RELATÓRIO TÉCNICO

QUICK-SHELL:

UM SISTEMA DE REGRAS DE PRODUÇÃO FORTEMENTE ACOPLADO A BANCOS DE DADOS

Esther de Castro Pacitti
NCE/UFRJ

Pedro Manoel Silveira
NCE/UFRJ

NCE — 04/92
Maio

Núcleo de Computação Eletrônica
Universidade Federal do Rio de Janeiro

Tel.: 598-3212 - Fax.: (021) 270-3842
Caixa Postal 2324 - CEP 20001
Rio de Janeiro - RJ

QUICK-SHELL: Um Sistema de Regras de Produção Fortemente Acoplado a Bancos de Dados

ESTHER DE CASTRO PACITTI
PEDRO MANOEL SILVEIRA

Núcleo de Computação Eletrônica — UFRJ
Caixa Postal 2324, Cep 20001, Rio de Janeiro, RJ, Brasil
E-mail: ncd10122@ufrj.bitnet, ncs01003@ufrj.bitnet

RESUMO

Shells para Sistemas Especialistas e Bancos de Dados empregam tecnologias com vários pontos em comum. O principal deles é que, nos dois casos, armazena-se e manipula-se conhecimento, seja na forma de fatos ou de regras de produção. Este artigo apresenta a arquitetura de uma *shell* para sistemas especialistas, baseada na utilização de regras de produção, fortemente acoplada a bancos de dados baseados no Modelo Entidade-Relacionamento. As regras e todo o processo de dedução, assim como o acesso ao banco de dados, são implementados numa linguagem sob o paradigma de orientação a objetos. Na construção das regras, são permitidas variáveis livres e existenciais, além de termos diretamente mapeáveis para conceitos e construções do Modelo E-R.

ABSTRACT

Expert Systems Shells and Databases employ technologies with quite a few points in common. Their main intersection is that both store and manipulate knowledge, either as facts or rules production. This article presents the architecture of an expert systems shell tightly-coupled with Entity-Relationship databases. Rule evaluation and the reasoning process are implemented under a object-oriented environment, which makes the method comprehensive and quite efficient. Free and existential variables are allowed for building rules and terms refering databases elements are directly mapped to E-R database primitives.

1. INTRODUÇÃO

QUICK-DB é um ambiente completo para definição, acesso, consulta, extração de relatórios e outras atividades relacionadas com a utilização de banco de dados. Construído sobre o paradigma do Modelo Entidade-Relacionamento, QUICK-DB oferece um conjunto completo de ferramentas de apoio a tais atividades, conforme ilustrado na Figura 1. O sistema inclui diversos componentes: dicionário de dados, diagramador gráfico, módulos de consulta e varredura, gerador de aplicações, gerador de banco de dados, interface para sistemas especialistas, entre outros. Há ainda um gerador de rotinas de acesso, permitindo acesso direto ao banco de dados a partir de alguma linguagem de programação hospedeira.

O objetivo deste trabalho é apresentar em maiores detalhes um modulo, denominado QUICK-SHELL, que permite a construção de sistemas especialistas [FORS 84, HARM 85] fortemente acoplados a uma base de dados E-R. Esta integração permite que regras de produção tenham acesso à base de dados de forma simples e natural.

O acoplamento entre bases de fatos O-A-V e bancos de dados E-R é natural devido à existência de termos com formas similares em ambas as disciplinas. Neste caso, particularmente, essas similares são exploradas e até reforçadas com a introdução de atributos referenciais e ordens de acesso (como extensões ao Modelo E-R).

Outros trabalhos apresentam abordagens distintas para a consecução de tal acoplamento [JARK 85, ZANI 86, KERS 90a, KERS 90b]. Neste último, o autor descreve um sistema que permite atributos da base de dados calculados através do processo de inferência. As bases de dados, entretanto, não são genericamente E-R como no nosso caso, além de persistirem as limitações de acesso impostas originalmente pelo modelo.

A característica principal do ambiente aqui apresentado é a total integração entre os vários componentes e a absorção de termos e construções E-R de forma clara e simples. Além disso, a introdução de ordens de acesso e atributos referenciais muda sensivelmente o cenário no que tange às possibilidades de acesso a base de dados. A semântica dos termos que podem ser construídos fica mais poderosa e é possível a especificação de varredura na base de dados, de modo a promover a busca de instâncias que atendam as premissas das regras.

É importante observar que a nossa proposta é muito mais do que uma simples junção de linguagens para sistemas especialistas com banco de dados. Regras de produção, no presente contexto, devem ser vistas como uma forma de atributos virtuais para a base de dados. Esta interpretação e a forma como foram implementados os conceitos de integração permitem que o acesso ao banco de dados fique sensivelmente enriquecido.

Este trabalho procura mostrar os aspectos mais relevantes da fundamentação teórica da implementação de tal acoplamento. A seção 2 caracteriza e delimita a família de banco de dados E-R que são suportados, detalhando os elementos básicos e algumas extensões que

foram por nós introduzidas de modo a obtermos construções robustas e úteis. A seção 3 apresenta como é visto o banco de dados do ponto de vista de QUICK-SHELL: os objetos e métodos disponíveis. Todo acesso à base de dados é feito através desses procedimentos. Na seção 4 é descrito o ambiente de integração. Ali são definidas as bases teóricas que suportam a implementação, descrita na seção 5. A seção 6 relata as conclusões e sugere algumas extensões do presente desenvolvimento.

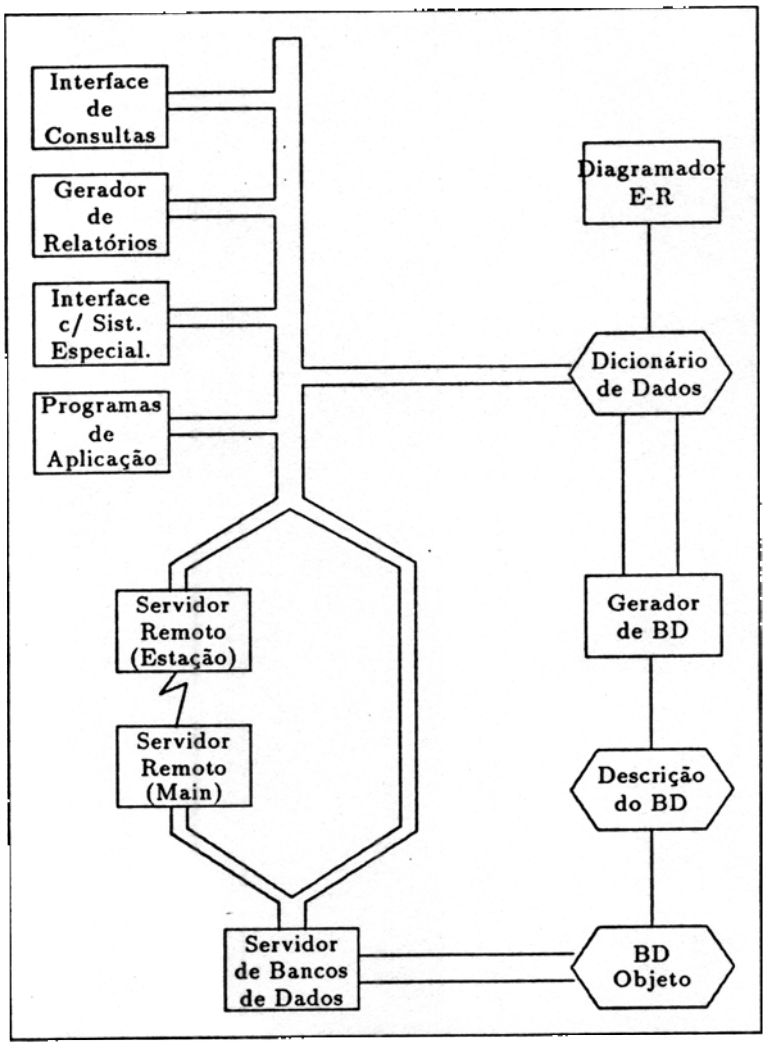

Figura 1. Modelo Funcional do QUICK-DB

2. O MODELO ENTIDADE-RELACIONAMENTO E EXTENSÕES

QUICK-SHELL trabalha com bancos de dados que seguem a disciplina de modelagem do Modelo E-R básico definido por Chen [CHEN 76]. Porém, algumas extensões foram introduzidas para tornar o sistema mais eficiente e completo. Esta seção apresenta, de modo resumido, as características do modelo resultante. Inicialmente são listados os elemento básicos e, em seguida, as extensões definidas.

ELEMENTOS BÁSICOS

Os elementos básicos para a definição de um esquema E-R são: [SILV 89]

a) *Constantes*, representadas por $c, c_1, c_2, ...$; e ϕ para o valor nulo.

b) *Conjuntos de Entidades* representados por $E, E_1, E_2, ...$

c) *Relacionamentos* identificados por $R, R_1, R_2, ...$ A cada relacionamento está associado um inteiro $dg(R)$, dito ser o grau de R, que é o número de entidades ligadas por R. Deste modo, $dg(R) \geq 2$, e $E_1, ...E_{dg(R)}$ é a lista das entidades associadas por R, ou lista das entidades participantes.

d) Para cada relacionamento R existe um conjunto de funções papéis da forma

$$\{\rho_1 : R \rightarrow E_1, ..., \rho_{dg(R)} : R \rightarrow E_{dg(R)}\}$$

onde E_i é uma entidade participante de R. Funções papéis são utilizadas para identificar instâncias das entidades ligadas por um relacionamento R. Uma função papel pode ser também imaginada como um atributo que retorna uma referência à instância da entidade participante que está associada com a instância do relacionamento ao qual o atributo se aplica.

e) *Conjuntos de Valores* $V_1, V_2, ...$, utilizados para definir os domínios dos atributos. A cada conjunto V está associado um conjunto de constantes $\{c_1, ..., c_n\}$.

f) Símbolos de atributos $\alpha_1, \alpha_2...$, que são funções do tipo

$$\alpha : X, I \rightarrow V$$

onde X é um conjunto de entidades ou relacionamento, I é um inteiro, e V é um conjunto de valores. Cada atributo α possui associado um inteiro $maxoc(\alpha)$, $1 \leq I \leq maxoc(\alpha)$. Este inteiro especifica o número de valores associados ao atributo, uma vez que o Modelo E-R admite atributos multivalorados. Assim, para atributos monovalorados este valor é 1 e ≥ 1 para os multivalorados.

g) *Restrições de Integridade* são expressões lógicas $\Theta_1, \Theta_2, ...$ Restrições de integridade abrangem uma ampla gama de propriedades de um esquema, tais como restrições de cardinalidades nos relacionamentos, existência de chaves, limitações de valores de atributos e outras de caráter específico.

EXTENSÕES

O Modelo E-R utilizado no âmbito deste projeto foi estendido pelos conceitos descritos abaixo.

g) O termo *generalização* [PECK 88] é empregado na área de banco de dados quando duas ou mais entidades podem ter suas diferenças abstraídas e, num nível superior, representadas como uma entidade única, a qual abriga os atributos que são comuns aos seus diferentes subtipos. O caminho inverso é chamado *especialização*. Neste trabalho, os conceitos acima foram implementados de forma simples, gerando uma solução robusta e eficiente. Todas as implicações ocasionadas pelos conceitos são suportadas, tais como herança de chaves e ordens (vide abaixo). Multipla herança de atributos não é permitida, ou seja, cada subtipo pode estar ligado a apenas um supertipo.

h) *Atributos referenciais* são atributos que retornam referências a outras entidades do esquema. Este conceito é importante e extremamente útil, uma vez que elimina, em muitos casos, a necessidade de introduzirmos relacionamentos $1 : n$ fictícios, que são melhor expressos com o presente conceito. Formalmente, atributos referenciais são funções da forma

$$\beta : X, I \rightarrow E$$

similarmente à definição no item f acima. Apenas, o contra-domínio de β é um conjunto de entidades E e não um conjunto de valores V como naquele caso. No exemplo da Figura 2, *professor* é um atributo referencial de CURSO e, quando aplicado a uma instância deste, retoma uma instância de PROFESSOR.

i) *Ordens de acesso* permitem acessar conjuntos de entidades ou relacionamentos de tal forma que a ordem em que as instâncias são obtidas fique previamente estabelecida.
Uma ordem pode ser parcial ou total. A ordem é total quando todos os elementos do conjunto de entidades ou do relacionamento que é objeto da ordem podem ser alcançados através da ordem. Ordens podem ser definidas sobre conjuntos de entidades, relacionamentos ou atibutos referenciais. Na Figura 2, a ordem *Alfabética*, definida sobre o conjunto de ALUNO, é uma ordem total. Todas as instâncias de ALUNO podem ser obtidas pela varredura da ordem.
Na mesma figura, a ordem *inscrições* é parcial, porque acessa apenas as instâncias de INSCRIÇÃO associadas a uma determinada instância de ALUNO. Isto é, para cada aluno *al*, *inscrições(al)* é como se fosse um atributo que retorna uma ordem, contendo a lista das instâncias de INSCRIÇÃO associadas àquele aluno. Da mesma forma, *cursos-dados* é parcial e definida sobre a relação inversa do atributo referencial *professor*, que tem CURSO como domínio. Ou seja, para um dado curso v_1 a ordem *cursos-dados*(v_1) retorna todo aluno v_2 tal que *curso*(v_2)= v_1.

j) *Entidades associativas* são relacionamentos que também assumem o papel de entidade participante em outros relacionamentos. Basicamente, entidades associativas têm o mesmo papel de relacionamentos, podendo ter atributos, funções-papéis, ordens e cardinalidades.

3. ACESSO AO BANCO DE DADOS: OBJETOS E MÉTODOS

O acesso à base de dados é feito através de um conjunto de métodos, aplicáveis às classes de objetos que formam a base. O programa que comporta a definição de cada método é gerado automaticamente a partir do diagrama E-R definido para o banco de dados. Das definições de entidades, relacionamentos, ordens e outras características do esquema, o gerador monta todas as rotinas necessárias para acesso a base de dados. Com isso, todos os nomes presentes no esquema têm métodos e/ou atributos correspondentes no programa gerado, o que torna a programação fácil e confortável para quem acessa a base de dados. Este conjunto de métodos é denominado *Servidor-ER*.

Há basicamente quatro classes de objetos: 1) *banco de dados*, 2) *conjunto*, 3) *ordem* e 4) *elemento*.

A classe banco de dados identifica a base de dados como um todo, e aceita os seguintes métodos básicos:

OPEN preparação para acesso;

CLOSE encerramento de acesso;

CREATE geração da base, com instâncias vazias.

Um objeto da classe banco de dados tem ainda um conjunto de atributos que servem para identificação e controles operacionais.

A classe conjunto representa cada um dos conjuntos de entidades E e relacionamentos R existentes na base de dados. Os seguintes métodos se aplicam:

COUNT número de elementos na instância do conjunto;

OPEN preparação para acesso;

CLOSE encerramento de acesso;

CREATE geração inicial;

POINTER obtenção de ponteiro para a instância de um elemento.

Há ainda um conjunto de métodos que servem para manipulação interna de arquivos na base de dados mas que, entretanto, são de acesso restrito e não são relevantes neste contexto.

A classe elemento abriga cada um dos elementos de determinada instância de um conjunto de entidades E ou de um relacionamento R, através da própria classe e de suas subclasses. Os métodos do tipo básico são:

REFERENCE referência física a um elemento dentro de sua instância;

SURROGATE referência lógica que identifica unicamente um elemento dentro da base.

Para cada conjunto de entidades E e relacionamento R, há um subtipo de elemento especificamente definido, que agrega os atributos de E ou R. Um subtipo de elemento, em geral, suporta os seguintes métodos:

STORE adição de elemento;

UPDATE regravação de elemento;

DELETE remoção de elemento;

INSTANTIATE obtenção da instância de um elemento.

Há ainda alguns métodos que tratam especificamente de elementos sujeitos a generalização e especialização. Nestes casos, observa-se que alguns dos métodos acima podem ou não ser aplicados, dependendo das circunstâncias.

Por fim, a classe ordem trata do acesso à base de dados. Uma instância da classe ordem pode ser imaginada como uma lista de referências a elementos da base. Assim, pode-se ter várias ordens definidas sobre um conjunto de entidades ou relacionamento. Através de cada uma dessas ordens, pode-se acessar os elementos segundo critérios de ordenação que são definidos por ocasião da confecção do projeto lógico do banco de dados. A classe ordem admite os seguintes métodos:

RESET posicionamento no início da ordem;

NEXT obtenção do próximo elemento;

EOO teste de fim de lista;

SEARCH busca de um elemento que contenha atributo igual ou posterior a determinada chave;

FIND busca de um elemento que contenha exatamente determinada chave;

OPEN preparação para acesso através da ordem;

CLOSE encerramento do acesso;

DELETEKEY remoção de associação de um elemento a uma determinada chave;

CHANGEKEY alteração de associação de um elemento a uma determinada chave;

ADDKEY inserção de associação de um elemento a uma determinada chave.

4. O AMBIENTE DE INTEGRAÇÃO

Uma vez apresentados os principais componentes do modelo E-R, e os métodos para acesso às bases E-R, podemos definir o ambiente QUICK-SHELL, que consiste da integração dos elementos E-R com os definidos para bases de conhecimento baseadas em triplas O-A-V e

regras de produção [HARM 85]. Normalmente, distingue-se duas classes de "conhecimento" em sistemas de regras de produção: a base de fatos e o conjunto de regras. Neste caso, a base de fatos divide-se em dois tipos: fatos representados através de triplas O-A-V e fatos oriundos da base de dados E-R. Esses fatos aparecem nas regras na forma de termos.

A seguir, apresentamos algumas definições desses elementos sintáticos de forma semi-formal.

TRIPLAS O-A-V

Uma *base de fatos* é constituída por um conjunto de triplas O-A-V da forma (o, α, c). A interpretação de (o, α, c) é definida como: o atributo α de um objeto o assume o valor c. Assim, pode-se considerar a expressão $\alpha(o) \rightarrow c$ equivalente à existência da tripla (o, α, c). [1]

REGRAS

Regras são compostas de literais e ações (descritos abaixo). Uma regra R é definida como

$$v_1 \in X_1, ..., v_m \in X_m$$

$$v_{m+1} \in X_{m+1}/O_{m+1}, ..., v_n \in X_n/O_n$$

$$A(v_1, ..., v_n) \leftarrow L_1(v_1, ..., v_n), ..., L_k(v_1, ..., v_n),$$

$k \geq 0$. X_i representa o domínio da variável v_i, que pode ser um conjunto de entidades ou relacionamento. O_i é uma referência a alguma ordem de acesso do banco de dados, seja total ou parcial.

As variáveis $v_1, ..., v_m$ são ditas variáveis livres, e podem aparecer na regra desde que instanciadas por ocasião da avaliação da mesma.

As variáveis $v_{m+1}, ..., v_n$ são ditas existenciais, porque tem seus domínios restritos às ordens estabelecidas, que são varridas por ocasião da avaliação da regra. Esta varredura faz com que as variáveis existenciais sejam instanciadas, passo a passo, até que a regra seja satisfeita ou se encerre a avaliação.

LITERAIS

Os literais, no ambiente QUICK-SHELL, permitem o uso de operadores relacionais e testes de pertinência. Testes de pertinência são necessários para verificar a existência de algum elemento num conjunto de entidades ou relacionamento. Os literais são definidos no mais

[1] Usaremos para atributos de objetos da base a mesma notação utilizada para atributos E-R. A diferenciação é facilmente percebida no contexto.

alto nível possível de modo a tornar transparente para o usuário as operações específicas do banco de dados [SILV 90]. Assim, foram definidas as seguintes formas para um literal L:

a) $t_1 \, \theta \, t_2$, onde θ representa um operador relacional;

b) $t \in X$, onde X representa um conjunto de entidades ou relacionamento;

c) $t_1 \in \alpha(t_2)$, onde α representa um atributo multivalorado.

Para os três literais definidos acima, t_i é um termo (definido abaixo).

TERMOS

Os termos podem ser categorizados em dois tipos, a saber:

Termos-E

Denotam instâncias de conjuntos de entidades e relacionamentos, e podem ser das seguintes formas:

a) uma variável v, que tem como domínio um conjunto de entidades ou relacionamento;

b) uma expressão $\rho(t)$, onde ρ é uma função papel e t é um Termo-E;

c) uma expressão $\beta(t)$, onde β é um atributo referencial e t é um Termo-E;

d) uma função chave $\Gamma(t_1, ...t_n)$, onde t_i é um Termo-C. Dado um conjunto de valores compondo o identificador, uma função chave corresponde a um operador da base de dados que retorna a instância correspondente. A cada função chave está associado um conjunto de entidades ou um relacionamento, o qual será tomado por base na busca da instância correspondente.

Termos-C

Denotam constantes e podem ser das seguintes formas:

e) uma expressão $\alpha(o)$, onde o é um objeto da base de fatos e α é um atributo de o;

f) um símbolo de função de grau n seguido por uma lista da forma $(t_1, ..., t_n)$, onde t_i é um Termo-C. Esta construção permite o uso de funções *standard*, tais como *seno, coseno, trunc*, etc;

g) uma constante c;

h) uma expressão $\alpha(t)$, onde α é um atributo de t, e t é um Termo-E; [2]

---

[2] O item h, estritamente falando, é da forma $\alpha(t, i)$, onde i referencia o i-ésimo valor da lista de valores de um atributo.

AVALIAÇÃO DE REGRAS E LITERAIS

Para a avaliação de um literal foi definida uma função que aplicada ao seu argumento, retorna um valor no domínio $\{0, 1\}$, correspondendo à avaliação falsa ou verdadeira, respectivamente, de seu argumento. O resultado da avaliação da regra depende da avaliação de seus literais [FORS 84]: é verdadeira se a avaliação de cada um de seus literais for verdadeira e falsa no caso contrário. Neste trabalho, assumimos que os termos não possuem fatores de certeza. Sua introdução, entretanto, não traria alterações para a estrutura básica das regras aqui apresentadas. Apenas, as funções de avaliação mencionadas anteriormente teriam de levar em consideração tais fatores.

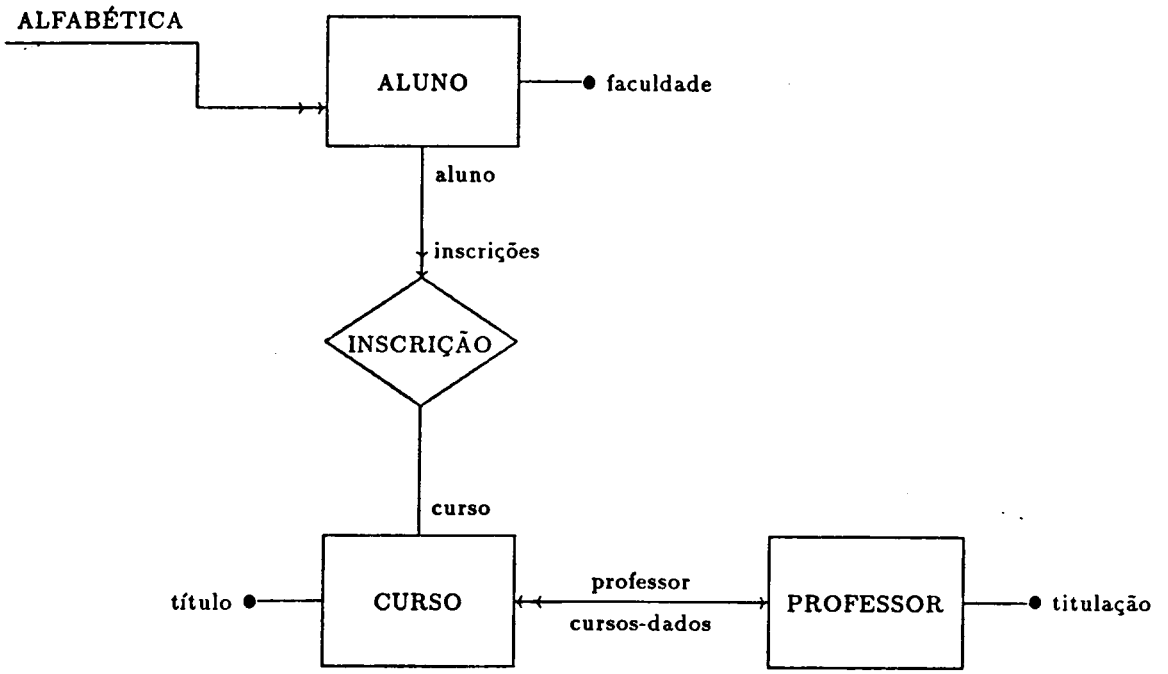

Figura 2. Um exemplo simplificado de um diagrama E-R

5. IMPLEMENTAÇÃO

Para a construção de regras na *shell*, definiu-se uma linguagem de regras onde é possível a definição dos objetos, das variáveis e das regras propriamente ditas. Esta linguagem é então traduzida para comandos de uma linguagem orientada a objetos (O-O) para execução das funções definidas para a *shell*.

A linguagem escolhida foi o Turbo Pascal 6.0 da Borland, que é uma versão com construções sob o paradigma de orientação por objetos. Desta forma, ao longo deste texto algumas construções do Pascal são utilizadas para ilustrar fielmente as soluções adotadas. É impor-

tante salientar que embora o ambiente escolhido seja o Pascal, as soluções expostas aqui são genéricas.

ASPECTOS DO TRADUTOR QUICK-SHELL

Utilizaremos a notação *objeto-b* e *atributo-b* para expressar objetos e atributos da base de fatos O-A-V e, analogamente, *objeto-p* e *atributo-p* no contexto de programação orientada a objetos (POO). A filosofia básica do sistema consiste em mapear cada objeto-b e a base de regras como um todo em objetos-p.

Cada objeto-b é traduzido em um objeto-p com o mesmo nome. Da mesma forma os atributos que compõem o objeto-p são exatamente os mesmos que descrevem o objeto-b. As ações, ou métodos, de cada objeto-p definido se referem às funções que obtêm os valores de cada atributo do referido objeto. Para exemplificar, considere a base da Figura 3. A tradução do objeto-b *processo* fica sendo

```
processo = object
  titulacao_exigida_      : integer;
  titulacao_              : string;
  parecer_                : string;
  function  titulacao_exigida  : integer;
  function titulacao           : string;
  function parecer             : string;
end;
```

O valor de um atributo é obtido através da avaliação da função correspondente. Esta função verifica se o atributo já foi instanciado ou não e verifica também se o atributo deve ser obtido por inferência ou através de pergunta ao usuário.

Na tradução das regras da linguagem para a linguagem de implementação, a estrutura do código gerado para cada tipo de termo aparece abaixo, segundo os tipos de termos definidos na Seção 4:

a) $v \rightarrow v$;

b) $\rho(v) \rightarrow v.\rho$;

c) $\beta(v) \rightarrow v.\beta$;

d) $\Gamma(t_1, ...t_n) \rightarrow X.nome.\text{FIND}(t_1, ..., t_n)$;

e) $\alpha(o) \rightarrow o.\alpha$;

f) $f(t_1, ..., t_n) \rightarrow f(t_1, ..., t_n)$;

g) $c \rightarrow c$;

h) $\alpha(v) \rightarrow v.\alpha$;

i) $att(t) \rightarrow X.\text{pointer}(t')\hat{}.att$.

Os itens a, e, g, h referem-se às traduções das variáveis, termos simples e constantes. Os itens b, c dizem respeito à tradução de funções papéis e atributos referenciais. O item d trata de chaves do banco de dados. A identificação de X e *nome*, naquele caso, deve ser obtida a partir da definição de Γ. O item i corresponde à tradução de termos compostos, onde X representa o domínio de t, e t é um termo. Neste caso, a identificação de X é obtida através de uma consulta ao dicionário de dados, uma vez que X é o domínio de t. t' representa a tradução do termo original t.

As ordens podem ser vistas como um tipo especial de termos que simbolizam funções que retornam listas de identificadores de instâncias de um domínio específico. Internamente, uma ordem é um objeto que encapsula uma memória interna e um conjunto de procedimentos e funções [TADA 90]. Assim, todos os objetos desse tipo possuem comportamento similar no sentido de que os mesmos métodos se aplicam. Na codificação das regras, uma ordem é traduzida conforme as transições mostradas abaixo:

a) $X.O \rightarrow X.O$;

b) $O(t) \rightarrow X.\text{pointer}(t')\hat{}.O$.

onde t simboliza um termo e t' sua tradução. O item a refere-se a ordens ditas *totais* porque têm acesso a todo conjunto domínio. No exemplo da Figura 2, ALUNO.*alfabética* é uma ordem total, que pode ser imaginada como uma lista, em ordem alfabética, de todas as instâncias daquela entidade. A segunda expressão representa uma ordem parcial, ilustrada no exemplo por *inscrições*. Esta possibilita, no domínio INSCRIÇÃO, o acesso a todas as instâncias relacionadas com um determinado aluno.

MECANISMO DE INFERÊNCIA

O processo de inferência se dá através do encadeamento das chamadas às rotinas traduzidas a partir das regras. Na tradução, ainda, são geradas chamadas a uma rotina de empilhamento que guarda a ordem de busca às regras de forma que o usuário possa, ao final da execução, verificar a ordem das regras disparadas. Esta facilidade pode ser percebida na linguagem através do comando *mostre*.

O agrupamento das regras é feito observando-se as suas conclusões. Regras que concluem valores de um mesmo atributo são agrupadas namesma rotina gerada. Se a primeira falhar, testa-se a segunda possibilidade, na ordem em que as regras aparecem na base.

Cabe ressaltar que a utilização de grafos AND/OR e do algoritmo de *backtracking* se dá

de forma implícita, pois a própria organização e representação do conhecimento através da orientação a objetos conduz a esta tarefa de forma simples e estruturada.

```
function processo.titulacao_exigida : integer;
begin
     if processo.titulacao_exigida_ = null_integer
     then begin
             if processo.titulacao = 'MESTRADO'
             then begin
                     processo.titulacao_exigida_  := 6
                  end
             else
             if processo.titulacao = 'DOUTORADO'
             then begin
                     processo.titulacao_exigida_  := 7
                  end
          end;
     titulacao.exigida := titulacao_exigida_ ;
end; {titulacao_exigida}

function processo.titulacao : string;
begin
     if processo.titulacao_ = null_string
     then begin
             write  ('Forneca titulacao(processo):');
             readln (processo.titulacao_);
          end;
     titulacao := processo.titulacao_ ;
end; {titulacao}
```

Figura 4. Parte do código gerado a partir da base exemplo.

EXEMPLOS

As figuras 4 e 5 mostram exemplos das rotinas que seriam geradas a partir das regras mostradas na Figura 3. Em todos os exemplos, há tipos e variáveis previamente definidos, que são gerados a partir da definição do banco de dados E-R. Na apresentação desta seção, os autores supõem uma certa familiaridade dos leitores com o Pascal O-O. Isto, entretanto, pode ser facilmente superado com uma breve consulta aos manuais do fabricante do referido *software*.

```
% Base Exemplo
% Declaracao de objetos
%
processo = objeto
    titulacao_exigida : inteiro [INFER];
    titulacao         : alfa    [PERG];
    parecer           : alfa    [INFER];
fim
%
#
% Base de Regras
%
Regra 1
%
SE    titulacao(processo) = 'MESTRADO'
ENTAO
      titulacao_exigida(processo) := 6
%
Regra 2
%
SE    titulacao(processo) = 'DOUTORADO'
ENTAO
      titulacao_exigida(processo) := 7
%
Regra 3
%
SE   para algum al  em  ALUNO      por  ALUNO.alfabetica
     para algum ins em  INSCRICAO  por  inscricoes(al)
          faculdade(al) = 'INFORMATICA'
       E  titulo(curso(ins)) = 'ALGEBRA'
       E  titulacao(professor(curso(ins))) >= titulacao_exigida(processo)
ENTAO
      parecer(processo) := 'FAVORAVEL'
#
% Comandos
%
DETERMINE parecer(processo)
MOSTRE
```

Figura 3. Base exemplo

As regras 1 e 2 geram a rotina `processo.titulacao-exigida`, uma vez que ambas possuem atribuções para o atributo *titulação-exigida* do objeto *processo*. Note que as duas regras foram condensadas numa estrutura *if-then-else*. O teste de validade para o valor do atributo permite que o sistema descubra se o atributo já foi calculado ou não em alguma chamada prévia.

Outro aspecto interessante é a obtenção do valor do atributo *titulação* do mesmo objeto *processo*. Na declaração da base de fatos O-A-V, na Figura 3, o atributo em questão é definido como sendo obtido através de pergunta ao usuário. Na rotina gerada correspondente, o sistema gera um procedimento de leitura do valor do atributo quando a rotina for invocada. Novamente neste caso, a possibilidade de que o atributo tenha sido obtido numa chamada anterior é levada em consideração.

Para a Regra 3 da Figura 3, a rotina gerada é um pouco mais complexa. Isso deve-se à utilização das variáveis existenciais *al* e *ins*, definidas sobre os domínios ALUNO e INSCRIÇÃO, respectivamente. É preciso varrer cada uma das ordens, cada uma por sua vez, até que se encontre alguma instanciação de *al* e *ins* que verifiquem as premissas da regra. Esta varredura é feita através dos métodos RESET, que estabelece as condições iniciais da varredura, e NEXT, que dá acesso ao elemento seguinte dentro da ordem. O método EOO auxilia na deteção da condição de fim de lista. As varreduras são expressas pelo uso do procedimento WHILE do Pascal. Pode-se também observar na Figura 5 aspectos da tradução de alguns tipos de termos, conforme explicado acima.

O comando *determine*, encontrado no exemplo das regras, é traduzido para uma chamada à rotina correspondente ao atributo examinado. Esta parte não aparece no exemplo, pois dependeria de algumas considerações inerentes à implementação em questão, tais como tipo de dispositivo de saída, ambiente de software e outros fatores. Observa-se ainda que as rotinas geradas podem ser transportadas para outros programas, estabelecendo uma maneira fácil e portátil para a implementação de atributos virtuais para a base de dados.

6. CONCLUSÕES

Este artigo apresentou aspectos do acesso a banco de dados a partir de uma *shell* para sistemas especialistas. Mostrou-se o modelo E-R básico utilizado, os métodos para acesso à base de dados através de uma linguagem O-O e os fundamentos teóricos da integração. Exemplos de programas que executam a base também foram apresentados.

Como mencionamos na Introdução, o presente trabalho deve ser visto como uma forma de consulta às bases de dados, abrindo o leque de opções dos usuários no que se refere a linguagens de consulta e atributos virtuais.

Quanto à implementação, tem-se no momento uma versão experimental em funciona-

```
function processo.parecer : string;
var
    ord00, ord01 : T_ORDEM;
    al           : T_ALUNO;
    ins          : T_INSCRICAO;
 begin
   if processo.parecer_ = null_string
   then begin
           ord00 := ALUNO.alfabetica;
           ord00.reset;
           acabou := false;
           while (not ord00.eoo) and (not acabou) do
           begin
              al.instantiate(ord00.next);
              ord01 := al.inscricoes;
              ord01.reset;
              while (not ord01.eoo) and (not acabou) do
              begin
                 ins.instantiate(ord01.next);
                 if  ( al.faculdade = 'INFORMATICA')
                 and (CURSO.pointer(ins.curso)^.titulo = 'ALGEBRA')
                 and (PROFFESSOR.pointer
                        (CURSO.pointer
                           (ins.curso)^.professor)^.titulacao
                      >= processo.titulacao_exigida)
                 then begin
                         parecer := 'FAVORAVEL';
                         acabou  := true;
                      end
              end {while}
           end {while}
        end;
end; {parecer}
```

Figura 5. Parte do código gerado a partir da base exemplo.

mento, capaz de gerar programas completos que utilizam a *shell* de acesso aos dados. Também está em curso a implementação de uma interface amigável, que obtém informalmente do usuário a definição da base de fatos e regras. De posse das estruturas de dados obtidas,

invoca-se as rotinas de geração de código.

O fato do tradutor do QUICK-SHELL gerar código compatível com o Pascal dá grande liberdade ao usuário, considerando-se que o código pode ser modificado, estendido e ligado a outros sistemas previamente desenvolvidos. Esta facilidade pode ser considerada bastante oportuna, dado que atualmente existem várias áreas de pesquisa em informática que desenvolvem projetos que exigem a utilização de sistemas especialistas internamente ao software desenvolvido.

Adicione-se a isto a possibilidade de, sem mudar substancialmente a estrutura do sistema, mudar a linguagem de programação na qual são gerados os programas que executam a *shell*. Por exemplo, a utilização da linguagem C++ está em estudos, uma vez que as transformações requeridas para a migração são facilmente exequiveis. Isto proporcionaria ao sistema alto grau de portabilidade.

Pretendeu-se também neste trabalho introduzir uma nova forma de implementação de *shells*, que mostrou ser mais fácil de ser desenvolvida e apresentou eficiência na execução.

7. REFERÊNCIAS

[CHEN 76] CHEN, P.: The Entity-relationship Model - Toward a Unified View of Data, *ACM Transactions on Database Systems 1*, (1976).

[FORS 84] FORSYTH, R.:, *Expert Systems: Principles and case studies*,. Chapman and Hill, 1984, USA.

[JARK 85] JARKE, MATTHIAS, VASSILIOU, YANNIS: Coupling Expert Systems with database Management Systems. In: Reitman, W. (ed). *Artificial Intelligence Applications for Business*, Norwood, NJ:Ablex, 1985, cap. 5 p. 65-85.

[KERS 90a] KERSCHERBERG, L.: Expert Database Systems: Knowledge/Data Management Environments for Intelligent Information Systems, *Information Systems* Vol. 15, No. 1, pp 151-160, 1990.

[KERS 90b] KERSCHERBERG, L.: KORTEX: An Expert database system shell for knowledge based entity relationship model, *Int. Conf. on the Entity/Relationship Approach*, pp 174-187, 1989.

[PECK 88] PECKHAM, M. and MARYANSKI, F.: Semantic Data Models, *ACM Computing Surveys 20*, 3(September 1988).

[SETZ 86] SETZER, W.: *Projeto Lógico e Projeto Físico de Bancos de Dados*, V Escola de Computação, Belo Horizonte, 1986.

[SILV 89] SILVEIRA, P. M.: A Formalization of The E-R Model, *Proceedings of the IX*

Conferencia Internacional de la Sociedad Chilena de la Ciencia de la Computation, Santiago, Chile, 1989.

[SILV 90] SILVEIRA, P. M.: Procedural Data Manipulation Operations for the E-R Model, *Proceedings of the XVI Conferência Latino Americana de Informática*, Assunção, Paraguay, 1990.

[TADA 90] TADAO, T.; HANS K. E. L.: *Programação Orientada a Objetos*, VII Escola de Computação, São Paulo - 1990.

[ZANI 86] ZANIOLO, C.: Object Oriented Database Systems and Knowledge Systems, *Expert Database Systems* pp 49-65, 1986.