CASE com Múltiplas Visões de Requisitos e Implementação Automática

Tathiana da Silva Barrére
Antonio Francisco do Prado

E-mail: tathiana@dc.ufscar.br
Universidade Federal de São Carlos - Departamento de Computação
Rod. Washington Luiz, Km 235 – CEP 13565-905 - São Carlos - SP - Brasil

Resumo

Este artigo apresenta uma ferramenta CASE orientada a objetos, denominada JMVCASE, que suporta múltiplas visões de requisitos de software em diferentes técnicas de métodos orientados a objetos, e a implementação automática, visando facilitar a análise e a modelagem de um sistema. A JMVCASE é composta por uma ferramenta gráfica, que suporta múltiplas visões de requisitos, e um sistema transformacional de software, que permite a geração automática de código em java, a partir de especificações em alto nível de abstração. O sistema transformacional também suporta a geração automática de bases de dados para o sistema, a partir da especificação dos requisitos. A JMVCASE integra também uma ferramenta com recursos para programação visual que permite a construção de interfaces gráficas para os sistemas desenvolvidos usando *Frame* ou *Applet*.

Palavras-chave: Engenharia de Software, Ambientes de Desenvolvimento de Software, Sistemas Transformacionais, Representação Canônica para Requisitos, Ferramentas CASE.

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*). Ferramentas CASE suportam desde a análise, projeto e construção de interfaces textuais e gráficas até a implementação do sistema usando bancos de dados.

Para investigar o processo de desenvolvimento de software orientado a objetos desde a análise até a implementação, foi desenvolvida a ferramenta CASE Orientada a Objetos (CASE OO) JMVCASE, com suporte a múltiplas visões dos requisitos e implementação automática.

A JMVCASE é composta basicamente de uma ferramenta gráfica, denominada JavaRC, que permite a especificação de requisitos de software usando técnicas de diferentes métodos orientados a objetos, e de um sistema transformacional de software, denominado Draco [Nei84][Pra92], responsável pela implementação automática do sistema. A JavaRC armazena as especificações do sistema numa descrição em linguagens definidas no Draco. A linguagem utilizada na descrição da especificação dos requisitos de software baseia-se numa Representação Canônica (RC) para requisitos, proposta por Alan Davis [Dav95]. Além da RC, é utilizada outra linguagem, denominada LBE, para completar a especificação dos modelos com a descrição do comportamento dos objetos, através das miniespecificações dos serviços em cada classe do sistema.

Outro componente integrado à JMVCASE é a ferramenta Visual Café Pro [Sym97], com a qual pode-se construir a interface do sistema, para aplicações que são executadas ou não na Internet.

Este artigo apresenta a ferramenta JMVCASE da seguinte forma: A seção 2 apresenta uma visão geral sobre a ferramenta. Na seção 3 são apresentados os componentes responsáveis pela obtenção das múltiplas visões e a implementação automática dos requisitos modelados. Para ilustrar a utilização da JMVCASE, é apresentado na seção 4 um estudo de caso sobre um sistema de Locadora de Automóveis. Na seção 5 são apresentadas, finalmente, as conclusões deste trabalho.

2. Visão Geral da Ferramenta CASE Orientada a Objetos - JMVCASE

A Figura 1 fornece uma visão geral da ferramenta JMVCASE, que possibilita o desenvolvimento de sistemas de software orientados a objetos, a partir da especificação dos requisitos até a implementação e construção da interface. Ela possui a ferramenta gráfica e textual JavaRC para a especificação dos requisitos, o sistema transformacional Draco para implementação automática, e a ferramenta Visual Café Pro para a construção de interfaces gráficas para o sistema.

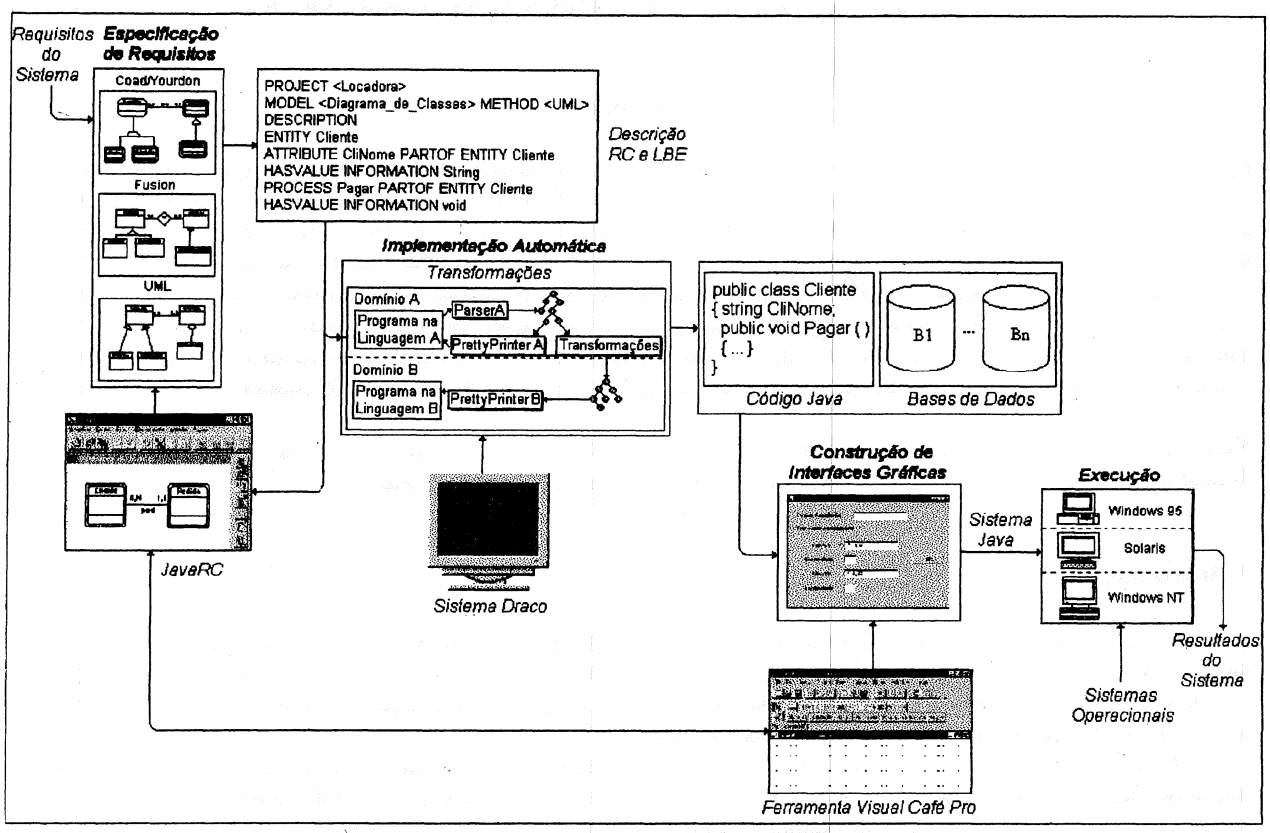

Figura 1 - Visão Geral da Ferramenta JMVCASE

A ferramenta JavaRC dispõe de três métodos de desenvolvimento de software orientados a objetos: Coad/Yourdon [Coa92], Fusion [Col94] e UML [Uml97]. Cada sistema modelado é persistido em uma descrição textual, segundo linguagens de modelagem definidas no sistema Draco. A linguagem para armazenar a descrição textual dos requisitos de software, utilizando técnicas da orientação a objetos, baseou-se numa Representação Canônica (RC) para requisitos, proposta por Alan Davis [Dav95]. Além da RC, foi também utilizada para completar a especificação dos modelos, uma linguagem de miniespecificação para os serviços contidos em cada modelo, denominada Linguagem Básica de Ensino (LBE).

Através da descrição textual do sistema nas linguagens RC e LBE, gerada pela JavaRC, o desenvolvedor pode obter uma nova visão do mesmo sistema segundo outro método orientado a objetos disponível. Para fazer a geração do código correspondente, o sistema Draco analisa esta descrição textual e transforma-a em linguagem executável, como java [Sun97]. Desta forma, obtém-se a implementação automática do sistema, a partir das especificações em alto nível de representação. As informações do sistema ainda podem ser persistidas em bases de dados, geradas também pelo Draco.

O código java e as bases de dados geradas podem ser integradas com o código gerado pela ferramenta Visual Café Pro, usada para a construção de interface gráfica dos sistemas. 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 das funções disponíveis nessas classes. Estando no Visual Café Pro, 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 [Sun97] permite que o desenvolvedor visualize, na Visual Café Pro, 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 Pro.

Finalmente o desenvolvedor pode fazer a execução do código java nas diferentes plataformas de hardware e software, produzindo os resultados especificados para o sistema. A próxima seção apresenta, com mais detalhes, a obtenção das múltiplas visões de requisitos e da implementação automática.

3. CASE com Múltiplas Visões de Requisitos e Implementação Automática

Na JMVCASE, um sistema modelado em determinado método de desenvolvimento de software orientado a objetos é armazenado em uma descrição textual nas linguagens de modelagem RC e LBE. Através desta descrição, pode-se obter uma nova visão do mesmo sistema segundo outro método OO disponível na JavaRC. Esta descrição também é analisado pelo sistema Draco e automaticamente transformada para linguagens executáveis, como java.

3.1. Ferramenta JavaRC

A ferramenta JavaRC provê uma 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.

A Figura 2 apresenta a 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 dispõe para fazer a modelagem no 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. A utilização de modelos gráficos facilita o entendimento do sistema, reduzindo a complexidade.

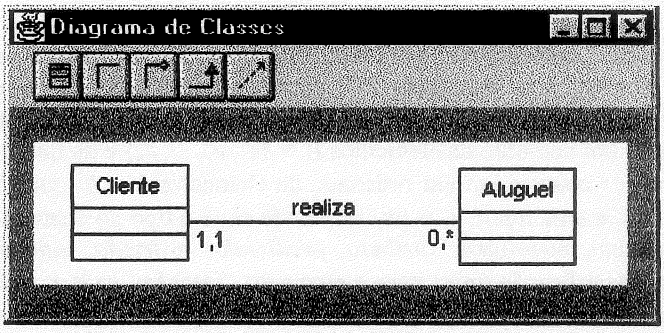

Figura 2 - Interface da ferramenta JavaRC para Modelagem de um Sistema

Modelos orientados a objetos, como o apresentado na Figura 2, são armazenados pela ferramenta JavaRC em uma descrição que utiliza as linguagens RC e LBE. A JavaRC utiliza esta descrição para produzir as 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, em situações nas quais o sistema já começou a ser desenvolvido;
- reuso total ou parcial de especificações; e
- minimização dos custos de manutenção.

Uma nova visão de um sistema pode ser obtida através de uma opção presente no menu principal da JavaRC. A Figura 3 apresenta a nova visão segundo o método Coad/Yourdon do sistema apresentado na Figura 2. 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.

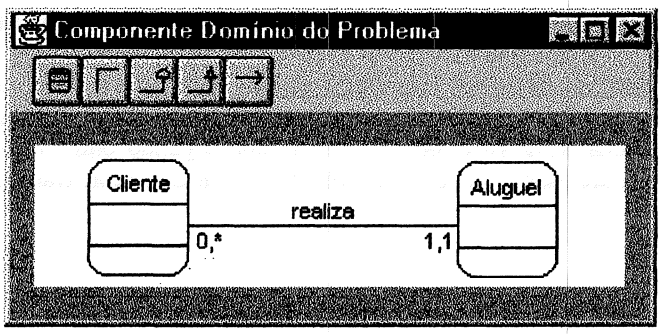

Figura 3 - Nova visão em Coad/Yourdon de um Sistema

Uma visão em determinado método contém técnicas que ao serem mapeadas para técnicas de outros métodos podem ser agrupadas ou separadas em diferentes representações. Por exemplo, no método Fusion, o ciclo de vida do modelo de interface contém a interface e as operações do sistema, enquanto que no método Coad/Yourdon, somente os nomes das operações aparecem no modelo de objetos. Estas operações são detalhadas separadamente através de miniespecificações.

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

3.1.1 Linguagens RC e LBE

Baseada na Representação Canônica (RC) para requisitos de software proposta por Alan Davis [Dav95], foi definida uma linguagem, denominada linguagem RC, para a especificação de requisitos. Esta RC é uma estrutura que permite analisar e armazenar requisitos, independentemente do método de especificação empregado. A RC é 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 $e \in E$, que não são necessariamente distintos. Além disso, cada $e_i \in E$ é uma *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) na qual rt_i é o tipo do 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 elemento final do relacionamento, $se_i \in E$ e $te_i \in E$ [Kir96].

Em resumo, a RC compõe-se dos seguintes elementos e relacionamentos:

Elementos:

- *entidade*: Algo existente no mundo real, relevante ao problema em questão.
- *processo*: Ação, tarefa, função ou atividade a ser realizada.
- *mensagem*: Algo que está sendo movimentado de um elemento para outro.
- *estado*: Modo no qual o sistema se comporta de maneira a conservar determinadas características.
- *atributo*: Característica ou descrição de algum outro elemento do modelo.
- *informação*: Valor ou um conjunto de valores referentes à aplicação considerada.
- *predicado*: Preposição ou um operador booleano, relacionados à aplicação tratada.
- *restrição*: Relacionamento que deve existir obrigatoriamente entre um ou mais elementos.
- *transição*: Conexão entre os agentes externos causadores de mudança no sistema e os resultados gerados.

Relacionamentos:

- *instanciação*: Indica que um elemento e_1 é uma generalização de outro elemento e_2.
- *parte-de*: Indica que um elemento e_2 é parte de um outro elemento e_1 no sistema considerado.
- *tem-valor*: Indica que um elemento e_1 tem valor de outro e_2 se e_2 recebe um valor específico atribuído a e_1.
- *envia*: Capta os requisitos relativos a um elemento e gera uma mensagem.
- *recebe*: Capta os requisitos de um elemento, a fim de aceitar uma mensagem.
- *estímulo*: Capta os elementos que causam a ocorrência de uma transição.
- *resposta*: Capta os elementos que serão modificados por uma transição.
- *operando*: Indica um relacionamento entre um predicado ou restrição e seus respectivos operandos.
- *equivalência*: Indica que dois elementos e_1 e e_2 são equivalentes se eles representam conceito ou algo idêntico no mundo real.
- *associação*: Identifica um relacionamento com características distintas dos anteriores.

A linguagem RC é expressa pelas combinações entre os elementos e os relacionamentos, e servirá para armazenar as informações relativas aos sistemas.

Para completar a especificação dos modelos, os serviços em cada classe são especificados na linguagem LBE, apropriada para detalhar o comportamento dos objetos. A LBE, com as características de pseudocódigo, permite que o desenvolvedor faça as miniespecificações dos serviços em cada classe, em um alto nível de abstração, facilitando o entendimento e a manutenção do sistema [Pra96].

As linguagens RC e LBE serviram de ligação entre a ferramenta JavaRC e o sistema Draco, tornando possível a implementação automática do sistema. O sistema Draco utiliza a descrição em linguagem RC para gerar o código correspondente ao modelo de objetos do sistema, e a parte da descrição em LBE para gerar o código dos serviços que definem o comportamento dos objetos.

3.2. O Sistema Transformacional Draco

O sistema transformacional Draco tem sua versão inicial como resultado da tese de doutorado apresentada por Neighbors [Nei84] e foi posteriormente reconstruído no Departamento de Informática da PUC-RJ [Pra92], com o objetivo de desenvolver, colocar em prática e testar o paradigma transformacional para o desenvolvimento de software orientado a domínios. Ele 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. Novas versões do Draco foram produzidas com melhorias que facilitam o seu uso pelo desenvolvedor [Lei94] [Lei95].

Um domínio encapsulado no sistema transformacional 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 utilizada no Draco, que é feita através de uma árvore de sintaxe abstrata chamada DAST (*Draco Abstract Syntax Tree*). Quando uma descrição escrita na linguagem de um domínio é analisada, o *parser* gera automaticamente a DAST correspondente. Para que uma descrição possa ser transformada pelo Draco, é necessário que ela esteja representada sob a forma interna.
- Um *prettyprinter* ou *unparser* responsável por mapear representações em sintaxe abstrata para a sintaxe concreta da linguagem, ou seja, a partir de uma DAST escreve novamente a descrição na linguagem do domínio.
- 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 [Ber96].

Sendo um sistema trasformacional orientado a domínios, o primeiro passo para a utilização do Draco é a construção dos domínios a serem utilizados. No caso da JMVCASE foram necessários os 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 [Lim97]. Estes domínios estão representado por:

- Um *parser* para a linguagem base 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 escrita na linguagem.

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 outras linguagens. Uma vez reconhecido o padrão LHS em uma especificação de entrada na linguagem RC ou LBE, esta transformação será automaticamente aplicada pelo Draco. Dessa forma, obtém-se a correspondente especificação na linguagem definida no RHS. No caso a linguagem alvo da implementação é java ou especificações em SQL que criam as bases de dados do sistema.

A seguir será apresentado um estudo de caso de um sistema de Locadora de Automóveis, que mostrará aspectos relacionados com a utilização da ferramenta JMVCASE.

4. Especificação e Implementação de Sistemas na JMVCASE - Estudo de Caso

Para mostrar o uso da JMVCASE, será apresentado um exemplo de um sistema de locadora de automóveis. Neste sistema os clientes são cadastrados no momento do primeiro aluguel. Quando o cliente escolhe o automóvel desejado, dentre os disponíveis, registra-se o seu aluguel. Quando o cliente devolve o automóvel, é cobrado o aluguel em função dos quilômetros rodados e dos dias em que permaneceu com o automóvel. Ao mesmo tempo é removida a instância do aluguel e é mudada a situação do carro para "disponível". Caso o cliente não possa pagar, a dívida do aluguel é registrada junto ao cliente.

Para usar a JMVCASE, inicialmente o desenvolvedor seleciona na JavaRC 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 de todas as técnicas deste método. Por exemplo, selecionado o método UML e a técnica Diagrama de

Classes desse método, a edição dos atributos das classes é feita como mostra a tela da Figura 4. A edição dos atributos da classe permite a definição do Nome e do Tipo do atributo, sua Visibilidade, e outras características opcionais que procuram cobrir as necessidades da programação orientada a objetos em java. Esta tela disponibiliza os tipos básicos da linguagem java, e tipos especiais para que o desenvolvedor possa escolher outros tipos correspondentes às classes de prateleira e definidas pelo desenvolvedor. Os serviços de uma determinada classe são editados como mostra a Figura 5. Para cada serviço o desenvolvedor especifica seu Tipo de Retorno, seus Parâmetros e sua Visibilidade.

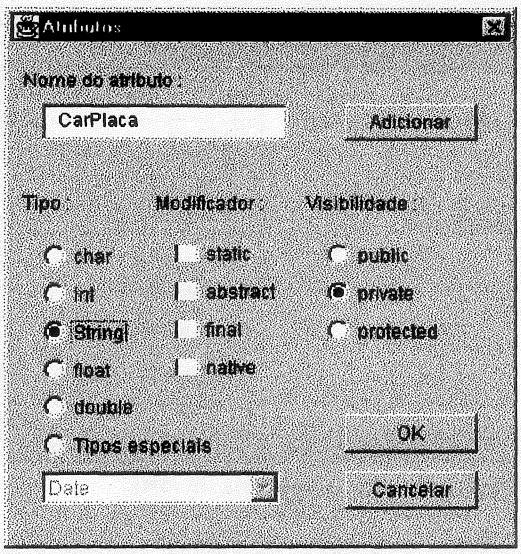

Figura 4 – Edição de Atributos na JavaRC

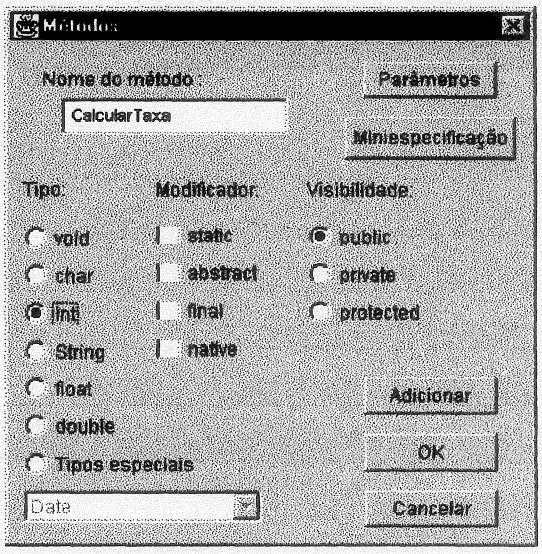

Figura 5 – Edição de Serviços na JavaRC

Para cada serviço editado o desenvolvedor pode definir sua miniespecificação na linguagem LBE. Na definição dos serviços das classes, o desenvolvedor interage com um novo diálogo que permitirá a sua descrição na linguagem LBE, como na Figura 6.

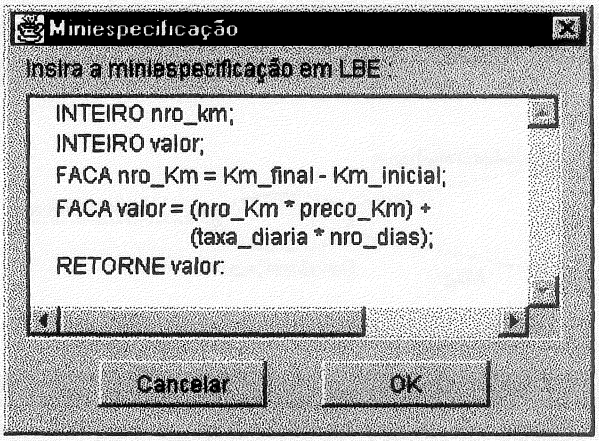

Figura 6 – Miniespecificação do Serviço *CalcularTaxa* da Classe *Carro*

A JavaRC, no caso do Diagrama de Classes do método UML, dispõe de componentes para representar as classes, agregações, associações, heranças e dependências através da seleção do item desejado na respectiva

barra de ferramentas. As definições das ocorrências de objetos de uma classe em relação a outras, são representadas pelas cardinalidades, junto às estruturas. A Figura 7 apresenta o modelo obtido da locadora, utilizando a técnica de Diagrama de Classes do método UML.

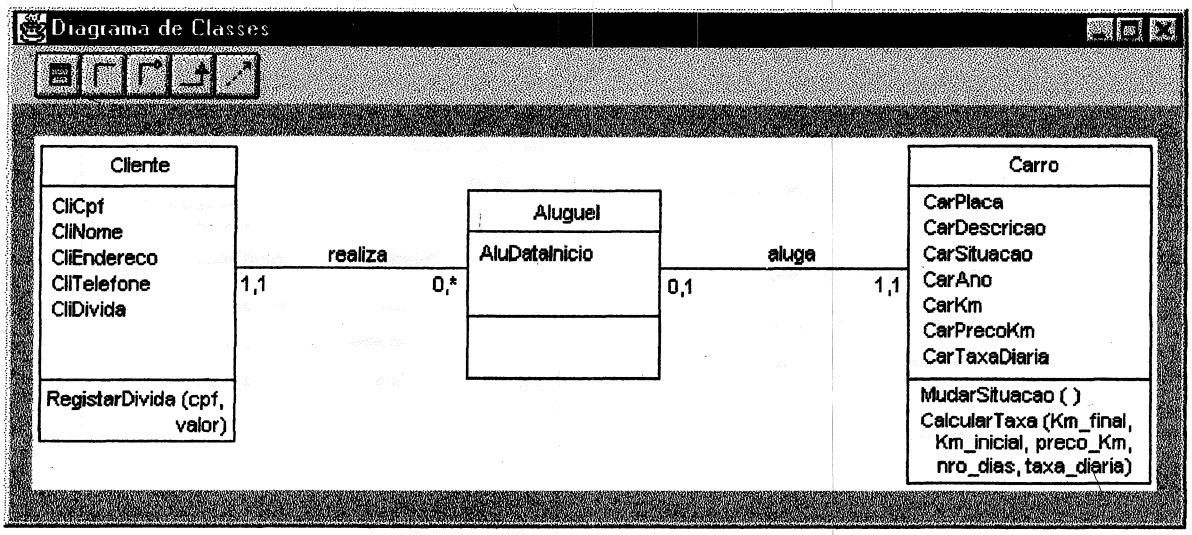

Figura 7 – Modelagem do Sistema de Locadora de Automóveis na JavaRC

A JMVCASE dispõe das outras técnicas que completam a modelagem do sistema segundo o método escolhido. Por exemplo, a Figura 8 mostra a modelagem do Diagrama de UseCase para a atividade *DevolverCarro*. Esse diagrama pode ser especificado na JMVCASE da mesma forma que foi feito com o Diagrama de Classes.

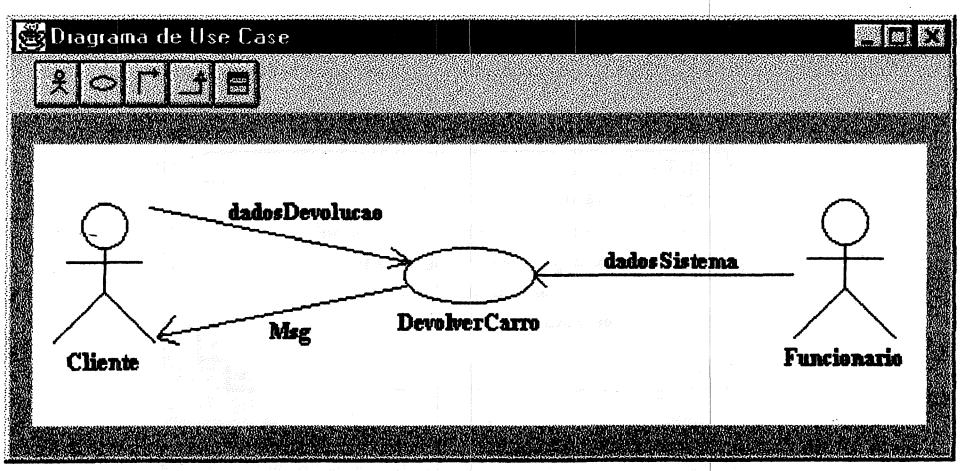

Figura 8 – Modelagem do Diagrama de UseCase para a Atividade *DevolverCarro*

Modelos do sistema orientado a objetos, usando as técnicas apresentadas, são armazenados, em descrições nas linguagens RC e LBE, pela ferramenta JavaRC. 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 9 pode-se identificar a descrição RC das técnicas utilizadas durante a modelagem e a descrição LBE dos serviços das classes.

```
PROJECT <Locadora_de_Automoveis>
MODEL <Diagrama_de_Classes> METHOD <UML>
DESCRIPTION
ENTITY Carro
ATTRIBUTE CarPlaca PARTOF ENTITY Carro HASVALUE INFORMATION String
ATTRIBUTE CarDescricao PARTOF ENTITY Carro HASVALUE INFORMATION string
PROCESS CalcularTaxa PARTOF ENTITY Carro HASVALUE INFORMATION int
ATTRIBUTE Km_final PARTOF PROCESS Recebe HASVALUE INFORMATION int
{{ expression lbej prog_comp
BEGIN
   INTEIRO nro_Km;
   INTEIRO valor;
   FACA nro_Km = Km_final - Km_inicial;
   FACA valor = (nro_Km *  preco_Km) + (taxa_diaria *  nro_dias);
   RETORNE valor;
END
}}
CONSTRAINT realiza
CONSTRAINT 1,1 PARTOF ENTITY Aluguel OPERAND ENTITY Cliente
CONSTRAINT 0,N PARTOF ENTITY Cliente OPERAND ENTITY Aluguel
END
GRAPHIC
ENTITY Carro HASVALUE POSITION 85,101
ENTITY Cliente HASVALUE POSITION 341,98
ENTITY Aluguel HASVALUE POSITION 236,243
CONSTRAINT aluga HASVALUE POSITION 105,101 317,101
CONSTRAINT realiza HASVALUE POSITION 341,128 260,246
END
END
MODEL <Diagrama_de_UseCase> METHOD <UML>
DESCRIPTION
ENTITY Cliente PARTOF ENTITY ATOR
ENTITY Funcionario PARTOF ENTITY ATOR
PROCESS DevolverCarro
MESSAGE dadosDevolucao PARTOF ENTITY Cliente
PROCESS DevolverCarro RECEIVES MESSAGE dadosDevolucao
END
END
END
```

Figura 9 – Trecho da Descrição RC e LBE do Sistema de Locadora de Automóveis

O primeiro trecho destacado na Figura 9 mostra como foram tratadas as descrições dos dois domínios RC e LBE num único programa. A descrição se 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 da linguagem 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[Fre96].

Na Figura 9 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.

A descrição em RC e LBE, gerada pela JavaRC, será utilizada pelo Draco para a geração do código em linguagem executável. Uma vez obtida a descrição de um sistema pode-se executar as transformações dos domínios RC e LBE, encapsuladas no Draco, que transformam esta descrição para java ou em outra linguagem disponível no Draco. Após aplicadas as transformações, sobre a descrição do modelo do sistema de Locadora de Automóveis, obtém-se o código correspondente em java. A Figura 10 mostra parte deste código gerado.

```
class Carro
{   String CarPlaca;
    String CarDescricao;
    int CarSituacao;
    int CarAno;
    float CarKm;
    float CarPrecoKm;
    float CarTaxaDiaria;

    public int CalcularTaxa (int Km_final, int Km_inicial, int preco_Km,
                             int nro_dias,  int taxa_diaria)
     {  int nro_Km, valor;
        nro_Km = Km_final - Km_inicial;
        valor = (nro_Km * preco_Km) + (taxa_diaria * nro_dias);
        return valor;
     }
// outros serviços
}
```

Figura 10 – Código Executável em Java Gerado para a Classe *Carro*

A qualquer instante da fase de especificação de requisitos, na JavaRC, o desenvolvedor pode salvar os modelos criados, gerando a respectiva descrição em RC e LBE. Esta descrição pode ser transformada em um programa executável em linguagem java, gerando um protótipo do sistema. Este protótipo pode ser executado e testado verificando assim a funcionalidade do sistema modelado. Novas versões do sistema podem ser obtidas com a evolução do protótipo, incluindo novas especificações, até obter uma versão final do sistema implementado.

O código java gerado pode ser utilizado pela ferramenta Visual Café Pro, integrada ao JMVCASE. A Figura 11 mostra uma das telas criadas no Visual Café Pro para a construção gráfica do sistema. O código java da interface é incorporado no código java gerado pela JMVCASE. Por exemplo, no sistema da locadora o desenvolvedor instancia a classe *Carro* e usa o serviço *CalcularTaxa* criado e especificado na JavaRC. Além do código java, foi gerada a base de dados para o sistema da locadora. Para acessar a base de dados o desenvolvedor deve utilizar os componentes visuais já disponíveis na ferramenta Visual Café Pro. Por exemplo, pode-se usar componentes que encapsulam comandos na linguagem SQL ou que fazem o acesso direto à base de dados.

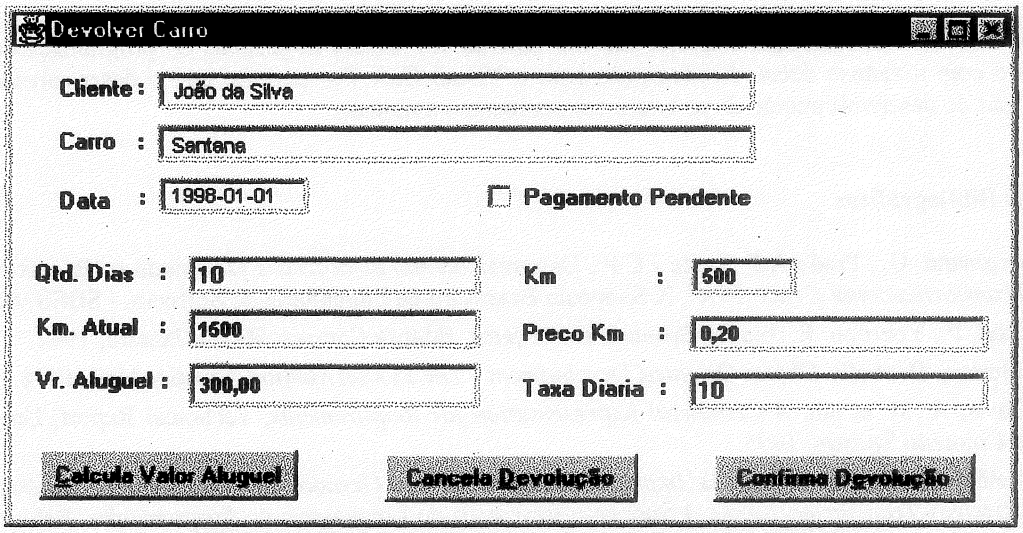

Figura 11 – Interface do Sistema de Locadora de Automóveis

5. Conclusões

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 gráfica integrada a um sistema transformacional de software.

Uma característica importante da JMVCASE 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 para Requisitos. A abordagem de múltiplas visões possibilita o uso integrado de diferentes métodos orientados a objetos na especificação de um sistema, auxiliando o usuário no processo de desenvolvimento e manutenção de sistemas. 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. Dessa forma, pode-se também reutilizar especificações de um sistema para outro. Equipes com conhecimento de métodos diferentes poderão integrar seus trabalhos usando a capacidade das múltiplas visões da JavaRC.

Outra característica importante desta pesquisa é a integração do sistema transformacional Draco na ferramenta. A integração da JavaRC com o Draco, através das linguagens RC e LBE definidas no Draco, facilitou a automatização do processo de desenvolvimento de software orientado a objetos e proporciona um alto grau de reuso da análise dos requisitos, aumentando a produtividade e facilitando a manutenção. Nesta integração foram utilizados recursos do sistema Draco, que permitem a transformação de programas escritos em uma linguagem para outros programas correspondentes, na mesma ou em novas linguagens. Esta integração permite ainda a geração automática de código em diferentes linguagens de programação dos domínios disponíveis no Draco, como java e SQL.

A combinação dos domínios das linguagens 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. Outra contribuição vem da exploração da tecnologia de transformação de software orientada a domínios.

Uma comparação da ferramenta JMVCASE com outras ferramentas CASE semelhantes no mercado, como FusionCASE [Fus95] e RationalRose [Rat97], mostra um ponto importante que diferencia MVCASE das outras ferramentas CASE, sua integração com o sistema transformacional Draco e com a ferramenta Visual Café Pro. Sua integração com o Draco permite a geração automática de código em qualquer linguagem definida no

Draco e permite ao desenvolvedor implementar um sistema em muitas linguagens de programação sem estar familiarizado com a sintaxe delas. A integração com o Visual Café Pro permite que o desenvolvedor realize todas as etapas do desenvolvimento de software em um mesmo ambiente.

Referências Bibliográficas

[Ber96] 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 - SBES 96, 1996.

[Coa92] Coad, P., Yourdon, E. *Análise Baseada em Objetos*, Editora Campus, Rio de Janeiro, 1992.

[Col94] Coleman, D. et alli. *Object-Oriented Development - The Fusion Method*, Prentice Hall, 1994.

[Dav95] Davis, A. M. et alli. *A Canonical Representation for Requirements*, Technical Report, University of Colorado at Colorado Springs, 1995.

[Fre96] 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, Belo Horizonte, 1996.

[Fus95] SoftCASE Consulting. *FusionCASE* – SPV, Version 1.3.1, 1995.

[Kir96] 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, São Carlos, 1996.

[Lei94] Leite, J. C. et alli. *Draco-PUC: A Technology Assembly for Domain Oriented Software Development*, III ICSR-IEEE, 1994.

[Lei95] Leite, J. C.; Prado, A. F.; Santana, M.; e Freitas, F. *O Uso do Paradigma Transformacional no Porte de Programas Cobol*, IX Simpósio Brasileiro de Engenharia de Software - SBES 95, 1995.

[Lim97] 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.

[Nei84] Neighbors, J. *Software Construction Using Components*, Tese de Doutorado, University of California at Irvine, 1984.

[Pra92] Prado, A. F. *Estratégia de Reengenharia de Software Orientada a Domínios*, Tese de doutorado, PUC-RJ, 1992.

[Pra96] 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.

[Rat97] *Using Rational Rose*, www.rational.com, 1997.

[Sun97] Sun Microsystems. *Tutoriais Java*, www.javasun.com, 1997.

[Sym97] Symantec Corporation. *Visual Café Pro*, www.symantec.com, 1997.

[Uml97] *UML Document Set*, www.rational.com/uml, 1997.