Estratégia Data-Driven no Desenvolvimento de Software

Daltro José Nunes*

Resumo

Este artigo mostra uma evolução da estratégia data driven de desenvolvimento de software pela definição exata do conceito de estruturas de dados, definidas como classes (domínios matemáticos). Para aplicar esta estratégia foi projetado um ambiente de desenvolvimento, denominado PROSOFT. Numa extensão dessa estratégia integrou-se a esse ambiente um método para geração de ferramentas através de especificações algébricas. O Ambiente garante um aumento de produtividade maior do que aquele obtido pela simples aplicação da estratégia, garantindo também, automaticamente, alguns objetivos de qualidade do software desenvolvido.

Abstract

This work shows an evolution of the software development data driven strategy. The concept of data structure is formalized as classes (mathematical domains). To apply this strategy it was designed a development environment called PROSOFT. With an extension of this strategy it is also possible the generation of tools using algebraic specifications. The environment garanties the increasing of the software productivity, greater than that obtained with the simple applications of the strategy. It garanties also automatically some qualities of the software.

1 Introdução

Se a definição de requisitos representa o enunciado de um problema, o software (programa ou sistemas de programas) representa a sua solução. Uma das maneiras mais eficientes de resolver o problema é aplicar a estratégia "data driven". Esta estratégia, colocada de forma muito simples,

*Universidade Federal do Rio Grande do Sul, Departamento de Informática Aplicada, Instituto de Informática, Av. Bento Gonçalves, 9500, Caixa Postal 15064, Porto Alegre - RS. E-mail: daltro@inf.ufrgs.br

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

programa = estrutura de dados + estruturas de controle

é, especialmente, recomendada quando se infere do problema a necessidade de, inicialmente, especificar as estruturas de dados antes das estruturas de controle. Esta estratégia não evoluiu muito pois, ela exigia uma definição precisa dos componentes do programa, especialmente das estruturas de dados, independente das linguagens de programação.

Um estudo aprofundado desta estratégia levou a uma definição de estruturas de dados e de um conjunto de funções sobre elas. Estes conceitos básicos foram implementados, experimentalmente, na linguagem Pascal e algumas ferramentas, constituindo um pequeno ambiente de desenvolvimento, foram construídas para facilitar o seu uso.

Para verificar, agora, a eficiência da aplicação desta estratégia, alguns problemas foram resolvidos e pôde-se verificar um ganho de tempo nas suas soluções, quando comparado com o tempo gasto para encontrar soluções para os mesmos problemas, usando outras estratégias. Algumas qualidades da solução, como por exemplo legibilidade, reusabilidade, amigabilidade, etc são logo verificadas. A eficiência da solução depende muito da eficiência da implementação dos conceitos básicos.

Carmo [Carmo91] em seu trabalho de diplomação fez um estudo comparativo do PROSOFT com diversos ambientes existentes como Smalltalk, OBJ, Axis, etc. mostrando várias vantagens.

2 Classe

O ambiente, PROSOFT, tem sua base formal no conceito de classe. Embora este conceito já seja bastante conhecido como tipos abstratos de dados compostos [Watt91], ou como domínios de Scott [Scott71], ele será apresentado aqui, de forma sucinta, com o objetivo de introduzir a notação e a terminologia empregada.

2.1 Definição

O conceito de classe é definido a partir de conjuntos bastante conhecidos na computação: o conjunto dos números inteiros (Z), o conjunto dos números reais (R), o conjunto dos Stringues etc. Estes conjuntos são denominados no PROSOFT de classes primitivas e são os mesmos disponíveis na linguagem hospedeira, no caso o Pascal.

A definição de classe é dada pela aplicação do conceito de indução matemática, da seguinte forma:

1. Os conjuntos Integer, String, Real, Char, Boolean, etc. são denominados de **classes primitivas**.
2. Se X é uma classe, então

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

$S = \{s | s \supseteq X\}$

é uma **classe de conjuntos**.

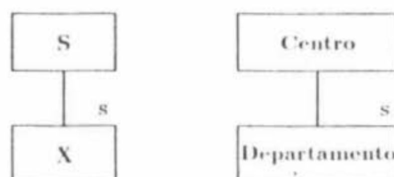
No PROSOFT, uma instância (elemento) de qualquer classe é denominada de **objeto**, ou seja, para esta classe, s é um objeto da Classe S se

$s \in S$

Sobre objetos desta classe podem ser aplicadas todas as operações definidas sobre conjuntos como mostrado abaixo:

Operação	Funcional	Exemplo
Union ($\cup$)	$S \times S \longrightarrow S$	$\{x_1, x_2\} \cup \{x_3\} = \{x_1, x_2, x_3\}$
Intersection ($\cap$)	$S \times S \longrightarrow S$	$\{x_1, x_2\} \cap \{x_2\} = \{x_2\}$
Complement ($\setminus$)	$S \times S \longrightarrow S$	$\{x_1, x_2\} \setminus \{x_2\} = \{x_1\}$
Containtion ($\subseteq$)	$S \times S \longrightarrow Bool$	$\{x_1, x_3\} \subseteq \{x_1, x_2, x_3\} = true$
Equal ($=$)	$S \times S \longrightarrow Bool$	$\{x_1\} = \{x_2\} = false$
power	$S \longrightarrow 2^S$	$power(\{x_1, x_2\}) = \{\emptyset, \{x_1\}, \{x_2\}, \{x_1, x_2\}\}$
union	$2^S \longrightarrow S$	$union(\{\{x_1\}, \{x_1, x_2\}\}) = \{x_1, x_2\}$
Belongs to ($\in$)	$X \times S \longrightarrow Bool$	$x_1 \in \{x_1, x_2\} = true$
Cardinality (car)	$S \longrightarrow N$	$car(\{x_1, x_2, x_3\}) = 3$

As classes S podem ser nomeadas para representar uma abstração do mundo real e podem, também, ter uma representação gráfica como mostrada no exemplo abaixo.



Esta representação define a classe Centro formada por conjuntos de Departamento que, também, é uma classe, mas ainda não definida. Se Departamento for definido como um conjunto de nomes de departamentos (conjunto primitivo), então, por exemplo, Centro poderia ser:

Centro = { {Inf. Aplicada, Inf. Teórica}, ... }

onde um elemento de Centro pode ser um centro particular, por exemplo, o Centro de Informática (CINF) definido assim:

CINF = {Inf. Aplicada, Inf. Teórica}

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

3. Se X e Y são classes e

$$W = \{\{(p,q) | p \in X' \wedge q \in Y'\} | X \supseteq X' \wedge Y \supseteq Y'\}$$

então

$$M = \{m \in W | (p_1, q_1), (p_2, q_2) \in m \wedge p_1 = p_2 \Longrightarrow q_1 = q_2\}$$

é uma **classe de mapeamentos**.

Neste caso, os pares de m são representados assim: $p \longrightarrow q$ e colocados entre colchetes [ ]. A classe M é, portanto, um conjunto de mapeamentos finitos.

Se $p \longrightarrow q$ é um par de m então ele pode também ser escrito assim, $q = m(p)$. Esta é a maneira de se obter q quando se conhece p.

Sobre os objetos desta classe podem ser aplicadas todas as operações definidas para o tipo mapeamento como:

Operação	Funcional	Exemplo
Domain (dom)	$M \longrightarrow 2^X$	$dom([x_1 \rightarrow y_1, x_2 \rightarrow y_2]) = \{x_1, x_2\}$
Range (rng)	$M \longrightarrow S^Y$	$rng([x_1 \rightarrow y_1, x_2 \rightarrow y_2]) = \{y_1, y_2\}$
Composition (Θ)	$M \times M \longrightarrow M$	$[y_1 \rightarrow y_2]\Theta[x_1 \rightarrow y_1] = [x_1 \rightarrow y_2]$
Restrict_to ($\|$)	$M \times 2^X \longrightarrow M$	$[x_1 \rightarrow y_1, x_2 \rightarrow y_2] \| \{x_1\} = [x_1 \rightarrow y_1]$
Restrict_with ($\setminus$)	$M \times 2^X \longrightarrow M$	$[x_1 \rightarrow y_1, x_2 \rightarrow y_2] \setminus \{x_1\} = [x_2 \rightarrow y_2]$
Merge ($\uplus$)	$M \times M \longrightarrow M$	$[x_1 \rightarrow y_1] \uplus [x_2 \rightarrow y_2] = [x_1 \rightarrow y_1, x_2 \rightarrow y_2]$
Override ($+$)	$M \times M \longrightarrow M$	$[x_1 \rightarrow y_1, x_2 \rightarrow y_2] + [x_2 \rightarrow y_3] =$ $[x_1 \rightarrow y_1, x_2 \rightarrow y_3]$

As classes M podem ser nomeadas para representar uma abstração do mundo real e podem também ter uma representação gráfica como mostrada no exemplo abaixo.

onde Disciplina e Professor nesta representação ainda não estão definidos. Se Disciplina e Professor forem definidos como tipos primitivos assim:

Disciplina = {Estruturas de Dados, Compiladores, ...}

Professor = {Gilberto, Thaden, ...}

então Encargo é definido assim:

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

Encargo = {[Estruturas de Dados → Thadeu, Compiladores → Gilberto, ...], ...}.

Se a classe de mapeamentos for definida mais fortemente assim

$$M' = \{m \in M | m(d_1) = m(d_2) \Longrightarrow d_1 = d_2\}$$

então a seta no desenho acima deve ser colocada nos dois sentidos ($\Longleftrightarrow$).

4. Se X é uma classe, então

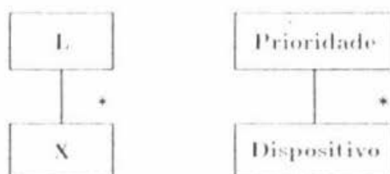
$$L = \{\langle x_1, \ldots, x_n | x \in X \rangle | n \in N\}$$

é uma **classe de listas**.

L é, portanto, uma classe de listas. Sobre os objetos desta classe podem ser aplicadas todas as operações definidas para o tipo lista como:

Operação	Funcional	Exemplo
Concatenation ($\frown$)	$L \times L \to L$	$\langle x_1, x_2 \rangle \frown \langle x_1 \rangle = \langle x_1, x_2, x_1 \rangle$
Projection ([ ])	$L \times N \to X$	$\langle x_1, x_2 \rangle[2] = x_2$
Head (hd)	$L \to X$	$hd(\langle x_1, x_2 \rangle) = x_1$
Tail (tl)	$L \to L$	$tl(\langle x_1, x_2 \rangle) = \langle x_2 \rangle$
Length (lng)	$L \to N$	$lng(\langle x_1, x_2, x_1 \rangle) = 3$
Índices ($inds$)	$L \to 2^N$	$inds(\langle x_1, x_2, x_1 \rangle) = \{1, 2, 3\}$
Elements ($elems$)	$L \to 2^X$	$elems(\langle x_1, x_2, x_1 \rangle) = \{x_1, x_2\}$
$last$	$L \to X$	$last(\langle x_1, x_2, x_1 \rangle) = x_1$
Replace ($+[\ \mapsto\ ]$)	$L \times N \times X \to X$	$\langle x_1, x_2 \rangle + [2 \mapsto x_1] = \langle x_1, x_1 \rangle$

As classes L podem ser nomeadas para representar uma abstração do mundo real e podem, também, ter uma representação gráfica assim:



Se a classe for restrita às listas de comprimento maior ou igual a 1, então isto deve ser indicado na representação gráfica substituindo o '*' pelo sinal '+'.

5. Se $X_1, \ldots, X_n$ são classes e $n \in N$, então

$$R = \{(x_1, \ldots, x_n) | x_i \in X_i \wedge i \in 1 \ldots 2\}$$

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

é uma **classe de registros**.

R é, portanto, uma classe de registros. Sobre este tipo de objeto pode ser aplicada a operação de seleção, definida da seguinte maneira: a partir das funcionalidades

$$s\text{-}_ : K \longrightarrow C$$

onde C é o conjunto das classes X_i e K, um conjunto de nomes, e

$$_ . _(\ ) : (K \longrightarrow C) \times K \times R \longrightarrow X_i$$

a operação de seleção é definida assim:

$$f\text{-}n((x_1, \ldots x_i, \ldots x_n)) = x_i$$

se $s\text{-}n = X_i$.

As classes R podem ser nomeadas para representar uma abstração do mundo real e podem, também, ter uma representação gráfica como mostrada no exemplo abaixo:

onde Rua, CEP, Cidade e Estado representam classes ainda não definidas.

6. Se $X_1, X_2, \ldots$ e X_n são classes, então

   $$\bigcup = X_1 \cup X_2 \cup \ldots \cup X_n$$

   é também uma classe, denominada de **classe disjuntiva**. $\bigcup$ representa, portanto, a união das classes X_i. Um objeto de $\bigcup$, u, é, portanto, também um objeto de X_i, para algum i. Sobre u podem ser aplicadas todas as operações que podem ser aplicadas sobre X_i.

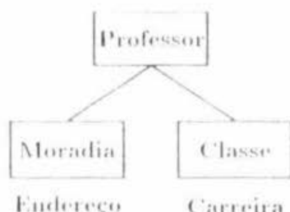
   Normalmente, dado um elemento de $\bigcup$, u, deseja se saber a qual classe X_i ele pertence. Dado que os objetos não possuem uma representação, eles estão em $\bigcup$ identificados por um "tag". Isto permite distinguir os diversos objetos entre si. Assim, os objetos são inseridos em $\bigcup$ acompanhados de um tag (nome da classe) e quando recuperados de $\bigcup$ são identificados pelo tag.

   A classe pode ser nomeada para representar uma abstração do mundo real e pode ter uma representação gráfica como mostra o exemplo abaixo.

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

2.2 Composição de Classes

A representação gráfica de uma classe é uma árvore onde as folhas *devem referenciar* classes previamente definidas (ou a serem definidas). As classes folhas da árvore podem fazer referência a qualquer outra classe ou, inclusive, a si própria (recursão). Desta forma, as classes são compostas de outras classes exceto as classes primitivas. O nome da classe folha deve refletir uma realidade cuja abstração é indicada por uma classe. No exemplo da figura abaixo, a classe Moradia faz referência à classe Endereço e Classe faz referência à classe Carreira.



3 Operações

O PROSOFT possui três tipos de operações:

- Operações primitivas, que são aplicadas sobre objetos das classes primitivas e portanto importadas da linguagem hospedeira;
- Operações básicas, que são aplicadas sobre objetos das classes básicas (conjunto, mapeamento, etc);
- Operações externas, como descritas abaixo.

Definida uma classe, deve-se mostrar, através de operações denominadas de operações externas, como um objeto da classe é gerado/transformado/corrigido.

Uma operação é definida sintaticamente assim:

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

$$op_i : X_1 \times X_2 \ldots X_n \longrightarrow X_j$$

onde X_i é uma classe.

4 Ambiente de Tratamento de Objetos — ATO

Uma classe e suas operações externas são encapsuladas em uma operação de alta ordem chamada de Ambiente de Tratamento de Objetos — ATO.

O domínio e a imagem de um ATO é, respectivamente, a união dos domínios e a união das imagens de suas operações externas. Isto significa que cada ATO recebe uma op como parâmetro, seus argumentos na forma de uma lista e devolve o mesmo objeto produzido pela operação.

$$ATO_k : (X_1 \times X_2 \ldots X_n \longrightarrow X_j) \times (X_1 \times X_2 \ldots X_n) \longrightarrow X_j$$

$$ATO_k(op_i, (x_1, x_2 \ldots x_n)) = op_i(x_1, x_2 \ldots x_n)$$

5 Interface de Comunicação do Sistema — ICS

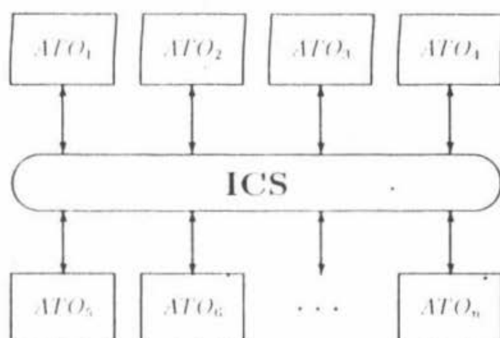
Os ambientes, por sua vez, são encapsulados em uma função de alta ordem chamada Interface de Comunicação do Sistema — ICS. Isto significa que a ICS recebe um ATO como parâmetro e devolve o mesmo objeto produzido pelo ATO.

$$ICS : ((X_1 \times X_2 \ldots X_n \longrightarrow X_j) \times (X_1 \times X_2 \ldots X_n) \longrightarrow X_j) \times X_1 \times X_2 \ldots X_n \longrightarrow X_j \times (X_1 \times X_2 \ldots X_n) \longrightarrow X_j$$

$$ICS(ATO_k, op_i, (x_1, x_2 \ldots x_n)) = ATO_k(op_i, (x_1, x_2 \ldots x_n))$$

A ICS é o mecanismo responsável pela troca de mensagens entre os ATOs. Se uma operação de um ATO precisar do resultado da execução de uma operação contida em outro ATO então, uma mensagem deve ser enviada àquele ATO com a operação e seus respectivos argumentos. A figura abaixo mostra o relacionamento entre os ATOs e a ICS.

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

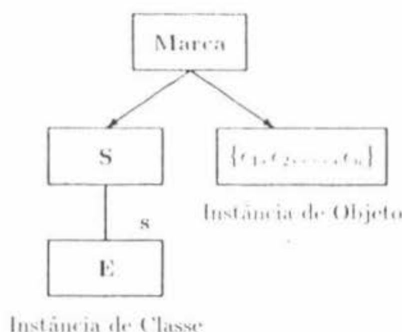


6 Implementação

Para representar as classes foi declarado um tipo na linguagem Pascal que é, basicamente, um record com uma variante para cada tipo de classe. Instâncias deste tipo são representações de classes. Estas instâncias são declaradas como constantes no ATO, implementado como uma função Pascal.

Para representar os objetos de classes foi declarado um tipo para cada tipo de objeto. Instâncias destes tipos são representações de objetos de classes. A declaração destes tipos foi bastante estudada para permitir a construção de algoritmos eficientes. Por exemplo, a representação de objetos da classe set é uma árvore. Assim, por exemplo, o algoritmo que verifica pertinência (verifica se um elemento está ou não no conjunto) é bastante eficiente.

Finalmente, foi declarado um tipo (denominado Marca) para fazer a ligação entre instâncias de classes e instâncias de objetos com a finalidade de permitir a construção de instâncias de objetos baseados em instâncias de classes. A figura abaixo mostra esta ligação.



As operações básicas (sobre sets, maps, etc.) foram implementadas para receber marcas e devolver marcas. Por exemplo

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

$intersection(m_1, m_2)$

tem como argumentos duas marcas para sets e devolve um set que é a intersecção de m_1 com m_2.

Diferentemente, as operações externas dos ATOs foram implementadas sobre instâncias de objetos. Desta forma, mantém-se o conceito de encapsulamento de dados: somente pode tratar uma instância de objeto aquele ATO que conhece a sua classe.

Quando um ATO é invocado via ICS para prestar um serviço (executar uma operação), a primeira providência tomada pelo ATO é ligar a instância de objeto à marca. Só depois o serviço é efetuado.

A ICS, implementada como uma função, simplesmente identifica o ATO origem (parâmetro da função) e chama o ATO correspondente.

7 Mecanismos de Controle

O PROSOFT possui três tipos de usuários: usuário de suporte, projetista de ATOs (chamado de Atista) e usuário final. O usuário de suporte basicamente implementa e otimiza as funções básicas e a ICS. Ele é um profundo conhecedor do PROSOFT e faz parte da equipe de desenvolvimento. O Atista é um usuário das funções produzidas pelo usuário de suporte para produção de novos ATOs, e o usuário final usa os ATOs já prontos.

Para facilitar a implementação de ATOs, foram criados alguns ATOs que estabelecem a comunicação homem máquina. Estes ATOs são: Menu e Cenário. Com o ATO Menu é possível geral novos menus e associar às suas opções nomes de ações (operações de Atos). Com o ATO Cenário o Atista pode abrir janelas e associá las a seus objetos.

Para cada novo ATO, obriga se o Atista ainda a implementar dois ATOs periféricos: um que mantém um diálogo com o usuário final para solicitar os argumentos da operação indicada pelo mesmo e outro que verifica se estes argumentos estão corretos (verifica a pré condição da operação)

Como funciona o PROSOFT? O PROSOFT espera que o usuário final aponte para uma opção de menu. Escolhida a opção, o ATO diálogo pede ao usuário final os argumentos da operação, que são imediatamente verificados. Finalmente, o ATO que contém a operação apontada é chamado. Todas estas transações são efetuadas exclusivamente via troca de mensagens entre ATOs.

O PROSOFT é um ambiente totalmente gráfico. Cada Atista é responsável por uma função de exibição que recebe um argumento do tipo objeto e o exibe. A função exibe do Cenário percorre todas as janelas e solicita serviço aos ATOS para exibirem os objetos associados a elas.

Para estabelecer uma comunicação entre homem e máquina o ATO Cenário possui uma função denominada Identifica Objeto, que recebe uma Coordenada, fornecida pelo Mouse, e devolve o objeto associado a janela que contém a coordenada fornecida.

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

8 PROSOFT Algébrico

Numa extensão do ambiente, foi introduzido no Projeto um modelo adicional para construção ATOs: o paradigma algébrico.

Foi projetado para o PROSOFT um compilador de especificações algébricas escritas na Linguagem Larch Compartilhada [Gut86] para Sistemas de Regras de Reescrita e um executor para esses objetos.

8.1 Visão Global

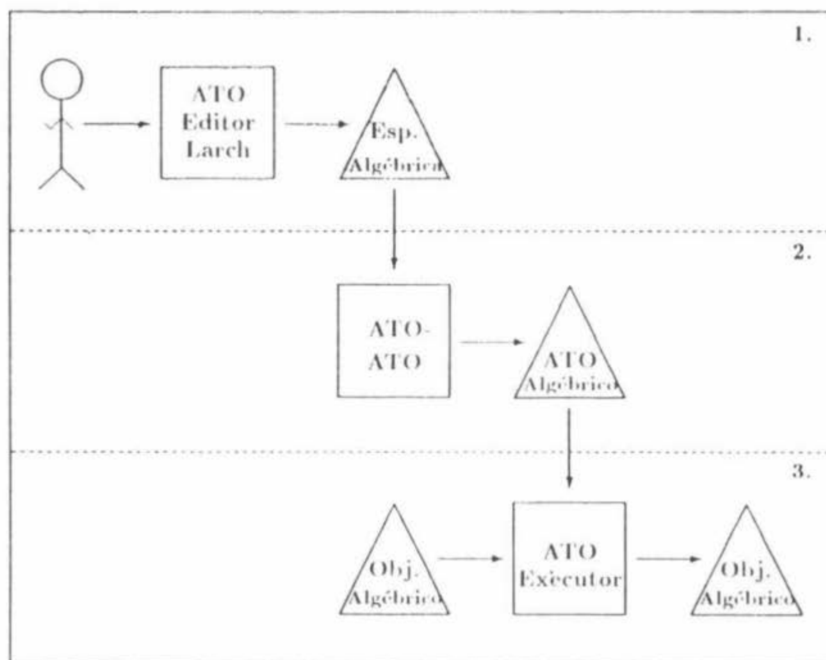


Figure 1: Visão Global

O PROSOFT Algébrico pode ser claramente dividido em três partes:

1. **ATO Editor Larch**
2. **ATO-ATO**
3. **ATO Executor**

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

O Editor Larch é um ATO não algébrico (desenvolvido em Pascal) que interage diretamente com o usuário na criação de especificações algébricas. O produto dessa ferramenta é uma especificação algébrica sintaticamente correta no formato de um objeto tradicional do PROSOFT. A classe que rege esse objeto é a do ATO Editor Larch.

O ATO-ATO (ou **Meta-ATO**) recebe como entrada especificações algébricas, analisa-as verificando quesitos como *completeza* e *consistência* e gera um Sistema de Regras de Reescrita (SRR) [Huert80] confluente na forma de um objeto tradicional do PROSOFT. Esse objeto é um ATO algébrico e é considerado uma nova ferramenta do sistema.

O ATO Executor recebe um objeto algébrico a ser tratado e o conjunto de regras (ATO Algébrico) que serão usadas nesse procedimento. A interação do ATO Executor com o sistema dá-se exclusivamente através da **ICS**.

8.2 Objetos Algébricos

Os objetos/dados gerados pelo sistema consistiam todos em instâncias de classes, já que o PROSOFT seguia apenas uma abordagem orientada a modelos. Com a criação do PROSOFT **Algébrico**, também farão parte do sistema objetos cujo conteúdo não segue o modelo de nenhuma classe especificada. Estes objetos são gerados pela execução de SRR e são denominados *objetos algébricos*.

Um objeto algébrico, usualmente representado na literatura por strings do tipo

push (push (newstack, NSD1), NSD2)

pode ser recursivamente definido como um identificador de uma operação com seus parâmetros, que, por sua vez, são outros objetos algébricos. No caso do PROSOFT **Algébrico**, os parâmetros de um objeto algébrico também podem ser objetos não algébricos.

8.3 Classe Algébrica

A Classe Algébrica é o modelo para a criação de todos os objetos algébricos do sistema. Ela foi concebida de tal forma que:

- os objetos algébricos, do ponto de vista sintático, possam ser manipulados como qualquer objeto. Isto significa que eles são gerados usando-se os construtores[1] já existentes no sistema;
- qualquer tipo de objeto possa ser especificado como argumento de um objeto algébrico;
- o tipo algébrico possa ser especificado como subclasse de um objeto não algébrico e

[1] Tipos usados na construção de classes e, conseqüentemente, de objetos.

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

- a diferença entre um objeto algébrico e um não algébrico seja de natureza semântica e não sintática: o objeto algébrico representa uma operação de um ATO Algébrico, e o objeto não algébrico representa uma instância de uma classe.

8.4 Sistemas de Regras de Reescrita (SRR)

Quando um usuário do sistema desejar criar uma nova ferramenta, ele deverá especificá-la algebricamente na linguagem Larch Compartilhada usando o Editor Larch e solicitar ao ATO-ATO a geração de um SRR.

Esse sistema gerado, através de um mecanismo de prototipação automática, é considerado como o 'código' da nova ferramenta.

No PROSOFT Algébrico, permite-se tanto que operações definidas algebricamente sejam usadas na criação de novas operações em Pascal como também é possível a referência a operações já implementadas em Pascal dentro de especificações algébricas. Dessa forma, no momento da criação de um novo ATO, todas as operações de todos os ATOs[2] do sistema podem ser usadas, e o fato de uma operação estar definida algebricamente ou não fica completamente transparente para o usuário.

8.5 Cache de Atos

O código objeto dos ATOs não algébricos está sempre em memória, pois é ligado junto com o código do programa principal.

O código dos ATOs algébricos, isto é, os SRR, são objetos PROSOFT e, em princípio, estão em disco.

Para viabilizar um tempo de resposta aceitável para a execução das funções algébricas, foi criado um Cache de SRR que consiste em um buffer com o código dos ATOs algébricos mais usados durante uma sessão do sistema.

O Cache é dimensionado por uma constante k correspondente ao número máximo de sistemas em memória ao mesmo tempo, e, sempre que um ATO for requisitado, se ele não estiver no cache, ele será lido do disco para esse buffer. Se o número de ATOs k for ultrapassado, antes de se completar a operação de leitura, será apagado do cache o ATO que menos recentemente foi usado.

8.6 Ordenação de Termos

A ordenação de termos [Der82] é importante para que se possa provar a terminação de um SRR. Um SRR que não possua essa propriedade gera um ATO incorreto cujas operações

[2] É importante ressaltar que o sistema atualmente possui uma vasta biblioteca de funções para manipulação de janelas, textos, menus e arquivos.

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

podem entrar em loop.

Essa etapa de verificação deve ser feita pelo ATO Ordem, que deve entrar em ação logo após a compilação e geração do SRR.

O processo de ordenação consiste em provar que, para todas as regras de um SRR, o lado esquerdo é maior que o lado direito. Essa relação de ordenação dos lados das equações parte de uma relação de ordem parcial que é construída em cima do conjunto de todas as operações do sistema analisado. Deve-se levar em conta que os valores constantes representam funções geradoras de seu sort e são também considerados operadores.

9 Conclusões

O PROSOFT mostrou-se ser um ambiente que contém algumas características das linguagens orientadas a objetos, das linguagens orientadas a modelo (tipo VDM), e das linguagens orientadas a propriedades (tipo OBJ). Sob o ponto de vista da aplicação da estratégia data-driven ele não tem similar e realmente aumenta a produtividade, principalmente pelo reuso de ATOs já existentes. Algumas qualidades do produto, como legibilidade, confiabilidade, etc. são obtidas automaticamente pela simples aplicação da estratégia.

Entretanto existem problemas que ainda não foram resolvidos como a representação gráfica dos objetos. A função de exibição tem sido a função mais difícil de ser implementada apesar dos inúmeros pacotes gráficos colocados à disposição. A razão disto é que não existe, ainda, uma teoria que ligue objetos abstratos a objetos gráficos, com a prova de que este represente aquele. Se isto fosse possível, os objetos abstratos poderiam até ter várias representações gráficas e ficaria, então, a cargo do usuário final escolher aquela que mais lhe conviesse.

Alguns ATOS foram construídos para auxiliar o Atista a construir software. Por exemplo, com o ATO Classe é possível construírem-se classes tal como mostrado anteriormente. Entretanto, para tornar o PROSOFT realmente um ambiente de desenvolvimento, ele deve conter outros ATOS. No momento um Hipertexto, que segue bastante as diretrizes apresentadas por Bigelow [Bigel88], está sendo projetado para recuperar classes, ATOs, operações, etc..

A integração no PROSOFT de ambientes construídos em outros paradigmas não é simples, pois isto implicaria uma conversão de paradigmas. Entretanto a integração no PROSOFT de novos ambientes é automática e atende os requisitos propostos por Ady[Ady88].

Os ambientes que geram código de alguma linguagem de programação não permitem uma análise crítica do código gerado. Por exemplo, não é possível verificar a priori se os programas contêm laços infinitos. Uma versão, ainda embrionária do PROSOFT, permite que o usuário especifique ambientes usando a linguagem LARCH [Gut86]. Um "trait" da linguagem é muito semelhante à especificação algébrica de tipos abstratos de dados. Entretanto, para tais especificações, existem algoritmos que verificam a completeza, a consistência, a confluência, etc., com a vantagem, ainda, delas serem executáveis quando seus axiomas forem transformados em regras de reescrita. Para permitir a realização de algumas experiências, foi construído

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor

um Ato, ATO LARCH, para que o usuário especifique seus ambientes, transforme-os em um sistema de regra de reescrita e as execute.

Gostaria de agradecer a Paulo Roberto Schmitt do Carmo pela ajuda dada na descrição do PROSOFT Algébrico.

Referências

[Carmo91] Carmo, P.R.S. Um estudo comparativo do PROSOFT. Trabalho de Diplomação, UFRGS, 1991

[Watt91] Watt, D.A. Programming Language Syntax and Semantics.C.A.R. Hoare Series Editor. Prentice Hall. 1991

[Dick90] Dick, A.J.J. An introduction to Knuth-Bendix Completion. The Computer Journal, v.34, n.1, 1991.

[Bigel88] Bigelow, J. Hypertext and CASE. IEEE Software 1988. pp 23-27

[Acly88] Acly, E. Looking Beyond CASE. IEEE Software. 1988 pp. 39-49

[Gut86] Guttag, J.V. Horning, J.J. Report on the LARCH shared language. Science of computer programming, v.6, n.2, pp.103-134, March 1986

[Der82] Dershowitz, Nachum. Orderings for term rewriting systems. Theoretical Computer Science, v.17, n.3, March 1982

[Huert80] Huert, G. Oppen, D.C. Equations and rewrite rules. Formal Languages: Perspectives and Problems. Academic Press: London, 1980.

[Scott71] Scott, D., and Strachey, C. Towards a Mathematical Semantics for Computer L anguages. in Proceedings of Symposium on Computers and Automata (ed. Fox, J.), Polytechnic Institute of Brooklyn Press, New York, Unided States, pp. 19-46

PDF compression, OCR, web optimization using a watermarked evaluation copy of CVISION PDFCompressor