Ambiente "CASE" com Múltiplas Visões de Requisitos e Implementação Automática utilizando o Sistema Transformacional Draco[1]

Maria Adriana Vidigal de Lima
Tathiana da Silva Barrére
Antônio Francisco do Prado
André Alves C. R. do Couto

Universidade Federal de São Carlos
Departamento de Computação
Caixa Postal 676 - CEP: 13565-905 São Carlos - SP
e-mail: draco@dc.ufscar.br

Resumo

Este artigo apresenta um ambiente CASE orientado a objetos, que suporta múltiplas visões de requisitos em diferentes técnicas de métodos orientados a objetos, visando facilitar a análise e a modelagem de um sistema, utilizando uma Representação Canônica para requisitos. O ambiente integra uma ferramenta com editor gráfico e textual, que suporta múltiplas visões de requisitos, e um sistema transformacional de software, que permite a geração automática de código em C++, a partir de especificações em alto nível de abstração.

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

Abstract

This article describes an object-oriented CASE environment that supports multiple views of requirements, in different techniques of object-oriented methods, aiming to facilitate the system analysis and modeling, using a Canonical Representation for requirements. The environment integrates a graphical and textual interface, that supports multiple views of requirements, and a transformational system of software, that allows automatic C++ code generation from specifications in high level abstraction.

Keywords: Software Engineering, Software Developing Environment, Transformational Systems, Canonical Representation for Requirements, CASE Tools.

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.

[1] Projeto financiado parcialmente pela FAPESP (Proc. 95/9892-1) e CNPq

Uma ferramenta CASE eficiente deve:

- prover várias metodologias e possibilitar o intercâmbio entre elas durante o desenvolvimento de um sistema;
- verificar erros lógicos;
- ser simples de usar;
- permitir a modelagem de todas as técnicas de especificação de requisitos, inclusive miniespecificação do comportamento dos objetos;
- possibilitar a modelagem de sistemas grandes;
- gerar código.

Para investigar o processo de desenvolvimento de software orientado a objetos desde a análise até a implementação, foi desenvolvido um ambiente CASE Orientado a Objetos (CASE OO), com suporte a múltiplas visões dos requisitos e implementação automática. Neste ambiente, o desenvolvedor pode projetar o sistema como um todo, com poucos detalhes e mais tarde refiná-lo, ou pode se concentrar em partes do sistema, criando porções reusáveis, que juntas formarão o sistema. Em qualquer ponto do desenvolvimento, pode-se gerar o código correspondente ao sistema ou a parte dele.

Este artigo apresenta o ambiente CASE OO da seguinte forma: A seção 2 apresenta uma visão geral sobre o ambiente. Na seção 3 é apresentado o ambiente CASE com múltiplas visões composto pela ferramenta gráfica e textual, usada para a especificação dos requisitos, e as linguagens internas do ambiente. Na seção 4 é apresentado o sistema transformacional Draco integrado no ambiente. Para ilustrar a utilização do ambiente integrado, é apresentado na seção 5 um estudo de caso sobre um sistema de Locadora de Automóveis. Na seção 6 é apresentada uma comparação com outros trabalhos e finalmente na seção 7 são apresentadas as conclusões deste trabalho.

2. Visão Geral do Ambiente CASE Orientado a Objetos

A figura 1 fornece uma visão geral do ambiente CASE OO, que possibilita o desenvolvimento de sistemas de software orientados a objetos, a partir dos requisitos até a implementação. Possui uma ferramenta gráfica e textual, denominada ToolRC, que está integrada a um sistema transformacional de software, denominado Draco. O sistema Draco implementa as idéias do paradigma de desenvolvimento de software orientado a domínios primeiramente proposto por Neighbors [12], e pesquisado por Prado [14] e Leite [9, 10, 11].

A ferramenta ToolRC dispõe de três métodos de desenvolvimento de software orientados a objetos: Coad/Yourdon [3], OMT [16] e Fusion [4]. Cada sistema modelado é armazenado em uma descrição textual, segundo linguagens definidas no sistema Draco. Através desta descrição, o desenvolvedor pode obter uma nova visão do mesmo sistema segundo outro método orientado a objetos disponível na ToolRC. Esta nova visão exibe o conjunto de todos os requisitos que têm correspondente no novo método escolhido pelo desenvolvedor.

A linguagem para 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 [5]. 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).

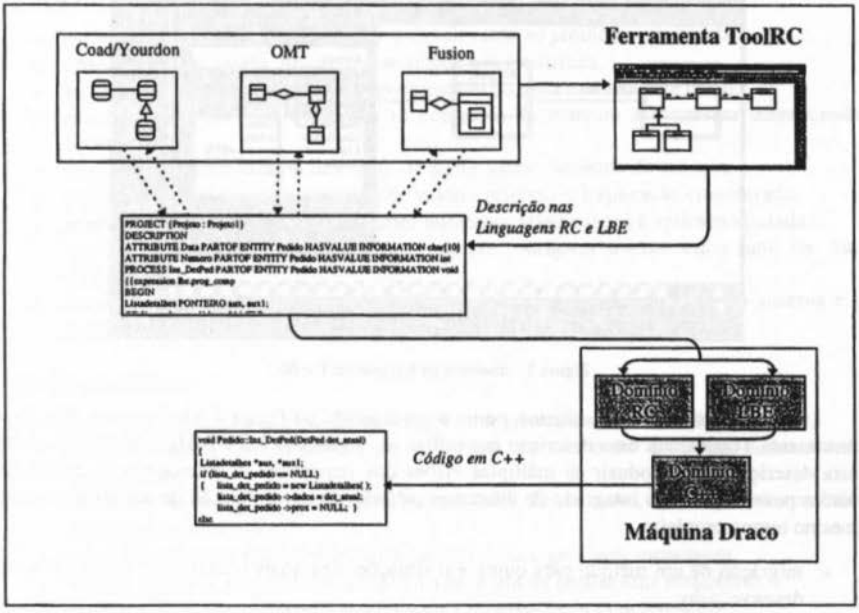

Figura 1 - Integração da Ferramenta ToolRC com o sistema Draco

Para gerar o código correspondente, a máquina Draco analisa a descrição textual do sistema nas linguagens RC e LBE, e aplica transformações sobre esta descrição, transformando-a em linguagem executável, como C++. Desta forma, obtém-se a implementação automática do sistema, a partir das especificações em alto nível de representação.

3. O Ambiente CASE com Múltiplas Visões

A ferramenta ToolRC provê uma interface gráfica e textual, para a modelagem de sistemas orientados a objetos. A ToolRC permite construir modelos dos métodos Coad/Yourdon, OMT e Fusion. O desenvolvedor escolhe um determinado método e então modela o sistema desejado segundo as técnicas deste método. As classes do sistema são mostradas no modelo sem os seus atributos e serviços, para não sobrecarregar o diagrama com detalhes que podem ser vistos separadamente. Os atributos e os serviços de determinada classe são consultados na janela Propriedades, que mostra ainda o tipo do atributo ou o retorno do serviço, como se pode ver na figura 2.

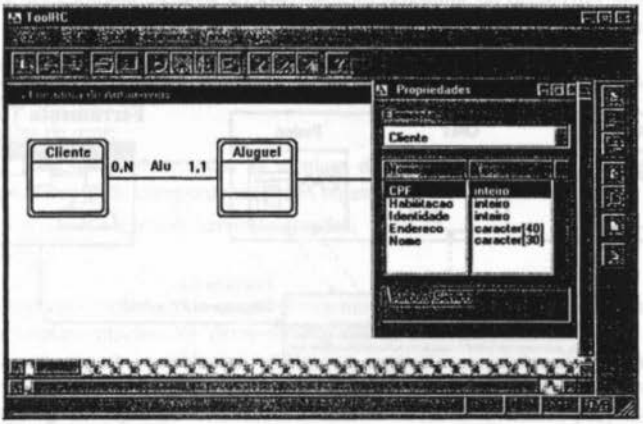

Figura 2 - Interface da ferramenta ToolRC

Modelos orientados a objetos, como o apresentado na figura 2, são armazenados pela ferramenta ToolRC em uma descrição que utiliza as linguagens RC e LBE. A ToolRC 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.

3.1. Representação Canônica para Requisitos de Software e Linguagem RC

A Representação Canônica (RC) para requisitos de software 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 $te_i \in E$ [8].

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 um 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.

Baseada na RC foi definida uma linguagem, denominada linguagem RC, para a especificação de requisitos. A linguagem RC é expressa pelas combinações entre os elementos e os relacionamentos, e servirá para armazenar as informações relativas aos sistemas.

3.2 Linguagem LBE

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 [15]. Desta forma, o desenvolvedor pode completar a especificação dos modelos de objetos com as miniespecificações dos serviços de cada classe. A figura 3 mostra um exemplo de miniespecificação usando a linguagem LBE.

```
FUNCAO Cliente:: insere (INTEIRO cod);
 Listacliente PONTEIRO aux;
 Listacliente PONTEIRO aux1;
 FACA codigo = cod;
 SE lista_cliente = VAZIO
     ENTAO FACA lista_cliente = NOVO Listacliente;
           FACA lista_cliente->dados = OBJETO_ATUAL;
           FACA lista_cliente->prox = VAZIO;
     SENAO FACA aux = lista_cliente;
           ENQUANTO aux->prox DIFERENTE_DE VAZIO FACA
             FACA aux = aux->prox;
           FIM_ENQUANTO
           FACA aux1 = NOVO Listacliente;
           FACA aux1->dados = OBJETO_ATUAL;
           FACA aux1->prox = VAZIO;
           FACA aux->prox = aux1;
 FIM_SE
FIM_FUNCAO
```

Figura 3 - Miniespecificação de serviço em LBE

As linguagens RC e LBE serviram de ligação entre a ferramenta ToolRC e o sistema Draco, tornando possível o desenvolvimento do ambiente CASE OO. A descrição textual nas linguagens RC e LBE é gerada automaticamente pela ToolRC após a especificação de um sistema. A linguagem RC armazena o modelo de objetos do sistema, independentemente do método utilizado na modelagem, e a LBE armazena o pseudocódigo definido pelo desenvolvedor para cada serviço 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.

4. O Sistema Transformacional Draco

O sistema transformacional Draco apresentado por Neighbors [12] e reconstruído no Departamento de Informática da PUC-RJ [9, 10, 11, 14], objetiva desenvolver, colocar em prática e testar o paradigma transformacional para o desenvolvimento de software orientado a domínios. 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. 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) [2,9,10]. 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.

4.1 Domínio RC

Baseado na proposta de uma Representação Canônica para requisitos, foi criado o domínio RC no sistema Draco. A linguagem do domínio RC permite a escrita de especificações de requisitos na forma canônica. O domínio desenvolvido no Draco para a RC está representado:

- Pela linguagem RC para a escrita de especificações de requisitos;
- Pelo *prettyprinter* para escrever a especificação na linguagem RC, a partir da representação interna do domínio RC (DAST);
- Por um conjunto de transformações que realiza a geração do código fonte do sistema na linguagem C++ a partir da especificação escrita na linguagem RC.

Uma vez construído o modelo de objetos, segundo determinado método orientado a objetos, na ferramenta ToolRC, e persistido numa descrição em linguagem RC, aplicam-se as transformações da biblioteca de transformações do domínio RC sobre esta descrição para obter o correspondente código C++. Dentre as transformações do domínio RC, destacam-se aquelas cujos objetivos são:

1. Incluir a definição das classes do modelo nos arquivos de implementação dessas classes, através do comando *include*;
2. Construir a função principal (main) do programa C++;
3. Definir cada nova classe do modelo de objetos;
4. Implementar estruturas de herança, e outros relacionamentos entre as classes;
5. Definir cada atributo de uma classe e o seu tipo; e
6. Definir cada serviço de uma classe, com seus argumentos e o objeto de retorno.

A figura 4 mostra um exemplo de uma transformação do domínio RC que identifica um atributo, seu tipo e sua classe, e gera o código correspondente em C++. A transformação possui o nome **IdentificaAtributoTipo1**, e é composta por um padrão de reconhecimento (LHS), o ponto de controle POST-MATCH, e um padrão de substituição (RHS). O LHS contém a regra da especificação a ser encontrada, **composeattrib**, definida no *parser* RC, seguida da especificação RC correspondente com as suas metavariáveis I,Y,X do tipo ID (identificador). O ponto de controle POST-MATCH permite que sejam realizadas operações sobre as metavariáveis, preparando-as para que o padrão de substituição seja aplicado. O RHS, encapsulado no template **GeraDefAtrib**, contém a regra de substituição, **member_declaration**, definida no *parser* do domínio C++, seguida da especificação de substituição que contém duas metavariáveis, TIPO e NATRIB com seus respectivos tipos.

```
TRANSFORM IdentificaAtributoTipo1

LHS: {{dast rc.composeattrib
        ATTRIBUTE [[ID I]] PARTOF ENTITY [[ID Y]] HASVALUE INFORMATION [[ID X]]
     }}
POST-MATCH: {{ dast cpp.statement_list
     char NomeAtrib[150], strRetorno[150], tmp[150], tmp1[150];
     char NomeTipo[150];
     COPY_LEAF_VALUE_TO(NomeAtrib , "I");
     COPY_LEAF_VALUE_TO(NomeClasse , "Y");
     COPY_LEAF_VALUE_TO(NomeTipo , "X");
        APPLY("CriaWorkspaces", "Y");
     strcpy(strRetorno , NomeTipo);
     strcpy(tmp, NomeClasse);
     strcat(tmp,"MembersProtected");
     TEMPLATE("GeraDefAtrib");
           SET_TEMPL_LEAF_VALUE("TIPO" , strRetorno);
           SET_TEMPL_LEAF_VALUE("NATRIB" , NomeAtrib);
           PLACE_AT(tmp);
     END_TEMPLATE;
}}

TEMPLATE GeraDefAtrib
   RHS: {{ dast cpp.member_declaration
         [[type_specifier TIPO]] [[IDENTIFIER NATRIB]] ;
}}
```

Figura 4 - Um exemplo de transformação da linguagem RC para C++.

Uma vez reconhecido o padrão LHS em uma especificação de entrada na linguagem RC, esta transformação será automaticamente aplicada pelo Draco. Dessa forma, obtém-se a correspondente especificação na linguagem C++ definida no RHS.

Da mesma forma foi construído o domínio da linguagem LBE, com seu *parser*, *prettyprinter*, e um conjunto de transformações que fazem a implementação das descrições dos serviços, em pseudocódigo LBE, na linguagem C++.

4.2 Domínio LBE

Utilizando o Draco, o desenvolvedor pode implementar automaticamente os serviços, descritos em pseudocódigo, na linguagem C++. A linguagem de pseudocódigo LBE permite que o desenvolvedor trabalhe em um nível mais abstrato na especificação das atividades de um sistema.

Na biblioteca de transformações do domínio LBE, podem ser identificadas transformações globais, e transformações de comandos e expressões. Devido às transformações globais, este transformador pode ser aplicado também em programas completos escritos em pseudocódigo, e não apenas em métodos separadamente. A figura 5 mostra dois exemplos de transformações, bem simples, do domínio LBE para o domínio C++. O primeiro exemplo mostra a transformação **Decisão**, na qual o LHS identifica o comando de seleção SE, seguido de condições e comandos em LBE e o RHS define a especificação correspondente do C++. No segundo exemplo, o LHS da transformação **Cópia** identifica o

comando COPIA seguido de parâmetros, e o RHS define a especificação de substituição no domínio C++. As transformações LBE, juntamente com as transformações RC completam o processo de implementação de um sistema.

```
Transform Decisao
 LHS: {{dast lbe.comando
  SE [[condicao cond]] ENTAO [[comando *cmds]] FIM_SE
 }}
 RHS: {{dast cpp.statement_list
  if ( [[expression cond]] ) { [[dstatement *cmds]] }
 }}

Transform Copia
 LHS: {{dast lbe.comando
  COPIA ( [[parametro *param]] );
 }}
 RHS: {{dast cpp.statement_list
  strcpy ( [[assignment_expression *param]] );
 }}
```

Figura 5 - Exemplo de transformações da linguagem LBE para C++.

A seguir será apresentado um estudo de caso de um sistema de Locadora de Automóveis, que mostrará aspectos relacionados com a utilização do ambiente de múltiplas visões e com a implementação automática.

5. Modelagem de Requisitos no Ambiente - Estudo de Caso

Trata-se de um sistema para uma Locadora de Automóveis que aluga carros ou motos. 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.

Para usar o ambiente, 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, selecionado o método Coad/Yourdon a edição das classes é feita como mostra a tela da figura 6. Entra-se com o nome da classe, *Automóvel* no caso, com os atributos *Ano, Código, Descrição, KM, Modelo, Taxa, Preço_Km* e *Situação* e os serviços encapsulados na classe. O campo de comentários é destinado à colocação de informações sobre a classe. A edição dos atributos da classe permite a definição do Tipo do atributo, seu Tamanho, e outras características opcionais que tentam cobrir as necessidades da programação orientada a objetos.

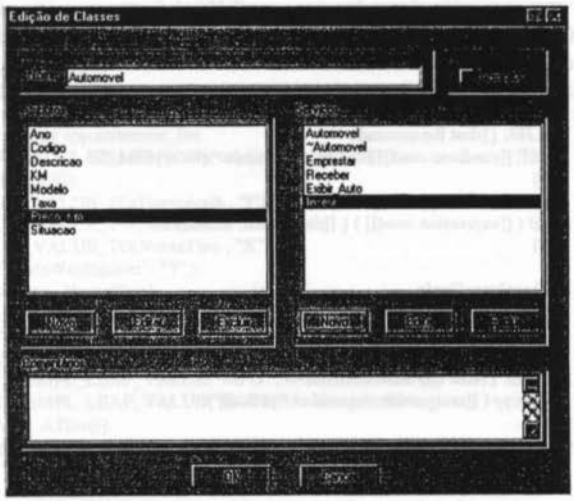

Figura 6 - Edição de classes na Ferramenta ToolRC

Os serviços de uma determinada classe são editados como mostra a figura 7. Para cada serviço definido na classe, o desenvolvedor pode especificar seu Tipo do Retorno e seus Parâmetros (nome e tipo). Para a miniespecificação do serviço, o desenvolvedor interage com um novo diálogo que permitirá a sua descrição na linguagem LBE. A figura 8 mostra um exemplo de miniespecificação.

Figura 7 - Edição de Serviços da Classe

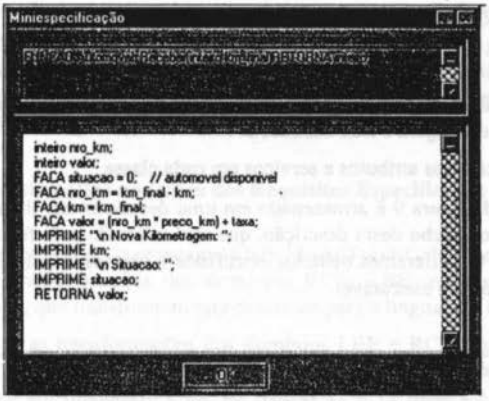

Figura 8 - Janela para a edição da miniespecificação de um serviço

Para a construção de um modelo de objetos, a ferramenta ToolRC permite a criação de classes, com seus atributos e serviços, e de estruturas de conexão de ocorrência e mensagem, e herança, como no caso do sistema de Locadora de Automóveis, modelado segundo o método Coad/Yourdon, conforme a figura 9.

Os atributos e serviços de cada classe podem ser vistos através da opção Propriedades do menu Exibir, que apresenta uma janela onde o desenvolvedor seleciona a classe desejada. Para cada classe selecionada são exibidos seus atributos (nomes e tipos), e seus serviços (nome, parâmetros e tipos dos parâmetros). As definições das ocorrências de objetos de uma classe em relação a outras, são representadas pelas cardinalidades, junto às estruturas.

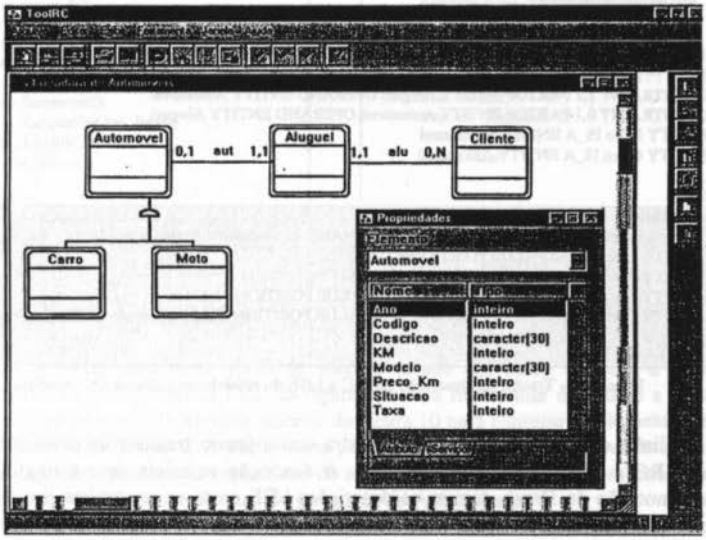

Figura 9 - Modelo em Coad/Yourdon do Sistema de Locadora de Automóveis

Algumas consistências são realizadas durante a construção dos modelos. Dentre elas destacam-se:

- a integridade das estruturas, que exigem sempre uma classe de origem e uma de destino;
- a obrigatoriedade da cardinalidade quando exigida pela estrutura, devendo ser colocada para a classe de origem e a de destino; e
- as dependências dos atributos e serviços em cada classe.

O modelo da figura 9 é armazenado em uma descrição nas linguagens RC e LBE. A figura 10 mostra um trecho desta descrição, que será utilizado pela ferramenta ToolRC, para obter novas visões em diferentes métodos orientados a objetos, e pelo Draco, para a geração do código em linguagem executável.

```
PROJECT <Locadora de Automoveis>
DESCRIPTION
ENTITY Automovel
ATTRIBUTE Situacao PARTOF ENTITY Automovel HASVALUE INFORMATION int
ATTRIBUTE KM PARTOF ENTITY Automovel HASVALUE INFORMATION int
ATTRIBUTE Ano PARTOF ENTITY Automovel HASVALUE INFORMATION int
ATTRIBUTE Modelo PARTOF ENTITY Automovel HASVALUE INFORMATION char[30]
PROCESS Receber PARTOF ENTITY Automovel HASVALUE INFORMATION int
{{
BEGIN
  inteiro nro_km;
  inteiro valor;
  FACA situacao = 0;  // carro disponivel
  FACA nro_km = km_final - km;
  FACA  km = km_final;
  FACA  valor = (nro_km+preco_km) + taxa;
  IMPRIME "\n Nova Kilometragem: ";
  IMPRIME km;
  IMPRIME "\n Situacao: ";
  IMPRIME situacao;
  RETORNA valor;
 END
}}
CONSTRAINT 1,1 PARTOF ENTITY Aluguel OPERAND ENTITY Automovel
CONSTRAINT 0,1 PARTOF ENTITY Automovel OPERAND ENTITY Aluguel
ENTITY Moto IS_A ENTITY Automovel
ENTITY Carro IS_A ENTITY Automovel
...
END
GRAPHIC
MODEL <OBJETOCOAD> METHOD <COAD> NAME <Locadora de Automoveis>
ENTITY Cliente HASVALUE POSITION 432,64
ENTITY Automovel HASVALUE POSITION 71,62
ENTITY Moto IS_A ENTITY Automovel HASVALUE POSITION 91,171
ENTITY Carro IS_A ENTITY Automovel HASVALUE POSITION 47,171
...
END
```

Figura 10 - Trecho da Descrição em RC e LBE do modelo do Sistema de Locadora

A linha destacada na figura 10 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[6].

5.1 Implementação Automática no Draco dos Requisitos Especificados

Uma vez obtida a descrição gerada pela ferramenta ToolRC do modelo de um sistema, pode-se executar as transformações dos domínios RC e LBE, encapsulados no sistema transformacional Draco, que transformam esta descrição para a linguagem C++.

Após aplicadas as transformações dos domínios LBE e RC, como as mostradas na figuras 4 e 5, sobre a descrição do modelo do sistema de Locadora de Automóveis, obtém-se o código correspondente em linguagem C++. As figuras 11 e 12 mostram parte deste código C++ gerado automaticamente pelo Draco.

```
#ifndef Automovel_H
#define Automovel_H
Class Automovel
{
 protected:
   int Ano;
   int Codigo;
   char Descricao[30];
   int KM;
   char Modelo[30];
   int Taxa
   int Preco_km;
   int Situacao
 public:
   Automovel();
   ~ Automovel ();
   int Emprestar();
   int Receber(int km_final);
   void Exibir_Auto();
   void Inserir();
 };
#endif
```

Figura 11 - Arquivo Automovel.h

```
#include " Automovel..h"
Automovel:: Automovel () {}
Automovel::~ Automovel () {}
int Automovel::Emprestar() {}
void Automovel::Exibir_Auto() {}
void Automovel::Inserir() {}

Int Automovel ::Receber(int km_final)
{
   int nro_km;
   int valor;
   Situacao = 0;   // carro disponivel
   nro_km = km_final - KM;
   KM = km_final;
   valor = (nro_km * Preco_km) + Taxa;
   cout << "\n Nova Kilometragem :";
   cout << KM;
   cout << "Situacao: ";
   cout << Situacao;
   return valor;
}
```

Figura 12 - Arquivo Automovel.cpp

5.2 Múltiplas Visões dos Requisitos

Para se obter uma nova visão do modelo criado em outro método, a ferramenta ToolRC utiliza a descrição RC e LBE da figura 10. Para redesenhar o modelo a ferramenta utiliza a descrição assinalada na parte inferior da figura 10 para capturar os elementos gráficos do modelo a ser transformado. A partir desta descrição, gera-se uma nova visão do modelo, com as correspondentes técnicas específicas do novo método. A figura 13 apresenta uma nova visão, no método OMT, para o sistema de Locadora de Automóveis.

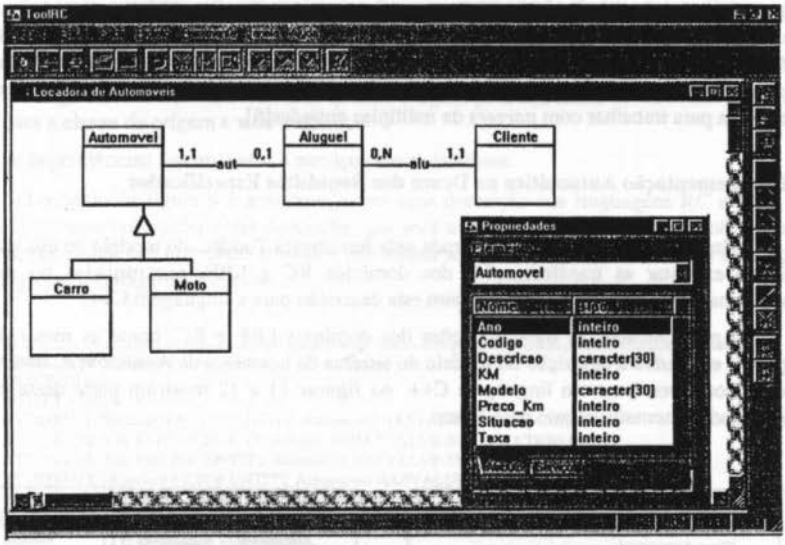

Figura 13 - Nova visão em OMT do Sistema de Locadora de Automóveis

A RC, com seus 9 elementos e 10 relacionamentos, facilitou a transformação entre as diferentes técnicas de representação dos requisitos orientados a objetos. A partir de uma descrição em RC, pode-se obter novas visões do modelo nos outros métodos orientados a objeto disponíveis na ferramenta ToolRC.

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.

6. Comparação com Outros Trabalhos

Hoje o mercado de ferramentas para desenvolvimento de software oferece opções como Paradigm Plus [1,13], Together C++ [1,17] e FusionCase [4,7] que permitem a representação de classes, e conexões entre elas, através de símbolos. Os símbolos utilizados pelas ferramentas representam a filosofia de uma determinada metodologia, e permitem uma visão global do sistema. As mudanças no sistema podem ser realizadas mais facilmente, pois a ferramenta CASE gera novamente o código, contendo as atualizações.

O ambiente CASE apresentado neste artigo, à semelhança destas e de outras ferramentas CASE, suporta o desenvolvimento de software provendo uma metodologia baseada em um conjunto de princípios que guiam o desenvolvedor na organização, projeto e construção de um sistema, facilitando sua manutenção.

Uma característica que diferencia o ambiente CASE OO de outras ferramentas CASE é que o ambiente apresentado suporta a modelagem em várias metodologias, proporcionando múltiplas visões dos requisitos através do uso de uma linguagem, baseada em uma Representação Canônica para Requisitos. O uso de uma Representação Canônica propicia a migração de um método para outro, bem como a integração de diferentes técnicas de especificação de requisitos de software. Outra vantagem da Representação Canônica vem da possibilidade de se reutilizar as especificações de um sistema na especificação de outro sistema similar.

Outra característica importante desta pesquisa é a integração do sistema transformacional Draco no ambiente. A integração da ToolRC com o Draco, através das linguagens RC e LBE definidas no Draco, facilitou a automatização do processo de desenvolvimento de software orientado a objetos, pois 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 não apenas em C++.

7. 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. O ambiente CASE separa os aspectos de interface, implementados na ferramenta ToolRC e os aspectos de manipulação simbólica, implementados por transformações realizadas pelo sistema transformacional Draco.

A abordagem de múltiplas visões possibilita o uso integrado de diferentes métodos orientados a objetos na especificação de um sistema, facilitando a análise e modelagem dos requisitos. 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.

A combinação ToolRC e Draco facilita o desenvolvimento de software orientado a objetos e proporciona um alto grau de reuso da análise dos requisitos, aumentando a produtividade e facilitando a manutenção.

O trabalho descrito faz parte de um projeto de pesquisa, que envolve ainda a geração automática de esquemas para Sistemas Gerenciadores de Bancos de Dados relacionais estendidos e orientados a objetos, a ampliação das funcionalidades da ferramenta ToolRC, a ampliação do ambiente para novos métodos de desenvolvimento de software orientado a objetos e a implementação automática em novas linguagens executáveis de outros domínios do Draco.

1. **Armbruster J. L.** *Comparing CASE Tools*, Dr. Dobb's Journal, vol. 20, número 6, p.76-86, Junho, 1995.

2. **Bergmann, U.** *Construção de um Domínio de Desenvolvimento de Software Orientado a Objetos Segundo o Paradigma Draco*. Dissertação (Mestrado), Instituto Militar de Engenharia, Rio de Janeiro, 1996.

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

4. **Coleman, D. et al.** *Object-Oriented Development - The Fusion Method*. Prentice Hall, 1994.

5. **Davis, A. M. et al.** *A Canonical Representation for Requirements*, 1995, Technical Report, University of Colorado at Colorado Springs, 1995.

6. **Freitas, F.G. et al.** *Aspectos Implementacionais de um Gerador de Analisadores Sintático para o Suporte a Sistemas Transformacionais*. I Simpósio Brasileiro de Linguagens de Programação, p.115-127, Belo Horizonte, 1996.

7. **FusionCASE - SPV, Version 1.3.1.** SoftCASE Consulting, 1995.

8. **Kirner, T. et al.** *Ambiente para Representação de Múltiplas Visões de Requisitos: O Metamodelo e uma Linguagem de Transformação*. X Simpósio Brasileiro de Engenharia de Software - SBES96, p. 207-222, São Carlos, 1996.

9. **Leite, J. C. et al.** *Draco-PUC: A Technology Assembly for Domain Oriented Software Development*, Third International Conference on Software Reuse - IEEE, Brazil, 1994.

10. **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, Recife, 1995.

11. **Leite, J.C.P. et al.** *Porting COBOL Programs Using a Transformational Approach*. Software Maintenance: Research and Practice, vol 9, p.3-31, 1997.

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

13. **Paradigm Plus 2.0 - Reference Manual**, Protosoft Inc., 1994.

14. **Prado, A. F.** *Estratégia de Reengenharia de Software Orientada a Domínios*, Tese de doutorado, PUC-RJ, Rio de Janeiro,1992.

15. **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.

16. **Rumbaugh, J. et al.** *Object Oriented Modeling and Design*, Prentice Hall, 1991.

17. **Together/C++ Professional 2.0**. Object International Software Ltd. Stuttgart, Alemanha, 1996.