Ferramenta "CASE" Orientada a Objetos com Múltiplas Visões e Implementação Automática [1]

Maria Adriana Vidigal de Lima
Antônio Francisco do Prado
Tathiana da Silva Barrére
André Alves C. R. do Couto
e-mail: draco@dc.ufscar.br

Universidade Federal de São Carlos - Departamento de Computação
Rodovia Washington Luiz, Km 235 Caixa Postal 676 13565-905 São Carlos-SP

Abstract

This paper presents an object-oriented CASE tool that integrates a graphical interface and a transformational system of software. The graphical interface supports multiple views of software requirements, in different techniques of object-oriented methods, and the transformational system allows automatic C++ code generation from specifications in high level abstraction.

Keywords: CASE tools, transformational systems, object-oriented software development.

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 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. Suportam desde a análise, projeto e construção de interfaces, até a implementação do sistema incluindo a persistência em banco de dados.

Hoje o mercado de ferramentas para desenvolvimento de software oferece opções que incluem principalmente ferramentas centradas em linguagens de programação e metodologias. Como exemplos de ferramentas centradas em linguagem, temos o Delphi[20], o Borland C++[19] e o Visual C++[14], que permitem a construção de interfaces gráficas e a programação orientada a objetos. Dentre as centradas em metodologias, vem ganhando destaque aquelas que utilizam o enfoque do paradigma da orientação a objetos, como FusionCase [4,7], MetaEdit[13], Paradigm Plus[1,16], e Rational Rose[22].

Motivado pela idéia de investigar o processo de desenvolvimento de software orientado a objetos, utilizando diferentes técnicas de representação dos requisitos, foi desenvolvida uma ferramenta CASE Orientada a Objetos (OO), com suporte a múltiplas visões de requisitos e implementação automática. Utilizando esta ferramenta, 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 obter o código em linguagem executável, correspondente ao sistema ou a uma parte dele.

Este artigo está organizado da seguinte forma: Na seção 2 é apresentada uma visão geral sobre a ferramenta desenvolvida, denominada ToolRC. Na seção 3 são apresentadas uma representação canônica para requisitos, e as linguagens utilizadas para a persistência dos modelos produzidos. Na seção 4 é apresentada a obtenção das múltiplas visões e a implementação em C++ dos requisitos modelados na ToolRC. Na seção 5 é apresentado um estudo de caso sobre um sistema de Distribuidora de Produtos modelado na ToolRC. Finalmente na seção 6 são apresentadas as conclusões deste trabalho.

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

2. A Ferramenta ToolRC - Visão Geral

A ferramenta ToolRC possibilita o desenvolvimento de sistemas de software orientados a objetos, desde a análise dos requisitos até a implementação, por estar integrada a um sistema transformacional de software, denominado Draco. Uma versão inicial da máquina Draco, que implementa as idéias do paradigma de desenvolvimento de software orientado a domínios, foi construída por Neighbors[15]. Posteriormente, num trabalho de reengenharia, a máquina Draco foi reconstruída na PUC-RJ, e uma 1ª versão foi obtida como resultado da tese de doutorado de um dos autores do presente artigo[17]. O paradigma Draco tem sido pesquisado por Leite, Sant'Anna e Freitas[11,12], e a versão atual da máquina Draco dispõe de mecanismos para realizar transformações complexas entre os diferentes domínios.

A ToolRC suporta a modelagem gráfica e textual dos requisitos do sistema, segundo técnicas da orientação a objetos, armazenando a especificação dos requisitos em uma descrição em linguagens definidas no Draco. A partir de uma descrição, utiliza-se o Draco para obter a implementação automática do sistema. A ToolRC pode ainda apresentar outras visões do sistema, segundo outros métodos orientados a objetos, o que possibilita o uso integrado de diferentes métodos na especificação de um sistema.

A linguagem para a persistência dos modelos de representação de requisitos de software, baseou-se numa Representação Canônica (RC) para requisitos, proposta por Alan Davis[5, 6, 9]. Além da RC, foi também utilizada, para completar a especificação dos modelos, a Linguagem Básica de Ensino (LBE), para a mini-especificação dos serviços contidos em cada classe do modelo.

3. Representação Canônica para Requisitos de Software

A Representação Canônica (RC) para requisitos de software é uma estrutura que permite analisar e armazenar requisitos, independentemente do método de especificação empregado. É composta de um conjunto de elementos $E=\{e_1,e_2,...\}$ e de um conjunto de relações $R=\{r_1,r_2,...\}$, onde cada relação r conecta um par ordenado de elementos $e,f \in E$, não 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: $(rt_i , rl_i, se_i , te_i)$ onde 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 os elementos inicial e final do relacionamento, $se_i \in E$ e $te_i \in E$ [10]. As Tabelas 1 e 2 [7] contém as representações gráficas e definições sucintas dos elementos e relacionamentos listados.

Nome	Representação Gráfica	Definição
Entidade	*nome_da_entidade*	Representa algo existente no mundo real, que seja relevante para o problema em questão.
Processo	*nome_do_processo*	Representa uma ação, tarefa, função ou atividade a ser realizada.
Estado	*nome_estado*	Representa um modo no qual o problema/sistema se comporta de maneira a conservar determinadas características.
Mensagem	< *mensagem* >	Algo que está sendo movimentado de um elemento para outro.
Atributo	{ *nome_do_atributo* }	Característica ou descrição de outro elemento do modelo.
Predicado	*prep.ou op.booleano*	Representa uma preposição ou um operador booleano, relacionados à aplicação tratada.
Restrição	*prep.ou op.booleano*	Define um relacionamento que deve existir obrigatoriamente entre um ou mais elementos.
Informação	[*informação*]	Valor ou um conjunto de valores referentes à aplicação.
Transição	*nome_da_trans*	Representa a conexão entre os agentes externos causadores de mudança no sistema e os resultados gerados.

Tabela 1. Elementos da Representação Canônica

Nome	Representação Gráfica	Definição
Parte de	*parte de*	Um elemento *e2* é parte de um outro elemento *e1*.
Instanciação	*instanciação*	Um elemento *e1* é uma generalização de outro elemento *e2*.
Tem valor	*tem valor*	Um elemento *e1* tem valor de *e2*, se *e2* recebe um valor atribuído a *e1*.
Envia	*envia*	Capta os requisitos relativos a um elemento e gera uma mensagem.
Recebe	*recebe*	Capta os requisitos de um elemento, a fim de aceitar uma mensagem.
Estímulo	*estímulo*	Capta os elementos que causam a ocorrência de uma transição.
Resposta	*resposta*	Capta os elementos que serão modificados pela ocorrência de uma transição.
Equivalência	*equivalência*	Dois elementos são equivalentes se representam algo idêntico no mundo real.
Operando	*operando*	Relacionamento entre um predicado e seus respectivos operandos.
Associação	*associação*	Relacionamento com características distintas dos relacionamentos anteriores.

Tabela 2. Relacionamentos da Representação Canônica

A figura 1 mostra um exemplo de mapeamento de um sistema em Coad/Yourdon para a Representação Canônica usando os 9 elementos e os 10 relacionamentos das tabelas 1 e 2. A linguagem RC é expressa pelas combinações entre estes elementos e relacionamentos, e servirá para descrever informações relativas aos modelos de objetos dos sistemas, e de outras técnicas de modelagem orientadas a objetos. Sobre estas descrições o Draco aplica transformações para obter a implementação automática do sistema.

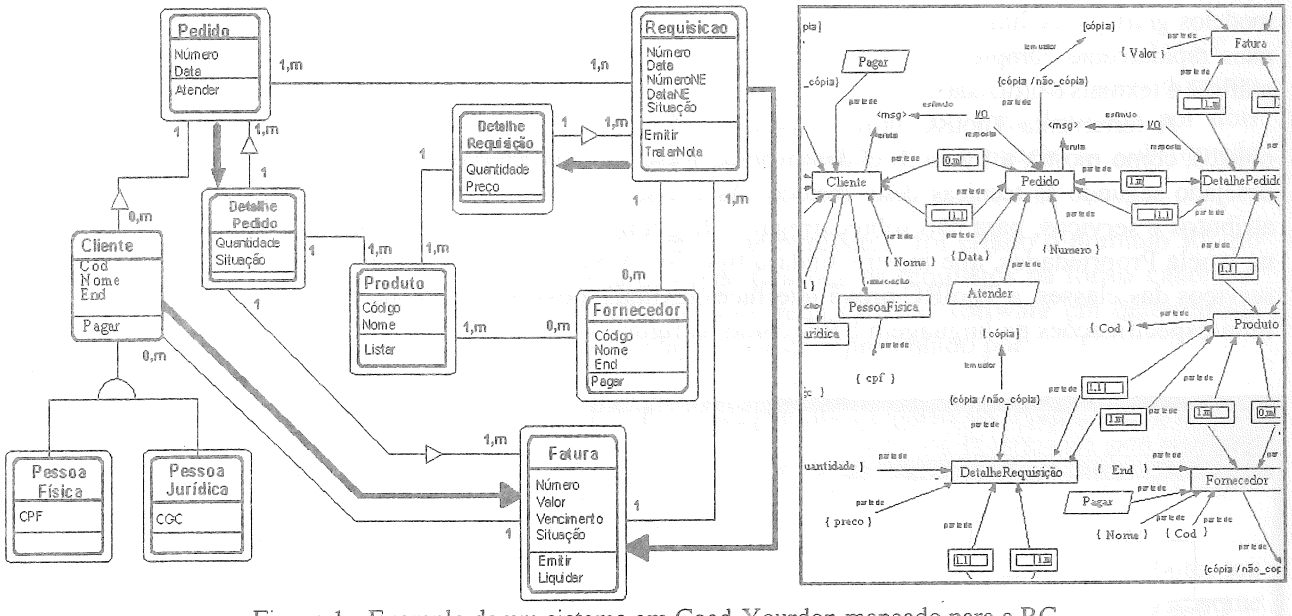

Figura 1 - Exemplo de um sistema em Coad-Yourdon mapeado para a RC

4. Ferramenta CASE OO com Múltiplas Visões

A ToolRC permite ao desenvolvedor modelar um sistema em determinado método de desenvolvimento de software orientado a objetos, e armazenar o modelo em uma descrição textual nas linguagens de especificação de requisitos RC, e de pseudocódigo LBE. Através desta descrição, pode-se obter uma nova visão do mesmo sistema segundo outro método OO disponível na ToolRC, conforme mostra a figura 2. Esta nova visão exibe o subconjunto de todos os requisitos visíveis sob o método OO escolhido. Descrições RC e LBE são automaticamente transformadas para linguagens executáveis, como C++, pelo sistema Draco. Atualmente, o desenvolvedor pode utilizar os seguintes métodos de desenvolvimento de software OO: Coad/Yourdon[3], OMT[18] e Fusion[4].

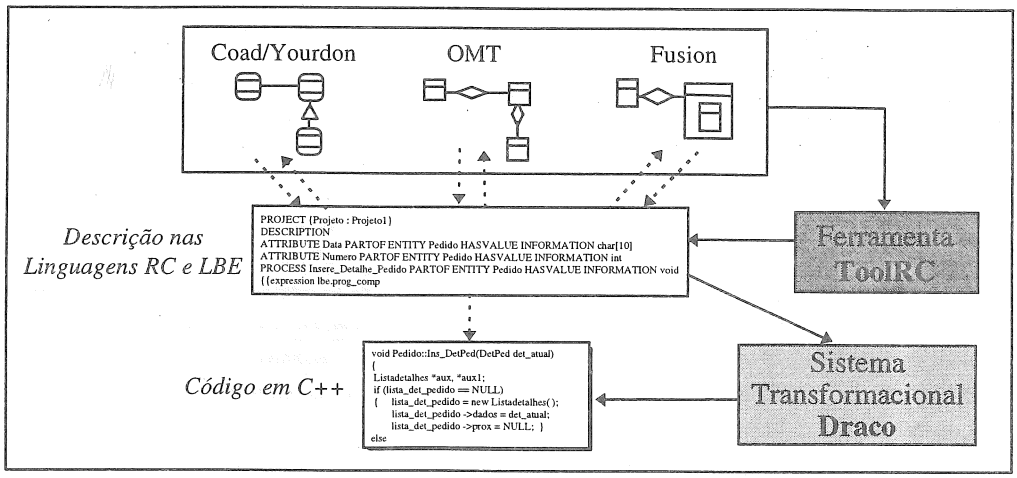

Figura 2 - Integração da Ferramenta ToolRC com o sistema Draco

4.1 A ferramenta ToolRC

As interfaces com o usuário são componentes importantes nas ferramentas CASE. Frequentemente a combinação entre interfaces gráficas e textuais é utilizada como canal para intercâmbio entre o sistema e o usuário. Vários conceitos da engenharia de software envolvem representações gráficas, e a utilização destes modelos gráficos facilita o entendimento do sistema, reduzindo a complexidade. Outros conceitos são melhor e mais rapidamente compreendidos através de interfaces textuais [8]. Por isso, a combinação entre interfaces gráficas e textuais é utilizada.

A ferramenta ToolRC provê interface gráfica e textual para a modelagem de sistemas orientados a objetos, como mostra a figura 3. 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 na janela Propriedades, que mostra ainda o tipo do atributo e tipo de retorno do serviço. Na definição dos serviços das classes, a ToolRC provê interfaces textuais, possibilitando que o desenvolvedor descreva suas mini-especificações na linguagem LBE, como na figura 4.

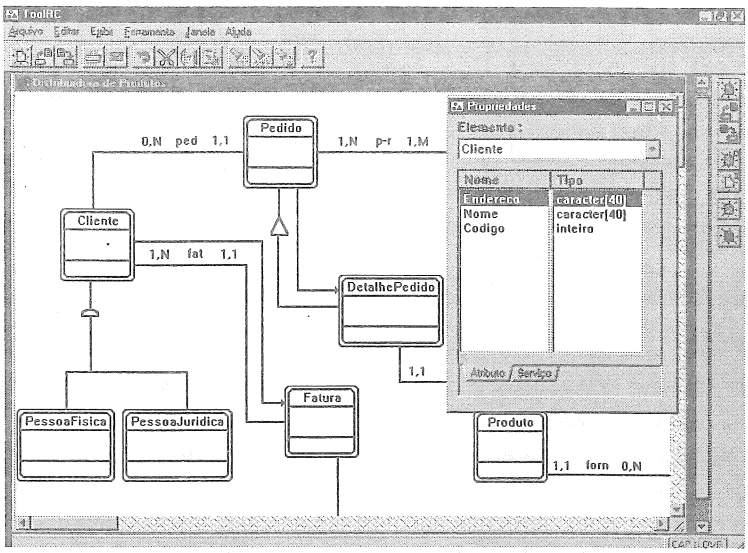

Figura 3 - Modelagem usando a ferramenta ToolRC

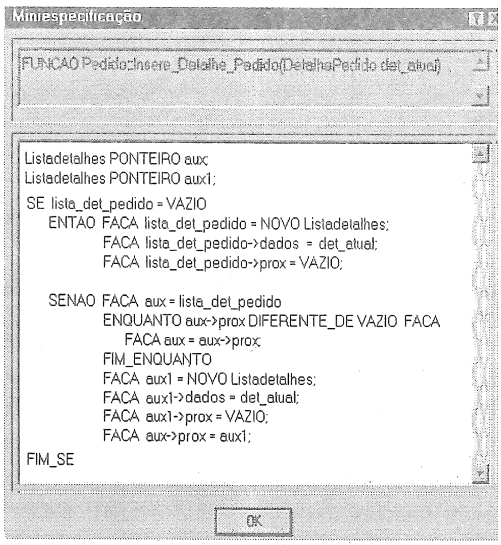

Figura 4 - Mini-especificação em LBE

A ToolRC obtém as informações relativas à um determinado sistema, e faz a persistência destas informações nas linguagens RC e LBE. As principais consistências são realizadas durante a construção dos modelos, destacando-se, no caso dos modelos de objetos:

- 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 para a de destino;
- As dependências dos atributos e serviços em cada classe.

4.2 O Sistema Transformacional Draco

O sistema transformacional Draco [15] 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 [11,12]. 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.
- 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].

Baseado na proposta de uma Representação Canônica de requisitos, foi criado o domínio RC no Draco. A linguagem do domínio RC permite a escrita de especificações de requisitos na forma canônica. Uma especificação na linguagem RC, representa a descrição de requisitos de software em determinada técnica de um método. O domínio para a Representação Canônica está representado por:

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

A figura 5 mostra o 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 domínio C++, seguida da correpondente especificação de substituição.

```
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], 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 5 - Exemplo de transformação da linguagem RC para C++.

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++.

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 6 mostra a transformação **Loop** do domínio LBE para o domínio C++. O LHS identifica o comando de controle ENQUANTO seguido da condição de fim da repetição e de comandos a serem executados dentro da repetição, 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 Loop
LHS :    {{ dast lbe.comando
           ENQUANTO [[ condicao cond ]] FACA [[ comando *cmds ]] FIM_ENQUANTO }}
RHS:     {{dast cpp.statement_list
           while ( [[ expression cond ]] ) { [[ dstatement *cmds ]] }    }}
```

Figura 6 - Exemplo de transformações da linguagem LBE para C++.

5. Estudo de Caso

Trata-se de um sistema de Distribuidora de Produtos, um dos estudos de caso usados para para testar a ToolRC. A descrição resumida do sistema é apresentada na figura 7:

> 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 produto existir no cadastro de produtos. Caso contrário o pedido é rejeitado.
>
> Semanalmente são emitidas requisições aos fornecedores, que ao serem atendidas são faturadas para os clientes.

Figura 7 - Descrição resumida do sistema de Distribuidora de Produtos

5.1 Análise e Modelagem de Requisitos no Ambiente de Múltiplas Visões

Inicialmente o desenvolvedor seleciona o método 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 8. Entra-se com o nome da classe, *Pedido* no caso, com os atributos *Código* e *Data* e os serviços encapsulados na classe. Na edição dos atributos da classe, define-se o tipo do atributo, seu tamanho, e outras características opcionais que cobrem as necessidades básicas da programação orientada a objetos.

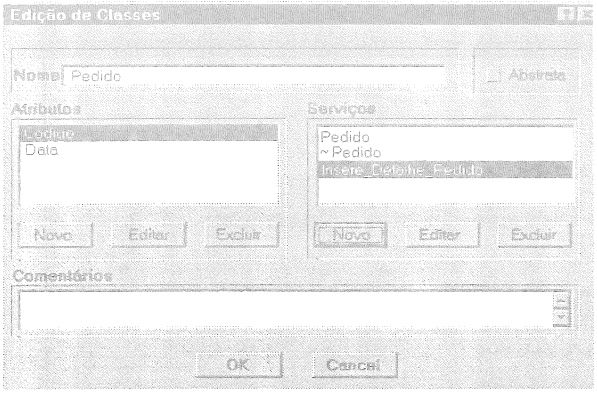

Figura 8 - Edição de classes na ToolRC

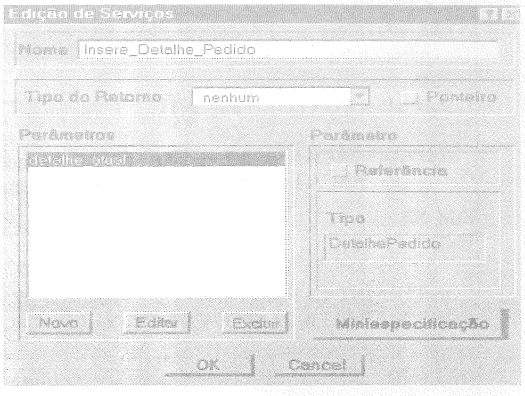

Figura 9 - Edição de Serviços na ToolRC

Os serviços de uma determinada classe são editados como mostra a figura 9. Para cada serviço definido na classe, o desenvolvedor pode especificar seu Tipo do Retorno e seus Parâmetros (nome e tipo). Para a mini-especificação do serviço, o desenvolvedor interage com um novo diálogo que permitirá a sua descrição na linguagem LBE. Um exemplo de mini-especificação para o método Insere_Detalhe_Pedido, da classe Pedido, que expressa o comportamento detalhado para se inserir os itens de um pedido, foi mostrado na figura 4. O modelo resultante, com as camadas de classes e objetos, atributos e estruturas, para o sistema de Distribuidora de Produtos é apresentado na figura 10.

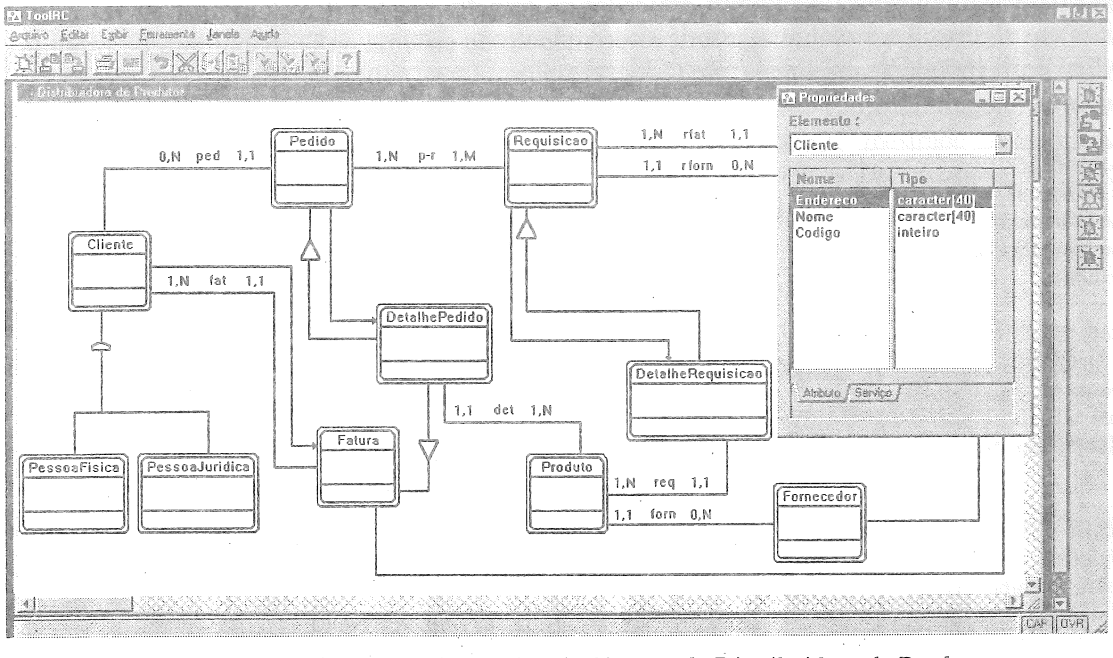

Figura 10 - Modelo em Coad/Yourdon do Sistema de Distribuidora de Produtos

O modelo da figura 10 é armazenado em uma descrição nas linguagens RC e LBE. A figura 11 mostra um trecho desta descrição, que será utilizada pelo Draco, para a geração do código em linguagem executável, e pela ferramenta ToolRC, para obter novas visões em diferentes métodos orientados a objetos. A descrição contém uma representação única para cada elemento ou relacionamento canônico, que podem ser vistos em diferentes técnicas de métodos orientados a objeto.

```
PROJECT {Projeto : Projeto1}
DESCRIPTION
ATTRIBUTE Data PARTOF ENTITY Pedido HASVALUE INFORMATION char[10]
ATTRIBUTE Numero PARTOF ENTITY Pedido HASVALUE INFORMATION int
PROCESS Insere_Detalhe_Pedido PARTOF ENTITY Pedido HASVALUE INFORMATION void
ATTRIBUTE det_atual PARTOF  PROCESS Insere_Detalhe_Pedido HASVALUE INFORMATION DetalhePedido
{{expression lbe.prog_comp
BEGIN
Listadetalhes PONTEIRO aux, aux1;
SE lista_det_pedido = VAZIO
   ENTAO  FACA lista_det_pedido = NOVO  Listadetalhes;
          FACA lista_det_pedido->dados = det_atual;
          .....
FIM_SE
END
}}
CONSTRAINT 1,1 PARTOF ENTITY Pedido OPERAND ENTITY DetalhePedido
CONSTRAINT 1,N PARTOF ENTITY DetalhePedido OPERAND ENTITY Pedido
.....
END
GRAPHIC
MODEL {OBJETOCOAD} METHOD {COAD} NAME {Modelo1 - Modelo de Objeto - Coad}
ENTITY DetalhePedido HASVALUE POSITION 198,351
ENTITY Pedido HASVALUE POSITION 404,105
.....
END
```

Figura 11 - Trecho da Descrição em RC e LBE do modelo do Sistema de Distribuidora

5.2 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 Draco, para transformar esta descrição para a linguagem C++. Após aplicadas as transformações dos domínios RC e LBE, como as mostradas nas figuras 4 e 5, sobre a descrição do modelo do sistema de Distribuidora de produtos, obtém-se o código correspondente em linguagem C++. As figuras 12 e 13 mostram parte deste código C++ gerado automaticamente pelo Draco.

```
#ifndef Pedido_H
#define Pedido_H
#include "Cliente.h"
#include "DetalhePedido.h"
class Pedido{

 protected:
    char Data[10];
    int Numero;
   Cliente *cli;
   DetalhePedido  *listadetalhes;

 public:
   Pedido();
   ~Pedido();
void Insere_Detalhe_Pedido (DetalhePedido det_atual);
};
#endif
```

Figura 12 - Arquivo Pedido.h

```
#include "Pedido.h"
Pedido::Pedido()  {}
Pedido::~Pedido() {}
void Pedido::Insere_Detalhe_Pedido(DetalhePedido det_atual)
{
 Listadetalhes *aux, *aux1;
 if (lista_det_pedido == NULL)
 {    lista_det_pedido = new Listadetalhes( );
      lista_det_pedido ->dados = det_atual;
      lista_det_pedido ->prox = NULL;  }
else
 {    aux = lista_det_pedido;
      while (aux->prox != NULL)
            aux = aux->prox;
      aux1 = new Listadetalhes ( );
      aux1->dados = det_atual;
      aux1->prox = NULL;
      aux->prox = aux1; } }
```

Figura 13 - Arquivo Pedido.cpp

5.3 Múltiplas Visões dos Requisitos

Para se obter uma nova visão, em outro método, do modelo produzido, a ferramenta ToolRC utiliza a descrição da figura 11. A parte inferior (GRAPHIC) assinalada na figura 11 é utilizada para capturar os elementos gráficos da nova visão, e suas posições. A partir desta descrição, gera-se uma nova visão do modelo, com os elementos gráficos específicos do novo método. A figura 14 apresenta uma visão no método OMT, do sistema de Distribuidora de Produtos.

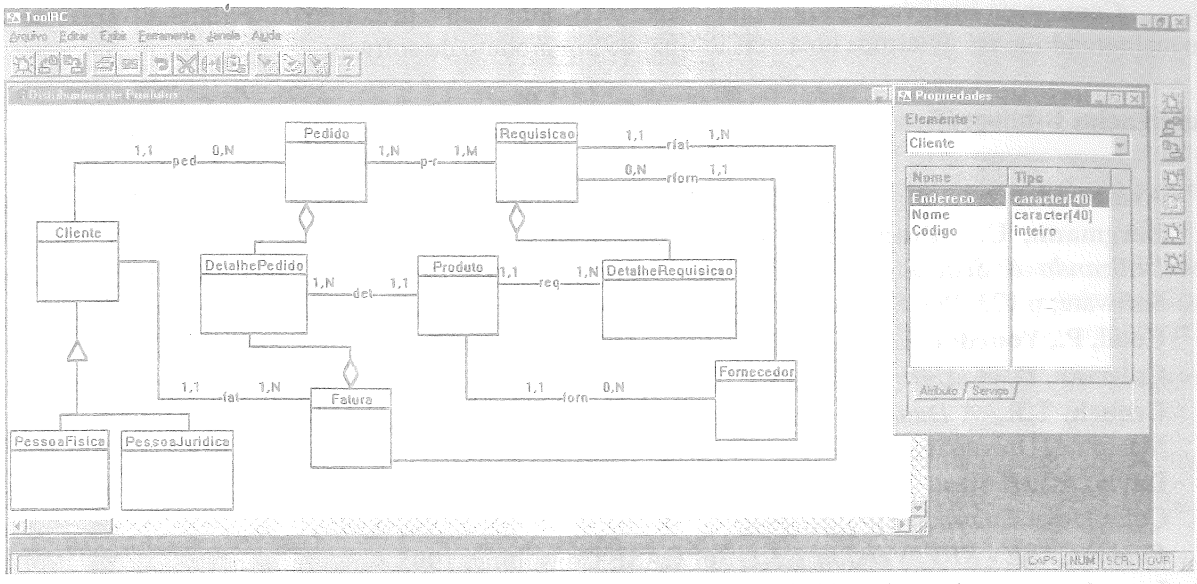

Figura 14 - Nova visão em OMT do Sistema de Distribuidora de Produtos

6. 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 com interface gráfica e textual integrada a um sistema transformacional de software. Esta integração, separa os aspectos de interface, implementados na ferramenta ToolRC e os aspectos de manipulação simbólica, implementados por transformações realizadas pelo sistema Draco.

A ferramenta ToolRC serviu para validar a pesquisa realizada na tese de mestrado "Uma linguagem de Modelagem baseada em uma Representação Canônica para Requisitos para o Desenvolvimento de Software Orientado a Objetos" da primeira autora do presente artigo. São contribuição deste trabalho:

- o domínio da linguagem de modelagem RC, construído no Draco, composto do parser, prettyprinter e conjunto de transformações da linguagem RC para a linguagem C++;
- a construção do conjunto de transformações da linguagem LBE para C++ no domínio LBE;
- a combinação dos domínios das linguagens RC e LBE, para transformar as descrições textuais geradas pela ToolRC em programas C++;
- construção da ToolRC integrada ao Draco, proporcionando o reuso da análise dos requisitos e facilidade de manutenção do sistema, múltiplas visões e implementação dos requisitos em C++;
- validação da Representação Canônica para requisitos, com seus 9 elementos e 10 relacionamentos utilizada para armazenar requisitos de software.
- disponibilização dos domínios RC e LBE no Draco, que permitirão a implementação automática em outras linguagens executáveis, como Pascal e Java.

Esta pesquisa faz parte de um projeto envolvendo alunos de mestrado e iniciação científica, contendo as seguintes atividades adicionais, em desenvolvimento:

- Geração automática de esquemas para Sistemas Gerenciadores de Banco de Dados relacionais estendidos e orientados a objetos;
- Ampliação da ferramenta para englobar novos métodos orientados a objetos, como o UML (*Unified Modeling Language*) [21];
- Implementação automática em novas linguagens de outros domínios do Draco.

Como meta final deste projeto, pretendemos evoluir com o protótipo da ferramenta, aperfeiçoando-a e melhorando-a, até obtermos uma versão final de um produto de software.

Referências Bibliográficas

1. **Armbruster J. L.** *Comparing CASE Tools*, Dr. Dobb's Journal, vol. 20, número 6, p.76-86, 1995.
2. **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, p.173-188, São Carlos, 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. **Crouch, T.R.** *A Database Design for Representing Requirements in ther Canonical Form*, Tese de Mestrado, University of Colorado at Colorado Springs, 1994.
6. **Davis, A. M. et al.** *A Canonical Representation for Requirements*, Technical Report, University of Colorado at Colorado Springs, 1995.
7. ***FusionCASE - SPV, Version 1.3.1***, SoftCASE Consulting, 1995.
8. **Isazadeh, H., Lamb D. A.,** *CASE Environments and MetaCASE Tools.* External Technical Report, Queen's University, Kingston, Ontario, 1997.
9. **Jordan, K., Davis A.,** *An Integrated View of Requirements*, Fifteenth IEEE International Conference on Computer Software And Applications, pp 472-478, Setembro, 1991.
10. **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, p.207-222, São Carlos, 1996.
11. **Leite, J. C. et al.** *Draco-PUC: A Technology Assembly for Domain Oriented Software Development*, III ICSR-IEEE, Brazil, 1994.
12. **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, Recife, 1995.
13. **MetaEdit+.** Reference Guide. Metacase Consulting, Juväskylä, Finland, 1996.
14. ***Microsoft Visual C++, Programmers Reference***, Microsoft Press, 1996.
15. **Neighbors, J.** *Software Construction Using Components,* PhD Thesis, University of California at Irvine, 1984.
16. **Paradigm Plus 2.0** - Reference Manual, Protosoft Inc., 1994.
17. **Prado, A. F.** *Estratégia de Reengenharia de Software Orientada a Domínios,* Tese de doutorado, PUC-RJ, 1992.
18. **Rumbaugh, J. et al.** *Object Oriented Modeling and Design*, Prentice Hall, 1991.
19. **Swan, T.** *Tom Swan's Mastering Borland C++ 5*, Sams Publishing, 1996.
20. **Todd, B., Novak, R., Saenz, B., Kellen V.** *Delphi 2: A Developer's Guide*, M&T Books, 1996.
21. **UML Document Set** - Version 1.0. *http://www.rational.com/uml.*
22. **Rational Rose/C++** - Rational Software Corporation. *http://www.rational.com.*