Desenvolvimento de Sistemas Orientados a Objetos Utilizando o Sistema Transformacional Draco-Puc

Ulf Bergmann [1]

Antonio Francisco do Prado [2]

Julio Cesar Sampaio do Prado Leite [3]

[1] IME - Instituto Militar de Engenharia
Praça General Tibúrcio s/n - Praia Vermelha
Rio de Janeiro-RJ - CEP 22290-270
e-mail: ulf@ime.eb.br

[2] UFSCAR - Universidade Federal de São Carlos
Caixa Postal 676
São Carlos - SP - CEP 13565-905
e-mail: prado@power.ufscar.br

[3] PUC - RJ - Pontifícia Universidade Católica do Rio de Janeiro
Rua Marques de São Vicente 225 - Gávea
Rio de Janeiro - RJ - CEP 22453-900
e-mail: julio@inf.puc-rio.br

ABSTRACT

Even not being an unanimity in Software Engineering, there is no way of denying the impact of Object Oriented Technology in all phases of the development process. Besides coming out many object oriented languages, many methods and techniques were born to support specification work and design of object oriented systems. This paper reports the building and how to use a support environment of object oriented development. The environment aggregates a graphical interface with the Draco-PUC machine transformational system. The environment characterizes itself by utilizing transformation technology that allows automatic C++ code generation from specifications of high level abstraction. The paper emphasizes Draco-PUC machine integration as well as its use for code generating. A real case based example is utilized to demonstrate the capacities and utilities of the proposed environment..

Key Words: Engenharia de Software, Ambientes de Desenvolvimento de Software, Sistemas Transformacionais, Reutilização.

1. Introdução

Ambientes de apoio ao desenvolvimento ou sistemas CASE tem despertado um grande interesse por parte da comunidade de software, tanto que existem várias conferencias dedicadas a este tópico. Também são vários os fornecedores comerciais de produtos que apoiam o

desenvolvimento, principalmente na parte da edição de diagramas. Alguns desses produtos oferecem também algum suporte na geração de código. Produtos como System Architect, Prosa Software, TurboCASE e ObjectTime são exemplos de sistemas CASE que lidam com orientação a objetos. Cada um deles tem vantagens e desvantagens e a escolha do melhor não é uma tarefa fácil. O presente artigo investiga aspectos implementacionais de ambientes correlatos aos listados acima, com a vantagem da utilização de um sistema transformacional apoiado em domínios.

O uso de um sistema transformacional baseado em domínios prove flexibilidade ao desenvolvedor de ambientes. Esta flexibilidade é muito importante quando se quer lidar com vários níveis de abstração assim como possibilita facilidades para a implementação de sistemas que utilizam meta-modelos. A utilização de vários níveis de abstração permite com que seja possivel retificar especificações através de vários passos de refinamento, escapando assim de uma abordagem do tipo tradutor que não só é pouco maleável como é também bem mais complexa. O uso de meta-modelos tem como principal vantagem a possibilidade de implementar ambientes baseados em diferentes métodos e técnicas. O aspecto de flexibilidade é secundado pelo não menos importante aspecto de reutilização, alcançado em função do uso de um sistema baseado em domínios.

Nosso artigo procura explorar justamente o uso de um sistema transformacional para suporte a montagem de um ambiente para o desenvolvimento orientado a objetos. Se por um lado a orientação a objetos não é uma unanimidade na engenharia de software, não há como negar seu impacto em todas as fases do processo de desenvolvimento. Não só surgiram várias linguagens de programação orientadas a objetos [1, 2, 3], como também surgiram vários métodos e técnicas de apoio ao trabalho de especificação e desenho de sistemas orientados a objetos [4, 5, 6, 7]. A novidade da nossa pesquisa é justamente a exploração do paradigma transformacional como base para a construção de ambientes de apoio ao desenvolvimento orientado a objetos. Em particular, centramos nossa atenção na máquina Draco-PUC [8] que vem sendo desenvolvida pelo projeto Draco-PUC, hoje um projeto multi-institucional. A máquina Draco-PUC é uma implementação parcial do paradigma de desenvolvimento por domínios primeiramente proposto por Neighbors[9].

Como maneira de ilustrar as vantagens de flexibilidade e reutilização de nossa proposta, utilizamos o modelo Fusion [10, 11] como base para a composição de uma linguagem de descrição de objetos e procedemos a construção de uma série de transformações. Fusion é um método que agrupa os principais conceitos de OMT [12], CRC [13], e Booch [14] juntamente com aspectos formais. Fusion prove um gramática para descrição de objetos, que foi a base para a montagem do domínio Fusion em Draco-PUC. A linguagem alvo escolhida foi C++[2] por esta já estar disponível como um domínio Draco-PUC. O presente artigo é um sumário dos resultados obtidos pelo primeiro autor em sua dissertação de mestrado [15] e está organizada em 5 Seções. A Seção 2 faz um resumo do sistema Draco-PUC. Na Seção 3 explicamos o processo de desenvolvimento que contextualiza nossa arquitetura de ambiente, bem como detalhamos os aspectos tanto da interface construída como das transformações utilizadas. O estudo de caso é apresentado na Seção 4. Concluímos resumindo nossos resultados e apontando para trabalhos futuros.

2. Sistemas Transformacionais

Sistemas Transformacionais [16 , 17 , 18]são formados por um conjunto de tecnologias que possibilitam a manipulação simbólica intra e inter-representações [19]. Estas manipulações são executadas através de passos formalmente bem definidos chamados de **regras de transformação**.

Uma regra de transformação é essencialmente formada por dois padrões: o padrão de reconhecimento (**LHS** - Left Hand Side) e o padrão de substituição (**RHS** - Right Hand Side). Para que uma regra de transformação seja aplicada, o sistema procura o padrão de reconhecimento. Ao ser encontrado, o mesmo é substituído pelo padrão de substituição. As regras podem possuir ainda restrições semânticas, impondo condições que devem ser seguidas antes ou após sua aplicação.

As aplicações das transformações são realizadas enquanto um objetivo final não for atingido. O objetivo é definido através de funções que calculam propriedades dos estados dos objetos, comparando-os com valores limitantes.

Algumas tecnologias que dão suporte à sistemas de transformação são gramáticas de atributos, algoritmos de casamento de padrões, heurísticas de busca, provadores de teoremas, sistemas de produção, sistemas reativos e planejadores.

Os principais sistemas transformacionais existentes são o Refine [20] , Popart [21] , Tampr [22] , TXL [23] e o Draco-PUC. Dada a capacidade do sistema Draco de utilizar linguagens de domínios para representar diferentes modelos de requisitos de software, decidiu-se investigar sua utilização no desenvolvimento de software orientado a objetos.

O projeto DRACO-PUC [8] é um sistema transformacional em desenvolvimento no Departamento de Informática da PUC-RJ com o objetivo de testar, desenvolver e colocar em prática o paradigma Draco [9, 24] para a construção de software.

Conforme é mostrado na Figura 2.1, os três componentes de um domínio encapsulado na máquina Draco são:

- *Parser*: que é o componente responsável por gerar a representação interna utilizada na máquina Draco. Esta representação é feita através de uma árvore de sintaxe abstrata chamada de **DAST** (Draco Abstract Syntax Tree). Para que um programa possa ser manipulado pela máquina Draco, é necessário que ele esteja representado sob a forma interna. O ambiente dispõe de um subsistema gerador de parser chamado **bpargen**, que auxilia na construção do parser a partir das definições da gramática da linguagem do domínio.
- *Pretty-Printer*: que mostra as informações armazenadas na forma interna utilizando a representação do meta-modelo. Um outro subsistema chamado **Ppgen** auxilia na geração automática do pretty-printer.
- *Transformações*: é o componente que manipula a DAST. As transformações podem ser de dois tipos:
  1. Transformações Verticais ou inter-domínios que transformam as aplicações descritas em um domínio em descrições de outro domínio.
  2. Transformações Horizontais ou intra-domínios que não mudam o domínio das aplicações e, normalmente, executam transformações de otimização ou de preparação para as transformações verticais.

  A especificação das regras de transformação é feita definindo-se os LHS e RHS. Além destes padrões, uma transformação pode disparar eventos e/ou alterar o fluxo de controle na aplicação das transformações através de pontos de controle, aos quais pode-se associar código para desempenhar tarefas. Os pontos disponíveis na máquina Draco cobrem todos os pontos de controle presentes em qualquer sistema transformacional.

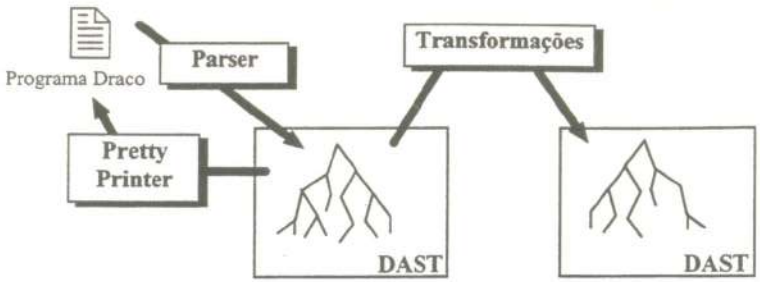


Figura 2.1 - Componentes da máquina Draco

3. Processo de Desenvolvimento

O método de desenvolvimento de software orientado a objetos proposto neste trabalho enfatiza a reutilização como forma de aumentar a produtividade e a qualidade do software produzido.

A Figura 3.1 apresenta o processo de desenvolvimento de software utilizado. A reutilização dos elementos encapsulados nos domínios da máquina DRACO é obtida através das linguagens correspondentes a cada domínio. A ferramenta chamada **DracoToolOO**, interfaceando com a máquina Draco, é utilizada na modelagem gráfica e textual dos sistemas.

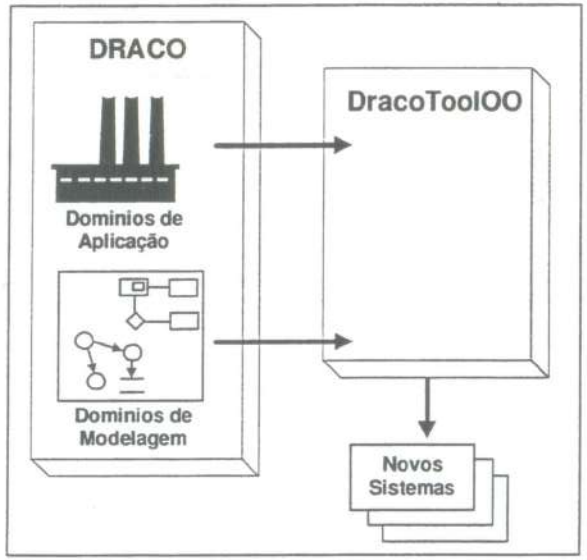


Figura 3.1 - Processo de Desenvolvimento

Modelos orientados a objetos editados na DracoToolOO são descritos por um conjunto de expressões de linguagens de diferentes domínios. Em qualquer ponto do desenvolvimento, pode-se utilizar a máquina DRACO para proceder as transformações necessárias à obtenção do código

fonte do sistema e, consequentemente, é disponibilizado um protótipo que pode ser utilizado para sua avaliação.

3.1 Níveis de Abstração

Durante o desenvolvimento de software foram identificados três níveis distintos de abstração, mostrados na Figura 3.2, pelos quais o sistema em desenvolvimento pode ser visualizado :

- Nível Usuário que possui o maior grau de abstração e pode ser facilmente compreendido pelo desenvolvedor do sistema. Sua representação é feita através de diagramas gráficos e textuais específicos de cada método de análise e projeto OO.
- Nível de Integração responsável pela integração, a nível semântico, dos diversos métodos de modelagem que podem ser utilizados no desenvolvimento de uma aplicação. Para que esta integração seja possível, é necessário utilizar uma Linguagem de Descrição de Objetos (LDO) que capte toda a semântica existente em um modelo OO, independentemente do método de análise e projeto utilizado na sua construção. A LDO escolhida foi o conjunto de gramáticas apresentadas no Método Fusion. A Figura 3.3 apresenta um trecho da gramática do método Fusion, que permite captar os aspectos estáticos de uma classe, definindo sua estrutura interna.
- Nível Executável que é a representação, a nível das linguagens de programação OO, do sistema modelado pelo desenvolvedor. O sistema visualizado segundo este nível pode ser diretamente compilado, disponibilizando protótipos que podem ser utilizados para verificar a exatidão do modelo junto aos usuários do sistema.

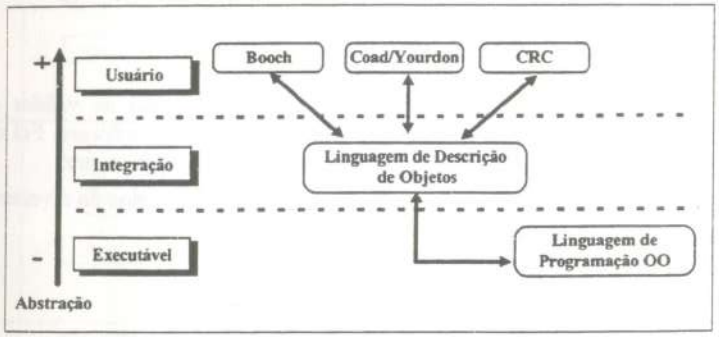


Figura 3.2 - Níveis de abstração

Cdesc	= **"class"** Name Inherit Property* **"endclass"**
Inherit	= **"isa"** Names
Names	= Name^{++} ","
Property	= Attribute | Method
Attribute	= **"attribute"** Mutability Name **":"** Sharing Binding Type
Method	= **"method"** Name ArgList [":" Type]
ArgList	= "(" Arg** "," ")"
Arg	= Name ":" Type

Figura 3.3 - Trecho da gramática de Descrição de Classes do Método Fusion estendida

A passagem entre os diversos níveis apresentados deve ser feita de forma simples e o mais transparente possível para o usuário. Para isto foram disponibilizadas ferramentas que possibilitem ao desenvolvedor a navegação livre entre os diversos níveis de abstração apresentados. A Figura 3.4 mostra como os componentes criados permitem esta navegação.

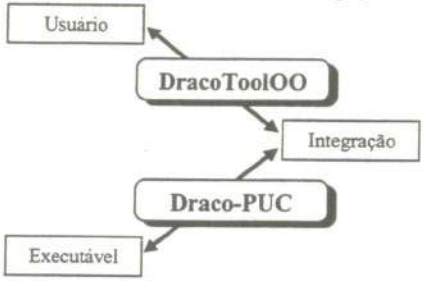


Figura 3.4 - Navegação entre os níveis de abstração

A ferramenta **DracoToolOO** [15] possibilita realizar graficamente a modelagem de um sistema, sendo responsável pela representação das informações gráficas na LDO. Na máquina **Draco-PUC** foi construído o domínio Fusion e o conjunto de transformações(inter-domínios) que permitem transformar programas descritos na linguagem do domínio Fusion em programas do domínio C++.

3.2 DracoToolOO

A ferramenta DracoToolOO foi desenvolvida com o objetivo principal de validar o método proposto, fornecendo apoio semi-automático para o desenvolvimento de software. Foi construído um protótipo utilizando a linguagem C++ em ambiente Windows que permite que:

- o sistema seja desenvolvido utilizando-se qualquer combinação de modelos de diversos métodos OO, tais como:
  - Grafo de Interação de Objetos do método Fusion
  - Modelo de Objetos do método Coad/Yourdon
  - Use Cases do método Objectory

  Para tanto, possibilita editar graficamente cada modelo componente do método, bem como realizar verificações de consistência entre os modelos. O usuário define o ambiente escolhendo quais os modelos adotados. Desta forma, o nível de granularidade do ambiente passa a ser o modelo e não o método de desenvolvimento.
- o usuário utilize as linguagens dos domínios encapsulados na máquina DRACO para proceder a especificação dos serviços que modelam o componente dinâmico dos objetos. Por exemplo, pode utilizar a linguagem de interface, de banco de dados ou mesmo pseudo-código.
- sejam incorporadas ferramentas gráficas específicas para cada domínio encapsulado na máquina DRACO. Exemplificando, quando o usuário for desenvolver uma janela para sua aplicação, ele pode acionar uma ferramenta para editar graficamente a interface e, a seguir, esta ferramenta transforma as informações da janela em expressões da linguagem de interface.
- os sistemas em desenvolvimento sejam descritos utilizando a LDO, ou seja, seja executada a

navegação entre os níveis de abstração Usuário e Integração.

- se controle a execução da máquina DRACO na aplicação das transformações necessárias à navegação entre os níveis de abstração Integração e Executável.

O mapeamento das informações apresentadas graficamente na sua descrição Fusion é executada segundo as seguintes regras:

1. Heranças e serviços de classe são descritos diretamente;
2. Atributos de classes são descritos como **variable**, **exclusive** e **bound**;
3. Associações se transformam em atributos das classes relacionadas. Na fase de projeto devem ser tomadas decisões referentes ao número de vias da associação, ou seja, se todas ou apenas uma das classes relacionadas armazena a referência ao relacionamento. Parte do mapeamento entre as associações existentes e a descrição Fusion correspondente é apresentada na Figura 3.5.
4. Agregações são descritas através da colocação de atributos na classe que agrega as demais, conforme mostra a Figura 3.6.

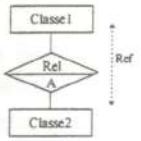
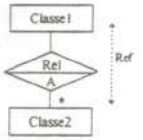
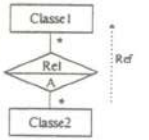
Tipo	Modelo de Objetos	Descrição Fusion correspondente
1:1 duas vias		class Classe1 attribute variable rel : shared unbound Classe2 end class class Classe2 attribute variable rel : shared unbound Classe1 attribute variable a : exclusive bound A end class
1:N duas vias		class Classe1 attribute variable rel : shared unbound col Classe2 attribute variable a : exclusive bound A end class class Classe2 attribute variable rel : shared unbound Classe1 end class
N:M uma via		class Classe1 attribute variable rel : exclusive bound col Rel end class class Rel attribute variable a : exclusive bound A attribute variable classe1 : shared unbound col Classe1 attribute variable classe2 : shared unbound col Classe2 end class

Figura 3.5 - Tabela de mapeamento de associações para a descrição Fusion

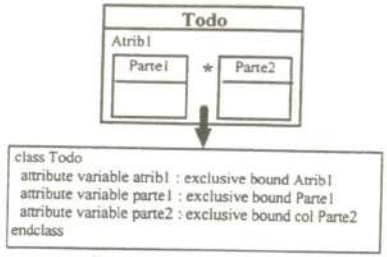


Figura 3.6 - Descrição Fusion para agregação

3.3 Domínio Fusion na Máquina Draco-PUC

Conforme descrito na seção 2, o domínio Fusion construído na máquina Draco possui os seguintes componentes:

- Uma linguagem de descrição de classes definida pelos analisadores léxico e sintático (Parser). Para cada regra do parser, existem ações semânticas para construção da DAST dos programas analisados.
- Um pretty-printer para a gramática da linguagem de descrição de classes. O pretty-printer permite exibir os programas na forma interna (DAST), orientado pela sintaxe do domínio.
- Uma biblioteca de transformações para transformar especificações na linguagem de descrição de classes para C++.

As transformações implementadas permitem gerar o código C++, equivalente à descrição de classes definidas utilizando-se a linguagem Fusion. A Figura 3.7 apresenta uma visão geral das principais transformações construídas, onde pode-se identificar 5 grupos distintos de transformações:

A. Transformações executadas no início do arquivo que contém as descrições

B. Transformações executadas todas as vezes que uma nova definição de classe for encontrada

C. Transformações executadas ao serem identificadas estruturas de heranças

D. Transformações executadas ao identificar-se um atributo

E. Transformações executadas ao ser identificado um método

Na figura 3.8 é apresentado um exemplo de uma regra de transformação criada. Esta regra identifica o início da definição de uma nova classe (linha *3*), cria o módulo fonte da classe (linhas *10* a *15*) e inclui a referência ao arquivo *header* da classe no módulos fonte da classe (linhas *16* a *21*).

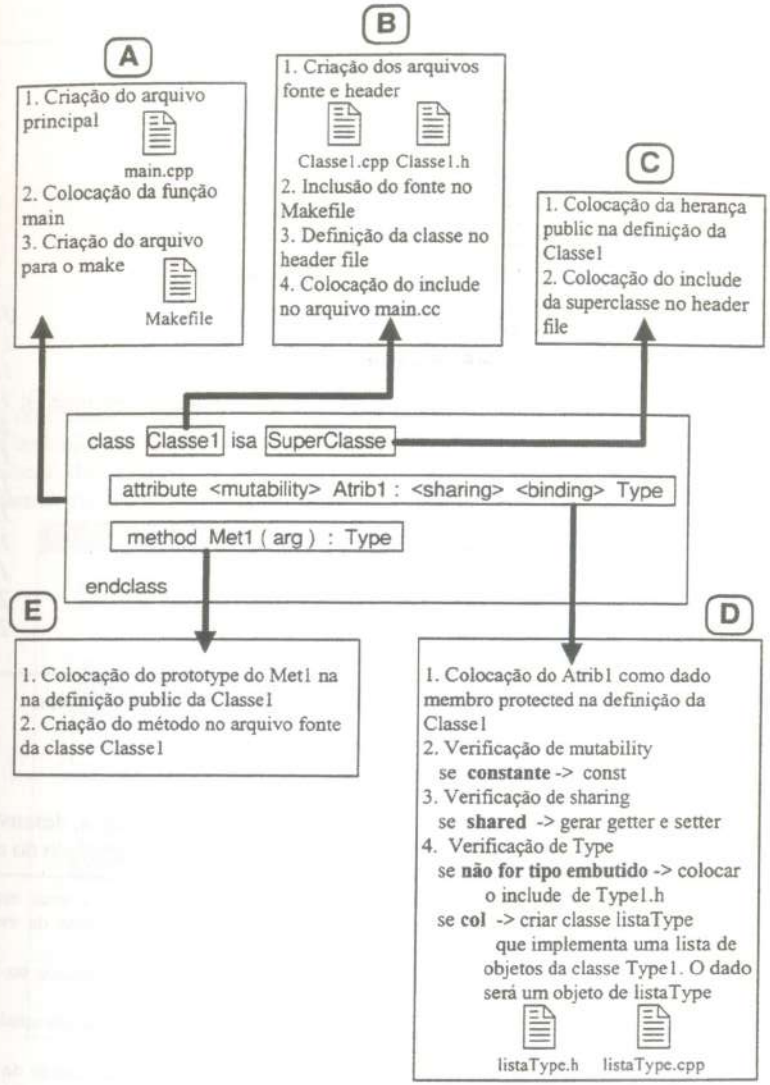


Figura 3.7 - Visão geral das transformações Fusion para C++

```
TRANSFORM IdentificaNomeClasse                                          1
 LHS: {{ dast FUSION.def_classe                                         2
    class [[ID I]]                                                      3
      }}                                                                4
 POST-MATCH: {{ dast CPP.statement_list                                 5
 COPY_LEAF_VALUE_TO(NomeClasse , "I");                                  6
 AppendSource(NomeClasse , 1);                                          7
 strcpy(NameSource, NomeClasse);                                        8
 strcat(NameSource, ".cc");                                             9
 CREATE_MODULE(NameSource);                                             10
 strcpy(tmp,NomeClasse);                                                11
 strcat(tmp,"_S_OBJ");                                                  12
 ModuleSource = CREATE_OBJECT_IN_MODULE("CPP", tmp,NameSource,1);       13
 ModuleSource->AddSon("statement_list_LIST#");                          14
 ModuleSource->GotoRawCode();                                           15
 TEMPLATE("TemplateGeraInclude");                                       16
 strcpy(tmp , "\"");                                                    17
 strcat(tmp , NomeClasse);                                              18
 strcat(tmp , ".h\"");                                                  19
 SET_TEMPL_LEAF_VALUE("NAME1" , tmp);                                   20
 INSTANTIATE_TEMPLATE_AS_LAST_SON(ModuleSource);                        21
 }}
```

Figura 3.8 - Exemplo de Regra de Transformação do Domínio Fusion

4. Estudo de Caso

Nesta seção será apresentado um estudo de caso, de um sistema de dietoterapia, desenvolvido segundo o método proposto na seção anterior. A Figura 4.1 apresenta o enunciado do caso.

> Deseja-se desenvolver um sistema computadorizado que permita a uma nutricionista executar os trabalhos necessários a prescrição de dietas e ao acompanhamento da evolução do tratamento dietoterápico prescrito, que atenda os seguintes requisitos:
>
> - Todo paciente é cadastrado, sendo coletadas as informações necessárias ao cálculo do seu peso ideal.
> - A cada consulta que o paciente comparece, é feita uma avaliação do seu estado atual e é prescrita uma dieta a ser seguida.
> - A dieta que é prescrita para o paciente é composta por uma coleção de alimentos escolhidos de um cadastro pela nutricionista. Cada alimento possui a sua quantidade de calorias, proteinas, glicideos e carboidratos tabeladas.

Figura 4.1 - Descrição do Sistema de Dietoterapia

A ferramenta DracoToolOO permite a modelagem do sistema em diversos métodos de análise e projeto OO. A Figura 4.2 apresenta diversas opções de modelos, mostrando a possibilidade de utilizar modelos de métodos distintos.

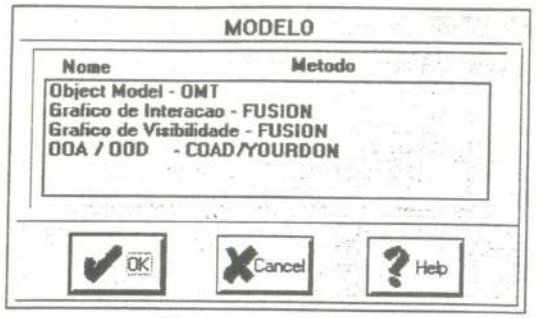


Figura 4.2 - Escolha de modelos

A edição de classes é apresentada na Figura 4.3. A Figura 4.4 mostra a inclusão da classe Paciente. Esta inclusão é efetuada em todos os modelos em uso na ferramenta. Qualquer alteração nas caracteristicas dos elementos representados, é automaticamente transmitida a todos os modelos, mantendo-se a consistência do sistema que está sendo modelado.

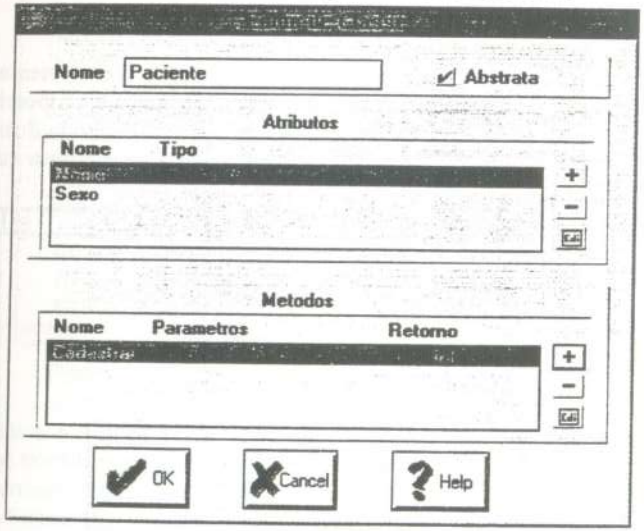


Figura 4.3 - Edição de classe

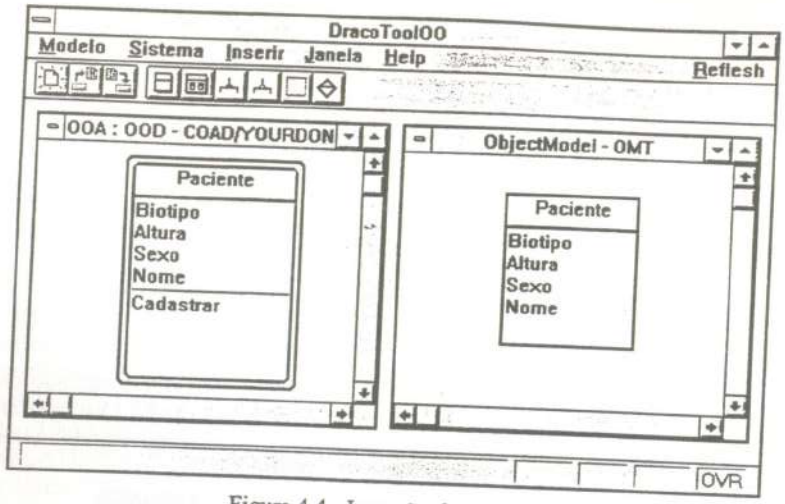


Figura 4.4 - Inserção de classes

O sistema resultante é o mostrado na Figura 4.5. A manipulação dos elementos do modelo pode ser feita diretamente na sua representação gráfica. A ferramenta proporciona ainda a descrição do sistema, utilizando a linguagem de descrição de classes do método Fusion. A Figura 4.6 mostra a descrição para o sistema de dietoterapia, pronta para ser submetida a máquina Draco e transformada em C++.

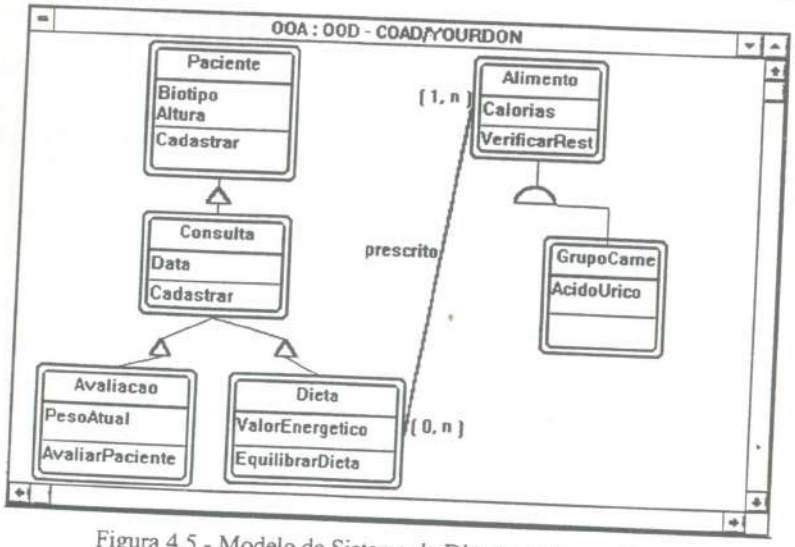


Figura 4.5 - Modelo do Sistema de Dietoterapia resultante


```
class prescrito
    attribute variable Quantidade : exclusive bound float
    attribute variable asAlimento : shared unbound Alimento
    attribute variable asDieta : shared unbound Dieta
endclass
class GrupoCarne isa Alimento
    attribute variable AcidoUrico : exclusive bound float
endclass
class Alimento
    method VerificarRest ( ) : int
    attribute variable Calorias : exclusive bound int
    attribute variable Nome : exclusive bound string
endclass
class Avaliacao
    method AvaliarPaciente (aConsulta:Consulta ) : int
    attribute variable PesoAtual : exclusive bound float
endclass
class Dieta
    method EquilibrarDieta ( ) : int
    attribute variable ValorEnergetico : exclusive bound float
    attribute variable asprescrito : shared unbound col prescrito
endclass
class Consulta
    method Cadastrar (aPaciente : Paciente ) : int
    attribute variable Data : exclusive bound string
    attribute variable agDieta : exclusive bound Dieta
    attribute variable agAvaliacao : exclusive bound Avaliacao
endclass
class Paciente
    method Cadastrar ( ) : int
    attribute variable Biotipo : exclusive bound char
    attribute variable Altura : exclusive bound float
    attribute variable Sexo : exclusive bound int
    attribute variable Nome : exclusive bound string
    attribute variable agConsulta : exclusive bound col Consulta
endclass
```

Figura 4.6 - Descrição Fusion do sistema de dietoterapia

Uma vez obtida a descrição do modelo na linguagem de descrição de classes do método Fusion, podem-se executar as transformações do domínio Fusion, encapsulado na máquina Draco-PUC, que transformam a descrição da Figura 4.6 para o domínio C++, conforme é mostrado para a classe Dieta na Figura 4.7.

Estas transformações permitem implementar a parte estática de um sistema. Os serviços podem ser implementados diretamente na linguagem alvo C++, no ambiente DracoToolOO, conforme se pode ver na Figura 4.8. Outra forma de se obter a implementação dos serviços é com minispecificações escritas em pseudo-código. Estas minispecificações podem ser transformadas automaticamente para uma linguagem C/C++, usando o domínio de pseudo-código encapsulado na máquina Draco.

```
#ifndef Dieta_H
#define Dieta_H
#include "lprescri.h"
class Dieta
{
 protected:
  float ValorEnergetico;
  listaprescrito asprescrito;
 public:
  int EquilibrarDieta( );
  Dieta();
  ~Dieta();
};
#endif
```

Figura 4.7 - Classe Dieta gerada pela máquina Draco

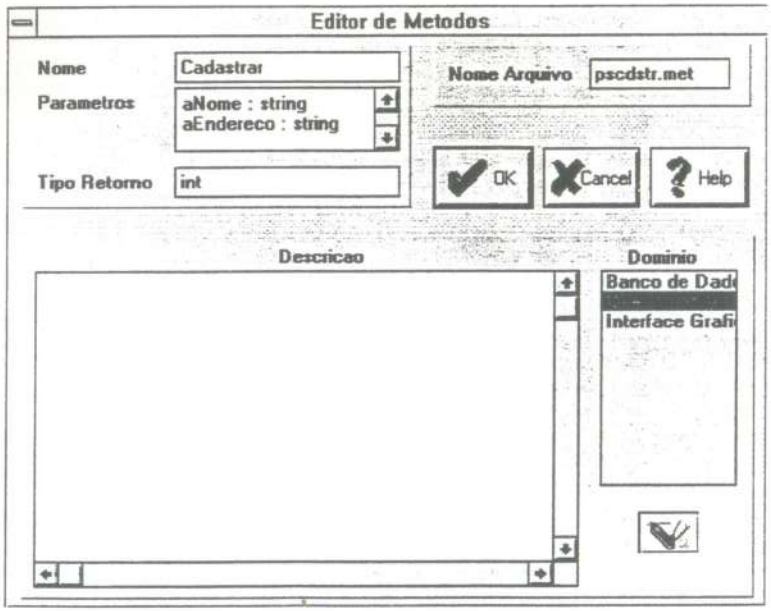


Figura 4.8 - Edição de serviços na DracoToolOO

5. Conclusão

Apresentamos nesse trabalho uma proposta de construção de ambientes de apoio a construção de sistemas orientados a objetos que baseia-se no uso de um sistema de transformação orientado a domínios. A arquitetura proposta separou aspectos de interface, implementados através de DracoToolOO e aspectos de manipulação simbólica, implementados por transformações em Draco-PUC. A linguagem de domínio, LDO, utilizada para mapear as especificações orientadas a

objetos foi construída com base na sintaxe descrita no método Fusion [10]. A contribuição desse trabalho reside principalmente na exploração da tecnologia de transformação apoiada em domínios com o objetivo de prover um ambiente para apoio ao desenvolvimento de sistemas de software orientados a objetos.

Um dos problemas que enfrentamos na implementação da LDO, foi que a principio, ao seguirmos a gramática descrita em [10], verificamos que perderíamos informações sobre agregações e associações entre classes. Esta perda de informações referentes a agregação e a associação entre classes, quando das suas transformações em atributos das classes relacionadas, dificultava a passagem do nível de integração para o nível do usuário. A solução utilizada foi estender a LDO, incluindo, além dos atributos e métodos de cada classe, declarações dos relacionamentos entre classes. Em conseqüência, a transformação dos relacionamentos em atributos das classes relacionadas passa a fazer parte do conjunto de transformações do domínio Fusion, encapsulado no sistema Draco-PUC.

Consideramos bastante positivo o resultado alcançado. O uso da máquina Draco-PUC e de seu sistema transformacional alem de mostrar-se eficaz para a manipulação e derivação de código C++ a partir de descrições em LDO, contribuiu também para que a própria máquina fosse testada. O seu uso como protótipo gerou algum trabalho extra, visto a carência de documentação e a necessidade de alguns ajustes durante o processo. Consideramos no entanto extremamente positiva nossa experiência. Pretendemos continuar trabalhando nesse ambiente em função dos primeiros resultados. Centraremos nossa atenção principalmente no uso de outros domínios para completar as informações estáticas com as informações dinâmicas dos objetos, isto é, de que maneira integraremos domínios de aplicação e de modelagem de maneira a facilitar a montagem de um ambiente de suporte a produção de sistemas orientados a objetos.

6. Agradecimentos

O trabalho reportado neste artigo não poderia ter sido feito sem o apoio da equipe do projeto DRACO-PUC, principalmente o realizado pelos alunos Marcelo SantÁnna e Felipe Gouveia. Sempre disponíveis e receptivos, "abriram" a máquina DRACO-PUC, desviando-se de suas tarefas rotineiras, contornando os problemas referidos na conclusão e permitindo que a máquina DRACO-PUC tivesse suas fronteiras extendidas ao ser utilizada em outras instituições.

7. Referências

[1] GOLDBERG, A., DAVID, R., *Smalltalk-80: The Language*, Addison-Wesley, 1989.

[2] ELLIS, M., STROUSTRUP, B., *The Annotaded C++ Reference Manual*, Addison-Wesley, 1990.

[3] MEYER, B., *Eiffel: The Language*, Prentice Hall, Englewood Cliffs, NJ, 1992.

[4] REINOSO G.B, HEUSE, C., *Comparando o Processo de Modelagem de Técnicas de Análise Orientada a Objetos*, Anais do VIII Simpósio Brasileiro de Engenharia de Software, Ed. SBC, 1994, pp. 177-193.

[5] CAMPO, M., PRICE, R.T., *O Uso de Técnicas Visuais e Navegacionais para a Compreensão de Frameworks Orientados a Objetos*, Anais do IX Simpósio Brasileiro de Engenharia de Software, Ed. SBC, 1995, pp. 175-190

[6] MOREIRA, A.M.D, CLARK, R.G., *Os Métodos Formais na Análise de Orientação a*

Objetos, Anais do VII Simpósio Brasileiro de Engenharia de Software, Ed. SBC, pp. 238-252, 1993.

[7] CARNEIRO, L.M.F, COWAN, D.D., LUCENA, C.J.P., *A Rationale for Both Nesting and Inheritance in Object-Oriented Design*, Anais do VII Simpósio Brasileiro de Engenharia de Software, Ed. SBC, 1993, pp. 223-237.

[8] LEITE, Julio et alii., *Draco-PUC: A Tecnology Assembly for Domain Oriented Software development*, III ICSR-IEEE, Brazil, 1994.

[9] NEIGHBORS, J., *The Evolution from Software Components to Domain Analysing*, Anais do V Simpósio Brasileiro de Engenharia de Software, Ed. SBC, 1991.

[10] COLEMAN, D. et al, *Object-Oriented Development: The Fusion Method*, Prentice-Hall, 1994.

[11] PENTEADO, R.A.D. *Um Método para Engenharia Reversa Orientada a Objetos*, Tese de Doutorado, USP, Instituto de Física e Química de São Carlos, 1996.

[12] RUMBAUGH, J. et al, *Object Oriented Modeling and Design*, Prentice-Hall Int. 1991.

[13] WIRFS-BROCK, Wilkerson, V., WIERNER L., *Designing Object Oriented Software*, Prentice-Hall, 1990.

[14] BOOCH, G., *Object Oriented Design with Applications*, Benjamin Cummings, CA, 1991.

[15] BERGMANN, U. ., *Construção de um Domínio de Desenvolvimento de Software Orientado a Objetos Segundo o Paradigma Draco*, Tese de Mestrado, IME, RJ, 1996.

[16] BAXTER, I., *Design (not code) Maintenance*, Palestra Convidada - Anais do VIII Simpósio Brasileiro de Engenharia de Software, Ed. SBC, 1994.

[17] PARTSCH, H., *Specification and Transformation of Programs*, Springer-Verlag, 1990.

[18] SANT'ANNA, M., *Uma Abordagem Prática do Paradigma Transformacional*, Monografia de Graduação, DI/PUC-RJ, 1993.

[19] SANT'ANNA, M. *Utilização de Sistemas Transformacionais como Ferramentas de Apoio a Técnicas de Reuso*, I Workshop de Reutilização, COPPE/UFRJ - PUC/RJ, 1994.

[20] REASONING SYSTEMS INCORPORATED. *REFINE User's Guide*, Reasoning Systems Incorporated, Palo Alto, 1992.

[21] WILE, D., *POPART: Producer of Parsers and Related Tools System Builders Manual*, Techinical Report, USC/Information Sciences Institute, 1993.

[22] BOYLE, J., *Abstract Programming and Program Transformations - An Approach to Reusing Programs*, in Software Reusability (Vol 1), Ed. Ted Biggerstaff, ACM Press, 1989.

[23] CORDY, J. & CARMICHAEL, I. *The TXL Programming Language Sysntax and Informal Semantics*, Techinical Report, Queen's University at Kingston - Canada, 1993.

[24] NEIGHBORS, J., *Software Construction Using Components*, Phd Thesis, University of California at Irvine, 1984.