PUC

ISSN 0103-9741

Monografias em Ciência da Computação
nº 16/95

Encapsulamento em Bancos de Dados e Engenharia de Software: Análise e Projeto de Refinamento

Paulo A. S. Veloso
Antonio L. Furtado

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22453-900
RIO DE JANEIRO - BRASIL

PUC RIO - DEPARTAMENTO DE INFORMÁTICA ISSN 0103-9741

Monografias em Ciência da Computação, Nº 16/95
Editor: Carlos J. P. Lucena Junho, 1995

Encapsulamento em Bancos de Dados e Engenharia de Software: Análise e Projeto de Refinamento *

Paulo A. S. Veloso
Antonio L. Furtado

* Trabalho patrocinado pelo Ministério de Ciência e Tecnologia da Presidência da República Federativa do Brasil.

In charge of publications:
Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC Rio — Departamento de Informática
Rua Marquês de São Vicente, 225 — Gávea
22453-900 — Rio de Janeiro, RJ
Brasil
Tel. +55-21-529 9386 Telex +55-21-31048 Fax +55-21-511 5645
E-mail: rosane@inf.puc-rio.br

ENCAPSULAMENTO EM BANCOS DE DADOS E EM ENGENHARIA DE SOFTWARE: ANÁLISE E PROJETO DE REFINAMENTO

Paulo A. S. VELOSO and Antonio L. FURTADO
e-mail {veloso, furtado}@inf.puc-rio.br

PUCRioInf MCC 16/95

Resumo. Analisam-se os aspectos fundamentais de proteção por encapsulamento, visando a esclarecer o papel dos formalismos envolvidos nesta disciplina e algumas conexões entre eles. Argumenta-se que esse processo consiste de uma tradução fiel seguida de um refinamento. Enfatiza-se o encapsulamento em projeto conceitual de aplicações de bancos de dados: passagem do nível de informação - com restrições de integridade declaradas - ao de manipulação - com essas garantidas por encapsulamento. Examinam-se métodos e heurísticas para análise e projeto, bem como um enfoque para o "frame problem" neste contexto.

Palavras chave: Encapsulamento, refinamento, projeto conceitual, especificações formais, tradução, interpretação, restrições de integridade, pré e pós condições, "frame problem".

Abstract. We analyze some fundamental aspects of protection by encapsulation. aiming at clarifying the roles of the formalisms involved in this discipline and some connections among them. We argue that this process amounts to a faithful translation followed by a refinement. We emphasize encapsulation in the conceptual design of data base applications, stressing the passage from the information level - with declared integrity constraints - to the manipulation level - where these constraints are guaranteed by encapsulation. We examine methods and heuristics for analysis and design, as well as an approach to the "frame problem" in this context.

Key words: Encapsulation, refinement, conceptual design, formal specifications, translation, interpretation, integrity constraints, pre and post conditions, "frame problem".

CONTEÚDO

1. INTRODUÇÃO

Este trabalho se propõe a analisar alguns aspectos fundamentais da idéia de proteção por encapsulamento, principalmente no contexto de bancos de dados. Argumenta-se que esse processo consiste de uma tradução fiel seguida de um refinamento. Examinam-se também aspectos metodológicos, relativos a análise e projeto, deste conceito.

Independência, proteção e integridade são objetivos importantes em Engenharia de Software e em Bancos de Dados. Encapsulamento é uma das ferramentas conceituais propostas para atingir estas metas. Proteção por encapsulamento baseia-se na idéia de ocultar (parte da) informação [Gehani & McGettrick '86; Parnas '79].

Tais idéias, por sua vez, conduzem naturalmente a tipos abstratos de dados e especificações formais. Um tipo abstrato de dados acha-se encapsulado por sua assinatura, a qual fornece certos caminhos de acesso a seus objetos, cuja representação interna permanece oculta e protegida [Guttag ,'77; Liskov & Zilles '77]. Para conectar tipos abstratos de dados usa-se uma tradução entre suas assinaturas [Turski & Maibaum '87; Veloso '87].

Projeto e análise de bancos de dados envolvem vários níveis, como informação, manipulação e representação (por exemplo, em um modelo de dados). Devido à diversidade de propósitos desses diversos níveis, enfoques distintos - desde declarativos a operacionais - são apropriados [Veloso et al. '84]. Como no caso de semântica de linguagens de programação, esses enfoques são complementares [Donahue '76; Hoare & Lauer '74], havendo porém necessidade de relacionar suas lógicas subjacentes.

No contexto de bancos de dados, encapsulamento aparece naturalmente ao se passar de um nível declarativo de informação a outro mais operacional de manipulação. No primeiro nível o estado fica geralmente oculto, enquanto no segundo é explicitamente manipulado por um repertório de operações [Casanova et al. '84; Furtado et al. '86]. Esta explicitação do estado corresponde a uma tradução fiel entre as lógicas subjacentes, a posterior disciplina de encapsulamento correspondendo a um refinamento.

Além dessas traduções - verticais - entre níveis, pode haver necessidade de outras - horizontais - entre formalismos complementares para um mesmo nível Tal é o caso, por exemplo, de formalismos axiomáticos e por pré e pós condições.

Há várias maneiras usuais de examinar propriedades de programas, estruturas ou bancos de dados. Uma delas é validação por testes, outra é por verificação e uma terceira tenta garantir as propriedades por construção. Encapsulamento, por suas ligações com especificações formais, parece bastante apropriado às duas últimas, se bem que não

exclua a primeira. Este trabalho enfatiza verificação e construção de encapsulamento.

Este trabalho pretende esclarecer o papel dos diversos formalismos envolvidos na disciplina de encapsulamento e das conexões entre eles. Examina-se mais de perto a passagem do nível de informação - com restrições de integridade declaradas - ao de manipulação - com essas garantidas por encapsulamento. Dá-se maior ênfase aos conceitos fundamentais envolvidos como base para métodos de análise e projeto, mas são examinados também una heurística para projeto de encapsulamento e um enfoque para o "frame problem" [Nilsson '73] neste contexto.

2. Exemplo: Apresentação Informal

Vamos usar um exemplo, propositadamente simples, para melhor ilustrar as idéias envolvidas. As considerações fundamentais, contudo, são gerais, conforme se verá, o mesmo valendo para diversos aspectos metodológicos de análise e projeto.

2.1 Aplicação: banco de dados acadêmico

Como um exemplo simples, consideremos um banco de dados acadêmico com informação sobre alunos e cursos durante período letivo. Esta informação diz respeito a alunos, cursos listados como oferecidos, e alunos inscritos em cursos.

Um curso pode ser oferecido, passando a ficar listado até que seja cancelado. Um aluno pode se matricular em um curso, quando passa a estar inscrito nele até que o abandone. Essa movimentação não deve violar os regulamentos da universidade, que exigem que: alunos só possam estar inscritos em cursos listados, e um aluno ativo deve permanecer ativo (inscrito em pelo menos um curso) durante o período).

Para se referir a essas entidades, alunos e cursos, alguns atributos são previstos. Digamos, cada aluno será identificado por um (único) número de registro e talvez seu nome; um curso fica caracterizado por seu título e código, este último sendo composto de sigla do departamento e número de disciplina. O efeito disso é descrever as entidades em termos de certos tipos básicos. Assim, pode-se imaginar que alunos e cursos serão representados sobre tipos primitivos de uma vez por todas. Um aluno ficará representado por um registro, como (regstr:94213967,nome:Pedro_Silva) e um curso por (ttl:Calculo,cod:(dept:Mat,num:1362)). Neste contexto, é natural que estes atributos fiquem estacionários durante a evolução do banco de dados.

Podemos encarar um banco de dados tanto a nível de informação, enfatizando seu conteúdo e a consultas, quanto a nível de manipulação, dando ênfase a como a informação será alterada [Casanova et al. '84; Furtado et al. '86; Veloso et al. '84].

2.2 Nível de Informação

A nível de informação o conteúdo do banco de dados indica os cursos listados e que alunos estão inscritos em que cursos. Do ponto de vista do modelo relacional, esta informação seria mantida em tabelas.

Os regulamentos da universidade impõem algumas restrições sobre a possível evolução dessa informação. Assim, o comportamento dinâmico do banco de dados, a nível de informação fica caracterizado pelas restrições de integridade impostas, no caso

estática: um aluno só pode estar inscrito em cursos listados;

dinâmica: cada aluno ativo deve permanecer ativo.

Notemos que a restrição estática se refere a cada estado do banco de dados, enquanto que a dinâmica envolve estados atuais e futuros.

Neste nível o banco de dados pode ser visto como consistindo de um conjunto Alc, de estados alcançáveis, e de um conjunto Trns, de transições.

2.3 Nível de Manipulação

Para manipular a informação armazenada lançamos mão de um repertório de funções - de atualização e de inicialização - que transformam os estados.

Em nosso caso, podemos considerar operações construtoras de oferecer curso, que passa a ser listado, e de matricular aluno em curso, ficando aquele inscrito neste. Podemos também ter operações como de cancelar curso, que deixa de ser listado, e de aluno abandonar curso, quando deixa de estar inscrito.

Neste nível, o banco de dados é visto através das consultas e das transformações do repertório provido.

2.4 Encapsulamento

Encapsulamento esconde certos detalhes de uma aplicação, manipulando-a apenas por operações de um repertório fornecido. Assim, ambos os conjuntos Trns, de transições, e Alc, de estados alcançáveis, ficam sob controle do repertório. O objetivo almejado é proteção: as restrições de integridade estão garantidas.

Este objetivo impõe algumas restrições sobre o comportamento das operações. Por exemplo, não se deveria matricular um aluno em um curso caso este não esteja já listado como oferecido. Assim, o efeito pretendido de matricular - inscrever aluno em curso - passaria a ser condicionado a que o curso esteja listado.

Deseja-se que, especificado certo comportamento apropriado para as operações de manipulação, as restrições de integridade fiquem automaticamente satisfeitas. Uma maneira de garantir isto é por uma verificação.

2.5 Proteção por Encapsulamento

Uma vez que a aplicação é manipulada apenas por operações do repertório fornecido e que esta disciplina de manipulação garante integridade, espera-se que a aplicação fique protegida. Assim, cada realização encapsulada deve fornecer uma realização aceitável para a aplicação.

Quando se examina mais de perto a situação, nota-se que esta envolve linguagens diferentes. Este processo de proteção por encapsulamento envolve uma troca de linguagens, de pouco impacto, e um refinamento por restrição de comportamento. Para esclarecer isto, vamos descrever melhor as linguagens envolvidas.

3. LINGUAGENS

Podemos apresentar uma aplicação de maneira mais estruturada através de assinaturas (poli-sortidas) envolvendo sortes, predicados e operações [Enderton '72; Ehrig & Mahr '85].

3.1 Linguagens de Informação

Uma linguagem de informação consiste de sortes (simbolizando domínios) e de predicados (simbolizando consultas).

Em nosso exemplo podemos reconhecer dois sortes:
Aln (representando alunos) e Crs (representando cursos).

Temos também os seguintes predicados :
predicado unário lstd, sobre o sorte Crs;
predicado binário: inscrt, sobre os sortes Aln e Crs.

Este elenco de símbolos, correspondendo ao esquema do banco de dados, constitui a assinatura de consulta. Acrescentando-se variáveis percorrendo os sortes, obtemos a linguagem de consulta. Podemos então expressar algumas propriedades por meio de sentenças desta linguagem. Por exemplo, a explicação de aluno ativo pode ser expressa sob a forma: atv(x), com x:Aln, abrevia $(\exists y{:}Crs)$ inscrt(x,y).

De maneira geral, uma assinatura Cnslt estática de consulta fica caracterizada por seus conjuntos de símbolos: Srt de sortes e Prd de predicados de consulta, cada um com seu perfil de argumentos $p\ (s_1,\ldots,s_m)$. Uma estrutura *I* para esta assinatura Cnslt fornece realizações para seus símbolos: domínios não vazios para seus sortes e relações para seus predicados [Enderton '72; Veloso '87].

Estado: implícito ou explícito

Na linguagem anterior o estado fica oculto; para expressar restrições dinâmicas lança-se mão de modalidades, como $\Box$ representando "sempre" (em qualquer estado futuro). Agora, podemos expressar, por exemplo, que sempre há um curso listado por $\Box(\exists y{:}Crs)$ lstd(y), uma sentença da extensão modal Cnslt$\Box$ [Casanova et al. '84].

Outra alternativa, mais expressiva, para raciocinar sobre atualizações internaliza o estado introduzindo na assinatura um sorte explícito Σ para

os estados. Esta extensão, dita explícita, será útil para expressar propriedades do encapsulamento. A propriedade acima seria agora expressa por $(\forall \eta{:}\Sigma.)\ (\exists y{:}Crs)\ lstd^*(y,\eta)$.

A razão para termos usado lstd*, ao invés de lstd, se deve a que este é um predicado unário sobre o sorte Crs, enquanto que aquele é um predicado binário, com argumentos Crs e estado. Veremos mais tarde que esta alteração, embora útil, é praticamente inócua: quase "confeito sintático".

Consultas implícitas e explícitas

A assinatura explícita tem um novo sorte explícito Σ para o estado, tendo cada consulta estática agora um novo argumento de sorte estado. Mais precisamente, a assinatura Cnslt* de consulta explícita consiste de:

Sortes: Aln e Crs.

Consultas explícitas como predicados envolvendo estado
predicado unário: lstd* (Crs,Σ) {invocado por lstd*(c:Crs,σ:Σ)};
predicado binário: inscrt* (Aln,Crs,Σ) {invocado por inscrt*(a:Aln,c:Crs,σ:Σ)}.

Note que cada consulta estática $p\ (s_1,\ldots,s_m)$ tem sua versão explícita $p^*\ (s_1,\ldots,s_m,\Sigma)$; assim, a assinatura explícita de consulta engloba a implícita.

3.2 Linguagens de Manipulação

Nossas linguagens ainda não têm símbolos para manipulação do banco de dados. Um repertório de manipulação provê um repertório de operações (simbolizando atualizações e inicializações).

Em nosso exemplo podemos reconhecer as seguintes operações
operações unárias: ofrcr(c), cnclr(c) {oferecer, cancelar curso},
operações binárias: mtrclr(a,c), abndnr(a,c) {matricular, abandona};
constante (operação nulária): inic {inicializa o banco de dados}.

Um repertório Rprt de manipulação sobre um conjunto dado de sortes fica caracterizado por seu repertório de operações: Inc de inicializações e Atlz, cada uma com seu perfil de argumentos o $(s_1,\ldots,s_n)$.

Repertório implícito e explícito

Esta é uma versão com estado implícito. Agora, podemos expressar algumas propriedades das atualizações em uma extensão modal apropriada. Em particular seus efeitos sobre predicados, por exemplo [offr(c)>lstd(c), numa notação similar à da lógica dinâmica [Harel '79].

A versão com estado explícito internaliza a idéia acima de transformações introduzindo um sorte Σ. explícito para estados. Agora podemos dar o destino das operações do repertório: o sorte estado Σ.

Podemos expressar propriedades de atualizações. Por exemplo, podemos descrever os efeitos pretendidos da atualização ofrcr sob a forma: $lstd^*(c,ofrcr^*(c,\sigma))$ expressando que curso c está listado no estado

atingido do estado σ oferecendo c. Podemos também expressar alguns efeitos de mtrclr por lstd*(c,σ)$\rightarrow$inscrt*(s,c,mtrclr*(s,c,σ)).

Mais precisamente, o repertório Rprt* explícito consiste de:

Sortes: Aln e Crs.

Inicialização como operação (nulária) fornecendo estado

inic* ()$\rightarrow\Sigma$ {invocada por inic*}

Atualizações como operações envolvendo estado e fornecendo estado

binárias: ofrcr*, cnclr*, ambas com perfil (Crs,Σ)$\rightarrow\Sigma$,

ternárias: mtrclr*, abndnr*, ambas com perfil (Aln,Crs,Σ)$\rightarrow\Sigma$.

Este repertório explícito incorpora o repertório Rprt explicitando o estado: cada inicialização i (s_1,... ,s_n) tem sua versão explícita i* (s_1,... ,s_n)$\rightarrow\Sigma$ e cada atualização u (s_1,... ,s_n) aparecendo como u* (s_1,...,s_n,Σ)$\rightarrow\Sigma$.

Algumas propriedades do repertório podem ser expressas nesta linguagem, por exemplo, ofrcr*(y,ofrcr*(y',η))$\approx$ofrcr*(y',ofrcr*(y,η)).

Estruturas encapsuladas

Uma estrutura R^* para o repertório Rprt* explícito fornece realizações para seus símbolos: domínios não vazios para seus sortes e funções para suas operações. Assim, dados valores $\underline{a}$ em R^*[Srt], cada inicialização i* (s_1,... ,s_n)$\rightarrow\Sigma$ e cada atualização u* (s_1,... ,s_n,Σ)$\rightarrow\Sigma$ gera uma invocação R^*[i*($\underline{a}$)] ou R^*[u*($\underline{a}$)]:$R^*[\Sigma]\rightarrow R^*[\Sigma]$, ambas com resultado no conjunto $R^*[\Sigma]$ de estados. Diremos que uma tal estrutura R^* está *encapsulada* quando seu conjunto $R^*[\Sigma]$ é finitamente gerado [Ehrig & Mahr '85] por suas invocações: cada estado $\sigma\in R^*[\Sigma]$ é o valor R^*[s] de alguma seqüência de invocações, i. e. de um traço [Veloso & Furtado '84].

Assim, uma estrutura R^* encapsulada realiza a idéia de encapsulamento: a aplicação é manipulada apenas através do repertório de operações provido. Este processo de encapsulamento pode ser encarado como impondo restrições aos conjuntos de estados e de transições. O conjunto Atlz caracteriza o conjunto Pss de um passo de transições, daí obtendo-se Trns como seu fecho transitivo Pss$^+$. Por sua vez, o conjunto Inc de inicializações caracteriza um conjunto Prtd de estados iniciais, sendo Alc sua imagem sob Trns.

Manipulação explícita

Uma assinatura de manipulação consiste de uma assinatura de consulta e de um repertório, sua versão explícita Mnpl* consistindo de Cnslt* e Rprt* além de um predicado binário trns (Σ,Σ) sobre o sorte estado.

Nesta linguagem explícita de manipulação, podemos expressar os efeitos pretendidos, como por exemplo ($\forall$y:Crs)($\forall\xi$:Σ) lstd*(y,ofrcr*(y,ξ)), bem como outras propriedades, como oferecimento de curso não altera inscrições. Cada consulta com estado implícito tem sua contrapartida com estado explícito: a $\varphi(v_1,\ldots,v_m)$ correspondendo $\varphi^*(v_1,\ldots,v_m,\xi)$ com $\xi:\Sigma$.

4. ESPECIFICAÇÕES

Uma especificação dá propriedades de seus símbolos com as quais se pode contar, uma vez que valem em suas realizações Gehani & McGettrick '86; Turski & Maibaum '87; Veloso '87]. Assim como temos diversos gêneros de linguagens, também podemos ter especificações de naturezas distintas: quanto ao nível e quanto à forma.

4.1 Especificação a nível de Informação

Uma especificação do comportamento a nível de informação consiste das restrições de integridade que devem ser obedecidas [Casanova et al. '84; Furtado et al. '86; Veloso et al. '84].

Especificação de informação na sintaxe implícita

Na linguagem com estado implícito, expressamos as restrições por meio de modalidades. Em nosso exemplo, podemos expressá-las pelas sentenças

Est: $(\forall y{:}Crs)\ [(\exists x{:}Std)\ inscrt\ (x,y)\rightarrow lstd(y)]$;

Dnm: $(\forall x{:}Std)\ [atv(x)\rightarrow \Box atv(x)]$

onde atv(x), com x:Aln, abrevia $(\exists y{:}Crs)\ inscrt(x,y)$.

A restrição estática deve ser satisfeita em qualquer estado, devendo ser portanto entendida como $\Box$Est. Os estados satisfazendo as restrições estáticas são ditos (estaticamente) válidos, formando um conjunto Val, e os satisfazendo as restrições dinâmicas são chamados de consistentes em relação a transições, formando um conjunto Tr_Cns.

Especificação de informação na sintaxe explícita

Na linguagem com estado explícito, as restrições acima se expressam naturalmente através de suas contrapartidas explícitas:

Est*(ξ): $(\forall y{:}Crs)\ [(\exists x{:}Std)\ inscrt^*(x,y,\eta)\rightarrow lstd^*(y,\xi)]$;

Dnm*(ξ): $(\forall x{:}Std)\ [atv^*(x,\xi)\rightarrow(\forall \zeta{>}\xi)atv^*(x,\zeta)]$,

onde atv*(x,η), com x:Aln e $\eta{:}\Sigma$, abrevia $(\exists y{:}Crs)\ inscrt^*(x,y,\eta)$ e $(\forall \zeta{>}\xi)atv^*(x,\zeta)$ abrevia $(\forall \zeta{:}\Sigma)\ [trns(\xi,\zeta)\rightarrow atv^*(x,\zeta)]$.

As restrições ficam então

Est*: $(\forall \xi{:}\Sigma)$ Est*(ξ), i. e.

$(\forall \eta{:}\Sigma)(\forall y{:}Crs)\ [(\exists x{:}Std)\ inscrt^*(x,y,\eta)\rightarrow lstd^*(y,\eta)]$;

Dnm*: $(\forall \xi{:}\Sigma)$ Dnm*(ξ), i. e.

$(\forall \eta{:}\Sigma)(\forall x{:}Std)\ \{atv^*(x,\xi)\rightarrow(\forall \zeta{:}\Sigma)[trns(\xi,\zeta)\rightarrow atv^*(x,\zeta)]\}$.

4.2 Especificação do Repertório

A nível de manipulação devemos descrever o comportamento das operações do repertório, principalmente como afetam as possíveis consultas. Isto sugere descrevê-las por meio de pré e pós condições [Gehani & McGettrick '86; Furtado et al. '86]. Vamos esboçar o aspecto de uma tal descrição para nosso exemplo de encapsulamento.

O efeito pretendido de mtrclr é que o aluno passe a estar inscrito no curso, desde que o curso esteja listado. Isto pode ser descrito por

op mtrclr*(a:Aln,c:Crs,$\sigma{:}\Sigma$)$\rightarrow\Sigma$

pré lstd*(c,σ)
pós inscrt*(a,c,σ')

4.3 Verificação da Proteção

A disciplina de encapsulamento pretende proteger a aplicação manipulando-a apenas pelas operações de um repertório fornecido, com o fito de garantir as restrições de integridade.

Uma verificação demonstra que este objetivo é realmente atingido. Com base na especificação Mnpl* do comportamento das operações do repertório, estabelecem-se as restrições estáticas e dinâmicas [Casanova et al. '84; Furtado et al. '86; Veloso et al. '84].

Para estabelecer as restrições estáticas, em geral verifica-se que
cada inicialização fornece um estado válido: $i^*(\underline{a}) \in Val$,
em nosso caso $Est^*(inic^*)$;
cada atualização preserva validade: se $\sigma \in Val$ então $u^*(\underline{a},\sigma) \in Val$,
por exemplo $[Est^*(\eta) \rightarrow (\forall x{:}Std)(\forall y{:}Crs)\ Est^*(mtrclr^*(x,y,\eta))]$.

As restrições dinâmicas podem ser estabelecidas por preservação de consistência sob atualizações: $\sigma \in Tr\_Cns$ então $u^*(\underline{a},\sigma) \in Tr\_Cns$,
por exemplo $(\forall x{:}Std)(\forall y{:}Crs)\ [Dnm^*(\eta) \rightarrow Dnm(mtrclr^*(x,y,\eta))]$.

Verificadas as restrições de integridade com base em Mnpl*, cada realização da especificação Mnpl*, que tem seu reduto a Rprt* encapsulado, satisfaz a essas restrições.

4.4 Do nível de Manipulação para o de Informação

Estabelecidas as restrições de integridade a partir da descrição de comportamento das operações do repertório, sabemos que uma estrutura encapsulada conforme este comportamento satisfaz às restrições. Deve, portanto, fornecer uma realização aceitável para a aplicação a nível de informação.

Porém, como mencionado, temos linguagens diferentes: as restrições são descritas a nível de informação usando modalidades, ficando o estado implícito; enquanto que o comportamento do repertório é descrito com o estado explícito. O encapsulamento consiste de duas etapas:
a explicitação do estado (uma tradução fiel),
o refinamento dos conjuntos de transições e de estados.

A fim de esclarecer a situação, vamos examinar mais de perto as semânticas das linguagens envolvidas.

5. Semântica das Linguagens de Consultas

A linguagem estática de consultas serve de base para duas extensões: uma modal, com estado implícito, e outra com estado explícito. Aquela se refere ao estado, oculto, indiretamente através de modalidades, enquanto que esta tem variáveis percorrendo o sorte estado. Assim é natural que tenham semânticas distintas [Casanova et al. '84].

5.1 Semântica da Linguagem de Consultas implícitas

Na linguagem de consultas implícitas faz-se referência ao estado, oculto, através de modalidades. Assim, temos uma extensão modal da linguagem estática de consultas por uma modalidade [] representando "sempre". A linguagem estática de consultas tem sua semântica por satisfação clássica, sendo a da extensão modal dada por meio de estados potenciais à la Kripke.

Mais precisamente, a assinatura Cnslt determina a classe de suas possíveis realizações, que dá a semântica da linguagem de consultas. Usamos $I \rhd \varphi(v_1,\ldots,v_m)\ [a_1,\ldots,a_m]$ para "I satisfaz $\varphi(v_1,\ldots,v_m)$ sob atribuição $a_1,\ldots,a_m$ para as variáveis $v_1,\ldots,v_m$", conforme a definição clássica de Tarski [Enderton '72].

Um conjunto de estados potenciais para o banco de dados corresponde a uma possível realização. Assim, cada sorte tem uma mesma realização estacionária. Diremos que um conjunto Pot de realizações para Cnslt é *estacionário* quando para I e J em Pot, I[Crs]=J[Crs], denotado por Pot[Crs], por exemplo. (Para cada símbolo c de constante, exigimos também designação rígida: para I e J em Pot, I[c]=J[c], denotado por Pot[c].)

Uma estrutura de transições para a extensão modal Cnslt[] é uma relação binária Trns sobre um tal conjunto estacionário Pot. Define-se satisfação [Rescher & Urquhart '71; Casanova et al. '84]

para uma fórmula estática (sem modalidades) $\varepsilon(\underline{v})$:
$(I,\mathrm{Trns}) \rhd [] \varepsilon(\underline{v})\ [\underline{a}]$ sse $I \rhd \varepsilon(\underline{v})\ [\underline{a}]$, i. e. I satisfaz $\varepsilon(\underline{v})$ sob atribuição $[\underline{a}]$,

para uma fórmula $[]\varphi(\underline{v})$: $(I,\mathrm{Trns}) \rhd [] []\varphi(\underline{v})\ [\underline{a}]$ sse para cada $J \in$ Pot com $\langle I,J\rangle \in$ Trns $(J,\mathrm{Trns}) \rhd [] \varphi(\underline{v})\ [\underline{a}]$.

Uma estrutura de transições Trns sobre Pot e um subconjunto não vazio Alc de Pot determinam uma estrutura Str=<Alc,Trns> de alcançabilidade. Para uma sentença θ, Str $\rhd [] \theta$ sse $(I,\mathrm{Trns}) \rhd [] \theta$ para cada $I \in$ Alc.

Dada uma estrutura Str=<Alc,Trns> de alcançabilidade, o conjunto de estados (estaticamente) válidos em relação às restrições estáticas Est é Str[Est]:={$I \in$ Alc/$I \rhd$ Est}, e o conjunto de estados consistentes em relação às restrições dinâmicas Dnm é Str[Dnm]:={$I \in$ Alc/$(I,\mathrm{Trns}) \rhd [] \mathrm{Dnm}$}.

5.2 Semântica da Linguagem de Consultas explícitas

Uma estrutura C para a assinatura Cnslt* fornece realizações para seus símbolos, inclusive um domínio não vazio para o sorte estado. Sua semântica é clássica: usamos $C \rhd^* \theta(\underline{v},\underline{\xi})\ [\underline{a},\underline{\sigma}]$ para "C satisfaz $\theta(\underline{v},\underline{\xi})$ sob atribuição $\underline{a}$ para as variáveis estáticas $\underline{v}$ e $\underline{\sigma}$ para as variáveis de estado $\underline{\xi}$".

Agora, o conjunto de estados válidos em relação à restrição estática Est* é C[Est*]:={$\sigma \in C[\Sigma]/C \rhd^* \mathrm{Est}^*(\xi)\ [\sigma]$ }, sendo o conjunto de estados

consistentes em relação à restrição dinâmicas Dnm* dado por $C[\text{Dnm}^*]:=\{\sigma\in C[\Sigma]/C \rhd^* \text{Dnm}^*(\xi)\ [\sigma]\ \}$.

6. Conexão entre as Linguagens de Consultas

As linguagens de consulta com estado implícito Cnslt[] e explícito Cnslt* têm semânticas distintas: a primeira através de estados potenciais à la Kripke e a segunda clássica à la Tarski. Apesar disso, há uma estreita conexão entre suas estruturas. Esta conexão vai permitir uma tradução fiel de fórmulas.

6.1 Indução de estruturas

Vamos examinar a indução reversa de estruturas: de estruturas para a assinatura Cnslt* em um conjunto de estruturas estacionárias para Cnslt.

Uma estrutura M para a assinatura Cnslt* induz um conjunto M_Σ de estruturas estacionárias para Cnslt como se segue.

Cada estado $\sigma\in M[\Sigma]$ induz uma σ-fatia:
uma estrutura M_σ em Pot, onde, por exemplo, para sorte Crs
$M_\sigma[\text{Crs}]:=M[\text{Crs}]$ e para predicado lstd $M_\sigma[\text{lstd}]:=\{c\in M[\text{Crs}]/<c,\sigma>\in M[\text{lstd}^*]\}$.
Tomamos M_Σ como consistindo das estruturas M_σ com $\sigma\in M[\Sigma]$.

Reciprocamente, um conjunto Pot de estruturas estacionárias para Cnslt induz uma estrutura Pot^Σ para a assinatura Cnslt*.

Tomamos $\text{Pot}^\Sigma[\Sigma]:=\text{Pot}$ e para os sortes estáticos, por exemplo,
$\text{Pot}^\Sigma[\text{Crs}]:=\text{Pot}[\text{Crs}]$ (o domínio estacionário de sorte Crs); para uma consulta, por exemplo, lstd* tomamos
$\text{Pot}^\Sigma[\text{lstd}^*]:=\{<c,I>\in \text{Pot}[\text{Crs}]\text{x}\text{Pot}/c\in I\text{lstd}]\}$.

Note-se que essas duas construções são inversas uma da outra: $(M_\Sigma)^\Sigma=M$ e $(\text{Pot}^\Sigma)_\Sigma=\text{Pot}$. Assim, temos uma bijeção entre conjuntos Pot de estruturas estacionárias para Cnslt e estruturas M para a assinatura Cnslt*.

Além disso, estas construções estabelecem uma correspondência entre:
relações binárias sobre $M[\Sigma]$ e relações binárias sobre M_Σ,
$T\subseteq M[\Sigma]\text{x}M[\Sigma]$ induzindo $T_\Sigma:=\{<M_\sigma,M_\tau>\in M_\Sigma\text{x}M_\Sigma/<\sigma,\tau>\in T\}$ sobre M_Σ;
subconjuntos de $M[\Sigma]$ e subconjuntos de M_Σ,
$A\subseteq M[\Sigma]$ induzindo $A_\Sigma:=\{M_\sigma\in M_\Sigma/\sigma\in A\}$ contido em M_Σ.

Assim, conjuntos de transições com estado implícito ou explícito se correspondem, e analogamente para conjuntos de estados.

6.2 Internalização e tradução

Podemos traduzir fórmulas da extensão modal em fórmulas da linguagem de consultas com estado explícito desde que tenhamos símbolos de predicado que internalizem conjuntos de transições.

Internalização e tradução de estrutura de transições

Para internalizar conjuntos de transições consideremos uma extensão Cnslt^t da assinatura Cnslt* obtida acrescentando-se um predicado binário trns (Σ,Σ) sobre o sorte estado.

Uma estrutura T para a assinatura Cnslt^t consiste de uma estrutura T_* para Cnslt*, seu reduto, junto com uma relação binária $T[\text{trns}] \subseteq T_*[\Sigma] \times T_*[\Sigma]$. Como tal, ela induz uma estrutura de transições $T_t := < T_*[\Sigma]_\Sigma, T[\text{trns}]_\Sigma >$ para Cnslt$^{\Box}$.

Traduzindo variáveis de maneira natural obtemos uma tradução da linguagem modal para a linguagem com estado explícito, que traduz uma fórmula $\varphi(\underline{v})$ em $\varphi^t(\underline{v},\xi)$ onde $\xi:\Sigma$, como se segue

para fórmula estática $\varepsilon(\underline{v})$: $\varepsilon^t(\underline{v},\xi)$ é $\varepsilon^*(\underline{v},\xi)$;

para uma fórmula $\psi(\underline{v})$ da forma $\Box\theta(\underline{v})$: $\psi^t(\underline{v},\xi)$ é
$(\forall\zeta:\Sigma)\ [\text{trns}(\xi,\zeta) \rightarrow \theta^t(\underline{v},\zeta)]$,
onde $\zeta:\Sigma$ é uma variável não ocorrendo em $\theta^t(\underline{v},\xi)$.

Uma conseqüência desses processos é o seguinte **lema de tradução:**

dada uma atribuição $\underline{a}$ para as variáveis estáticas $\underline{v}$ e $\sigma \in T[\Sigma]$
$(T_\sigma, T[\text{trns}]_\Sigma) \rhd \Box\theta(\underline{v})\ [\underline{a}]$ sse $T \rhd^* \theta^t(\underline{v},\xi)\ [\underline{a},\sigma]$.

Internalização e tradução de estrutura de acessibilidade

Tomemos uma extensão Cnslt^a da assinatura Cnslt^t obtida acrescentando-se um predicado unário alc (Σ) sobre o sorte estado, a fim de internalizar conjunto de estados.

Uma estrutura A para assinatura Cnslt^a consiste de uma estrutura A_t para Cnslt^t, seu reduto, junto com um subconjunto $A[\text{alc}] \subseteq A_t[\Sigma]$. Uma tal estrutura será dita *própria* quando $A[\text{alc}] \neq \varnothing$. Assim temos uma bijeção entre estruturas próprias A para Cnslt^a e estruturas de alcançabilidade para Cnslt$^{\Box}$: A induzindo $A_a := < A[\text{alc}]_\Sigma, A_t[\text{trns}]_\Sigma >$ para Cnslt$^{\Box}$.

Com tradução natural para variáveis obtemos uma nova tradução da linguagem modal para a linguagem com estado explícito, traduzindo uma fórmula $\varphi(\underline{v})$ em $\varphi^a(\underline{v},\xi)$, onde $\xi:\Sigma$, com $\varphi^a(\underline{v},\xi)$ sendo $[\text{alc}(\xi) \rightarrow \varphi^t(\underline{v},\xi)]$; usaremos também seu fecho universal sobre o estado φ^Σ: $(\forall\xi:\Sigma)\ \varphi^a(\underline{v},\xi)$

Como conseqüência temos um **lema de tradução:**

para $\sigma \in A[\Sigma]$: $\sigma \in A[\text{alc}]$ sse $A_\sigma \in A[\text{alc}]_\Sigma$, e

para uma sentença θ: $A_\sigma \in A_a[\theta]$ sse $A \rhd^* \theta^a(\xi)\ [\sigma]$ e
$A_a \rhd \Box\theta$ sse $A \rhd^* \theta^\Sigma$ i. e. $A \rhd^* (\forall\xi:\Sigma)\ [\text{alc}(\xi) \rightarrow \theta^t(\xi)]$.

6.3 Tradução fiel entre sintaxes implícitas e explícitas

Agora, vamos rever o efeito combinado dessas construções. Temos uma tradução fiel preservando definibilidade. Portanto, podemos substituir a linguagem de Cnslt$^{\Box}$ pela de Cnslt^a sem perda de poder dedutivo ou expressivo.

De fato, temos uma bijeção natural entre estruturas próprias A para Cnslt^a e estruturas de alcançabilidade para Cnslt$^{\Box}$: A para Cnslt^a induzindo $A_a := < A[\text{alc}]_\Sigma, A_t[\text{trns}]_\Sigma >$ para Cnslt$^{\Box}$, bem como uma tradução de fórmulas da linguagem de Cnslt$^{\Box}$ em fórmulas da linguagem de Cnslt^a. Assim esta correspondência preserva comportamentos definíveis e conseqüências de propriedades.

Tradução fiel: dadas sentenças θ e χ da linguagem de Cnslt$^{\Box}$,
$\chi \rhd^{\Box} \theta$ sse $\chi^{\Sigma} \rhd^{*} \theta^{\Sigma}$.

Por outro lado, dada uma estrutura Str para Cnslt$^{\Box}$, uma sentença θ da linguagem de Cnslt$^{\Box}$ define um conjunto Str[θ] de estados potenciais. Temos uma estrutura A para Cnslt^{a} correspondente a Str e uma fórmula $alc(\xi) \wedge \theta^{t}(\xi)$ da linguagem de Cnslt^{a} que define a imagem deste conjunto.

Definibilidade: em estruturas Str e A correspondentes,
$A_{\sigma} \in$ Str[θ] sse $\sigma \in A[alc(\xi) \wedge \theta^{t}(\xi)]$.

A situação é similar à conhecida redução da lógica poli-sortida à versão usual sem sortes mas com predicados de relativização [Enderton '72]. (Aqui, não precisamos tanto de predicados de relativização devido ao caráter estacionário das estruturas.) A relação entre a lógica subjacente à extensão modal Cnslt$^{\Box}$ com estado implícito e a subjacente à linguagem de consultas com estado explícito é a de uma tradução fiel entre lógicas: esta tem poder expressivo englobando o daquela mas a tradução preserva conseqüências.

A finalidade dos predicados alc e trns é permitir expressar propriedades dos estados alcançáveis e das transições. O predicado trns já está na assinatura Mnpl* de manipulação explícita, enquanto que alc vem do encapsulamento. Este encapsulamento é a parte de refinamento propriamente dito.

7. SISTEMA DE MANIPULAÇÃO

A descrição por pré e pós condições de uma operação fornece parte da informação sobre seu comportamento dinâmico.

Consideremos, por exemplo, o caso de mtrclr*, descrito por pré-condição $lstd^{*}(c,\sigma)$ e pós-condição $inscrt^{*}(a,c,\tau)$ onde $\tau = mtrclr^{*}(a,c,\sigma)$. Quando a pré-condição não vale no estado σ, por inércia a atualização não deve alterar o estado: se $\neg lstd^{*}(c,\sigma)$ então $mtrclr^{*}(a,c,\sigma)=\sigma$. Quando, ao contrário, vale pré-condição no estado σ então vale a pós-condição fornecida no novo estado: se $lstd^{*}(c,\sigma)$ então $inscrt^{*}(a,c,\tau)$.

7.1 Problema do "frame"

O que não costuma ser explicitamente dito, mas é geralmente suposto, é que as pós-condições enumeradas são as únicas afetadas pela execução da operação. Para raciocinar sobre efeitos das operações é preciso informação tanto sobre a parte não afetada quanto sobre a afetada.

Porém, a descrição explícita de tudo o que não é afetado, além de grande, tende a obscurecer a descrição das operações. Trata-se do chamado "frame problem" [Nilsson '73].

Em nosso caso temos uma maneira satisfatória de contornar o problema, conseguindo o melhor de ambos os enfoques. Baseia-se na idéia [Veloso & Furtado '86] de deixar a cargo de um módulo geral subjacente a parte comum da descrição, como inércia.

7.2 Especificação estruturada de Manipulação

A atualização mtrclr*(a,c,σ) usa o predicado lstd*(c,σ) para testar a pré-condição e afeta o predicado inscrt*(a,c,τ). Isto sugere descrever uma tal atualização sob a forma seguinte

op mtrclr*(x:E,y:t,η:Σ)→Σ
 usa lstd*(y,η)
 afeta inscrt*(x,y,η');
  pré lstd*(y,η)
  pós inscrt*(x,y,η')

Note-se que a parte afeta dessa descrição fornece um conjunto finito Aft[mtrclr*(x,y,η)]={inscrt*(x,y,η')} de fórmulas da linguagem. As regras subjacentes geram a partir dessa descrição estruturada o seguinte conjunto Axm[mtrclr*] de axiomas (com quantificação universal implícita):

¬lstd*(y,ξ)→mtrclr*(x,y,ξ)≈ξ {inércia }
[lstd*(y,ξ)∧mtrclr*(x,y,ξ)≈ζ}→inscrt*(x,y,ζ) {efeito pretendido}
[mtrclr*(x,y,ξ)≈ζ∧¬(x'≈x∧y'≈y)]→[inscrt*(x',y',ξ)↔inscrt*(x',y',ζ)] {frame}
mtrclr*(x,y,ξ)≈ζ→[lstd*(y',ξ)↔lstd*(y',ζ)] {frame}

7.3 Especificação explícita de Manipulação

A especificação explícita de manipulação é obtida da descrição por pré e pós condições por meio das regras subjacentes de geração de axiomas.

Ela contém, por exemplo, o seguinte conjunto Axm[cnclr*] de axiomas (com quantificação universal implícita) para cnclr*:

(∃x:Std) inscrt*(x,y,η)
(∃x:Std)inscrt*(x,y,ξ)→cnclr*(y,ξ)≈ξ {inércia }
[¬(∃x:Std)inscrt*(x,y,ξ)∧cnclr*(y,ξ)≈ζ]→¬lstd*(y,ξ) {efeito pretendido}
[cnclr*(y,ξ)≈ζ∧¬y'≈y]→[lstd*(y',ξ)↔lstd*(y',ζ)] {frame}
cnclr*(y,ξ)≈ζ→[inscrt*(x',y',ξ)↔inscrt*(x',y',ζ)] {frame}

Com essa explicitação das propriedades supostas temos uma especificação $Mnpl^G$ do comportamento do repertório por meio de sentenças da linguagem de manipulação com estado explícito.

7.4 Verificação das Restrições de Integridade

A partir da especificação explícita $Mnpl^G$ de manipulação pode-se proceder à verificação das restrições de integridade, de acordo com o método usual já esboçado.

Estabelecemos as restrições estáticas, verificando, por exemplo, que
(∀y:Crs)[¬lstd*(y,inic*)∧(∀x:Std)¬inscrt*(x,y,inic*)] ⊳* Est*(inic*);
Axm[mtrclr*] ⊳* (∀ξ:Σ) [Est*(ξ)→(∀x:Std)(∀y:Crs) Est*(mtrclr*(x,y,ξ))].

Para estabelecer as restrições dinâmicas, verifica-se, por exemplo, que
$Mnpl^G$ ⊳* (∀ξ:Σ)(∀x:Std)(∀y:Crs) [Dnm*(ξ)→Dnm*(mtrclr*(x,y,ξ))].

Assim temos assegurado que todo estado alcançável é válido (Alc⊆Val) e que toda transição em Pss^+ é consistente (Pss^+⊆Tr_Cns).

8. Outros Aspectos de Encapsulamento

Vimos como verificar a satisfação das restrições de integridade, estáticas ou dinâmicas a partir da especificação de manipulação. Agora, vamos examinar outros aspectos relativos a essa disciplina: indução, inernalização e projeto.

8.1 Encapsulamento e indução

Para tentar estabelecer outras propriedades do banco de dados encapsulado, por exemplo para ver que todo estado válido é alcançável ($Val \subseteq Alc$), usualmente se recorre a argumentos indutivos. Tais argumentos podem ser feitos na meta-linguagem [Casanova et al. '84], mas alguns deles poderiam ser internalizados. Isto seria conveniente para se tirar real proveito das especificações. Pois, assim poderemos nos valer de suporte de máquina, na forma de demonstradores semi-automáticos de teoremas, por exemplo [Chang & Lee '73].

Lembremos que o processo de encapsulamento consiste do refinamento de Trns às invocações de atualizações e de Alc como os estados denotados por traços. Esse processo é normalmente expresso na meta-linguagem, por exemplo $Trns=Pss^{+}$.

A idéia da disciplina de encapsulamento pode também ser expressa pelo seguinte trecho de programa não determinístico com guardas [Gries '81]

begin {inicialização}
  [] invoque $inicialização_1$,
  ...
  [] invoque $inicialização_m$
end {inicialização} ;
do {atualização}
  [] invoque $atualização_1$,
  ...
  [] invoque $atualização_n$
od {atualização}

8.2 Encapsulamento internalizado

O processo de encapsulamento pode ser encarado como impondo sucessivas restrições aos conjuntos de estados e de transições. Como tal, ele consiste da explicitação dos seguintes itens:
repertório de invocações de atualizações,
passo de transição,
transição;
repertório de invocações de inicializações,
estados iniciais,
estados alcançáveis.
Para raciocinar sobre o comportamento do repertório é conveniente captar esse processo de restrições por meio de um conjunto Encpsl de axiomas na linguagem de $Cnslt^{a}$.

Explicitação do repertório de invocações de atualizações

Para cada atualização $u^*(s_1,\ldots,s_n,\Sigma)\rightarrow\Sigma$ de Rprt* temos um axioma definindo sua execução. Por exemplo:

$ofrcmnt(\xi,\zeta)\leftrightarrow(\exists y{:}Crs)\ ofrcr^*(y,\xi)\approx\zeta$

$mtrcl(\xi,\zeta)(\xi,\zeta)\leftrightarrow(\exists x{:}Aln)(\exists y{:}Crs)\ mtrclr^*(x,y,\xi)\approx\zeta.$

Explicitação do passo de transição

Um passo de transição fica caracterizado pela escolha de uma atualização de Rprt*. Em nosso caso:

$pss(\xi,\zeta)\leftrightarrow ofrcmnt(\xi,\zeta)\vee mtrcl(\xi,\zeta)\vee cnclmnt(\xi,\zeta)\vee abndn(\xi,\zeta)$

Explicitação das transições

O conjunto de transições Trns=Pss$^+$ é o menor contendo Pss que é transitivo. Para captar "o menor" usamos o esquema de indução ι_ψ para fórmulas $\psi(\alpha,\omega)$ com $\alpha,\omega{:}\Sigma$.

λ: $(\forall\xi,\zeta{:}\Sigma)\ [pss(\xi,\zeta)\rightarrow trns(\xi,\zeta)]$ {Pss⊆Trns}

τ: $(\forall\xi,\eta,\zeta{:}\Sigma)\ [trns(\xi,\eta)\wedge trns(\eta,\zeta)\rightarrow trns(\xi,\zeta)]$ {Trns transitivo}

ι_ψ: $\{(\forall\xi,\zeta{:}\Sigma)[pss(\xi,\zeta)\rightarrow\psi(\xi,\zeta)]\wedge(\forall\xi,\eta,\zeta{:}\Sigma)[\psi(\xi,\eta)\wedge\psi(\eta,\zeta)\rightarrow\psi(\xi,\zeta)]\}\rightarrow$

$\rightarrow(\forall\alpha,\omega{:}\Sigma)\ [trns(\alpha,\omega)\rightarrow\psi(\alpha,\omega)]$

Explicitação do repertório de invocações de inicializações

Para cada inicialização $i^*(s_1,\ldots,s_n)\rightarrow\Sigma$ de Rprt* temos um axioma $é\_i(\zeta)\leftrightarrow(\exists\underline{x}{:}Srt)\ i^*(\underline{x},)\approx\zeta$ definindo sua execução. Em nosso caso

$é\_inic(\zeta)(\zeta)\leftrightarrow inic^*\approx\zeta.$

Explicitação dos estados iniciais

Um estado inicial fica caracterizado pela escolha de uma inicialização de Rprt*. Em nosso caso $prtd(\zeta)\leftrightarrow é\_inic(\zeta)$.

Explicitação dos estados alcançáveis

O conjunto Alc de estados alcançáveis é a imagem de Prtd sob Trns.

$alc(\zeta)\leftrightarrow\{prtd(\zeta)\vee(\exists\xi{:}\Sigma)[alc(\zeta)\wedge trns(\xi,\zeta)]\}$

Entre as conseqüências de temos um princípio de indução para alc:

α_θ: $(\forall\zeta{:}\Sigma)[prtd(\zeta)\rightarrow\theta(\zeta)]\wedge(\forall\xi,\zeta{:}\Sigma)[\theta(\xi)\wedge pss(\xi,\zeta)\rightarrow\theta(\zeta)]\}\rightarrow$

$\rightarrow(\forall\eta{:}\Sigma)\ [alc(\eta)\rightarrow\theta(\eta)].$

Note-se que esses axiomas são satisfeitos em uma estrutura encapsulada expandida por conjunto Alc e relação Trns conforme pretendido.

8.3 Projeto de Encapsulamento

O método usual para estabelecer restrições de integridade sob encapsulamento consiste em mostrar que certos conjuntos de estados potenciais são fechados sob operações. Por exemplo, para garantir as restrições estáticas mostra-se que Val é fechado sob as operações do repertório, e para as dinâmicas que Tr_Cns é fechado sob as atualizações.

Esse método sugere uma heurística para obter pré e pós condições para as operações [Veloso '91]. O exame dos efeitos pretendidos de cada operação e das restrições sugere refinamentos das descrições de molde

a se poder garantir as restrições. Os efeitos pretendidos dos construtores, como ofrcr* e mtrclr*, são sucessivamente refinados para se ter Alc$\subseteq$Val e Val$\subseteq$Alc. Este refinamento prossegue com a consideração de Trns$\subseteq$Tr_Cns e das outras operações.

Por exemplo, partimos dos efeitos pretendidos

para inic*: $(\forall y{:}Crs)[\neg lstd^*(y,inic^*)\wedge(\forall x{:}Std)\neg inscrt^*(x,y,inic^*)]$,

para ofrcr*(y,ξ): lstd*(y,ofrcr*(y,ξ)) e

para mtrclr*(x,y,ξ): inscrt*(x,y,mtrclr*(x,y,ξ)).

A consideração do que seria necessário para se ter Alc$\subseteq$Val sugere um primeiro refinamento

para $ofrcr^*(y,\xi)\approx\zeta$: $lstd^*(y',\xi)\rightarrow lstd^*(y',\zeta)$ e $inscrt^*(x',y',\zeta)\rightarrow inscrt^*(x',y',\xi)$

para mtrclr*(x,y,ξ): pré-condição lstd*(y,ξ) e

pós-condição

$mtrclr^*(x,y,\xi)\approx\zeta\rightarrow\{inscrt^*(x',y',\zeta)\leftrightarrow[inscrt^*(x',y',\xi)\vee(x'\approx x\wedge y'\approx y)]\}$

Agora, examinando o que seria necessário para se ter Val$\subseteq$Alc chega-se a uma descrição desses três construtores. Então, a consideração de Trns$\subseteq$Tr_Cns sugere sucessivos refinamentos para as outras operações [Veloso '91; Veloso & Furtado '86; Casanova et al. '84; Furtado et al. '86].

9. CONCLUSÃO

Este trabalho analisou alguns aspectos fundamentais da idéia de proteção por encapsulamento, principalmente no contexto de bancos de dados. Argumentou-se que esse processo consiste de uma tradução fiel seguida de um refinamento. Examinaram-se também aspectos metodológicos deste conceito.

Esta análise se concentrou em traduções verticais, entre níveis, e horizontais, entre formalismos complementares de um mesmo nível. Outros aspectos enfatizados foram verificação e corretude por construção.

Uma das finalidades deste trabalho é esclarecer o papel dos diversos formalismos envolvidos na disciplina de encapsulamento. Estes são em geral complementares, havendo tradução fiel entre eles, embora alguns, mais explícitos, tenham maior poder expressivo.

Encapsulamento envolve uma tradução fiel entre as lógicas subjacentes seguida de um refinamento por restrição de comportamento. Aqui examinou-se mais de perto a passagem do nível de informação - com restrições de integridade declaradas - ao de manipulação - com essas garantidas por encapsulamento.

Foram enfatizados os conceitos fundamentais envolvidos. Assim provê-se não só base, mas também métodos de análise e projeto, conforme ilustrado. Exemplos são a heurística de projeto de encapsulamento e o enfoque sugerido para o "frame problem", assuntos que são objeto de pesquisa em andamento.

REFERÊNCIAS

Broy, M.; Pepper, P. - Program development as a formal activity. IEEE Trans. Software Engin., **SE-7** (1)14-22, 1981.

Casanova, M. A.; Veloso, P. A. S.; Furtado, A. L. - Database specification formalisms: an eclectic perspective. Proc. 3rd ACM Symposium on Principles of Database Systems (110-118). Waterloo, Canadá, 1984.

Chang, C. L.; Lee, R. C. T. - Symbolic Logic and Mechanical Theorem Proving. Academic Press, New York, 1973.

Donahue, J. E. - Complementary Definitions of Programming Language Semantics. Springer-Verlag, Berlin, 1976.

Enderton, H. B. - A Mathematical Introduction to Logic. Academic Press; New York, 1972.

Ehrig, H.; Mahr, B. - Fundamentals of Algebraic Specifications 1: Equations and Initial Semantics. Springer-Verlag, Berlin, 1985.

Furtado, A. L. ; Casanova, M. A. ; Veloso, P. A. S. - Application-oriented approaches. In Furtado, A. L. ; Neuhold, E. J. (eds.) Formal Techniques for Data Base Design (p. 5- 44). Springer-Verlag, Berlin, 1986.

Furtado, A. L. ; Maibaum, T. S. E. ; Veloso, P. A. S. - Especificações abstratas (de bancos de dados) via níveis de traço. Rev. Bras. de Computação **1** (3) 179-193, 1981.

Gehani, N. and McGettrick, A. D. - Software Specifications Techniques. Addison-Wesley, Reading, 1986.

Guttag, J. V - Abstract data types and the development of data structures. Comm. Assoc. Comput. Mach., **20** (6), 396-404, 1977.

Harel, D. - First-order Dynamic Logic. Springer-Verlag, Berlin, 1979.

Hoare, C. A. R.; Lauer, P. E.- Consistent and complementary formal theories of the semantics of programming languages. Acta Informatica **3** 135-153, 1974.

Liskov, B.; Zilles, S. - An introduction to formal specifications of data abstractions. In Yeh, R. T. (ed.). Curent Trends in Programming Methodology, vol. I (1-32). Prentice-Hall, 1977.

Lucena, C. J. P. de; Martins, R. C. B.; Veloso, P. A. S. - Construção de programas por transformadores de dados: introdução e estudo de caso. Rev. Bras. de Computação **3** (1) 5-17, 1983/1984.

Nilsson, N. J. - Problem-solving Methods in Artificial Intelligence. McGraw-Hill, New York, 1973.

Parnas, D. L. - Designing software for ease of extension and contraction. IEEE Trans. Software Engin., **5** (2), 128-138, 1979.

Rescher, N.; Urquhart, A. - Temporal Logic. Springer-Verlag, Berlin, 1971.

Turski, W. M.; Maibaum, T. S. E. - The Specification of Computer Programs. Addison-Wesley, Wokingham, 1987.

Veloso, P. A. S. - Verificação e Estruturação de Programas com Tipos de Dados. Edgard Blücher, São Paulo, SP;1987.

Veloso, P. A. S. - Enforcing integrity constraints by design of repertoire of updates. Imperial College, Dept. of Computing, Res. Rept. DoC 91/24, 1991.

Veloso, P. A. S.; Casanova, M. A.; Furtado, A. L.- Formal data base specification: an eclectic perspective. PUC - RJ, Dept. Informática, MCC 1/84, 1984.

Veloso, P. A. S.; Furtado, A. L. -Stepwise construction of algebraic specifications. In Gallaire, H.; Minker, J.; Nicholas, J. M. (eds.). Advances in Data Base Theory, vol. 2 (321-352). Plenum, New York, 1984.

Veloso, P. A. S. ; Furtado, A. L. - Towards simpler and yet complete formal specifications. In Langefors, B.; Verrijn-Stuart, A. A.; Bracchi, G. (eds.) Trends in Information Systems (257-271). North-Holland, Amsterdam, 1986.