ANTENOR FERREIRA FILHO

O SISTEMA DE EFEITOS COLATERAIS EM THM

Dissertação apresentada ao Curso de MESTRADO EM SISTEMAS E COMPUTAÇÃO da Universidade Federal da Paraíba, em comprimento às exigências para obtenção do Grau de Mestre.

ULRICH SCHIEL

Orientador

F368s Ferreira Filho, Antenor
O sistema de efeitos colaterais em THM / Antenor Ferreira Filho. - Campina Grande, 1987.
82 p.

Dissertacao (Mestrado em Sistemas e Computacao) - Universidade Federal da Paraiba, Centro de Ciencias e Tecnologia

1. Banco de Dados 2. Linguagem de Maquina 3. Programacao - Modelo Temporal - Hierarquico 4. Dissertacao I. Schiel, Ulrich, Dr. II. Universidade Federal da Paraiba - Campina Grande (PB) III. Título

CDU 004.652.2(043)

O SISTEMA DE EFEITOS COLATERAIS EM THM

ANTENOR FERREIRA FILHO

DISSERTAÇÃO APROVADA EM 23.10.87

COMISSÃO EXAMINADORA

ULRICH SCHIEL - Ph.D

- Presidente -

SUELY MENDES DOS SANTOS - Ph.D

- Examinador -

MARCUS COSTA SAMPAIO - M.Sc

- Examinador -

CAMPINA GRANDE
OUTUBRO - 1987

Para Debora, Renata e Tally

SUMÁRIO

AGRADECIMENTOS

Aos professores

Ulrich Schiel, pela confiança em mim depositada, seu contagiante otimismo e conhecimento acadêmico, que me fizeram superar todas as dificuldades ao longo deste trabalho.

Joseluce F. Cunha, co-orientadora informal.

Aos funcionários da Universidade do Amazonas

Alberto Nogueira de Castro Jr.,
João Bosco Leão Carneiro,
José Luiz N. de Mello,
Mariney Silva Sampaio,
Osmar da Costa Macêdo,
Welington Chevreuil, cujo apoio técnico e material possibilitou o término deste trabalho.

Especiais

Antenor Ferreira,
Raimunda Vasconcelos Ferreira, pelo apoio.

O SISTEMA DE EFEITOS COLATERAIS EM THM

RESUMO

Mostramos neste trabalho como se pode evitar que as operações que manipulam um Banco de Dados (BD), tornam sua extensão (os dados) inconsistentes com a sua intenção (o esquema conceitual). Consideramos o esquema conceitual descrito pelo Modelo Temporal-Hierárquico (THM) e o sistema aqui proposto é acoplado as operações primitivas que manipulam o BD, gerando chamadas a operações especiais que testam a validade da operação e, se necessário, geram operações de efeitos colaterais.

CAPÍTULO I

INTRODUÇÃO

1. Generalidades.

Um BD é muito sensível a atualizações, pois estas podem tornar o BD inconsistente com o esquema que o descreve. Para evitar esta inconsistência, duas medidas tem que ser tomadas, para cada operação que altera o BD:

a) Verificar quais alterações que tal operação pode acarretar e como recompor a consistência com o esquema conceitual;

b) Recusar a operação caso ela esteja em conflito com alguma restrição do esquema conceitual.

Neste trabalho é desenvolvido um sistema automático de manipulação da integridade semântica de um esquema conceitual, descrito com o (THM) [Sc1,CS1,CS2]. As medidas a) e b) acima são formalizadas como axiomas de efeitos colaterais e axiomas dinâmicos, respectivamente [Sc2]. Utilizamos os formalismos da lógica dinâmica e da lógica temporal. Estes axiomas são acoplados as operações primitivas da Linguagem de Manipulação de Dados do modelo (THM/DML), transformando cada uma destas operações em chamadas a sub-operações em THM/DML que realizam os efeitos colaterais e axiomas dinâmi-

cos.

2. O Modelo Temporal-Hierárquico - THM

Apresentaremos aqui uma rápida introdução ao THM. Muitos dos conceitos e definições presentes serão melhor detalhados no capítulo II (Uma Descrição Abstrata ao Modelo THM).

O THM é um modelo de dados que permite três abstrações: generalização, agregação e agrupamento. Considera aspectos temporais, parâmetros com tempo e uma relação especial entre classes chamada "pré-pós"; é um modelo estático e dinâmico completo, possuindo uma linguagem própria, a Linguagem do Esquema Conceitual (CSL), proveniente de "Conceptual Schema Language" e a modelagem de eventos e disparadores.

Há um mapeamento do Universo de Discurso (porção do mundo real de interesse para a aplicação) para o nível conceitual, mapeando objetos para entidades, tipos de objetos para classes, propriedades e associações para relacionamentos, processos para operações e ocorrências para eventos e disparadores.

A imagem completa do Universo de Discurso é o esquema conceitual e a base de informações.

2.1 A Linguagem THM/CSL

Usada para descrever o esquema conceitual. É composta pela Linguagem de Descrição de Dados (DDL) e pela Linguagem de Manipulação de Dados (DML), provenientes de "Data Description Language" e "Data Manipulation Language", respectivamente [Sc1].

2.1.1 THM/DDL

A THM/DDL é utilizada para descrever a parte estrutural do esquema conceitual, incluindo a definição de classes com as suas relações, parâmetros, papéis, agregações e agrupamentos. Esta descrição feita pela THM/DDL é mapeada para uma série de tabelas chamada ECD (Esquema Conceitual de Dados).

2.1.2 THM/DML

Utilizada para definir as operações que manipulam os dados do esquema definido. Estas operações escritas em THM/DML são mapeadas para um conjunto de tabelas, chamado Esquema Conceitual de Operações (ECO). Como vamos utilizar a THM/DML, apresentaremos resumidamente a sua sintaxe.

Sintaxe THM/DML

operation <nome-operação> '('<ipl>;<opl>')'

pre-conditions

<predicado>

body

<comando>

post-conditions

<predicado>

{otherwise (warning/cancel/error)}

O predicado nas pré-condições e nas pós-condições é um conjunto de cláusulas, sendo que cada cláusula é um predicado simples (ps) ou uma disjunção de ps. A sintaxe de ps é obtido como a sequir:

- um termo é:

i) entidades constantes ou variáveis;

ii) se CrD Λ c ε C então r(C) é um termo;

iii) se r é um relacionamento entre classes, então r( ) é um termo;

iv) se t é um termo numérico então qualquer expressão aritmética envolvendo t é um termo.

- se t_1 Λ t_2 são termos um ps é obtido:

i) se C é uma classe "$t_1 \varepsilon$ C" é um ps;

ii) se # é um dos símbolos =, >, >=, < v <=, então t_1 # t_2 é um ps;

iii) se p é um ps $\sim$ (p) é um ps;

iv) se CrD, c ε C $\wedge$ d ε D, então r(c,d) é um ps;

v) se um ps p envolve somente uma classe e esta classe tem parâmetros de tempo e t é uma entidade temporal, então "p at t" é um ps.

Os comandos no procedimento, aqui chamado "body", são uma sequência de comandos simples (operações primitivas ou operações previamente definidas) e controles de comandos do tipo:

for each (<entidade>|class<nome-classe>)
such that <condição> do <comando>

for each (<entidade> in (<grupo>|<nome-classe>) do
<comandos>

Semântica

"ipl" é a lista dos parâmetros de entrada, que são passadas pelo usuário e "opl" a de saída, são entidades criadas no BD pela execução da operação.

Se a operação é chamada para execução, os predicados de pré-condições são testados e se são verdadeiros, os comandos do procedimento são executados. Se uma pré-condição falha, o programador pode dar um dos três parâmetros da cláusula "otherwise".

"warning" - a operação é executada, mas uma mensagem adicional é enviada.

"cancel" - a operação é cancelada.

"error" - todo o processo envolvendo esta operação é cancelado, inclusive a própria operação.

O parâmetro "default" é "cancel".

Se a operação após ser executada não satisfizer os predicados da pós-condições, então serão tomados procedimentos idênticos aos da pré-condições.

2.2 - Exemplo:

Para ilustrar o uso das linguagens THM/DDL è THM/DML, apresentaremos um exemplo utilizado em [Sc1] (Fig. 1.1).

THM/DDL

```
class CARRO
  member relationships
    tem-placa      : NUMERO-PLACA
    é-do-modelo    : MODELO
    produzido-em   : ANO-FABRICAÇAO
    produzido-por  : FABRICANTE
```

```
    tem-numero     : NUMERO-CHASSI
  keys are produzido-em, tem-numero, tem-placa
  with roles situação-de-propriedade gives subclasses
    CARRO-DO-FABRICANTE, CARRO-NO-REVENDEDOR, CARRO-EM-USO,
    CARRO-DESTRUIDO explicit

class FABRICANTE
  class relationships
    consumo-max : CONSUMO
    qty         : QUANTIDADE
  member relationships
    tem-nome      : NOME-FABRICANTE(1,1)
    produz        : CARRO (1,*)
    proprietário  : CARRO-DO-FABRICANTE (1,*)
    fabrica       : MODELO (1,*)
  keys are tem-nome
  aggregated in FABRICAÇÃO-ANUAL

class CARRO-EM-USO
  class relationships
    pre-class  : CARRO-NO-REVENDEDOR exclusive
    post-class : CARRO-DESTRUIDO, CARRO-NO-REVENDEDOR  ex-
    clusive
  member relationships
    é-possuido     : PESSOAS (0,*) with old values
    é-grupo-proprietário : GRUPO-PESSOAS (0,1)
  in role situação-de-propriedade
```

```
    parameters with time and lifetime : 3 anos

  class PESSOAS
    member relationship
      possui : CARRO-EM-USO (0,*) with old values
    type set of string

  class GRUPO-PESSOAS
    member-relationships
      é-do-grupo-proprietário : CARRO-EM-USO (1,*)
    type integer
    grouping of PESSOAS explicit

        THM/DML

operation INICIO-DATABASE( )
  body
    establish FABRICANTE qty 0

operation PERMISSÃO-PARA-OPERAR (m-name)
  pre-conditions
    not((m-name) in FABRICANTE
    qty( ) < 5
  body
    insert (m name) into FABRICANTE
    establish FABRICANTE qty qty( ) + 1
  post-conditions
```

```
    card(proprietario(m-name)) > 0
    otherwise warning ('fabricante' m 'não possui carros')

operation DEFINE-MODELO(mod,fc,m)
  pre-conditions
    not(mod in MODELO)
    fc in CONSUMO
  body
    insert mod into MODELO
    establish mod tem-cons-comb fc
    establish mod tem-construtor m

operation REGISTRA-CARRO(rn,m,mod,sn,py)
  pre-conditions
    not(rn in NUMERO-PLACA)
    m in FABRICANTE
    mod in MODELO
    clock.mes=janeiro v py=clock.ano
    not(clock.mes=janeiro) v (py=clock.ano v py=clock.ano-1)
  body
    let c be (rn)
    insert c into CARRO-DO-FABRICANTE at clock.data
    insert rn into NUMERO-PLACA
    insert sn into NUMERO-CHASSI
    establish c tem-numero sn
    establish c produzido-por m
    establish c é-do-modelo mod
```

establish c proprietario m

establish c produzido-em py

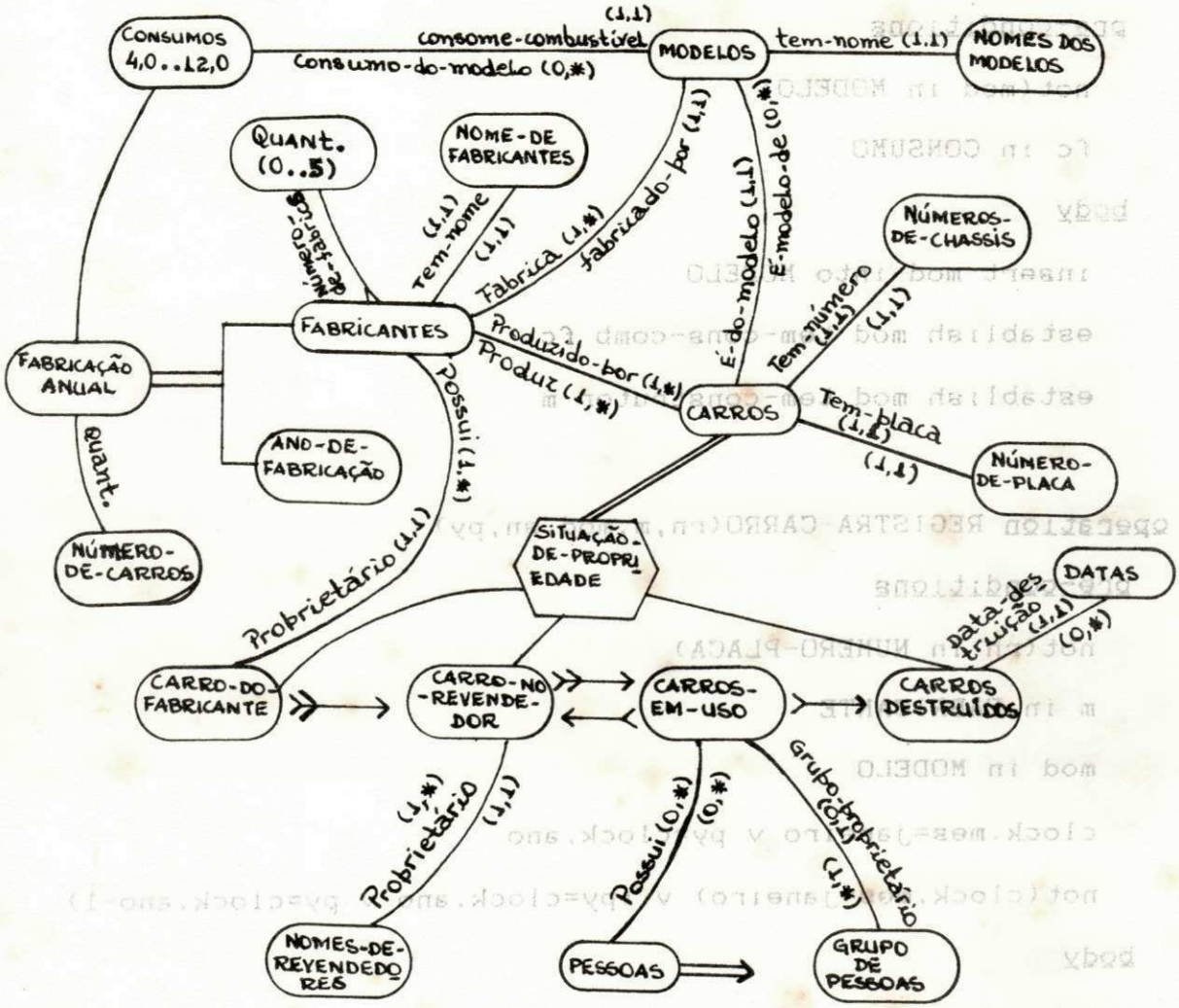

Fig. 1.1

CAPíTULO II

UMA DESCRIÇÃO ABSTRATA DO MODELO THM

Os conceitos e definições deste capítulo tem como referência [Sc2].

1. Conceitos Estáticos

Os conceitos estáticos do modelo de dados são especificados por uma série de axiomas, chamados estáticos, que devem ser obedecidos em todos os estados do BD. Para garantir que um BD permaneça satisfazendo esses axiomas, restrições são feitas as operações primitivas por meio de axiomas, chamados dinâmicos, e as alterações provocadas pelas operações primitivas são controladas pelos axiomas de efeitos colaterais.

O único primitivo básico é a entidade, que pode ser interpretada como a representação de um objeto do mundo real para a base de informação no nível conceitual. As entidades neste nível são idéias abstratas e são identificadas pelas suas propriedades ou relacionamentos com outras entidades.

Uma classe é um par C = <Ic,Mc>, onde Ic é a identificação de C composto de seu nome,Mc, e eventualmente outras informações que servem para identificar a classe. Mc é o

conjunto de entidades chamados membros de C.

Dada duas classes C e D, uma relação r de C para D é um sistema r = <Nr,Rr,MINr,MAXr>, onde Nr é o nome de r ,e Rr ⊂ Mc x Md, MINr é um inteiro positivo e MAXr é um inteiro positivo ou um símbolo especial denotado por "*" (maior valor possível), e são denominados de cardinalidade minimal e maximal da relação r, respectivamente, e indicam que cada membro da classe C está relacionado com no mínimo MINr e no máximo MAXr elementos de D.

Se CrD é uma relação e c é um membro de C, nós definimos r(c) como o conjunto de elementos de D relacionados com C, isto é: r(c) = {d ε D/<c,d> ε Rr}, e r(c,d) é interpretado como um predicado que é verdadeiro se e somente se <c,d> ε Rr.

Para dar uma caracterização da generalização e especialização, nós definimos papel como um predicado disjuntivo, $p(e) = p_1(e) v \ldots v p_n(e)$, que pode ser aplicado a entidades e da classe genérica G e tal que p é verdadeiro se e somente se e é um membro da subclasse C .

2. Hierarquias

Em alguns casos as organizações hierárquicas de entidades podem ser consideradas no sentido de níveis de abstrações [CS1]. Alguns detalhes das entidades de 'nível mais baixo' são suprimidos e elas são usadas para formarem enti-

dades de 'níveis mais altos'. Desta forma as classes podem ser generalizadas, agregadas ou agrupadas.

2.1 Generalização

A idéia de 'papel' do THM aplicado a uma classe, gera subclasses . Por exemplo, o papel sexo aplicado a classe PESSOA gera as subclasses MULHER e HOMEM. O inverso da generalização é a especialização (Fig. 2.1).

2.2 Agregação

Uma classe pode ser formada por entidades de diferentes classes, possivelmente relacionadas entre si. Por exemplo, a classe CASAL pode ser formada por entidades das classes HOMEM e MULHER. O inverso da generalização é a desagregação (Fig. 2.2).

2.3 Agrupamento

Podemos agrupar os membros de uma classe e considerar cada grupo como um membro de uma classe de 'nível mais alto'. Por exemplo, podemos agrupar PROFESSOR em DEPARTAMENTO e DEPARTAMENTO em INSTITUTO. O inverso do agrupamento é dis-

solução (Fig. 2.3).

Fig. 2.1

Fig. 2.2

Fig. 2.3

Matematicamente esses conceitos correspondem a subconjuntos de um conjunto, produto cartesiano e conjunto-potência.

Para poder testar a correspondência entre classes e entidades no relacionamento hierárquico, alguns predicados especiais são usados:

para classes

is_a(A,B) - A é subclasse de B

is_part(A,B) - A é componente de uma classe agregada B

is_elem(E,G) - os membros de G são conjuntos de membros de E

para entidades

part(p,c) - p é um componente da entidade agregada c

elem(e,g) - a entidade e é elemento de um grupo g

3. Conceito de Tempo

Em nosso contexto, o tempo é considerado como uma classe T de tuplas $t = (t_1 : u_1, \ldots, t_\eta : u_\eta)$ tal que:

i) há um conjunto de constantes $p_2, \ldots, p_\eta$, chamados períodos;

ii) $t_1, \ldots, t_\eta$ são inteiros positivos, tais que $t_i < p_i$;

iii) u_i são strings, chamados unidades;

iv) para i = 1,2,...,n p_{i+1}: $u_{i+1} = 1 : u_i$.

A menor unidade u_η é chamada granularidade de uma unidade de tempo. Isto significa que uma unidade de tempo não é algo infinitamente pequeno, mas sim, um intervalo.

Há uma unidade de tempo especial que reflete o momento presente,que nós chamamos clock.

Outro conceito interessante do THM é a possibilidade de fazer algumas considerações acerca do passado e do futuro de uma classe. Nós podemos enunciar a relação pré-pós entre classes, denotada como C >-- -> D, significando que entidades deletadas de C podem ser inseridas em D, e entidades inseridas em D podem ser originárias de C. C é uma pré-classe de D e D é uma pós-classe de C. Se nós trocarmos pode por deve, nós temos uma classe pré e pós exclusiva, denotadas como C >>--->> D.

4. Axiomas Estáticos

A1 : Nc = Nd => C = D (classes distintas, têm nomes distintos)

A2 : (CrD $\wedge$ CsE $\wedge$ Nr = Ns) => r = s (uma classe não pode ter duas relações com o mesmo nome)

A3 : MAXr <> * => MINr <= MAXr

A4 : $c \in Mc \Rightarrow (\#(\langle c,d\rangle/\langle c,d\rangle \in Rr) \geq MINr \wedge (MAXr \neq * \Rightarrow \#(\langle c,d\rangle/\langle c,d\rangle \in Rr) \leq MAXr))$

A5 : $r(c,d) \Rightarrow \exists C, D\ (c \in C \wedge d \in D \wedge CrD)$ (entidades somente podem ser relacionadas se as classes correspondentes tambem estiverem)

Se nós admitirmos a existência de um membro chamado "ninguém", podemos enunciar o inverso do axioma 5.

A5´ : $CrD \wedge c \in C \Rightarrow \exists d \in D / (r(c,d))$

Cada relação possui um inverso

A6 : $CrD \Rightarrow \exists s\ (DsC \wedge (r(c,d) \Leftrightarrow s(d,c))$

A generalização G das classes C1,C2,...,Cn pelo papel p é dado por:

A7 : para i=1,2,...,n $is\_a(Ci,G,p) \Leftrightarrow ((g \in G \wedge p_i(g)) \Leftrightarrow g \in Ci$

Isto caracteriza as classes Ci como subclasses de G com $M_{C_i} \subset M_G$.

Um papel aplicado a uma classe D é disjuntivo se as subclasses são disjuntas.

A8 : para 1 <= i,j <= n $\wedge$ i <> j, disjunctive(D,C1,...,Cn,p) <=> ($p_i(e) \Rightarrow \neg p_j(e)$) (se p é disjunctive, então Ci $\cap$ Cj = $\emptyset$ para i <> j)

Se cada entidade da classe genérica está em pelo menos uma subclasse, o papel é dito covering.

A9 : covering(G,C1,...,Cn,p) <=> ($e \in G \Rightarrow \exists i, p_i(e)$) (se p é covering, então $\bigcup$ Ci=G)

Uma classe A é uma agregação de classes C1,...,Cn se:

A10 : aggregated_by(A,C1,...,Cn) <==> $M_A \subset M_1 \times \ldots \times M_n$

A é uma agregação de C1,...,Cn por relações r_{ij} se e somente se exatamente as entidades relacionadas por r_{ij} estão na classe agregada:

A11 : aggregated-by(A,C1,...,Cn,(r_{ij})) <=>

i) $\langle c_1,\ldots,c_n \rangle \in A$ <=> ($\langle c_1,\ldots,c_n \rangle \in C1 \times \ldots \times Cn$) $\wedge$ ($r \in (r_{ij}) \Rightarrow \exists\ 1 <= k,l <= n, r(c_k,c_l)$) (os membros componentes de A devem estar na classe correspondente e relacionados entre si.)

ii) CrD => ($r \in (r_{ij}) \Rightarrow C,D \in (C1,\ldots,Cn)$)

iii) todas as classes componentes estão transitivamente relacionadas:

para $i,j = 1,\ldots n$, $\exists$ K1,...,Km (C_i r $C_{i_{k_1}}$ r $C_{k_1 k_2}$ r $C_{k_2 k_3}$ ... r $C_{k_m j}$ $\wedge$ $Ck_1,\ldots,Ck_m \varepsilon$ (C1,...,Cn)

Uma classe G é um agrupamento de outra classe C por um predicado p, se os seus elementos são conjuntos de elementos de C que satisfazem a este predicado:

A12 : is_elem(C,G,p) <=> $G \subset \wp(C) \wedge ((g \varepsilon G \wedge x \varepsilon C \wedge x \varepsilon g) \Rightarrow p(x,g)) \wedge p(x,g) \Rightarrow (x \varepsilon C \Leftrightarrow x \varepsilon g))$

Um agrupamento é disjuntivo se cada entidade ocorre no máximo em um grupo:

A13 : disjunctive(C,G,p) <=> (is_elem(C,G,p) $\wedge$ ($g_1, g_2 \varepsilon$ G => $g_1 \cap g_2 = \emptyset$)

Um agrupamento G de C é covering se todas as entidades da classe elemento ocorre no mínimo em um grupo:

A14 : covering(C,G,p) <=> is_elem(C,G,p)$\wedge \bigcup$ G = C

A classe T (de tuplas $t=(t_1 : u_1,\ldots,t_n : u_n)$) tem uma relação de ordem natural de pontos de tempo, dados pelos conceitos de "antes" e "depois". Uma classe I é uma classe de intervalo de tempo se ela é um agrupamento de uma classe de tempo T, tal que cada ponto d de tempo entre dois pontos de tempo no intervalo, está neste

grupo:

A15 : interval(I,T) <=> (p,r ε T $\wedge$ p,r ε i ε I) => (p<q<r => q ε i)

Uma classe com tempo é uma classe agregada C' = CxI de uma classe C e de uma classe de intervalo de tempo I, tal que:

A16 : timed(C',C,I) <=> interval(I,T) $\wedge$ aggregated-by(C',C, I) (<c,i_1>,<c,i_2> ε C' => ($i_1 = i_2$ v $i_1 \cap i_2 = \emptyset$))

A17 : timed(C,C,I) => (x ε C <=> $\exists$ t (<x,(t,*)> ε C'))

Uma relação com valores passados de C para D é um sistema r'=<N,R',MIN,MAX>, tal que:

i) <N,R'> é uma classe com R' $\subset M_C \times M_D \times I$, onde I é uma classe de intervalos de tempo.

ii) <c,d,i_1>,<c,d,i_2> ε R' => ($i_1 = i_2$ v $i_1 \cap i_2 = \emptyset$)

iii) para cada ponto do tempo t ε i ε I, se Rt é uma t-projeção de R' sobre $M_C \times M_D$, isto é, Rt={s ε $M_C \times M_D$ / $\exists$ i (t ε i $\wedge$ <s,i> ε R')), então r = <N, Rt,MIN,MAX> é uma relação de C para D.

iv) V c ε C' $\exists$ t_0 V t (t >= t_0 $\wedge$ t <= clock => $\exists$ i,d (t ε i $\wedge$ <c,d,i> ε R')

A18 : old(r') <=> condições i) até iv) acima ocorre.

A19 : excl-pre(C,D) <=> ($x \notin D \wedge o(x \in D) \Rightarrow x \in C$)

(C >--->>)

A20 : excl-post(C,D) <=> ($x \in C \wedge o(x \notin C) \Rightarrow o(x \in D)$

(C >>---> D)

O operador temporal o(p), significa : no próximo estado p é verdadeiro.

5. Aspectos Dinâmicos

Considere a representação de todos os estados do Universo de Discurso no passado, no presente e no futuro, e chamemos isto de UDD(Descrição do Universo de Discurso). UDD é composto de:

UE = {e/ e é uma entidade da UDD}

UC = {C/ C é uma classe da UDD}

UR = {r/ r é uma relação da UDD}

O UE (Universo de Entidades) é a união de conjuntos disjuntos, chamados tipos de entidades. Para uma entidade e, o tipo ao qual ela pertence é denotado por Te, e nós dizemos que é do tipo Te.

O UC (Universo de Classes) é decomposto em subconjuntos disjuntos, chamados metaclasses, tal que há uma bijeção

entre tipos de entidades e metaclasses. Se MC é o conjunto de todas as metaclasses e ET o conjunto de todos os tipos de entidades, nós temos o mapeamento.

rep : ET --> MC

int : MC --> ET

As operações básicas do THM são inserir e deletar entidades, estabelecer e remover relações. Se T e S são tipos de entidades e M=rep(T) uma metaclasse correspondente, então nós definimos:

1) insert ins : TxM --> M

(e,C) --> $C \cup (e)$

2) delete del : TxM --> M

(e,C) --> C-(e)

3) estabilish est : TxSxUR --> UR

(e,g,r) --> $r \cup (<e,g>)$

4) remove rem : TxSxUR --> UR

(e,g,r) --> r-(<e,g>)

Duas operações especiais para grupos e seus elementos são necessárias: g-insert(e,g), torna e um novo elemento de g e g-delete(e,g) deleta e de g.

O prefixo "actual" utilizado nas operações primitivas, significa que esta operação primitiva está em execução.

6. Eventos e Disparadores (Triggers)

O uso de um 'clock' interno e outros testes internos para o início automático de uma operação são performados em THM/DML por meio de eventos e disparadores [Sc1].

Sintaxe

Evento

event <nome-evento> (<lista-parâmetros>)
  on <ocorrência>
  condition <predicado>

Disparador

trigger <nome-disparador>
  - o restante é idêntico ao da operação.

exemplo.

classe FATURA
  member relationships
    para-cliente : CLIENTE
    prazo-pagamento : DIAS
    data-compra : DATA

```
event expiração(i:FATURA):
  on muda(clock.day)
  condition prazo-pagamento(i) < clock.day - data-compra(i)
            i in FATURA-ABERTA
            not (i in FATURA-VENCIDA)

trigger advertência:
  pre_condition
    expiração(i)
    let c be para-cliente(i)
    not (c in CLIENTE-REGULAR)
  body
    enviar-advertência(c,i)
    insert i into FATURA-VENCIDA
```

7. Os Axiomas Dinâmicos

Os axiomas dinâmicos e os de efeitos colaterais, utilizam lógica dinâmica através de duas fórmulas:

1) $p \vdash [op] \Rightarrow q$ (no estado com p verdadeiro, op é permitido somente se q é verdadeiro após a execução de op)

2) $p \vdash [op_1] \Rightarrow [op_2]$ (no estado com p verdadeiro, a execução de op_1 implica na execução de op_2)

DA1 : $\vdash$ [estabilish(x,y,r)] $\Rightarrow \exists C,D\ (x \in C \wedge y \in D \wedge CrD \wedge (MAXr = \star \vee \#(\langle x,z\rangle / r(x,z)) < MAXr)$ (establish deve

manter a cardinalidade maximal)

DA2 : $\vdash$ [remove(x,y,r)] $\Rightarrow$ #(<x,z>/r(x,z)) > MINr (remove deve manter a cardinalidade minimal)

DA3 : is_a(C,D,p) $\vdash$ [insert(x,C)] $\Rightarrow$ $p_C(x)$ (insert deve manter o papel)

DA4 : covering(D,C1,...,Cn) $\wedge$ is_a(Ci,D,p) $\Rightarrow$ $x \notin Ci$) $\vdash$ [insert(x,D)] $\Rightarrow \exists i$ ($p_{C_i}(x)$) (numa generalização covering não se pode permitir uma entidade somente na classe genérica)

DA5 : timed(C',C,I) $\vdash$ [insert(x,C)] $\Rightarrow$ $\langle x,(t,*)\rangle \notin C'$ (numa classe com tempo, não se pode inserir uma entidade atualmente presente.)

DA6 : is_elem(C,G,p) $\vdash$ [g-insert(x,g)] $\Rightarrow$ p(x,g) (os elementos de um grupo devem obedecer o predicado deste grupo.)

DA7 : CrD $\wedge$ MIN(r) = n $\vdash$ [insert(x,C)] $\Rightarrow \exists y_1, y_2, \ldots, y_n$ ($y_i \in D \wedge r(x,y_i)$) (as entidades devem obedecer a cardinalidade mínima.)

DA8 : CrD $\vdash$ [delete(x,C)] $\Rightarrow \nexists y \in D$, r(x,y) (após deletar uma entidade de uma classe, todos os relacinamentos

desta entidade, devem ser deletadas.)

8. Os Axiomas de Efeitos Colaterais

Ao ser efetuada uma operação primitiva, esta pode afetar a integridade semântica do esquema conceitual de dados. Esta operação pode afetar as classes e os relacionamentos. Esta modificação motivada por uma operação primitiva é o que chamamos de efeitos colaterais. Os axiomas de efeitos colaterais nos informam os possíveis efeitos colaterais e nos dão as operações primitivas que neutralizarão estes efeitos.

A aplicabilidade do efeito colateral depende da posição hierárquica da classe afetada. Serão apresentados os possíveis efeitos colaterais para cada posição relativa da classe em três estruturas hierárquicas: generalização, agregação e agrupamento, com seus inversos, chamados especialização, decomposição e dissolução, respectivamente.

Generalização

SE1 : $is\_a(C,D) \wedge x \notin D) \vdash [insert(x,C)] \Rightarrow [insert(x,D)]$

(toda entidade das subclasses tem de estar na superclasse.)

SE2 : is_a(C,D) $\wedge$ disjunctive(D,C1,C2,...,Cn) $\wedge$ $\exists$ i (1 <= i <= n $\wedge$ x $\in$ Ci $\wedge$ C <> Ci) |- [insert(x,C)] => [delete(x,Ci)] (a disjunção não pode ser violada.)

SE3 : is_a(C,D) $\wedge$ covering(D,C1,...,Cn) $\wedge$ (Ci <> C => x $\notin$ Ci) |- [delete(x,C)] => [delete(x,D)] (a deleção não pode violar uma generalização covering.)

SE4 : x $\in$ C $\wedge$ is_a(C,D,p) |- ([estabilish(x,y,r)] v [remove(x,y,r)] => ($p_C(x)$ $\wedge$ o($\sim p_C(x)$) => [delete(x,C)] $\wedge$ ($\sim p_C(x)$ $\wedge$ o($p_C(x)$) => [insert(x,C)] (uma entidade ao sofrer alterações em seus relacionamentos, deve ser movida para uma classe compatível com seus novos relacionamentos.)

Especialização

SE5 : is_a(C,D,p) $\wedge$ $p_C(x)$ |- [insert(x,D)] => [insert(x,C)] (toda entidade da classe genérica deve estar em todas as subclasses compatíveis.)

SE6 : is_a(C,D) $\wedge$ x $\in$ C |- [delete(x,D)] => [delete(x,C)] (toda entidade das subclasses tem de estar na superclasse.)

Agregação

SE7 : aggregated(y,x_1,...,x_n,r_1,...,r_m) Λ y ε D Λ (1<= i <= n => x_i ε Ci) |- [delete(x_i,Ci)] => [delete(y,D)] (sem um elemento a entidade agregada deve ser deletada.)

SE8 : aggregated(y,x_1,...,x_n,r_1,...,r_m) Λ y ε D Λ (1 <= i <= n => x_i ε Ci) Λ r_κ ε (r_1,...,r_m) |- [remove(x_i,x_j,r_κ)] => [delete(y,D)] (numa agregação por relacionamentos, estes devem ser mantidos.)

SE9 : aggregated_by(D,C1,...,Cn,r_1,...r_m) Λ x_i ε C_i Λ $\exists$ r_κ (r_1,...,r_m) (Ci r_κ Cj Λ $\sim$ related(x_i,x_j,r_κ)) |- [estabilish(x_i,x_j,r_κ)] => [insert(<x_1,...,x_n>,D)]

Decomposição

SE10 : aggregated_by (D,C1,...,Cn) |- [insert(y,D)] => (1 <= i <= n Λ part(x_i,y) => [insert(x_i,Ci)]) (os elementos de uma entidade agregada devem estar na classe componente.)

SE11 : aggregated_by(D,C1,...,Cn,r_1,...,r_m) |- [insert(y,D)] => (1 <= i <= n Λ part(x_i,y) => [insert(x_i,Ci)] Λ (is_related(Ci,Cj,r_κ) Λ r_κ ε(r_1,...,r_m) => [estabilish(x_i,x_j,r_κ)])) (os elementos de uma entidade agregada devem estar na classe componente e os seus

relacionamentos estabelecidos.)

Agrupamento

SE12 : is_elem(C,G,p) $\wedge$ g $\in$ G $\wedge$ p(x,g) |- [insert(x,C)] <=> [g_insert(x,g)] (se x satisfaz o predicado de g então x é da classe elemento (C) se e só se ele está em g.)

SE13 : covering(C,G,p) $\wedge$ $\nexists$ g (p(x,g))|- [insert(x,C)] => [insert({x},G) $\wedge$ p(x,{x})) (num agrupamento covering cada entidade da classe elemento deve estar em pelo menos num grupo.)

SE14 : is_elem(C,G,p) $\wedge$ x $\in$ g |- [delete(x,C)] => [g_delete(x,g)]

Dissolução

SE15 : is_elem(C,G) |- [insert(g,G)] => (x $\in$ g $\wedge$ x $\notin$ C => [insert(x,C)]) (os elementos de um grupo devem estar na classe elemento.)

SE16 : covering(C,G) |- [delete(g,G)] => (x $\in$ g $\wedge$ $\nexists$ h (h $\in$ G $\wedge$ h <> g $\wedge$ x $\in$ h) => [delete(x,C)]

SE17 : disjunctive(C,G) |- [g_insert(x,g)] => $\exists$ h (h <> g $\wedge$ x ε h => g_delete(x,h)]) (num agrupamento disjuntivo uma entidade só pode pertencer a um grupo.)

Outros Efeitos Colaterais

SE18 : related(x,y,r) $\wedge$ x ε C |- [delete(x,C)] => [remove(x, y,r)] (ao deletar uma entidade, todos os seus relacionamentos devem ser removidos.)

SE19 : $\sim$ related(y,x,inv_r) |- [establish(x,y,r) => [establish(y,x,inv_r)] (toda relação tem um inverso.)

Alguns Efeitos Colaterais Envolvendo Aspectos Temporais

SE20 : timed(C',C,I) |- [insert(x,C)] <=> [insert(<x,(clock,*)>,C')]

SE21 : timed(C',C,I) |- [delete(x,C)] <=> (<x,(t,*)> ε C'=> [delete(<x,(t,*)>,C')] $\wedge$ [insert(<x,(t,clock)>,C')]) (numa classe com tempo uma entidade deletada deve ser movida para o passado.)

SE22 : old(r') $\wedge$ t=clock |- [establish(x,y,r_t)] <=>

[insert(<x,y,(t,*)>,R´)]

SE23 : old(r´) ∧ t=clock |- [remove(x,y,rt)] <=> (<x,y,(t_0 ,*)> ε R´ => [delete(<x,y,(t_0 ,*)>,R´)] ∧ [insert(<x, y,(t_0 ,t)>,R´)]

SE24 : excl_pre(C,D) |- [insert(x,D)] => [delete(x,C)]

SE25 : excl_post(C,D) |- [delete(x,C)] => [insert(x,D)]

teorema: Os axiomas dinâmicos DA1-DA8 e os de efeitos colaterais SE1-SE25 são necessários para manter a consistência de um BD de acordo com os axiomas A1-A20.

CAPÍTULO III

O SISTEMA DE EFEITOS COLATERAIS

Nos modelos de dados conceituais aparecem dois aspectos distintos, em geral misturados nos modelos descritivos: tratam-se das estruturas de informações e das manipulações das informações [Se].

O THM é um modelo de dado que possui um Esquema Conceitual de Dados (ECD), onde está descrito a parte estrutural do esquema conceitual, incluindo a definição de classe com suas relações, parâmetros, papéis e hierarquias e um Esquema Conceitual de Operações (ECO), onde está definido as operações que manipulam os dados do esquema.

Ao se manipular um BD corre-se o risco de afetar a sua integridade. O THM possui meios de evitar esta perda. Por meio dos axiomas dinâmicos é possível verificar se uma operação desejada é conflitante com alguma restrição do esquema conceitual, com isso ele evita que operações não apropriadas sejam realizadas; e por meio dos axiomas de efeitos colaterais se tem conhecimento de como recompor a consistência com o esquema conceitual; assim sabemos quais operações serão necessárias para recompor o esquema conceitual.

O Sistema de Efeitos Colaterais em THM é todo baseado nos axiomas dinâmicos e de efeitos colaterais. Ele se baseia

no teorema enunciado no capítulo II. Então este sistema checa cada operação primitiva, para verificar se esta é válida, ou seja, se os axiomas DA1-DA8 são respeitados e quais operações primitivas são necessárias, se for o caso, para manter a integridade do esquema conceitual, observando os axiomas de efeitos colaterais SE1-SE25.

1. Implementação

O sistema é implementado se utilizando basicamente ferramentas do THM. Os axiomas dinâmicos e de efeitos colaterais são transformados em operações escritas em THM/DML e armazenadas no ECO. Vamos dividir o seu desenvolvimento em duas fases:

Fase 1

Aqui nós temos uma série de operações escritas em THM/DML. Cada operação desta, possui no procedimento uma ou mais operações primitivas. Cada operação primitiva deve ser submetida as sequintes verificações:

i) se a operação é válida: o que pode ser verificado por meio dos axiomas dinâmicos;

ii) que operações primitivas devem ser executadas para manter o esquema conceitual inalterado.

Para que i) e ii) seja realizado, substituimos todas as operações primitivas de todas as operações por duas chamadas; uma à uma operação que verifica i) e uma outra que verifica ii) e para para fazer esta substituição implementamos um programa que chamamos PROG-SUBST.

Fase 2

Nesta fase os axiomas dinâmicos e de efeitos colaterais foram divididos em dois grupos. A questão era: como fazer para verificar se uma operação primitiva poderia ser executada e quais operações primitivas devem ser executadas para neutralizar os efeitos colaterais. Observamos que os axiomas dinâmicos e de efeitos colaterais são caracterizados pela operação primitiva e estas por sua vez com a posição hierárquica das classes envolvidas na operação, então resolvemos agrupar os axiomas de acordo com a operação primitiva a ser executada para testar os axiomas dinâmicos e para os axiomas de efeitos colaterais, além disso nós os separamos de acordo com a hierarquia das classes envolvidas e de casos especiais, como veremos a sequir:

Grupo dos axiomas dinâmicos

operação primitiva	axiomas
insert	DA3, DA4, DA5 e DA7

establish	DA1
remove	DA2
g-insert	DA6
delete	DA8

Grupo dos axiomas de efeitos colaterais

operação primitiva	hierarquia	axiomas
insert	generalização	SE1 e SE2
	especialização	SE5
	desagregação	SE10 e SE11
	agrupamento	SE12 e SE13
	dissolução	SE15
	caso especial	SE24
delete	generalização	SE3
	especialização	SE6
	agregação	SE7
	agrupamento	SE14
	dissolução	SE16
	casos especiais	SE18 e SE25
establish	generalização	SE4
	agregação	SE9
	casos especiais	SE19 e SE22

remove	generalização	SE4
	agregação	SE8
g_insert	dissolução	SE17

Isto feito, fizemos o mapeamento destes axiomas para operações THM/DML.

1.1 - O Primeiro Grupo

Este grupo de operações verifica a validade das operações primitivas. O grupo está dividido em cinco operações, sendo que cada operação agrupa todos os Axiomas Dinâmicos relacionados com uma operação primitiva em particular.

Operação 1 : verifica os Axiomas Dinâmicos (AD) da operação primitiva insert

Operação 2 : verifica os AD da operação primitiva delete.

Operação 3 : verifica os AD da operação primitiva establish.

Operação 4 : verifica os AD da operação primitiva remove.

Operação 5 : verifica os AD da operação primitiva g_insert.

Operação 1 (inserir)

```
operation Insert_DA(x,C)
  body
    DA3(x,C) - operação 1.1
    DA4(x,C) - operação 1.2
    DA7(x,C) - operação 1.3
    (a operação 1 é composta de 4 chamadas à operações: ope-
ração 1.1, 1.2, 1.3 e 1.4).
```

Operação 1.1

Função: procurar uma ou mais superclasses D de C, com os respectivos papéis, e caso encontre, verificar DA3, via operação 1.1.1

```
operation DA3(x,C)
  body
    for each class D and role p such that is_a(C,D,p)
      do
        Axioma_DA3(x,C,p)  * operação 1.1.1
```

Operação 1.1.1

```
operation Axioma_DA3(x,C,p)
  pre_condition
    role_compatible(x,C,p) otherwise error
  body
    insert x into C
    (x deve ser compatível com C pelo papel p).
```

```
/* A OPERAÇÃO 1.2 NÃO FOI IMPLEMENTADA */
```

Operação 1.3

Função: obter todas as classes D que estão relacionadas com C e os respectivos relacionamentos r.

```
operation DA7(x,C)
  body
    for each class D and rel r such that CrD
     do
       Axioma-DA7(x,r,C,D)

operation Axioma-DA7(x,r,C,D)
  body
    let n be min(r)
    insert x in C
  post_condition
    card(r(x)) >= n otherwise error
```

(a cardinalidade mínima para cada relacionamento r, que x pode ter, com cada classe D deve ser respeitada).

Operação 2 (deletar)

Função: verificar as classes D que estão relacionadas com C, e verificar DA8, via operação 2.1

```
operation delete_DA(x,C)
  body
```

```
    for each class D and rel r such that CrD
      do
        Axioma_DA8(x,r,C,D)   * operação 2.1
```

Operação 2.1

```
operation Axioma_DA8(x,r,C,D)
  body
    delete x from C
  post_condition
    not (y in D) or not (x r y) otherwise error
```

(não deve haver mais nenhum relacionamento com x após sua deleção).

Operação 3 (estabelecer)

Função: verificar o axioma DA1, para saber se a cardinalidade de x não ultrapassa o máximo permitido.

```
operation Establish_DA(x,y,r)
  pre_condition
    not ((x r y) and (x in C) and (y in D)) or CrD otherwise
error
  body
    let n be max(r)
    let m be card(<x,z>/r(x,z))
    establish(x,y,r)
  post_condition
    max = * or m < n otherwise error
```

pectiva operação primitiva.

Operação 1 : verifica os Efeitos Colaterais (SE) da operação primitiva insert.

Operação 2 : verifica os SE da operação primitiva delete.

Operação 3 : verifica os SE da operação primitiva establish.

Operação 4 : verifica os SE da operação primitiva remove.

Operação 5 : verifica os SE da operação primitiva g_insert.

Operação 1 - grupo da operação INSERIR

```
operation Insert_SE(x,C)
  body
    Insert_Gen_SE(x,C)     * SE1 e SE2 - 1.1
    Insert_Esp_SE(x,C)     * SE5 - 1.2
    Insert_Dec_SE(x,C)     * SE10 e SE11 - 1.3
    Insert_Agrup_SE(x,C)   * SE12 e SE13 - 1.4
    Insert_Dis_SE(x,C)     * SE15 - 1.5
    Insert_SE24_SE(x,C)    * SE24 - 1.6
```

A operação 1 é composta de cinco chamadas à operações, que chamaremos: operação 1.1, 1.2, 1.3 e 1.4 que se baseiam na posição hierárquica da classe afetada e operação 1.5 para outros casos.

Operação 1.1 (generalização)

Função: verificar se a classe C possui uma ou mais superclasses D, e se existir, saber se os efeitos colaterais SE1 e SE2 ocorrem, verificados respectivamente pelas operações 1.1.1 e 1.1.2.

```
operation Insert_Gen_SE(x,C)
  body
    for each class D and role p such that is_a(C,D,p)
      do
        Inserir_SE1(x,D)      * operação 1.1.1
        Inserir_SE2(x,p,C,D) * operação 1.1.2
```

Operação 1.1.1

```
operation Inserir_SE1(x,D)
  pre_condition
    not x in D
  body
    Insert_DA(x,D)     * Axiomas Dinâmicos(DA) - operação 1
    Insert_SE(x,D)     * Efeitos Colaterais(SE) - operação 1
```

(se x não está na superclasse D, então ele deve ser inserido em D).

Operação 1.1.2

```
operation Inserir_SE2(x,p,C,D)
  pre_condition
    disjunctive(p,C,D)
```

body

for each class B such that is_a(B,D,p) and x in B

do

Delete_DA(x,B) * DA - operação 2

Delete_SE(x,B) * SE - operação 2

(se D é superclasse de C pelo papel p, e as subclasses de D por p, são disjuntivos, então x deve ser deletado de todas as outras subclasses).

Operação 1.2 (especialização)

Função: verificar se a classe C é superclasse de uma ou mais classes A compatíveis com a entidade x.

operation Insert_Esp_SE(x,C)

body

for each class A and role p such that is_a(A,C,p) and role_compatible(x,p,A) and not x in A

do

Insert_DA(x,A) * DA - operação 1

Insert_SE(x,A) * SE - operação 1

(sse x não está em A, mas é compatível, então deve ser inserido em A).

Operação 1.3 (decomposição)

Função: verificar se x é uma entidade agregada de uma classe C, e se assim for, verificar os efeitos colaterais SE10 e SE11, através das operações 1.3.1 e 1.3.2

```
operation Insert_Dec_SE(x,C)
  body
    Inserir_SE10(x,C)  * operação 1.3.1
    Inserir_SE11(x,C)  * operação 1.3.2
```

Operação 1.3.1

```
operation Inserir_SE10(y,D)
  body
    for each class C such that is_part(C,D)
      do
        let a be is_part(y,C)
        Insert_DA(a,C)    * DA - operação 1
        Insert_SE(a,C)    * SE - operação 1
```

(deve ser inserida toda entidade a que compõe a entidade agregada y na sua respectiva classe).

Operação 1.3.2

```
operation Inserir_SE11(y,D)
  body
    for each class U,V and rel r such that is_part(U,D)
    and is_part(V,D) and UrV
      do
        let u be is_part(y,U)
        let v be is_part(y,V)
        Establish_DA(u,v,r)     * DA - operação 3
        Estabelish_SE(u,v,r)    * SE - operação 3
```

UFPb-BIBLIOTECA/PRAI

(se D é um agregado por relacionamentos, então, toda entidade a que compõe a entidade agregada y, deve ter seus relacionamentos atualizados).

Operação 1.4 (agrupamento)

Função: verificar se C é classe elemento de uma ou mais classes agrupadas G, e se assim for vamos verificar SE12 e SE13, pelas operações 1.4.1 e 1.4.2.

```
operation Insert_Agrup_SE(x,C)
  body
    for each class G such that is_elem(C,G,p)
      do
        Inserir_SE12(x,G)      * operação 1.4.1
        Inserir_SE13(x,C,G,p) * operação 1.4.2
```

Operação 1.4.1

```
operation Inserir_SE12(x,G)
  body
    for each entity g such that g in G and  role_compatible(
     x,g)
      do
        G_Insert_DA(x,g)    * DA - operação 5
        G_Insert_SE(x,g)    * SE - operação 5
```

(Deve-se inserir x em toda entidade g de G compatível com x).

Operação 1.4.2

```
body
  for each entity x such that elem(x,g) and not(x in C)
    do
      Insert_DA(x,C)    * DA - operação 1
      Insert_SE(x,C)    * SE - operação 1
```

(toda entidade x que pertence a g e não pertence a C, deve ser inserida em C).

Operação 1.6 (pré-post)

Função: verificar um caso especial possível, para tanto examinaremos SE24.

```
operation Insert_SE24_SE(x,C)
  body
    for each class D such that excl_pre(D,C)
     do
       Delete_DA(x,D)    * DA - operação 2
       Delete_SE(x,D)    * SE - operação 2
```

se C é exclusivo-pré, então, a inserção em C obriga a deleção em D.

Operação 2 - grupo da operação deletar

```
operation Delete_SE(x,C)
  body
    Delete_Gen_SE(x,C)     * SE3 - 2.1
    Delete_Esp_SE(x,C)     * SE6 - 2.2
    Delete_Agreg_SE(x,C)   * SE7 - 2.3
    Delete_Agrup_SE(x,C)   * SE14 - 2.4
```

(se D é superclasse de C pelo papel p, e nenhuma subclasse F pelo papel p contém x, então x deve ser deletado de D).

Operação 2.2 (especialização)

Função: verificar se D é superclasse de uma ou mais classes C que contém x, e caso isso ocorra,então x deve ser deletado destas classes C que contém x.

```
operation delete_Esp_SE(x,D)
  body
    for each class C and role p such that is_a(C,D,p) and x
     in C
      do
        Delete_DA(x,C)
        Delete-SE(x,C)
```

Operação 2.3 (agregação)

Função: Verificar se há uma ou mais classes D formadas pela agregação de classes do qual C faça parte, e se isto ocorrer, verificar o axioma SE7, via operação 2.3.1.

```
operation Delete_Agreg_SE(x,C)
  body
    for each class D such that aggregated(D,C)
      do
        Deletar_SE7(x,D)  * operação 2.3.1
```

Operação 2.3.1

```
operation Deletar_SE7(x,D)
  body
    for each y such that part(x,y) and y in D
      do
        Delete_DA(y,D)  * DA - operação 2
        Delete_SE(y,D)  * SE - operação 2
```

(toda entidade y de D que possuir x como componente deve ser deletada).

Operação 2.4 (agrupamento)

Função: verificar se C forma uma ou mais classes agrupadas G (agrupamento), e se houver, verificar SE14 pela operação 2.4.1.

```
operation Delete_Agrup_SE(x,C)
  body
    for each class G and role p such that is_elem(C,G,p)
      do
        Deletar_SE14(x,G)  * operação 2.4.1
```

Operação 2.4.1

```
operation Deletar_SE14(x,G)
  body
    for each entity g such that g in G and elem(x,g)
      do
        g_delete x from g
```

(x deve ser deletado de toda entidade g de G que possuir x).

Operação 2.5 (dissolução)

Função: verificar se a classe G é uma classe agrupada, tendo como classe elemento uma classe C. Caso haja, verifica-se SE16 pela operação 2.5.1.

```
operation Delete_Dis_SE(g,G)
  body
    for each class C such that is_elem(C,G) and  covering(C,
     G)
      do
        Deletar_SE16(g,C,G)  * operação 2.5.1
```

Operação 2.5.1

```
operation Deletar_SE16(g,C,G)
  body
    for each entity x such that elem(x,g)
      do
        Del_SE16(x,C,G)  * operação 2.5.1.1
    (submeter toda entidade x de g à operação 2.5.1.1).
```

Operação 2.5.1.1

```
operation Del_Se16(x,C,G)
  body
    Delete_DA(x,C)  * DA - operação 2
    Delete_SE(x,C)  * SE - operação 2
  post_condition
    not (h in G) or not(x in h)
```

(se x não estiver em nenhuma entidade de grupo h de G, x deve ser deletado de C).

Operação 2.6

Função: verificar dois casos especiais possíveis, o SE18 e SE25, através das operações 2.6.1 e 2.6.2.

```
operation Delete_Other_SE(x,C)
  body
    Deletar_SE18(x)     * operação 2.6.1
    Deletar_SE25(x,C)   * operação 2.6.2
```

Operação 2.6.1

```
operation Deletar_SE18(x)
  body
    for each entity y and rel r such that related(x,y,r)
     do
        Remove_DA(x,y,r)  * DA - operação 4
        Remove_SE(x,y,r)  * SE - operação 4
```

(ao se deletar uma entidade x, também se faz necessário remover todos os seus relacionamentos).

Operação 2.6.2

```
operation Delete_SE25(x,C)
  body
    for each class D such that excl_post(C,D)
      do
        Insert_DA(x,D)    * DA - operação 1
```

```
    Insert_SE(x,D)    * SE - operação 1
```

(ao se deletar uma entidade x de uma classe C - 'post' exclusiva em relação a uma ou mais classes D, devemos inserí-la em D).

Operação 3 - grupo da operação ESTABELECER

```
operation Establish_SE(x,y,r)
  body
    Establish_Gen_SE(x,y,r)     * SE4 - 3.1
    Establish_Agreg_SE(x,y,r)   * SE9 - 3.2
    Establish_Other_SE(x,y,r)   * SE19 e SE22 - 3.3
```

A operação 3 é composta de três chamadas à operações, que são as operações 3.1, 3.2 e 3.3 que se baseiam na posição hierárquica da classe que está sendo alterada e operação 3.4 para outros casos.

Operação 3.1 (generalização)

Função: encontrar as classes C que possuem x.

```
operation Establish_Gen_SE(x,y,r)
  body
    for each class C such that x in C
      do
        Estabelecer_SE4(x,y,r,C)  * operação 3.1.1
```

Operação 3.1.1

Função: verificar se C possui uma ou mais superclasses D, e se houver, verificar SE4, via operação 3.1.1.1

```
operation Estabelecer_SE4(x,y,r,C)
  body
    for each class D and role p such that is_a(C,D,p)
      do
        Estab_SE4(x,C,p)  * operação 3.1.1.1
```

Operação 3.1.1.1

```
operation Estab_SE4(x,C,p)
  body
    Delete_DA(x,C)    * DA - operação 2
    Delete_SE(x,C)    * SE - operação 2
  post_condition
    not role_compatible(x,C) otherwise cancel
```

(se x não é mais compatível com C, então x deve ser deletado deste).

```
/* A OPERAÇÃO 3.2 NÃO FOI IMPLEMENTADA */
```

Operação 3.3

Função: verificar um caso especial possível, o SE19, via operação 3.3.1

```
operation Establish_Other_SE(x,y,r)
```

```
  body
    Estabelecer_SE19(x,y,r)  * operação 3.3.1
```

Operação 3.3.1

```
operation Estabelecer_SE19(x,y,r)
  pre_condition
    not (related(y,x,inv_r))
  body
    Establish_DA(y,x,inv_r)  * DA - operação 3
    Establish_SE(y,x,inv_r)  * SE - operação 3
```

(se não existir a relacionamento inverso, inv_r, de y para x, então este deve ser estabelecido).

Operação 4 - grupo da operação REMOVER

```
operation Remove_SE(x,y,r)
  body
    Remove_Gen_SE(x,y,r)     * SE4 - 4.1
    Remove_Agreg_SE(x,y,r)   * SE8 - 4.2
```

(a operação 4 é composta de três chamadas à operações, que chamaremos: operação 4.1 e 4.2 baseadas na posição hierárquica da classe afetada).

Operação 4.1 (generalização)

Função: encontrar as classes C que possuem x

```
operation Remove_Gen_SE(x,y,r)
  body
    for each class C such that x in C
```

```
    do
      Remover_SE4(x,y,r,C)  * operação 4.1.1
```

Operação 4.1.1

Função: verificar se C possui uma ou mais superclasses D, e se houver, verificar SE4, via operação 4.1.1.1

```
operation Remover_SE4(x,y,r,C)
  body
    for each class D and role p such that is_a(C,D,p)
      do
        Remov_SE4(x,p,r,C)  * operação 4.1.1.1
```

Operação 4.1.1.1

```
operation Remov_SE4(x,p,r,C)
  body
    Delete_DA(x,C)    * DA - operação 2
    Delete_SE(x,C)    * SE - operação 2
  post_condition
    not (role_compatible(x,C,p)
```

(se x não é mais compatível com C, então x deve ser deletado deste).

Operação 4.2 (agregação)

Função: procurar classes C e F que sejam relacionadas, contendo x e y respectivamente, tal que C esteja relacionado com F, e se encontrarmos, verificar SE8 pela operação 4.2.1.

```
operation Remove_Agreg_SE(x,y,r)
  body
    for each class C,F such that x in C and y in F and CrF
      do
        Remover_SE8(x,y,r,F,C)  * operação 4.2.1
```

Operação 4.2.1

```
operation Remover_SE8(x,y,r,F,C)
  body
    for each class D such that aggregated_by_rel(D,C,F)
      do
        Remoção_SE8(x,y,D)  * operação 4.2.1.1
```

(aqui procuramos as classes D agregadas pela relação {rj}, cujas classes C e F fazem parte das classes que agregam, e passamos como dado para a operação 4.2.1.1).

Operação 4.2.1.1

```
operation Remoção_SE8(x,y,D))
  body
    for each entity z in D such that part(x,z) and part(y,z)
      do
        Delete_DA(z,D)  * DA - operação 2
        Delete_SE(z,D)  * SE - operação 2
```

(toda entidade z de D, que tiver como componente x e y deve ser deletada).

Operação 5 - grupo da operação G_INSERIR

```
operation G_Insert_SE(x,g)
  body
    G_Insert_Dis_SE(x,g)     * SE17 - 5.1
```

(esta operação possui apenas uma chamada a operação 5.1, e se baseia na posiçao hierárquica da classe).

Operação 5.1 (dissolução)

Função: encontrar uma ou mais classes C que contém x, e verificar se a classe C é uma classe que forma uma ou mais classes agrupadas G, disjuntivas, e sendo este o caso, verificar SE17, via operação 5.1.1

```
operation G_Insert_Dis_SE(x,g)
  body
    for each class C,G such that x in C and is_elem(C,G) and
    disjunctive(C,G)
      do
        G_SE17(x,g,G)  * operação 5.1.1
```

Operação 5.1.1

```
operation G_SE17(x,g,G)
  body
    for each h such that elem(x,h) and h in G and h <> g
      do
        g_delete x from h
```

(se existir outra entidade h de G, diferente de g, que contém x, então x deve ser deletado desta classe).

CAPÍTULO IV

EXEMPLOS

Para ilustrar o Sistema de Efeitos Colaterais, vamos dar alguns exemplos, tomando como base o esquema abaixo.

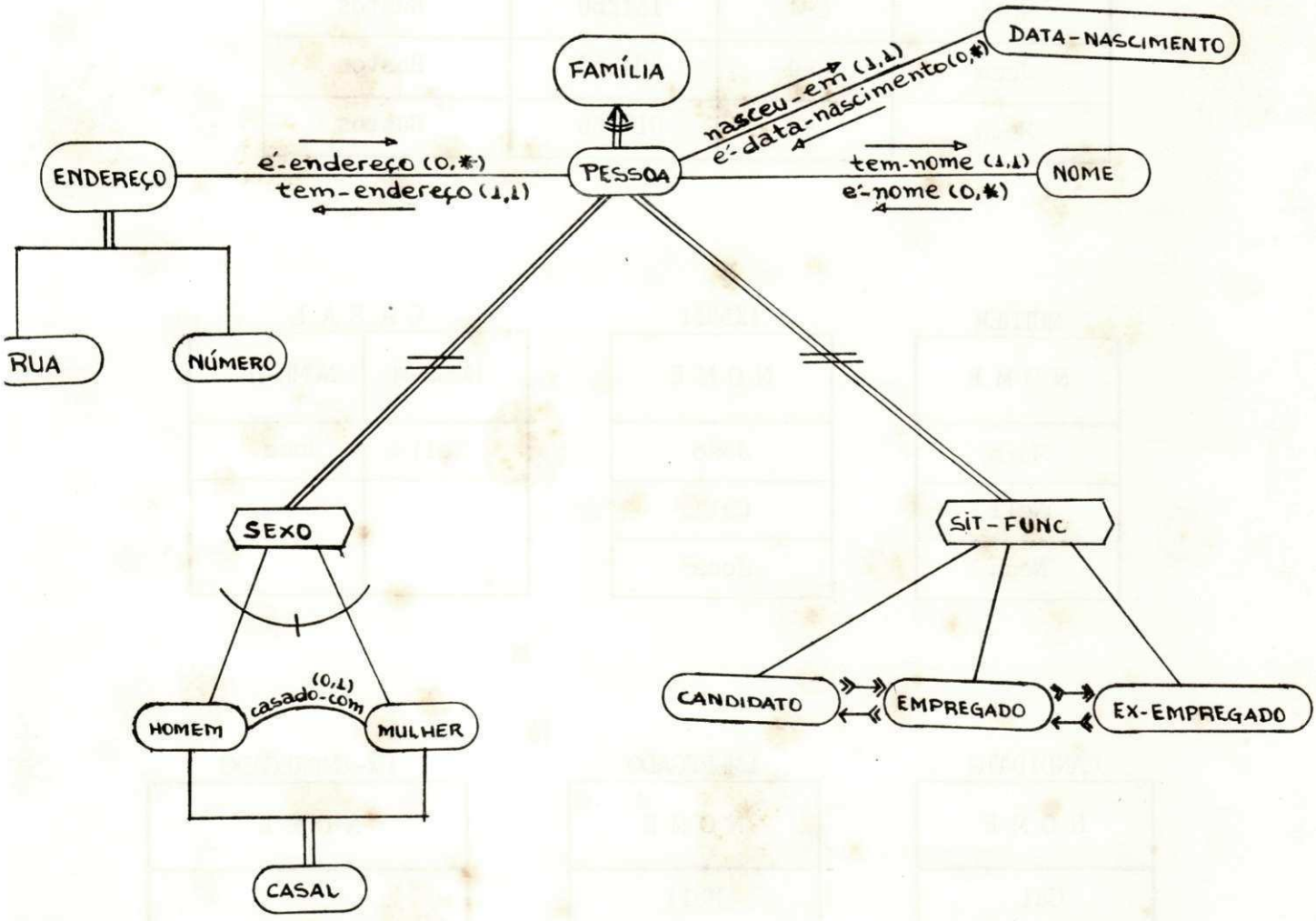

Fig. 4.1

O esquema acima, é mapeado para o modelo interno relacional. Seja o sequinte Banco de Dados

P E S S O A

N O M E	ENDEREÇO	NASCIMENTO	FAMÍLIA
João	B-1	011060	Silva
Cal	A-5	101050	Matos
Mara	B-1	301180	Fulô
Neli	C-4	151260	Bastos
Joca	C-4	201260	Bastos
Noca	C-4	011186	Bastos

MULHER

N O M E
Mara
Neli
Neca

HOMEM

N O M E
João
Cal
Joca

C A S A L

NOME-M	NOME-H
Neli	Joca

CANDIDATO

N O M E
Cal

EMPREGADO

N O M E
Neli
Joca

EX-EMPREGADO

N O M E

FAMÍLIA

SOBRENOME
Silva
Matos
Fulô
Bastos

Caso 1. Inserir Juca na classe PESSOA.

Dados

nome : Juca

sobrenome : Bastos

nascimento : 131087

sexo : masculino

A operação primitiva seria: 'insert Juca into PESSOA'. Esta operação primitiva seria desdobrada em duas operações pelo programa 'PROG-SUBST'.

PROG-SUBST

insert Juca into PESSOA ----------> Insert_DA(Juca,PESSOA)

Insert_SE(Juca,PESSOA)

Insert_DA(Juca,PESSOA), é uma chamada ao grupo de operações que verificam os axiomas dinâmicos:

DA3(Juca,PESSOA)

DA4(Juca,PESSOA)

DA7(Juca,PESSOA)

DA3(Juca,PESSOA) verifica se a classe PESSOA é subclasse de alguma classe, o que não ocorre.

DA4(Juca,PESSOA) verifica se a classe PESSOA é superclasse das classes HOMEM e MULHER pelo papel 'sexo', e superclasse de CANDIDATO, EMPREGADO e EX-EMPREGADO pelo papel 'situação-funcional'. Como (pelo esquema) sabemos que a especialização por 'sexo' é do tipo 'covering', é necessário que após Juca ser inserido em PESSOA, ele deva ser inserido em alguma sub-classe de PESSOA, compatível com seu sexo.

DA7(Juca,PESSOA) verifica se após a inserção de Juca na classe PESSOA, é satisfeita a condição de cardinalidade mínima com todas as classes que PESSOA está relacionada.

Insert_SE(Juca,PESSOA), chama o grupo de operações que verificam os axiomas de efeitos colaterais:

Insert_Gen_SE(Juca,PESSOA)

Insert_Esp_SE(Juca,PESSOA)

Insert_Dec_SE(Juca,PESSOA)

Insert_Agrup_SE(Juca,PESSOA)

Insert_Dis_SE(Juca,PESSOA)

Insert_SE24_SE(Juca,PESSOA)

Insert_Gen_SE(Juca,PESSOA) não se aplica, pois PESSOA não é subclasse de nenhuma classe.

Insert_Esp_SE(Juca,PESSOA) detecta que a classe PESSOA é superclasse das classes HOMEM e MULHER pelo papel 'sexo', e verifica se Juca é compatível com alguma dessas classes. Como Juca é homem, então ele deve ser inserido na classe HOMEM, o que acarreta uma nova operação primitiva 'insert Juca into HOMEM', que será desdobrada em:

Insert_DA(Juca,HOMEM)

Insert_SE(Juca,HOMEM)

Insert_DA(Juca,HOMEM) chama as operações:

DA3(Juca,HOMEM)

DA4(Juca,HOMEM)

DA7(Juca,HOMEM)

DA3(Juca,HOMEM) exige que Juca seja compatível com HOMEM pelo papel 'sexo'.

DA4(Juca,HOMEM) não se aplica (não foi implementado)

DA7(Juca,HOMEM) obriga que seja respeitada a cardinalidade mínima entre as entidades da classe HOMEM com as entidades das classes com as quais estiver relacionada.

Insert_SE(Juca,HOMEM), chama todas as operações relacionadas com esta operação primitiva, mas a úniça que vamos ver é a Insert_Gen_SE(Juca,PESSOA), que exige Juca em PESSOA, mas nesse caso, a função 'actual insert' informa que Juca está sendo inserido em PESSOA.

A operação Insert_Esp_SE(Juca,PESSOA) detecta ainda que a classe PESSOA é superclasse das classes CANDIDATO, EMPREGADO e EX-EMPREGADO pelo papel situação funcional´ e verifica se Juca é compatível com alguma dessas classes, o que não ocorre.

Insert_Dec_SE(Juca,PESSOA) não se aplica.

Insert_Agrup_SE(Juca,PESSOA) encontra a classe que é formada pelo agrupamento da classe PESSOA, no caso FAMILIA, e chama a operação Inserir_SE12(Juca,PESSOA,FAMILIA).

Inserir_SE12(Juca,PESSOA,FAMILIA) verifica se existe alguma família da classe FAMILIA compatível com Juca, e como há, Juca deve ser inserido nesta família. Temos então nova operação primitiva ´g_insert Juca into Bastos´, que se desdobrará nas operações:

G_Insert_DA(Juca,Bastos)

G_Insert_SE(Juca,Bastos)

G_Insert_DA(Juca,Bastos) verifica se Juca é compatível com Bastos.

G_Insert_SE(Juca,Bastos) não se aplica.

Insert_Agrup_SE(Juca,PESSOA) depois chama Inserir_SE13(Juca,PESSOA,FAMILIA,Bastos) que verifica se a classe FAMILIA em relação a classe PESSOA é do tipo ´covering´, neste caso é (veja esquema), então deve ser verificado se Juca pertence a alguma família, e caso não pertença, deve ser criada uma

nova família, mas no nosso caso Juca é um Bastos.

Insert_Dis_SE(Juca,PESSOA) e Insert_SE24_SE(Juca,PESSOA) não ocorrem.

Conclusão: para que a operação seja executada com sucesso, o procedimento original deverá contar no mínimo com as operações primitivas:

g_insert Juca into Bastos
insert Juca into HOMEM
estabilish Juca nasceu-em 131087
estabilish Juca tem-nome Juca Bastos
estabilish Juca tem-endereço C-4

Caso 2. Estabelecer a relação João casado-com Mara.

A operação primitiva seria : establish João casado-com Mara.

PROG-SUBST

establish João casado-com Mara ----------> Estabilish_DA(
João,Mara,casado-com)
Estabilish_SE(
João,Mara,casado-com)

Establish_DA(João,Mara,casado_com) verifica se a cardinalidade máxima da relação 'casado_com' não foi ultrapassada, ou seja, só se poder estar casado com uma e uma só pessoa.

Establish_SE(João,Mara,casado_com) é composto pelas chamadas as operações:

Establish_Gen_SE(João,Mara,casado_com)

Establish_Agreg_SE(João,Mara,casado_com)

Establish_Other_SE(João,Mara,casado_com)

Establish_Gen_SE(João,Mara,casado_com) não se aplica.

Establish_Agreg_SE(João,Mara,casado_com) se aplica, isto é, ao estabelecermos o relacionamento 'casado_com' entre João e Mara, está sendo formado um novo casal, João e Mara estarão sendo agregados para formar uma única entidade da classe CASAL. Esta nova operação obriga ao exame dos axiomas dinâmicos e de efeitos colaterais, mas não vamos tratá-los.

Establish_Other_SE(João,Mara,casado_com) obriga que seja estabelecido o relacionamento inverso, caso já não esteja estabelecido.

Caso 3. Deletar Cal da classe CANDIDATO.

A operação primitiva seria: delete Cal from CANDIDATO

delete Cal from CANDIDATO ----------> Delete_DA(Cal,CANDIDA-
TO)
Delete_SE(Cal,CANDIDA-
TO)

Delete_DA(Cal,CANDIDATO) verifica se todos os relacionamentos da entidade Cal na classe CANDIDATO serão desfeitos após a deleção. Como candidato não possue relacionamentos, o axioma dinâmico não se aplica.

Delete_SE(Cal,CANDIDATO) chama as operações:

Delete_Gen_SE(Cal,CANDIDATO)

Delete_Esp_SE(Cal,CANDIDATO)

Delete_Agreg_SE(Cal,CANDIDATO)

Delete_Agrup_SE(Cal,CANDIDATO)

Delete_Dis_SE(Cal,CANDIDATO)

Delete_Other_SE(Cal,CANDIDATO)

Delete_Other_SE(Cal,CANDIDATO) é a única que se aplica, obrigando a inserção de Cal na classe EMPREGADO quando este é deletado de CANDIDATO.

CAPÍTULO V

COMENTÁRIOS E CONCLUSÕES

O mapeamento do ECO para operações SQL está em desenvolvimento, e somente após a conclusão deste trabalho, que faz parte de uma tese em fase final de desenvolvimento, o 'Sistema de Efeitos Colaterais em THM' poderá ser testado. Os axiomas dinâmicos e de efeitos colaterais que envolvem tempo não foram implementados, pois eles farão parte da implementação das operações primitivas, considerando que esses axiomas serão melhor implementados em tempo de execução, pois estão intimamente ligados ao tempo real da operação.

O sistema que aqui foi proposto garante a consistência de um esquema conceitual em THM. Há possibilidade do usuário criar o seu próprio esquema de efeitos colaterais, por meio de eventos e disparadores. Este esquema seria submetido a um processo semelhante pelo que passa qualquer grupo de operações, inclusive a submissão ao Sistema de Efeitos Colaterais.

As operações em THM são executadas paralelamente, não havendo portanto ordem de execução.

Os axiomas DA4 e SE9 não foram implementados em THM/DML, pois não foi possível encontrar um significado desses axiomas na linguagem.

O Sistema de Efeitos Colaterais em THM, no futuro, poderá vir a sofrer otimizações, já que o nível de recursividade da implementação atual é muito alta, o que refletirá na sua eficiência.

REFERÊNCIAS BIBLIOGRÁFICAS

[CS1] - Cunha,J.F e Schiel,U. - "Representação Relacional do Modelo Semântico de Dados - THM", Informazônia,1986.

[CS2] - Cunha,J.F e Schiel,U. - "Aspectos Dinâmicos do Modelo Temporal Hierárquico - THM", 2SBBD, Porto Alegre, 1987.

[HS] - Horndasch,A. e Schiel,U. - "THM/CSL : A Language for Conceptual Schema Specification", Bericht 5/83, Univ. Stuttgart, 1983.

[ISO] - ISO TC97/SC5/WG3 "Concepts and Terminology for the Conceptual Schema and the Information Base", J.J. Griethuysen,1982.

[Sc1] - Schiel,U. - "The Temporal-Hierarchic Data Model",Bericht 10/82, Univ. Stuttgart,1982.

[Sc2] - Schiel,U. - "An Abstract Introduction to the Temporal-Hierarchic Data Model (THM),in : Proc 9º Int. Conf. on Very Large Data Bases, Florence,1983.

[Sc3] - Schiel,U. - "A Semantic Data Model for Conceptual Schemas and their Mapping to Internal Relational Schemas, Univ. Stuttgart, Doctoral Thesis,1984.

[Se] - Setzer,V.W. - Bancos de Dados, Editora Edgard Blucher Ltda,S.Paulo, 1986.

[SFNC]- Schiel,U. et alli - "Towards Multilevel and Modular Conceptual Schema Specifications", Bericht 2/82, Univ. Stuttgart, 1982.

```
{ Este programa le um conjunto de operacoes escritas em THM/DML, gravadas
  num disquete, procurando operacoes primitivas. Encontrando uma opera-
  cao primitiva, esta e substituida por uma chamada a uma operacao que
  fara a verificacao dos axiomas dinamicos, e insere uma nova linha con-
  tendo uma chamada a uma operacao que fara a verificacao dos efeitos co-
  laterais.}


program PROG_SUBST; {Inicio do PROGRAMA.}

  const

    LIM1   =  255;
    LIM2   =   50;
    IND_I  =    1;
    IND_S  =    5;
    ESPACO = #32;

  type

    CAD1   =  String[LIM1];
    CAD2   =  String[LIM2];
    VET    =  Array[IND_I..IND_S] of CAD2;

  var

    vetor        { Contem as operacoes primitivas.}
                 :    VET;
    arq_oper,    { Arquivo de entrada. Contem as operacoes THM.}
    arq_se       { Arquivo de saida. Acrescida com as chamadas as procedures.}
                 :   Text;
    linha,       { Linha lida de arq_oper e alterada, se possuir operacao pri-
                   mitiva.}
    linha_SE     { Contem chamada aos axiomas de efeitos colaterais.}
                 :   CAD1;
    Tem_Op       { Informa se na linha tem operacao primitiva.}
                 :   Boolean;

  procedure Inicializar_Vetor(var vetor: VET);
  { Procedimento que armazena em 'vetor' as operacoes primitivas.
      Parametro S : vetor }
    begin
      vetor[1]:='g_insert';
      vetor[2]:='insert';
      vetor[3]:='delete';
      vetor[4]:='remove';
      vetor[5]:='estabilish';
    end; {Inicializar_Vetor}


  procedure Trata_Linha(var linha,linha_SE:CAD1;var vetor:VET;
            var Tem_Op:Boolean);
  { Procedimento que trata a linha como um todo.
      Parametro E   : vetor;
      Parametro S   : linha_SE, Tem_Op
      Parametro E/S : linha }
    var
      i,       { indica posicao em que se encontra a operacao no vetor.}
```

```
  posicao, { indica a posicao na linha.}
  k        { tamanho da linha.}
           : Integer;

function Comentario(linha:CAD1;k:Integer):Boolean;
{ Funcao que identifica linha de comentario.
    Parametro E  : linha,k; }
  var
    i     { variavel auxiliar para percorrer a linha.}
          : Integer;
  begin
    i:=1;
    While((linha[i]=ESPACO)and(i(=k)) do i:=i+1;
    if(linha[i]='*')
      then
        Comentario:=TRUE
      else
        Comentario:=FALSE; {if}
  end; {Comentario}

procedure Tp_Operacao(linha:CAD1; vetor:VET; var i,posi:Integer);
{ Procedimento que encontra, caso haja, uma operacao primitiva,
  devolvendo a posicao que se encontra na linha e o indice  que
  indica qual a posicao do vetor em que ela se encontra.
    Parametro E  : linha,vetor;
    Parametro S  : i,posi; }
  begin
    i:=1;
    if (Pos('g_delete',linha)=0 )
       then begin
            While((Pos(vetor[i],linha)=0) and (i(=IND_S)) do i:=i+1;
            if (i(=IND_S)
               then
                 posi :=Pos(vetor[i],linha);
            end
       else
            i:=IND_S + 1;
  end; {Tp_Operacao}


procedure Trata_Operacao(var linha,linha_SE:CAD1;vetor:VET;
                         var posi:Integer;i,k:Integer);
{ Procedimento que trata a operacao como um todo, devolvendo
  a linha com as insercoes necessarias, e a nova a ser inse_
  rida.
    Parametro E   : i,k,vetor;
    Parametro E/S : linha,posi;
    Parametro S   : linha_SE; }
  var
    nova_linha   { variavel auxiliar tipo do tipo 'cadeia'.}
                 : CAD1;
    Par1,        { armazena o primeiro parametro,}
    Par2,        { o segundo parametro parametro e }
    Par3         { o terceiro parametro.}
                 :CAD2;


  procedure Traz_Parametro(linha:CAD1;var posi:Integer;k:Integer;
                                                  var Par:Cad2);
  { Procedimento que devolve um parametro.
```

```
  Parametro E   : linha,k;
  Parametro S   : Par;
  Parametro E/S : posi; }
var
n,        { comprimento da linha.}
gd_posi   { posicao, na linha, do parametro.}
          : Integer;

procedure Trazendo_Par(linha:CAD1;var posi:Integer;k:Integer);
{ Procedimento que traz a posicao, na linha, do termino do
  parametro.
    Parametro E   : linha,k;
    Parametro E/S : posi; }

  procedure Trata_P(var linha:CAD1;var posi:Integer;k:Integer);
  { Procedimento que pula brancos e '(' da linha.
    Parametro E   : k;
    Parametro E/S : linha,posi; }
    begin
      posi := posi+1;
      While(((linha[posi]='(')or(linha[posi]=ESPACO))and(posi(=k))
           do posi := posi+1;
    end; {Trata_P}


  function Operador(caract:Char):Boolean;
  { Funcao que informa se um caracter e ou nao um operador.
      Parametro E  : caract; }
    begin
      if((caract='*') or (caract='+') or (caract='-') or (caract='/'))
        then
          Operador := TRUE
        else
          Operador := FALSE;
    end; {Operador}

  begin  {Trazendo_Par}
    While((linha[posi]=ESPACO) and (posi (= k)) do posi:=posi+1;
    While(not(Operador(linha[posi])))and(linha[posi]()ESPACO)and
         (linha[posi] () ')')and(posi(=k) do  posi:=posi+1;
    if(Operador(linha[posi])) {if_1}
      then begin
           Trata_P(linha,posi,k);
           Trazendo_Par(linha,posi,k);
           end
      else begin
           While(((linha[posi]=ESPACO) or (linha[posi]=')')) and
                                  (posi (= k)) do posi:=posi+1;
           if(Operador(linha[posi])) {if_2}
             then begin
                  Trata_P(linha,posi,k);
                  Trazendo_Par(linha,posi,k);
                  end; {if_2}
           end; {if_1}

  end; {Trazendo_Par}


begin {Traz_Parametro}
  gd_posi := posi;
```

```
    Trazendo_Par(linha,posi,k);
    n:=posi-gd_posi;
    Par:=Copy(linha,gd_posi,n);
  end; {Traz_Parametro}


procedure Passa_Into_From(linha:CAD1;var posi: Integer);
{ Procedimento que pula 'into' e 'from'.
    Parametro E   : linha;
    Parametro E/S : posi; }
  begin
    While(linha[posi] = ESPACO) do posi:=posi+1;
    While(linha[posi] <> ESPACO) do posi:=posi+1;
  end; {Passa_Into_From}


procedure Traz_Par_Ins_Del(linha:CAD1;var posi:Integer;k:Integer;
                                          var Par1,Par2:CAD2);
{ Procedimento que trata as  operacoes 'insert'  e 'delete'.
  Devolvendo Par1 e Par2.
    Parametro E   : linha,k;
    Parametro S   : Par1,Par2;
    Parametro E/S : posi; }
  begin
    Traz_Parametro(linha,posi,k,Par1);
    Passa_Into_From(linha,posi);
    Traz_Parametro(linha,posi,k,Par2);
  end; {Traz_Par_Ins_Del}


function Monta_Ins_Del(linha,nova_linha:CAD1;Par1,Par2:CAD2;
                       var posi:Integer;k:Integer):CAD1;
{ Funcao que devolve uma linha que contem uma operacao 'insert'
  ou 'delete' com as alteracoes feitas.
    Parametro E   : linha,nova_linha,Par1,Par2,k;
    Parametro E/S : posi; }
  begin
    nova_linha:=Concat(nova_linha,Par1,',',Par2,')');
    linha:=Copy(linha,posi,k);
    Monta_Ins_Del:=Concat(nova_linha,linha);
  end; {Monta_Ins_Del}


procedure Traz_Par_Est_Rem(linha:CAD1;var posi:Integer;k:Integer;
                                      var Par1,Par2,Par3:CAD2);
{ Procedimento que trata as operacoes 'estabilish' e 'remove'.
  Devolvendo Par1,Par2 e Par3.
    Parametro E   : linha,k;
    Parametro S   : Par1,Par2,Par3;
    Parametro E/S : posi; }
  begin
    Traz_Parametro(linha,posi,k,Par1);
    Traz_Parametro(linha,posi,k,Par2);
    Traz_Parametro(linha,posi,k,Par3);
  end; {Traz_Par_Est_Rem}


function Monta_Est_Rem(nova_linha,linha:CAD1;
          Par1,Par2,Par3:CAD2;var posi:Integer;k:Integer):CAD1;
{ Funcao que devolve uma linha que contem uma operacao 'estabilish'
```

```
  ou 'remove' com as alteracoes feitas.
    Parametro E   : linha,nova_linha,Par1,Par2,Par3,k;
    Parametro E/S : posi; )
  begin
    nova_linha:=Concat(nova_linha,Par1,',',Par2,',',Par3,')');
    linha:=Copy(linha,posi,k);
    Monta_Est_Rem:=Concat(nova_linha,linha);
  end; {Monta_Est_Rem}


begin {Trata_Operacao}
  nova_linha := ESPACO;
  nova_linha := Copy(linha,1,posi-1);

  case i OF

  1 :  begin
       nova_linha:=Concat(nova_linha,'g_insert_DA(');
       linha_SE:='G_Inserir_SE(';
       posi:=posi+8;
       end; {1}
  2 :  begin
       nova_linha:=Concat(nova_linha,'insert_DA(');
       linha_SE:='Inserir_SE(';
       posi:=posi+6;
       end; {2}
  3 :  begin
       nova_linha:=Concat(nova_linha,'delete_DA(');
       linha_SE:='Deletar_SE(';
       posi:=posi+6;
       end; {3}
  4 :  begin
       nova_linha:=Concat(nova_linha,'remove_DA(');
       linha_SE:='Remover_SE(';
       posi:=posi+6;
       end; {4}
  5 :  begin
       nova_linha:=Concat(nova_linha,'estabilish_DA(');
       linha_SE:='Estabelecer_SE(';
       posi:=posi+10;
       end; {5}
  end; {case}

  case i OF

  1..3 :
    begin
    Traz_Par_Ins_Del(linha,posi,k,Par1,Par2);
    linha:=Monta_Ins_Del(linha,nova_linha,Par1,Par2,posi,k);
    linha_SE:=Concat(linha_SE,Par1,',',Par2,')');
    end; {1..3}
  4..5 :
    begin
    Traz_Par_Est_Rem(linha,posi,k,Par1,Par2,Par3);
    linha:=Monta_Est_Rem(nova_linha,linha,Par1,Par2,Par3,posi,k);
    linha_SE:=Concat(linha_SE,Par1,',',Par2,',',Par3,')');
    end; {4..5}
  end; {case}

end; {Trata_Operacao}
```

UFCG

```
  begin {Trata_Linha}
    k:=Length(linha);
    Tem_Op := FALSE;
    if(not Comentario(linha,k))
      then begin
           Tp_Operacao(linha,vetor,i,posicao);
           if (i <= 5)
              then begin
                   Trata_Operacao(linha,linha_SE,vetor,posicao,i,k);
                   Tem_Op := TRUE;
                   end;

           end;
  end; {Trata_Linha}


begin {THM_SE}
  Inicializar_Vetor(vetor);
  Assign(arq_oper,'OPER.THM');
  Assign(arq_se,'OPERSE.THM');
  Reset(arq_oper);
  Rewrite(arq_se);
  While(not EOF(arq_oper)) do
    begin
      ReadLn(arq_oper,linha);
      WriteLn('Linha lida ',linha);
      Trata_Linha(linha,linha_SE,vetor,Tem_Op);
      WriteLn('Impressa ',linha);
      WriteLn(arq_se,linha);
      if (Tem_Op)
         then
           WriteLn(arq_se,linha_SE);
    end; {While}
  close(arq_oper);
  close(arq_se);
end. {THM_SE}
```