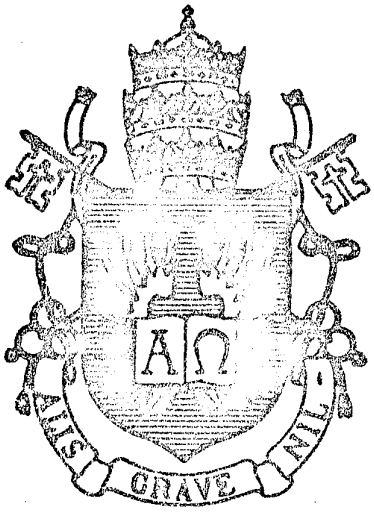


PUC

Série: Monografias em Ciência da Computação
Nº12/88

# UM NOVO PARADIGMA PARA O PROBLEMA DE REUTILIZAÇÃO DE SOFTWARE

José Reinaldo Silva
Carlos José Pereira de Lucena

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
MARQUÊS DE SÃO VICENTE, 225 – CEP 22453
RIO DE JANEIRO – BRASIL

PUC/RJ - Departamento de Informática

Série: Monografias em Ciência da Computação, Nº12/88
Editor: Paulo A. S. Veloso outubro de 1988

# UM NOVO PARADIGMA PARA O PROBLEMA DE REUTILIZAÇÃO DE SOFTWARE*

José Reinaldo Silva**
Carlos José Pereira de Lucena

* Trabalho parcialmente financiado pelo CNPq e pela SID Informática.
** Depto. de Engenharia Elétrica, Escola Politécnica, USP.

Responsável por publicações

Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC/RJ - Depto. de Informática
Rua Marquês de São Vicente, 225 - Gávea
22453 - Rio de Janeiro, RJ

Resumo

Este trabalho apresenta as idéias centrais de um novo paradigma para o tratamento do problema de reutilização de software baseado em uma teoria formal de metáforas intitulada Transferencia Semântica Restringida [1]. Fugindo dos enfoques usuais para o problema de reutilização, que têm produzido resultados desanimadores, os autores esperam contribuir na busca de soluções para este problema central da Engenharia de Software. O trabalho apresenta um exemplo completo da adaptação de novos conceitos da área de Ciência da Cognição à reutilização de software.

# 1. Introdução

A solução do problema de reutilização de software propicia a criação de uma nova cultura de programação, na medida em que ao mesmo tempo incentiva a modularização e a busca de soluções de mais alto nível [2]. As abordagens convencionais para o problema de reutilização não são muito eficientes porque a maioria dos métodos disponíveis para o tratamento das fases de especificação e "design" têm ainda caráter muito geral. A eficiencia destes métodos é em geral, inversamete proporcional a suas abrangencias [3], o que deixa em aberto a possibilidade de novas abordagens.

Sucintamete, o problema de reutilização consiste em:

i) Achar a componente de software desejada;

ii) Resgatar suas especificações;

iii) Introduzir as modificações necessarias (se houver);

iv) Integrar a componente ao resto do código.

O primeiro item poderia ser utilizado para caracterizar as diversas abordagens ao problema de reutilização uma vez que [illegible] (vide o final do texto) [illegible].

Metodologias "building blocks", "subroutine libraries", e outros voltados para linguagens dirigidas para objetos, se propõe a achar uma peça de código reutilizavel, e têm obtido bons resultados para aplicações bem particulares.

A dicotomia entre abrangencia e eficiencia é gerada pela estreita relação entre os items i) e ii), isto é, quanto mais informação se tem sobre a componente potencialmente reutilizável mais eficiente se torna o processo de reutilizaçào.

Isto sugere a reunião das etapas i) e ii) em uma etapa única. Esta é uma das premissas da abordagem que estamos propondo: conseguir um metodo eficiente para realizar a busca por uma componente reutilizável com um mínimo de informação prévia sobre a componente.

Como para os outros metodos citados acima o tratamento proposto pode ser caracterizado pela metodologia utilizada nesta fase - que descrevemos adiante.

Uma vez encontrado um modulo candidato a reutilização resta ainda a seguinte questão: é compensador reutilizar a componente, isto é, prosseguir com as etapas iii) e iv)?. Com a atual disciplina de programação esta questão tem se mostrado absolutamente não trivial [4]. Portanto, a continuação desta pesquisa consiste em prover a ferramenta em desenvolvimento para fase de especificação e reutilização de "designs", de um mecanismo de apoio à realização da análise do impacto e complexidade de manutenção e integração de um componente selecionado.

Como já foi mencionado anteriormente a nossa proposta de abordagem pode ser caracterizada pela metodologia que utlizaremos para "buscar uma componente reutilizável".

Teremos como premissa o fato de que na maioria das situações de interesse prático, a reutilização se faz necessaria justamente quando a quantidade de informações sobre a componente potencialmente reutilizável é bastante limitada (a fase de especificação e design).

Por outro lado, não estamos supondo a existencia de um modulo a que se possa aplicar o modelo "use-as-it-is". Temos portanto o problema de mapear uma descrição da componente desejada usando um subconjunto de assertivas à priori verdadeiras nesta descrição.

Resumidamente, gostariamos de mapear uma teoria a partir de um conjunto de seus axiomas - onde os termos teoria e axioma tem o significado usual utilizado em Teoria de Modelos [5].

A aplicação de "pattern matching" para este fim tem a desvantagem de fornecer quase sempre um numero muito grande de possibilidades em cada tentativa de mapeamenteo, comprometendo assim a performance do algoritimo - apesar da facilidade de implementação deste approach.

A nossa intuição para buscar na Ciencia da Cognição um novo paradigma para reutilização reside no fato de que analogias e modelos científicos podem ser vistos como casos especiais de *metáforas*. Indurkhya [1] enumera, entre outras, as seguintes caracteristicas de metáforas que ele adotou para o desenvolvimento da sua teoria, e que motivaram o nosso interesse no seu trabalho como um subsídio para o problema da reutilização:

- metáforas usam termo(s) pertencentes a um domínio, chamado domínio fonte, para se referir a objeto(s) pertencentes a um domínio, possivelmente diferente, chamado o domínio objeto;
- o domínio fonte e objeto não precisam ser diferentes nem disjuntos; eles podem ser os mesmos ou podem se superpor parcialmente;
- uma metáfora funciona transferindo coerentemente um conjunto de relações estruturais do domínio fonte para o domínio objeto;
- a adequação de uma metáfora é determinada, pelo menos em parte, pela quantidade da estrutura do domínio fonte que é transferida coerentemente para o domínio objeto.

Acreditamos que a aplicação de teoria de metáforas pode tornar a modularização do software que é produzido relativamente independente da modularizaçào da componente potencialmente reutilizavel, permitido assim a elaboraçào de um metodo mais abrangente.

A utilização deste enfoque traz ainda a vantagem de permitir um tratamento mais formal do problema e, por conseguinte, a elaboração de soluções mais portáveis.

Em contrapartida, a complexidade de implementação aumenta bastante com a aplicação do CST[1] e do seu modelo computacional, o AST (Aproximate Semantics Transference)[6]. Entretanto, ambos podem ser formalizados utilizando logica de primeira ordem, e portanto, são passíveis de implementação usando programação logica.

Na próxima seção aparesentaremos, em linhas gerais, a formalização do CST enfatizando os conceitos mais importantes do ponto de vista de reutilização.

## 2. O Método de Transferencia Semântica : Uma Teoria Formal de Metáforas.

Nesta seção apresentaremos um metodo alternativo para tratar a busca por uma componente reutilizavel : o metodo de transferencias semânticas, fundamentado em teoria de metáforas.

Metáforas são entidades indispensaveis em processos de interação cognitiva, onde se supõe transferencia de conhecimento, e porisso estão presentes em todas as linguagens. O uso comum de expressões figurativas na linguagem natural, como "chove canivetes" por exemplo, é o melhor exemplo de como uma associação simples de palavras pode subistituir toda uma explicação elaborada sobre as condições do tempo. Uma extensão deste conceito intuitivo de metáfora tem sido aplicado no desenvolvimento das ciencias [7] e no estudo de processos cognitivos [8].

Um outro exemplo simples, embora menos coloquial, é o uso de relações familiares (como "pai", "filho", "parente", etc.) para se referir a uma especie particular de graíos, as arvores. Quado nos referimos a dois nós A e B dizendo que "A é o pai de B" estamos usando a relação estrutural "pai de" válida para pessoas, para representar o seguinte: "existe a ligação cujos vertices são os nós A e B onde A tem profundidade p e B tem profundidade p+1". Note-se a diferença de terminologia nos dois casos.

No exemplo acima dois dominios podem ser identificados : i) o dominio das relações familiares, que consiste de um vocabulario - composto de sentenças como "pai" , "é filho de", e relações do tipo "x e y são irmãos se ambos têm o mesmo pai" - e de relações estruturais (como por exemplo, "cada filho tem somente um pai", etc.); ii) o dominio referente aos graíos simples direcionados, cujo vocabulario inclue sentenças como "nós", "existe uma ligação entre x e y", etc. e de relações estruturais do tipo

$$\forall x \forall y \, [\, \text{ligação}(x,y) \rightarrow \neg \, \text{ligação}(y,x) \,]$$

que expressa o fato de que o graío é orientado. Uma abordagem formal deste exemplo é feita em [1].

Podemos então caracterizar uma metáfora como uma descrição de um dominio (chamado dominio alvo) em termos de outro dominio (chamdo dominio fonte). No exemplo mencionado acima o dominio fonte é o dominio das relações familiares e o dominio alvo é o dominio que representa as árvores.

Black [8] formalizou a teoria de metáforas como uma parte de sua "teoria da interação". Entretanto, a computabilidade da teoria não foi tratada, significando que: se por uma lado uma metáfora pode ser um mecanismo cognitivo eficiente - no sentido de poder conter varios conceitos e relações em uma imagem simples expressa em um vocabulario mais familiar - , há, por outro lado, o risco de que uma metáfora possa conduzir a interpretações erradas sobre o dominio alvo.

Voltando ao exemplo acima - onde o dominio alvo representa todas as arvores e florestas, e o dominio fonte representa todas as relações familiares, vamos considerar a sentença: "B é ancestral de F". Esta sentença poderia ser *mapeada* no dominio alvo com o seguinte interjpretação: "B tem profundidade menor do que F", enquanto que a interpretação pretendida era "B está no caminho que vai de A a F" (figura 1).

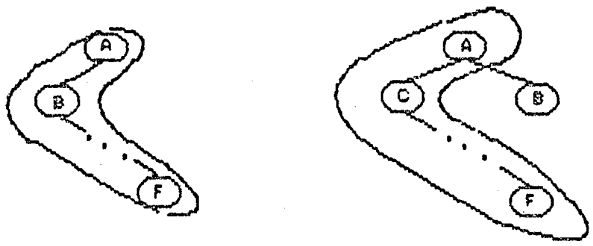


figura 1: Duas diferentes interpretações para a metáfora "B é ancestral de F".

Para garantir que a metáfora seja apropriada, isto é, que a interpretação correta seja tomada, basta, neste caso, fazer com que a metáfora "transporte" para o dominio alvo as relações estruturais do dominio fonte. Neste exemplo, a relação,

$$\text{ancestral}(X) :- \text{pai}(X) \vee \text{ancestral}(\text{pai}(X))$$

é suficiente para garantir que a interpretação correta seja tomada no dominio alvo.

A confiabilidade destes processos de mapeamento de interpretações sobre o dominio alvo usando metáforas foi restaurada por Indurkhya [1] com o Constrained Semantic Transference (CST).

O CST é uma formalização da teoria de metáforas enunciada por Black [8], onde o conceito de consistencia[1] é usado como critério para a escolha das metáforas apropriadas, isto é, as que preservam as estruturas do dominio fonte.

A seção seguinte contém algumas das definições mais importantes do CST principlamente os mapeamentos que transferem conhecimento entre dominios.

## 2.1 Uma formalização da teoria de metáforas

Um conceito elementar para a transferencia semantica é o de "vocabulario". Se V é um conjnto de palavras (constantes) sobre uma dada linguagem, e

$$\mathfrak{V} = \{o_0, o_1, \ldots, p_0, p_1, \ldots\}$$

(1) O termo consistencia deve ser entendido no senúdo usado em Teoria de Modelos: um conjunto de sentenças S sobre um vocabulario V é dito consistente se admite algum modelo. Para maiores detalhes veja C. C. Chang e H. J. Keisler, "Model Theory", North-Holland Publishing Co., 1978

é um conjunto de tipos, onde os $o_i$ são operações i-narias e os $p_i$ são predicados de ordem i. Seja f uma função $f : V \dashrightarrow \mathbb{N}$ que associa um tipo a cada constante em V. Um vocabulario é um par $\langle V, f \rangle$. Um exemplo simples de um tal vocabulario é

$$V = \{nó, raiz, aresta, folha\}$$

$$f(X) = \begin{cases} p_1 & \text{se } X = raiz, folha, nó \\ p_2 & \text{se } X = aresta \end{cases}$$

Def] Um dominio é definido como uma quadrupla $\langle V, f, S, S_d \rangle$ onde o par $\langle V, f \rangle$ representa o vocabulario sobre o qual se define o dominio. S é um conjunto de sentenças sobre este vocabulario, chamado restrições estruturais (structural constraints), que restringem as interpretações semanticas sobre as sentenças que se pode formar com o vocabulario $\langle V, f \rangle$. $S_d$ é um subconjunto de S cujos elementos são derivações (definições) de palavras em $\langle V, f \rangle$.

No dominio das arvores orientadas mencionado acima a sentença

$$\forall x \, [\, folha(x) \Leftrightarrow \forall y \, [\, nó(y) \Rightarrow \neg \, aresta(x,y) \,] \,]$$

pertence a S, uma vez que restringe as interpretações possiveis do "token" folha(x) aos casos em que não existem arestas começando em x. Por outro lado, esta mesma sentença serve como uma definição de "folha" e portanto pertence a $S_d$.

Um mapeamento entre dois dominios $D_1 = \langle V_1, f_1, S_1, S_{d1} \rangle$ e $D_2 = \langle V_2, f_2, S_2, S_{d2} \rangle$ é uma função parcial $F$: $V_1 \dashrightarrow V_2$. Assim, se o domínio $D_1$ for o domínio das arvores direcionadas e $D_2$ for o dominio das relações familiares, um mapeamento entre estes dominios poderia fazer a transferencia nó --> pessoa, em que um nó no grafo direcionado é associado com uma pessoa na arvore genealogica. Evidentemente, nó e pessoa devem ter a mesma ordem, isto é, $f_1(nó(x)) = f_2(pessoa(x)) = p_1$. A conservação do tipo ( $p_1$) é essencial para se assegurar a coerencia da metáfora.

Def] Um mapeamento $F$ entre os dominios $D_1 = \langle V_1, f_1, S_1, S_{d1} \rangle$ e $D_2 = \langle V_2, f_2, S_2, S_{d2} \rangle$ é dito *admissível* se, para todo v pertencente a $V_1$, se v pertence a $F$ então $f_1(v) = f_2(F(v))$.

Portanto, qualquer subestrutura S (S subconjunto de $S_1$ ) pode ser transformada em uma nova subestrutura, desde que cada constante e/ou predicado em S pertença a $S_2$ ou pertença a um mapeamento admissível $F$ de $D_1$ em $D_2$. A transformabilidade não é entretanto condição suficiente para garantir a coerencia de uma metáfora.

Em primeiro lugar, uma metáfora $T = \langle F, S \rangle$ é caracterizada por um mapeamento admissível $F$ e por uma subestrutura transformável S do dominio fonte que se deseja projetar sobre o dominmio alvo. Uma tal metáfora é dita coerente se o conjunto de sentenças $S' = F(S) \cup S_2$ for consistente[2]. Evidentemente, o conceito de consistencia não é computável e portanto, um prototipo de ferramenta de reutilização baseada em metáforas deve utilizar de métodos alternativos para verificação da coerencia das transforamações semanticas executadas. Este problema será discutido na seção seguinte.

Um caso especial de uma metáfora coerente - bastante útil no caso de reutilização de software - é quando $S_2$ é um modelo para $F(S)$. Neste caso, a metáfora é dita fortemente coerente.

Finalmente, enfatizaremos o conceito de inversibilidade de uma metáfora, que nos será útil para, uma vez identificada a descrição (especificação) da componente a ser reutilizada, permitir ao usuário incorporar uma subestrutura desta componente ao dominio fonte. Em outras palavras estariamos reutilizando também as especificações da componente reutizável.

Teorema] Seja $T = \langle F, S \rangle$ uma metáfora onde $F$ é um mapeamento injetivo entre os dominios $D_1$ e $D_2$. Então existe um mapeamento inverso $F^{-1}$ de $D_2$ em $D_1$ e uma metáfora inversa $T^{-1} = \langle F^{-1}, F(S) \rangle$.

Se a metáfora é fortemente coerente, a metáfora inversa não é necessariamente fortemente coerente, isto é, embora pelo teorema acima a coerencia seja preservada por inversão o mesmo não acontece com o caso especial de metáforas fortemente coerente. Isto implica que, se $S' = F(S)$, podemos garantir apens a consistencia de $F^{-1}(S') \cup S1$ e portanto coloca a possibilidade de um mecanismo de reutilização baseado em metáforas poder fornecer ferramentas de apoio à fase de especificação de software.

Apesar desta possibilidade não ter ser sido ainda plenamente explorada neste trabalho as operações definidas sobre metáforas, como "Aumento", servem como suporte para esta hipótese. As orações sobre metáforas estão definidas na seção seguinte.

## 2.2 Operações sobre metáforas

Como vimos na seção anterior, o conceito de coerencia de uma metáfora está intimamente associado ao conceito de consistencia de um conjunto de sentenças. De acordo com o teorema de Lindenbaum [5], qualquer conjunto de sentenças sobre um dado vocabulário que seja consistente, pode ser extendido até formar um conjunto maximal. Assim, dado uma metáfora coerente $T = \langle F, S \rangle$ de D1 em D2, onde

$$D_1 = \langle V_1, f_1, S_1, S_{d1} \rangle \text{ e}$$
$$D_2 = \langle V_2, f_2, S_2, S_{d2} \rangle \ .$$

(2) Veja nota (1) na pagina 2.4

e sendo $v_1$ uma constante pertencente a $V_1$ e $d_1$ sua respectiva derivação (definição), que naturalmente deve ser uma sentença pertecentea $S_{d1}$. Seja também uma constante $v_2$ que não pertence a $V_2$. Adimitimos ainda que $v_1$ não pertence ao mapeamento $F$. A operação *Aumento* é tal que

$$Aumento(D_1, D_2, T, v_1, d_1, v_2) = \langle T', D'_2 \rangle$$

onde $D'2 = \langle V'_2, f'_2, S'_2, S'_{d2} \rangle$ é o novo dominio alvo extendido, isto é,

$V'_2$ = $V_2 \cup \{ v_2 \}$;
$f'_2$ = $f_2$ acrescido da relação $f'_2(v_2) = f_1(v_1)$ que conserva a adimissibilidade do mapeamento $F$;
$S'_2$ = $S_2$ acrescido à subestrutura $\{ F'(d_1) \}$;
$S'_{d2}$ = $S_{d2}$ acrescido da derivação $F'(d_1)$;

e $T' = \langle F', S' \rangle$ é uma nova metáfora, onde

$S'$ = $S \cup \{ d_1 \}$
$F'$ = $F$ acrescido do mapeamento $\{ v_1 \dashrightarrow v_2 \}$ que conserva inversibilidade

Este tipo de operação pode ser extendido para cobrir os casos em se deseja "aumentar" o dominio alvo de um conjunto de constantes $A_2$, associado a um conjunto de sentenças $A_1$ pertencentes ao dominio fonte por uma bijeção $I$. Neste caso, a operação é chamada "positing structure" [1].

Do ponto de vista da reutilização de software, o importante é que se temos como dominio fonte um conjunto de especificações em fase de elaboração, como dominio alvo um conjunto de descrições (especificações) de componentes, e, se existe uma metáfora coerente e inversível T entre eles, então, através de $T^{-1}$ e da operação de aumento, é possível "fazer crescer" o dominio fonte, isto é, manter mecanismos de apoio à fase de especificação de software acoplados às ferramentas de reutilização.

A construção de um prototipo de ferramenta de reutilização baseada no método de transferencias semânticas descrito acima é irrealizável desde que a noção de consistencia - e consequentemente a de coerencia - não é computável. No entanto, uma versão computável do método de transferencia semantica, o método de trasferencias semânticas aproximadas, pode ser utilizado. Este método é descrito na próxima seção.

## 3. Transferencias Semânticas Aproximadas : Uma Teoria Computável de Metáforas

Além dos problemas de implementação existentes para desenhar uma ferramenta de reutilização baseada no método de transferencia semântica restringida, existe ainda um problema de computabilidade relacionado com o fato do CST fundamentar-se em um conceito não computável: o conceito de coerencia.

O conceito de coerencia, fundamental para esta teoria, é, por sua vez, baseado no conceito de consistencia que é não-computável quando aplicado a dominios abetos, isto é, que podem crescer e consequentemente ampliar suas restrições de estrutura como é o caso de reutilização em que estamos interessados. Portanto, torna-se necessário utilizar metodos aproximados para verificar a consistencia dos dominios.

O método de transferencias semânticas aproximadas [6] consiste do CST onde o conceito de consistencia é subistituido pelo de w-consistencia. Pode-se chegar a este conceito utilizando o principio da resolução-refutação[10].

Pelo teorema da resolução, se S é um conjunto de clausulas, S é "unsatisfiable" se e somente se a n-ésima resolução de S contém a clausula vazia,

$$\square \in \mathcal{R}^n(S)$$

para algum n maior ou igual a zero, e onde a n-ésima resolução de S é dada por[10]

$$\mathcal{R}^n(S) = \begin{cases} S & \text{se } n = 0 \\ \mathcal{R}\,(\mathcal{R}^{n-1}(S)), & \text{se } n \geq 1 \end{cases}$$

O conceito de w-coerencia é definido como,

Def] Um conjunto de clausulas S é dito w-inconsistente se a clausula vazia ocupa um nivel menor ou igual a w com relação a S. S é w-consitente se não é w-inconsistente.

onde uma clausula X ocupa o nivel n com relação a S se n é o menor numero natural tal que X pertence a n-ésima resolução de S.

Finalmente, o conceito de w-consistencia para um dominio é definido como,

Def] Um dominio é w-consistente, para um dado numero natural w, se o conjunto das suas restrições de estrutura é w-consistente.

Na seção seguinte analisaremos um exemplo aplicando o CST a dois modulos de software.

## 4. Um exemplo de reutilização usando transferencias semânticas

Nesta seção analisaremos um exemplo simples (mas suficientemente realista) de reutilização, cujo objetivo é mostrar a potencialidade do uso de transferencia semântica. Evidentemente existe um longo caminho à percorrer entre este exemplo e uma implementação prática de ferramentas de reutilização baseadas em metáforas. Algumas idéias e perspectivas de desenvolvimento de um protótipo de ferramenta como esta - um "processador de metáforas" - serão discutidas no final desta seção.

Tomaremos como exemplo um caso em que o domínio alvo é constituido da descrição de um componente de software que chamaremos "Help_Assistant". Trata-se de um conjunto de procedimentos que, quando chamados por um programa hospedeiro, emite mensagens, avisos e/ou menus e janelas que auxiliam o usuário a encontrar o comando pretendido.

Na versão simplificada que utilizaremos existem basicamente três tipos de procedimentos que o Help_Assistant pode acessar,

A (Procedimento de Erro):

> Um procedimento do tipo 'A' é requisitado quando um erro faral ocorre no programa hospedeiro. Neste caso, o Help_Assistant notifica o usuário da ocorrencia de um erro fatal, emite uma mensagem esclarecendo o tipo de erro ocorrido, e, devolve ao programa principal o "ponto de retorno", isto é, o ponto para onde o controle da execução deve ser transferido. Protanto, um procedimento do tipo 'A' pressupõe uma interrupção na sequencia de execução do programa hospedeiro.

B (Avisos):

> Um procedimento do tipo 'B' é chamado quando um erro não-fatal ocorre no programa hospedeiro ou quando se faz necessário passar uma mensagem ou aviso ao usuário durante a execução de uma sequencia de instruções do programa que utiliza os recursos do Help_Assistant, sem contudo interromper esta mesma sequencia.

C (Assistencia ao Usuário):

> Este tipo de procedimento é invocado quando o usuário precisa ou requer um processo interativo de modo a encontrar os comandos ou acessos pretendidos. O acesso a este tipo de procedimento pode ou não ser originado à partir da ocorrencia de um erro fatal.

O dominio fonte é constituido da descrição (ainda incompleta) de uma interface amigável para banco de dados. A interface consta de um menu principal que pode acessar janelas (forms) que por sua vez auxiliam o usuário a montar os queries para o BD. No caso de ocorrer um erro na composição de um query ou uma tentativa de acesso ilegal ao BD, a interface deverá emitir mensagens de erro interrompendo a sequecia de execução. Cada "form" é caracterizado por um "tipo" ('msg' ou 'form'), conforme a ação requerida: 'msg' se um erro de acesso ocorre, ou 'form' se se pretende colocar uma janela na tela capaz de aceitar um comando escolhido pelo usuário.

Mostraremos à seguir a descrição formal dos dominios fonte e alvo descritos acima.

Interface $D_1$ (Interface para banco de dados)

Vocabulário $V_1$ = { buffer, apontador, comando, parametros, entrada, saida, Interface, Form_File, igual, msg, form }

$f_1(X) = p_0$ X = Interface, Form_File, [illegible]
$f_1(X) = p_1$ X = buffer, comando, apontador
$f_1(X) = p_2$ X = saida, igual, parametros
$f_1(X) = p_3$ X = entrada

onde f é um mapeamento de V em { $o_1$, $o_2$, . . ., $p_1$, $p_2$, . . . } e $o_i$ é um operador de ordem i e $p_i$ um predicado de aridade i.

Restrições Estruturais

$E_1^{(1)}$ : Vx Vy [entrada(x, y, Interface) <=> parametros(x, y)]

$E_1^{(2)}$ : Vx [saida(x, Interface) => buffer(x)]

$E_1^{(3)}$ : Vx Vy [entrada(x, y, Interface/Form_File) <=> [entrada(x, y, Interface) Λ buffer(x)]]

$E_1^{(4)}$ : Vx [saida(x, Interface/Form_File) => [[entrada(y, z, Interface) Λ igual(y, msg)] => apontador(x)]]

$E_1^{(5)}$ : Vx [saida(x, Interface/Form_File) => [[entrada(y, z, Interface) Λ igual(y, form)] => [comando(w) Λ igual(x, w)]]]

Derivações $D_1^{(1)} = E_1^{(1)}$; $D_1^{(2)} = E_1^{(3)}$

onde cada derivação é uma relação estrutural que "define" uma dada constante do vocabulario <V, f>.

Dominio $D_2$ (Help Assistant)

Vocabulario $V_2$ = { *record, entrada, saida, msg, Help_Assistant, Busca, Gerenciador_de_Janelas, comando, igual, endereço, vazio, A, B, C* }

$f_2(X) = p_0$ X = Help_Assistant, Busca, Gerenciador_de_Janelas, A, B, C

$f_2(X) = p_1$ X = msg, comando, endereço, vazio

$f_2(X) = p_2$ X = saida, record, igual

$f_2(X) = p_3$ X = entrada

$E_2^{(1)}$ : $\forall x \forall y$ [entrada(x, y, Help_Assistant) <=> record(x, y)]

$E_2^{(2)}$ : $\forall x$ [saida(x, Help_Assistant) => msg(x)]

$E_2^{(3)}$ : $\forall x \forall y$ [entrada(x, y, Help_Assistant/Busca) <=> <=> entrada(x, y, Help_Assistant)]

$E_2^{(4)}$ : $\forall x$ [saida(x, Help_Assistant/Busca) => msg(x)]

$E_2^{(5)}$ : $\forall x \forall y$ [entrada(x, y, Help_Assistant/Gerenciador_de_Janelas) <=> [entrada( y, z, Help_Assistant)] $\wedge$ msg(x)]]

$E_2^{(6)}$ : $\forall x$ [saida(x, y, Help_Assistant/Gerenciador_de_Janelas) => [[entrada(y, z, Help_Assistant)] $\wedge$ igual(y, C)] => [comando(w) $\wedge$ igual(x, w)]]]

$E_2^{(7)}$ : $\forall x$ [saida(x, y, Help_Assistant/Gerenciador_de_Janelas) => [[entrada(y, z, Help_Assistant)] $\wedge$ igual(y, A)] => endereço(x)]]

$E_2^{(8)}$ : $\forall x$ [saida(x, y, Help_Assistant/Gerenciador_de_Janelas) => [[entrada(y, z, Help_Assistant)] $\wedge$ igual(y, B)] => vazio(x)]]

Derivações $D_2^{(1)} = E_2^{(1)}$; $D_2^{(2)} = E_2^{(3)}$; $D_2^{(3)} = E_2^{(5)}$

Nos dois dominios acima, a estrutura das componentes de software foi encontrada seguindo o fluxo de dados nas sub-componentes. Vale ressaltar entretanto que a abordagem para o problema de reutilização proposto neste trabalho não depende do método de desenvolvimento usado na

construção das componentes. A escolha do fluxo de dados (método de Jackson [9]) neste exemplo serve a dois propósitos :

i) é um dos recursos mais conhecidos por engenheiros de software, e é relativamente simples, isto é, não gera um numero muito extenso de restrições de estrutura;

ii) nos dá a oportunidade de mostrar um exemplo em que os dois vocabulários - o fonte e o alvo - têm constantes comuns sem no entanto se reduzirem ao mesmo vocabulário.

A importancia do item i) acima é óbvia. Quanto ao item ii) devemos registrar que, se $V_1 \cap V_2 = \emptyset$ só existe uma possibilidade de se encontrar uma metáfora $T = <F,S>$, qual seja, mapeando todos os termos de um conjunto de de sentenças $S \subset S_1$ nos seus correspondentes em $S_2$. Isto implica que encontraremos uma componente reutilizável somente se as restrições de estrutura destas forem um "matching" perfeito de $S_1$ pela metáfora T. Em outras palavras, trata-se de um processo "use-as-is".

No nosso exemplo, as constantes msg, entrada, saida, são comuns aos dois dominios. Podemos por exemplo, escolher a metáfora $T = <F', S>$, onde F' é o mapeamento,

{ (buffer -> msg), (Interface -> Help_Assistant), (Form_File -> Gerenciador_de_Janelas), (parametros -> record), (apontador -> endereço), (form -> C), (msg -> A) }

(Note que o termo $msg \in V_1$ é uma constante (aridade zero) enquanto o termo $msg \in V_2$ é um predicado de aridade 1 e portanto não são "termos comuns".)

Pode-se notar facilmente que F' é admissível e que a metáfora $T = <F',S>$, onde S é o conjunto de todas as restrições de estrutura do dominio $D_1$, leva a um conjunto de sentenças $S'_2$ do dominio alvo onde

$$S'_2 = \{ E_2^{(1)} , E_2^{(2)} , E_2^{(5)} , E_2^{(6)} , E_2^{(7)} \}$$

Este conjunto de sentenças define o fluxo de dados que entra em Help_Assistant e pasa pela componente Gerenciador_de_Janelas. Isto sugere que na elaboração da interface para banco de dados podemos reutilizar a componente Gerenciador_de_Janelas.

Na verdade o uso da notação "Help_Assistant"/Gerenciador_de_Janelas", apesar de bastante ilustrativa implica que a identificação da subrotina a ser reutilizada é imediata. Esta identificação é um dos problemas de implementação a que nos referimos. Entretanto, acreditamos que este problema pode ser resolvido com a escolha de uma arquitetura adequada para a base de conhecimento do sistema especialista que execute a transferencia semântica.

Um problema mais grave é o de eliminar varios dos mapeamentos inconsistentes como por exemplo

[ F' - {(msg -> A)} ] U { (msg -> msg) }

que é não-admissível, ou

F' = { (buffer -> msg), (Interface -> Gerenciador_de_Janelas), (msg -> C), (form -> A),
(Form_File -> Help_Assistant), (parametros -> record) }

que pode levar a conclusão de que não existe componente reutilizável. Escolher convenientemete os mapeamentos é um problema que acreditamos poder abordar usando heurísticas.

Finalmente, temos o problema de associar o processo de elaboração das restrições de estrutura ao processo de desenvolvimemto de software, isto é, como transformar especificações em relações formais.

Pretendemos abordar este aspecto acoplando o processo de construção do dominio alvo (o dominio contendo a descrição das componentes potencialmente reutilizáveis) com o ciclo de desenvolvimeto do software, mais especificamente, com a fase de especificação. Assim, supondo que estamos tratando com programas formalmente especificáveis, as restrições de estrutura que compõe o dominio alvo poderão ser adquiridas pelo sistema durante a fase de especificação.

# Referencias

[1] Bipin Indurkhya; "Constrained Semantic Transference: A formal Theory of Metaphors"; Synthese, 68, 1986.

[2] A. I. Wasserman and S. Gutz; "The Future of Programming", Comm. of ACM, March 1982.

[3] T. Biggerstaff and Charles Richter; "Reusability Framework, Assesment and Directions"; IEEE Software, March 1987.

[4] Scott Woodfield et al.; "Can Programmers Reuse Software?"; IEEE Software, July 1987.

[5] C. C. Chang and H. Jerome Keisler; "Model Theory"; North-Holland Publishing Co., 1978

[6] Bipin Indurkhya; "Aproximate Semantics Transference: A Computacional Theory of Metaphors and Analogies"; Cognitive Science, December 1987.

[7] Khun, T. S.; "Metaphor and Science" in Metaphorand Thought, Cambridge University Press, Cambredge, U. K., 1979

[8] Black, M.; "Models and Metaphors", Cornell University Press, Ithaca, 1962

[9] Jackson, M. A.; "Principles of Program Design", Academic Press, London, 1975

[10] Robinson, J. A.; "A Machine-Oriented Logic Based on the Resolution Principle", Journal of the ACM, vol. 12, 1, 1965

Seguem os parágrafos com falha de impressão na página 1:

O primeiro item poderia ser utilizado para caracterizar as diversas abordagens ao problema de reutilização uma vez que o êxito de todas elas depende da eficiencia com que é feita a busca por uma componente reutilizável.

Metodos como "building blocks", "subroutine libraries", e outros voltados para linguagens dirigidas para objetos, se propõe a achar uma peça de código reutilizavel, e têm obtido bons resultados para aplicações bem particulares.