY Pedido
ATTRIBUTE Numero PARTOF ENTITY Pedido HASVALUE INFORMATION int
ATTRIBUTE Data PARTOF ENTITY Pedido HASVALUE INFORMATION String
PROCESS CalcularTotal PARTOF ENTITY Pedido HASVALUE INFORMATION float
ATTRIBUTE ped PARTOF PROCESS CalcularTotal HASVALUE INFORMATION Pedido
  {{ expression lbej.prog_comp
  BEGIN
   INTEIRO tam;
   INTEIRO i;
   REAL valor;
   FACA i=0;
   FACA valor=0;
   FACA tam = ped.tam();
   ...
}}
CONSTRAINT realiza
CONSTRAINT 1,1 PARTOF ENTITY Pedido OPERAND ENTITY Cliente
CONSTRAINT 0,N PARTOF ENTITY Cliente OPERAND ENTITY Pedido
END
GRAPHIC
ENTITY Cliente HASVALUE POSITION 85,101
ENTITY Pedido HASVALUE POSITION 341,98
ENTITY Fatura HASVALUE POSITION 236,243
CONSTRAINT realiza HASVALUE POSITION 105,101 317,101
CONSTRAINT recebe HASVALUE POSITION 341,128 260,246
END
END
MODEL <Diagrama_de_UseCase> METHOD <UML>
DESCRIPTION
ENTITY Cliente PARTOF ENTITY ATOR
PROCESS FazerPedido
MESSAGE dadosPedido PARTOF ENTITY Cliente
PROCESS FazerPedido RECEIVES MESSAGE dadosPedido
END
END
END
```

Figura 12 - Trecho da Descrição em RC e LBE do Sistema Distribuidora de Produtos

O primeiro trecho destacado na Figura 12 mostra como foram tratadas as descrições dos dois domínios RC e LBE num único programa. A descrição inicia com a linguagem RC, e usando a notação do Draco, tem-se as descrições LBE entre os metassímbolos "{{" e "}}". Seguindo o metassímbolo "{{", tem-se o nome da regra de retorno do *parser* RC, **expression**, o nome do domínio LBE com a regra inicial do *parser* LBE, **prog_comp**. Este procedimento permite que o Draco faça a recuperação da análise do *parser* RC quando terminar a análise do *parser* LBE. Assim quando for encontrado o metassímbolo "}}" tem-se um retorno para o *parser* RC que continua a análise normalmente. A máquina Draco está preparada para trabalhar com *parsers* de múltiplas entradas [5].

Na Figura 12 pode-se ver ainda a descrição da parte geométrica dos modelos, que permite a sua recuperação gráfica, pela JavaRC. Para a modelagem gráfica dos elementos de um modelo, foi incluído o elemento POSITION na linguagem do domínio RC, que armazena as posições destes elementos no modelo gráfico.

4.2. Implementação Automática em Java

A qualquer instante do desenvolvimento do sistema, utilizando a MVCASE, o desenvolvedor pode salvar os modelos criados, gerando a respectiva descrição em RC e LBE. Uma vez obtida a descrição, pode-se executar as transformações dos domínios RC e LBE, encapsulados no Draco, para transformar esta descrição para outras linguagens, no caso a linguagem java. A aplicação das transformações gera a implementação do sistema que pode ser executado e testado verificando assim a sua funcionalidade. Novas versões do sistema podem ser obtidas alterando ou incluindo novas especificações, até obter uma versão final do sistema implementado.

Após aplicadas as transformações dos domínios RC e LBE, sobre a descrição do modelo do sistema Distribuidora de Produtos, obtém-se o código correspondente em linguagem java. A Figura 13 mostra parte deste código gerado automaticamente pelo Draco.

```
class Pedido {
   public String Data;
   public int Numero;
   public Vector detped = new Vector();

   public Pedido () {}
   public float CalcularTotal (Pedido ped) {
      int tam = ped.tam ();
      int i=0;
      float valor=0;
      while (i<tam) {
         d=(DetalhePedido)ped.detped.elementAt(i);
         valor = valor+d.preco;
         i=i+1;
      }
      return valor;
   }
   // outros serviços
}
```

Figura 13 – Código Executável em Java Gerado para a Classe *Pedido*

Após a implementação automática do código java o desenvolvedor pode construir a interface gráfica do sistema. A Figura 14 mostra uma das telas criadas no Visual Café dbDE para o sistema Distribuidora de Produtos. O código java gerado pelo Draco é incorporado ao código gerado pela ferramenta Visual Café dbDE. Por exemplo, no sistema da locadora o desenvolvedor instancia a classe *Pedido* e usa o serviço *CalcularTotal* criado e especificado na JavaRC. Além do código java, foi gerada a base de dados para o sistema utilizando o bando de dados relacional Sybase. Para acessar a base de dados o desenvolvedor utiliza os componentes visuais já disponíveis na ferramenta Visual Café dbDE.

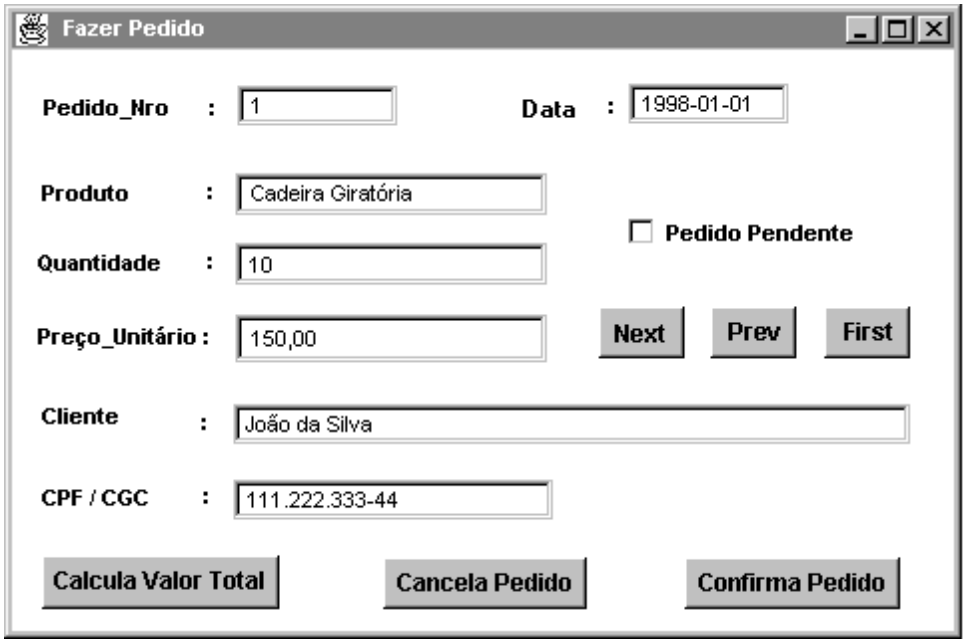

Figura 14 – Interface do Sistema Distribuidora de Produtos

4.3. Múltiplas Visões dos Requisitos

A abordagem de múltiplas visões possibilita o uso integrado de diferentes métodos na especificação de um sistema, e ao mesmo tempo propicia:

- migração de um método para outro, quando o sistema já começou a ser desenvolvido;
- reuso total ou parcial de especificações; e
- minimização dos custos de manutenção.

Para se obter uma nova visão, em outro método, do modelo produzido, a ferramenta JavaRC utiliza a descrição nas linguagens RC e LBE como a da Figura 12. A partir desta descrição, gera-se uma nova visão do modelo, com os elementos gráficos específicos do novo método. A geometria dos elementos em novas visões baseiam-se nas posições armazenadas na descrição RC, sendo a mesma para qualquer visão do modelo nos diferentes métodos orientados a objeto. A Figura 15 apresenta uma visão no método Coad/Yourdon, do Diagrama de Classes do sistema Distribuidora de Produtos.

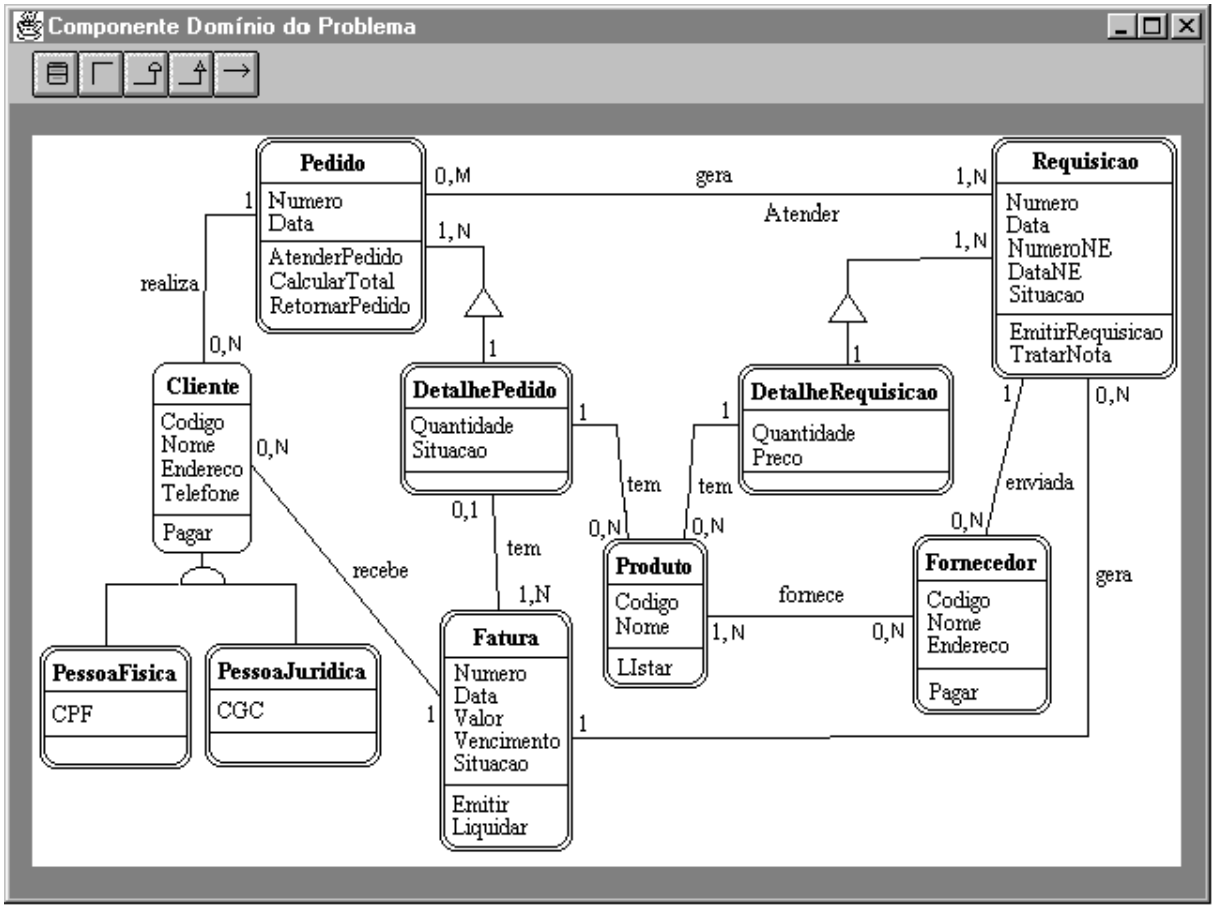

Figura 15 - Nova visão em Coad/Yourdon do Sistema Distribuidora de Produtos

5. Conclusão

Os resultados obtidos com esta pesquisa, demonstraram a viabilidade de se combinar as idéias do desenvolvimento de software orientado a objetos com as da implementação automática, usando uma ferramenta CASE com interface gráfica e textual integrada a um sistema transformacional de software. Esta combinação agiliza e facilita o processo de desenvolvimento e manutenção de software.

Uma característica importante da MVCASE é que a ferramenta apresentada suporta a modelagem em várias metodologias, proporcionando múltiplas visões dos requisitos através do uso de uma linguagem, baseada em uma Representação Canônica de requisitos. A abordagem de múltiplas visões possibilita o uso integrado de diferentes métodos orientados a objetos na especificação de um sistema, reduzindo os custos de manutenção e facilitando o uso de outras especificações já existentes. O uso integrado de vários métodos permite que o desenvolvedor faça o uso mais adequado das diferentes técnicas para especificação de requisitos, através da transformação de uma técnica para outra.

Outra característica também importante da MVCASE vem de sua integração com a ferramenta Visual Café dbDE e com o sistema transformacional Draco. A integração da MVCASE com o Draco, através das linguagens RC e LBE, permite a geração automática de código em qualquer linguagem definida no Draco, permitindo que o desenvolvedor implemente um sistema em diferentes linguagens de programação sem ter que conhecer profundamente suas sintaxes. Explorando a capacidade do Draco pode-se gerar os comandos

SQL para criação das bases de dados a partir das especificações dos requisitos. Esta integração também proporciona um alto grau de reuso da análise dos requisitos, aumentando a produtividade e facilitando a manutenção.

A combinação dos domínios RC e LBE definidos no Draco, validou a possibilidade de se trabalhar com vários domínios em uma mesma aplicação. Uma rede integrada de vários domínios pode ser construída, gerando grandes recursos para o desenvolvedor.

A integração da ferramenta Visual Café dbDE com a MVCASE, permite que o desenvolvedor realize todas as etapas do desenvolvimento de software em um mesmo ambiente.

Referências Bibliográficas

1. Bergmann, U. , Prado A.F., Leite J.C.P., *Desenvolvimento de Software Orientado a Objetos Utilizando o Sistema Transformacional Draco-PUC*, X Simpósio Brasileiro de Engenharia de Software, 1996.
2. Coad, P., Yourdon, E. *Análise Baseada em Objetos*, Editora Campus, 1992.
3. Coleman, D. et alli. *Object-Oriented Development - The Fusion Method*, Prentice Hall, 1994.
4. Davis, A. M. et alli. *A Canonical Representation for Requirements*, Technical Report, University of Colorado at Colorado Springs, 1995.
5. Freitas, F.G. et alli. *Aspectos Implementacionais de um Gerador de Analisadores Sintático para o Suporte a Sistemas Transformacionais*, I Simpósio Brasileiro de Linguagens de Programação, 1996.
6. Jepson, B. *Programando Banco de Dados em Java*, Makron Books, 1997.
7. Kirner, T. et alli. *Ambiente para Representação de Múltiplas Visões de Requisitos: O Metamodelo e uma Linguagem de Transformação*, X Simpósio Brasileiro de Engenharia de Software, 1996.
8. Leite, J. C. et alli. *Draco-PUC: A Technology Assembly for Domain Oriented Software Development*, III ICSR-IEEE, 1994.
9. Leite, J. C.; et alli *O Uso do Paradigma Transformacional no Porte de Programas Cobol*, IX Simpósio Brasileiro de Engenharia de Software - SBES 95, 1995.
10. Lima, M. A. V. *Especificação e prototipação de um ambiente para modelagem de sistemas orientados a objeto, usando uma Representação Canônica*, Tese de Mestrado, UFSCar, 1997.
11. Neighbors, J. *Software Construction Using Components*, Tese de Doutorado, University of California at Irvine, 1984.
12. Prado, A. F. *Estratégia de Reengenharia de Software Orientada a Domínios*, Tese de Doutorado, PUC-RJ, 1992.
13. Prado, A. F., Silva, T. E. - *O Uso do Sistema Transformacional DRACO no Desenvolvimento de Softwares Orientados a Objetos*, VII Semana de Informática da Universidade Estadual de Maringá, 1996.
14. Sun Microsystems. *Tutoriais Java*, www.javasun.com, 1997.
15. Symantec Corporation. *Visual Café dbDE*, www.symantec.com, 1997.
16. *UML Document Set*, www.rational.com/uml, 1997.