JOSELUCE DE FARIAS CUNHA

MAPEAMENTO DO MODELO SEMÂNTICO DE DADOS **THM** PARA O MODELO RELACIONAL

Dissertação apresentada ao Curso de MESTRADO EM INFORMÁTICA da Universidade Federal da Paraíba, em cumprimento às exigências para obtenção do Grau de Mestre.

ULRICH SCHIEL

Orientador

C972m Cunha, Joseluce de Farias
Mapeamento do modelo semantico de dados THM para o modelo relacional / Joseluce de Farias Cunha. - Campina Grande, 1988.
89 f.

Dissertacao (Mestrado em Informatica) - Universidade Federal da Paraiba.

1. Computacao - Base de Dados 2. Banco de Dados 3. Dissertacao I. Schiel, Ulrich, Dr. II. Universidade Federal da Paraiba - Campina Grande (PB) III. Título

CDU 004.65(043)

MAPEAMENTO DO MODELO SEMANTICO DE DADOS **THM**
PARA O MODELO RELACIONAL

JOSELUCE DE FARIAS CUNHA

DISSERTAÇÃO APROVADA EM 14/03/88

ULRICH SCHIEL
Orientador

RUBENS NASCIMENTO MELO
Componente da Banca

MARCUS COSTA SAMPAIO
Componente da Banca

CAMPINA GRANDE
MARÇO - 1988

MAPEAMENTO DO MODELO SEMÂNTICO DE DADOS **THM** PARA O MODELO RELACIONAL

JOSELUCE DE FARIAS CUNHA

DISSERTAÇÃO APROVADA EM 14/03/88

ULRICH SCHIEL
Orientador

RUBENS NASCIMENTO MELLO
Componente da Banca

MARCUS COSTA SAMPAIO
Componente da Banca

Ao meu querido esposo Vinício e aos nossos filhos Juliana, Priscila e Júnior. Pelo amor que lhes tenho e pelas horas que deixei de estar com eles para me dedicar a este trabalho.

AGRADECIMENTOS

- Ao meu orientador Dr. Ulrich Schiel não só pela orientação e pelo encorajamento nas horas mais difíceis, mas também pelos conhecimentos que oportunamente me transmitiu, contribuindo significativamente para o meu desenvolvimento acadêmico.

- Aos demais colegas do DSC que me apoiaram e contribuiram direta ou indiretamente com este trabalho. Gostaria de ressaltar a contribuição especial de alguns destes colegas:

- Do professor Marcus Sampaio, que com sua vasta experiência em Banco de Dados e em escrita, se dispôs a criticar este documento, fazendo sugestões que valorizaram bastante a apresentação do mesmo;

- Do professor Giuseppe Mongiovi, que sempre se dispôs a encontrar soluções para as minhas dificuldades na fase de implementação e documentação;

- Do professor Antonio Everaldo, pela sua compreensão em liberar-me das atividades da vice-coordenação do Curso de Processamento de Dados;

- De Lilian Diniz, pelo apoio e pela amigável contribuição na digitação deste texto.

MAPPING OF THE THM DATA SEMANTIC MODEL FOR THE RELATIONAL MODEL

ABSTRACT

In order to mantain the conceptual modelling effort, the semantics of this description must be transmitted to the internal operational level. The goal of this work is to convert applications described through the Temporal_Hierarchic Data Model (THM) to the internal level.

The mapping formalized and implemented in this contribution, translates the semantics of the data structures described by THM to the Relational Model and converts the semantic operations given by the application, to executable ones, guaranteing that these operations will not affect data base integrity.

MAPEAMENTO DO MODELO SEMÂNTICO DE DADOS THM PARA O MODELO RELACIONAL

RESUMO

Todo o esforço de modelagem semântica é desperdiçado se ficar no nível conceitual, sem ser transmitido para o nível operacional. A finalidade deste trabalho é explorar o Modelo Semântico de Dados Temporal-Hierárquico (THM) e transferir para o nível operacional a semântica modelada com ele.

O mapeamento aqui formalizado consiste em traduzir os conceitos semânticos abordados pelo THM para o Modelo Relacional, com o objetivo de criar bancos de dados de acordo com a semântica da aplicação e permitir que aqueles sejam manipulados por operações definidas conceitualmente, garantindo que estas não irão alterar a integridade do banco de dados.

SUMÁRIO

MAPEAMENTO DO MODELO SEMÂNTICO DE DADOS **THM** PARA O MODELO RELACIONAL

1. INTRODUÇÃO

Uma das missões mais difíceis na implementação de um sistema de informação é absorver as "coisas" e os "fatos" do universo do discurso que terão representação significativa no nível de implementação. Com o surgimento de Bancos de Dados a implementação ficou mais organizada, facilitando o acesso aos dados. Esta implementação passou a ser feita em um dos modelos tradicionais : Relacional , Hierárquico ou de Redes. Cada um com suas facilidades em termos de organização e acesso aos dados. Porém, todos com o problema de escassêz semântica.

Para facilitar a tradução do mundo real para o nível de implementação e, de certa forma, os dados representarem um pouco da semântica, surgiram os modelos semânticos de dados. Entre os primeiros, o que mais se destaca é o modelo de Entidades e Relacionamentos desenvolvido por Chen [Ch-76]: trata-se de um modelo que representa apenas a estrutura dos dados, mas é de fácil aplicação, de forma que têm surgido muitas propostas para o enriquecimento semântico deste modelo. A partir dele surgiram muitos outros, todos buscando uma representação que retrate a semântica da aplicação. Entre estes modelos temos o Modelo Temporal-Hierárquico (THM), desenvolvido por Ulrich Schiel [Sc-84] que se destaca pela abrangência de fatos do mundo real que ele pode modelar e por considerar também a modelagem da dinâmica da aplicação (operações e eventos).

Para que a proposta de um modelo semântico de dados se torne viável e realmente útil, a sua transformação para um Sistema de

Gerenciamento de Banco de Dados (SGDB), baseado em um dos modelos tradicionais, deve ser estudada e solucionada. Infelizmente pouco tem sido feito nesta direção. Apresentamos neste trabalho uma proposta de transformação do Modelo THM para o Modelo Relacional, tendo em vista uma interface para um SGDB relacional. Baseamo-nos em uma implementação preliminar para o sistema relacional POREL da Universidade de Stuttgart [Pr-84].

Inicialmente apresentamos de forma sintetizada o THM, mostrando primeiro suas facilidades para a modelagem estrutural e depois as facilidades para a modelagem operacional. Expomos ainda uma visão geral da modelagem de sistemas pelo THM, para mostrar como nosso trabalho se enquadra na configuração geral do sistema. Em seguida, apresentamos um algoritmo para transformar a modelagem estrutural em esquema relacional. Finalizando, apresentamos o algoritmo de transformação das operações conceitualmente aplicáveis ao esquema de dados, para operações executáveis juntamente com uma interface que permite ao usuário executar estas operações.

2. O MODELO DE DADOS TEMPORAL-HIERÁRQUICO

O Temporal-Hierarchic Model - **THM**, é um modelo semântico de dados, que se baseia na abordagem de relacionamentos binários e dispõe de conceitos para modelar os aspectos estáticos (estruturais) e aspectos dinâmicos (operacionais) de um sistema de informação.

O modelo será apresentado de forma resumida, mas suficiente para o entendimento dos conceitos abordados nos capítulos seguintes. Mais detalhes podem ser vistos em [Sc-82, Sc-84].

2.1 Modelagem Dos Aspectos Estáticos

Corresponde à formalização das estruturas de um sistema de informação denominada Esquema Conceitual de Dados (ECD), cujos principais conceitos abordados são: a **entidade** como membro de uma **classe** e o **relacionamento** (binário).

Os membros (entidades) de uma classe são identificados por relacionamentos chaves com outras classes.

São distinguidos três tipos de relacionamento:

a)**relacionamento membro-a-membro**: descreve uma relação entre os membros de duas classes.
Por exemplo, "tem_salário" relaciona entidades da classe EMPREGADO a entidades da classe SALÁRIO;

b)**relacionamento classe-a-membro**: relaciona uma classe a um

membro de outra classe.

Por exemplo, "salário_médio" relaciona a classe EMPREGADO como um todo, a uma entidade da classe SALARIO.

c)**relacionamento classe-a-classe:** relaciona duas classes independentes de membros individuais. São os relacionamentos hierárquicos **é-um**, **parte-de** e **elemento-de** e o relacionamento temporal **pré-pós**.

A cada relacionamento é associada uma cardinalidade expressa pelo par de valores (n,m), o qual significa que cada membro da primeira classe (classe origem) está associado a no mínimo n e no máximo m membros da segunda classe (classe relacionada)

classe é um conjunto de membros com propriedades comuns. São distinguidos dois tipos de classe:

a)**classe composta:** seus membros não são identificados por si mesmo e sim pelos relacionamentos com os membros de outra classe.

Por exemplo, as entidades da classe PESSOA são identificadas pelo relacionamento "tem_nome" com a classe NOME, logo é uma classe composta;

b)**classe de domínio:** seus membros são representados por sua própria identificação.

Por exemplo, as entidades da classe NOME se identificam por si mesmo "joão", "carlos"...

São distinguidas duas classes de domínio: dinâmica cujos membros têm representação explícita e podem ser mudados; estática que não tem representação explícita de todos os seus membros.

A Figura 2.1 mostra a representação gráfica destes conceitos, onde os blocos ovalados representam as classes, as arestas simples representam os relacionamentos de membros, as arestas com saída dupla representam os relacionamentos de classe e os valores entre parênteses indicam respectivamente a cardinalidade mínima e máxima do relacionamento.

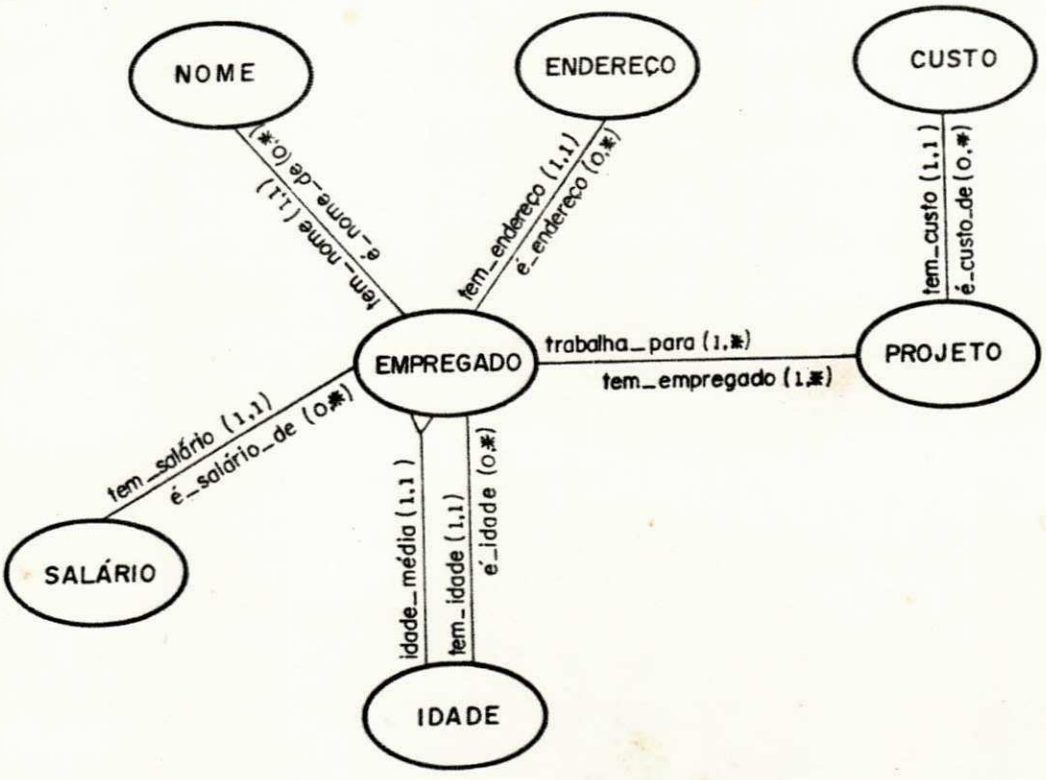


Figura 2.1 Classes e Relacionamentos

Como conceitos de abstração ou relacionamentos hierárquicos entre classes são considerados:

a) **generalização** : Um papel aplicado a uma classe gera subclasses.
Por exemplo, o papel "sexo" aplicado a classe PESSOA gera as subclasses HOMEM e MULHER.

b) **agregação** : as entidades de uma classe podem ser formadas pela composição de entidades de diferentes classes.
Por exemplo, DATA é formada por entidades das classes DIA, MES e ANO.

c) **agrupamento**: Os membros de uma classe podem ser agrupados, e cada grupo ser uma entidade da classe de nível mais alto
Por exemplo, os membros da classe EMPREGADO podem ser agrupados formando membros da classe EQUIPE.

A Figura 2.2 mostra a representação gráfica das hierarquias.

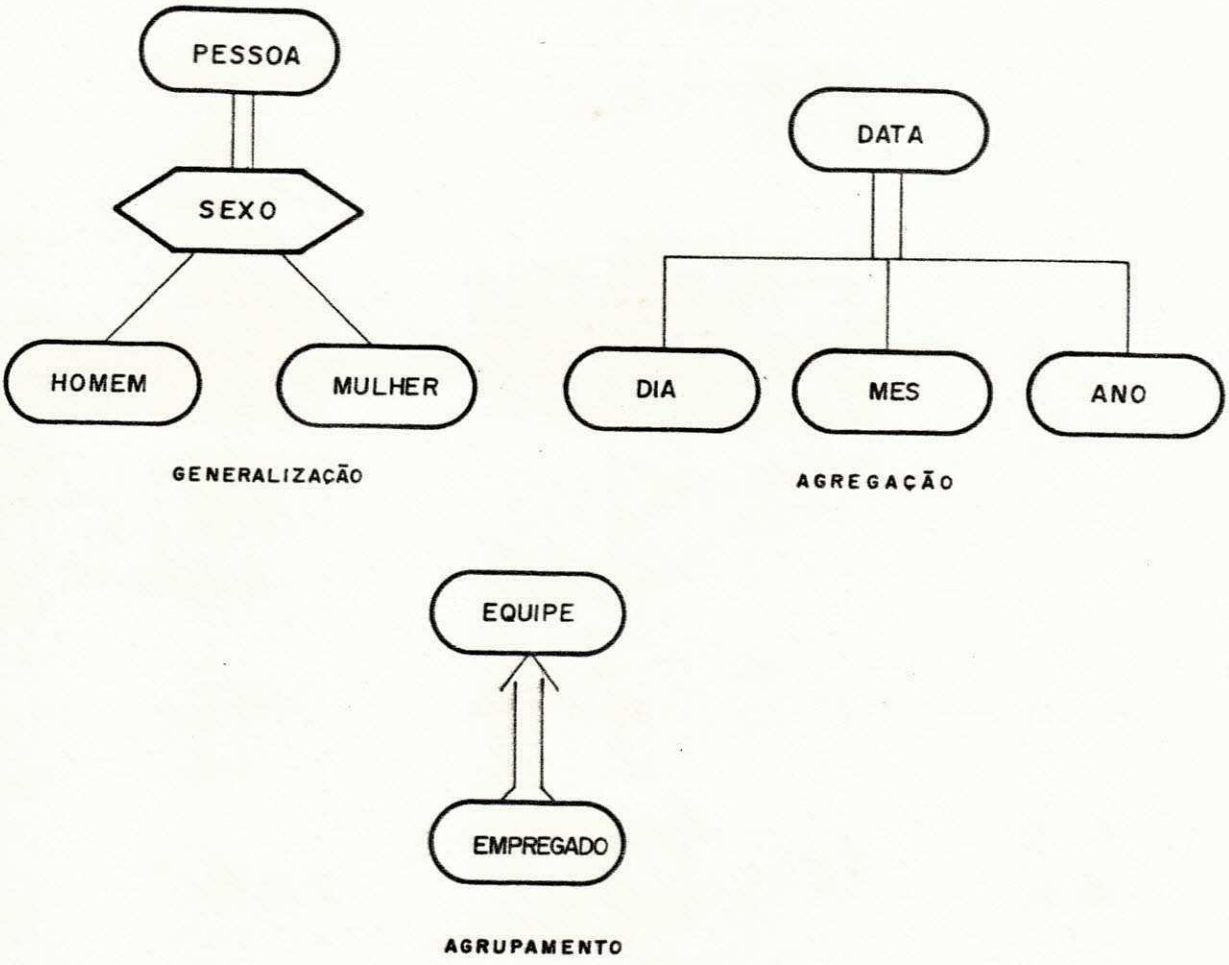
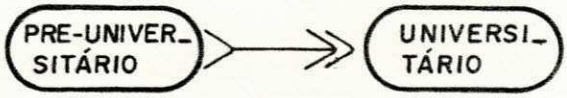


Figura 2.2 Hierarquias

O tempo é modelado no ECD através dos relacionamentos temporais pré-pós, classes com tempo (with time) e relacionamentos com valores anteriores (with old values)

a)relacionamentos pré-pós : para os casos onde as entidades

de uma classe passam a pertencer a uma segunda classe, e as entidades da segunda classe originam-se da primeira Por exemplo, as entidades da classe PRÉ_UNIVERSITÁRIO futuramente podem passar para a classe UNIVERSITÁRIO e todas as entidades desta classe no passado pertenceram àquela. Este relacionamento é representado da seguinte forma:



A seta simples na origem e dupla no destino indica que o relacionamento é pré-simples/pós-exclusivo.

b)classes com tempo : Adicionando-se o parâmetro with time na definição de uma classe, uma nova dimensão é adicionada a suas entidades e os atributos DE e ATE especificam o tempo de existência da entidade na classe. Por exemplo, se quisermos manter o histórico das entidades da classe ALUNOS, a classe será representada da seguinte forma:

c)relacionamento com valores anteriores: quando é necessário manter o histórico de um relacionamento, acrescenta-se a este o parâmetro with old values. Desta forma, quando o relacionamento for alterado, o valor anterior será preservado juntamente com informações sobre o período de sua existência (DE, ATE).

Para especificar todos estes conceitos, o THM dispõe de uma linguagem para a especificação de esquemas conceituais denominada THM/LEC (Linguagem de Esquemas Conceituais modelados pelo THM) a qual é dividida em duas partes : A THM/LDD (Linguagem de Definição de Dados do THM) para descrever o esquema conceitual de dados que incluem as estruturas definidas até agora e, a THM/LMD (Linguagem de Manipulação de Dados do THM) para descrever o esquema conceitual de operações. A sintaxe da linguagem pode ser vista nos apêndices A e B.

2.2 Modelagem dos Aspectos Dinâmicos.

Corresponde a descrição de operações (ações) aplicáveis a um sistema de informação, denominada Esquema Conceitual de Operações (ECO).

Para o ECO, o THM oferece o conceito de **operações** que só são executadas sob certas pré-condições. Além disto, podem ser descritos **eventos**, que sob certas condições, disparam automaticamente operações denomidadas **triggers**. Um **sistema de eventos e triggers** inerente à aplicação e um **sistema de efeitos colaterais** definido para o modelo garantem a integridade estrutural do esquema conceitual.

Destes conceitos, só abordaremos as operações. O desenvolvimento do sistema de eventos e o do sistema de efeitos colaterais podem ser vistos respectivamente em [Tr-87] e [Fr-87].

As operações são codificadas em THM/LMD e denominadas

operações_THM, através das quais o usuário pode manipular o banco de dados com a segurança de que a integridade dos dados será mantida. Isto é garantido pela própria estrutura das operações-THM (mais detalhada no capítulo-5), que possue um corpo (body) de instruções composto de operações primitivas e/ou chamadas a outras operações-THM, que só é executado se as pré-condições se verificarem. Isto leva o banco de dados de um estado inicial consistente a um estado final consistente.

As operações primitivas são as instruções básicas para inserir ou eliminar entidades de uma classe e para estabelecer ou remover relacionamentos. As pré-condições equivalem a uma conjunção de predicados primitivos que correspondem às consultas básicas que verificam se uma entidade pertence a uma classe, se duas entidades estão relacionadas e outras consultas que dizem respeito as hierarquias. Relacionamos a seguir os predicados e operações primitivas:

Predicados primitivos

a) e in C (at time)

Verifica se a entidade e pertence a classe C em um determinado intervalo de tempo.

b) e1 r e2 (at time)

Verifica se as entidades e1 e e2 estão relacionadas por r em um determinado intervalo de tempo.

c) is-part(e1,e2)

Verifica se a entidade **e1** é componente da entidade agregada **e2**.

d) **is_elem(e1,e2)**

Verifica se a entidade **e1** é um elemento da entidade de grupo **e2**.

Existem outros predicados primitivos que não citaremos. Os mesmos são usados exaustivamente pelo sistema de efeitos colaterais [Fr-87] e servem para verificar a estrutura hierárquica das classes em relação ao ECD.

Operações primitivas

Existem 4 operações básicas:

a) **insert e into C (at time)**

Insere a entidade **e** na classe **C** a partir de um instante especificado.

b) **delete e from C (at time)**

Elimina a entidade **e** da classe **C** a partir do instante especificado.

c) **establish e1 r e2**

Estabelece o relacionamento **r** entre as entidades **e1** e **e2** **(relacionamento de membros)**;

establish C r e

Estabelece o relacionamento **r** entre a classe **C** como um todo e a entidade **e**.

d) **remove** e1 r e2

Elimina o relacionamento r entre as entidades **e1** e **e2**;

remove C r e

Elimina o relacionamento r entre a classe C e a entidade **e**.

Mais duas operações primitivas podem ser consideradas :

move e **from** C1 **to** C2

Como a seqüência das operações

delete e **from** C1

insert e **into** C2

update e1 r e2 **to** e3

Como a seqüência das operações

remove e1 r e2

establish e2 r e3

Note-se que os aspectos temporais são considerados tanto nos predicados primitivos como nas operações primitivas. São opcionais porque nem todo relacionamento ou toda classe são definidos com parâmetros temporais. Quanto aos relacionamentos entre pré-classe e pós-classe, estes são mantidos pelo Sistema de Efeitos Colaterais [Fr-87] através das operações **insert** e **delete**.

3. FASES DE PROJETO DE BANCO DE DADOS COM O THM

Apresentamos neste capítulo uma visão geral das etapas para a modelagem de um sistema de informação com o THM, tendo como objetivo situar o enquadramento do mapeamento proposto no contexto geral do sistema-THM.

Existem 5 fases distintas como mostra a Figura 3.1. Destas fases, fazem parte do nosso trabalho:

- A definição da estrutura de tabelas para o armazenamento do esquema conceitual definido em THM/LEC,
- O mapeamento do esquema conceitual para o esquema relacional,
- A indicação da operacionalização do sistema.

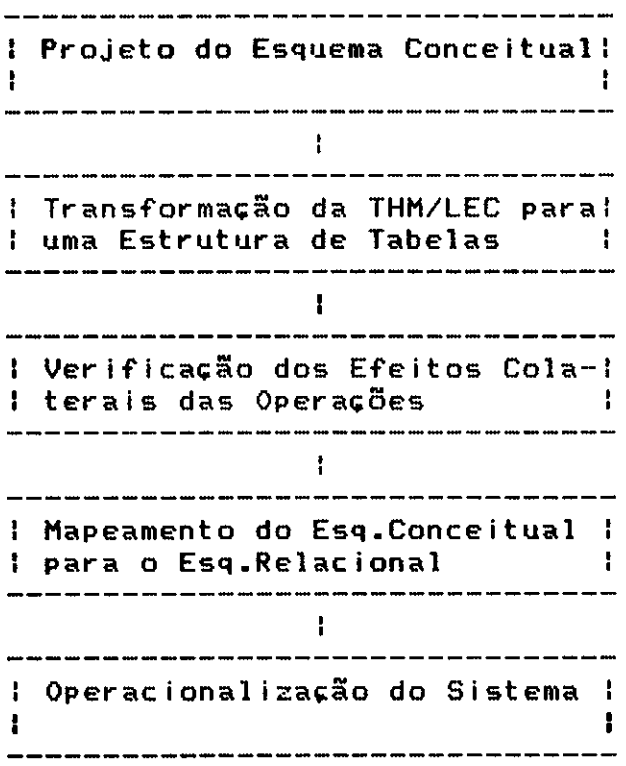


Figura 3.1 Fases de Projeto

A fase do projeto do esquema conceitual corresponde à definição do sistema. Nesta fase pode ser aplicada a chamada Metodologia-THM descrita em [MS-85] e [Sc-87], que consiste na formalização de um sistema de informação partindo do universo do discurso até obter-se uma descrição formal do mesmo. O último passo da metodologia é voltado para a descrição do esquema conceitual através da linguagem THM/LEC.

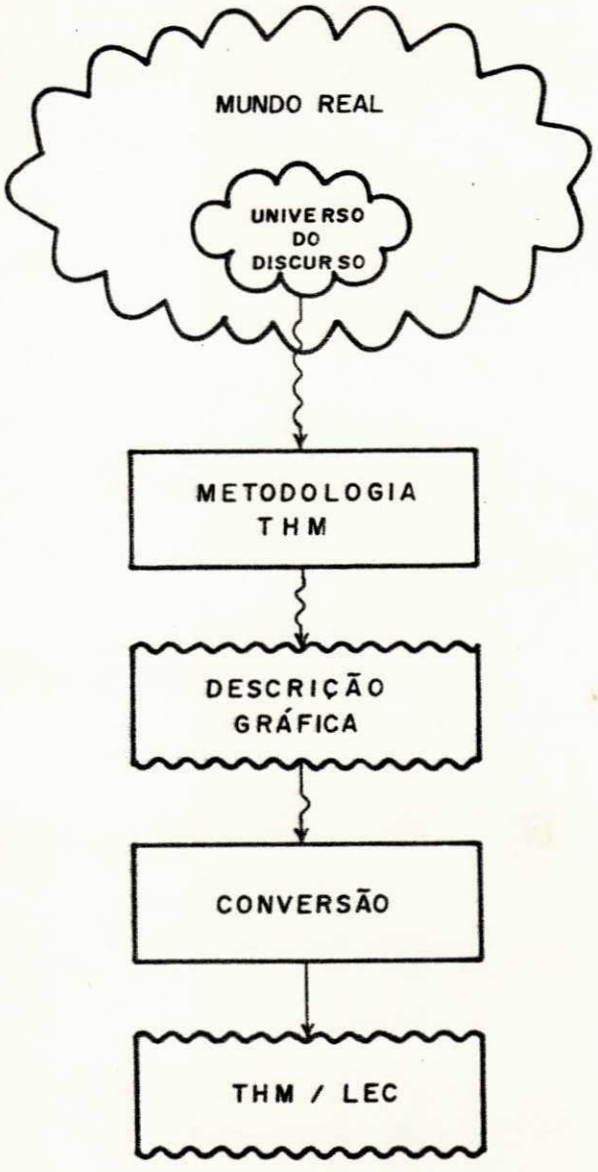


Figura 3.2 : Formalização de uma Aplicação do Mundo Real com Metodologia-THM (Projeto do Esquema Conceitual)

Na segunda fase, a codificação do esquema conceitual em THM/LEC é submetida a um programa que interpreta esta descrição e a transforma em duas estruturas de tabelas distintas que representam respectivamente o Esquema Conceitual de Dados (ECD) e o Esquema Conceitual de Operações (ECO). As duas estruturas de tabelas sao obtidas em duas etapas: na primeira etapa a THM/LDD é compilada gerando como saída um conjunto de tabelas que armazenam o ECD; na segunda etapa a THM/LMD é compilada gerando outro conjunto de tabelas que armazenam o ECO.

As tabelas são gravadas em arquivos que servirão de entrada para o mapeamento (Veja Figura 3.3). Mais detalhes sobre estas tabelas serão vistos no capítulo 4 e 5.

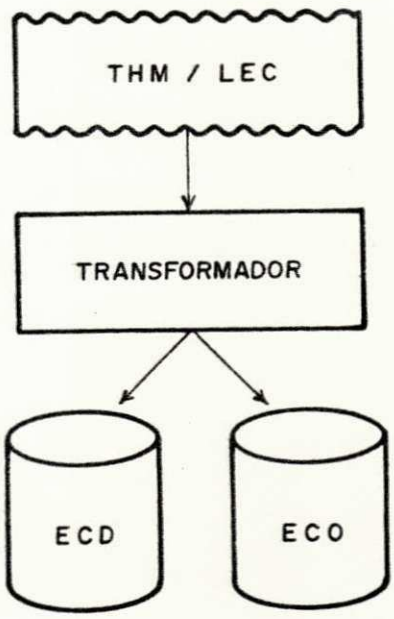


Figura 3.3 : Fase de Transformação do Esquema Conceitual em Tabelas

Na 3a. fase, o sistema de efeitos colaterais [Fr-87] atua no ECO (Figura 3.4) verificando se cada uma das operações mantém a semântica estrutural definida no ECD. Quando alguma operação é definida de forma incompleta, podendo causar inconsistência dos dados com o esquema, o sistema de efeitos colaterais acrescenta novas declarações a esta operação, garantindo a integridade dos dados.

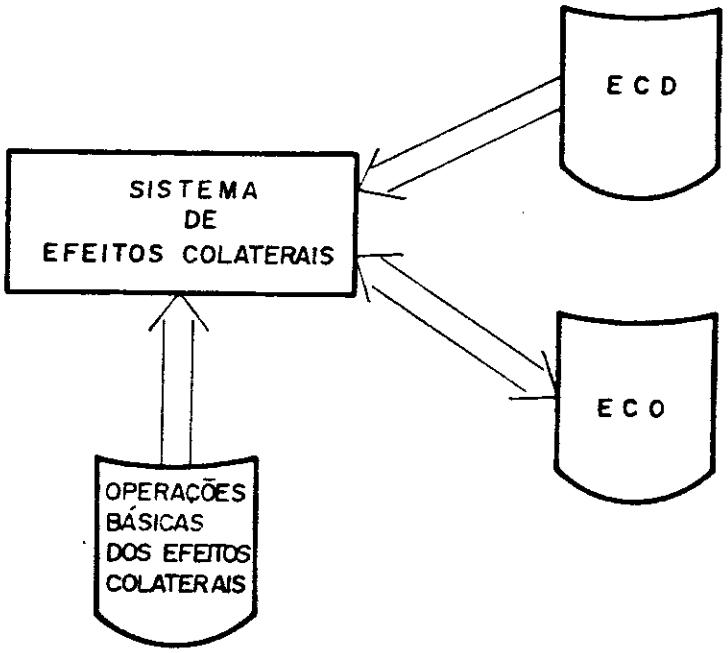


Figura 3.4 : Atuação do Sistema de Efeitos Colaterais

Na 4a. fase, faz-se o mapeamento do esquema conceitual para o esquema interno relacional. O mapeamento é dividido em duas etapas: o mapeamento do ECD e o mapeamento do ECO como mostra a Figura 3.5.

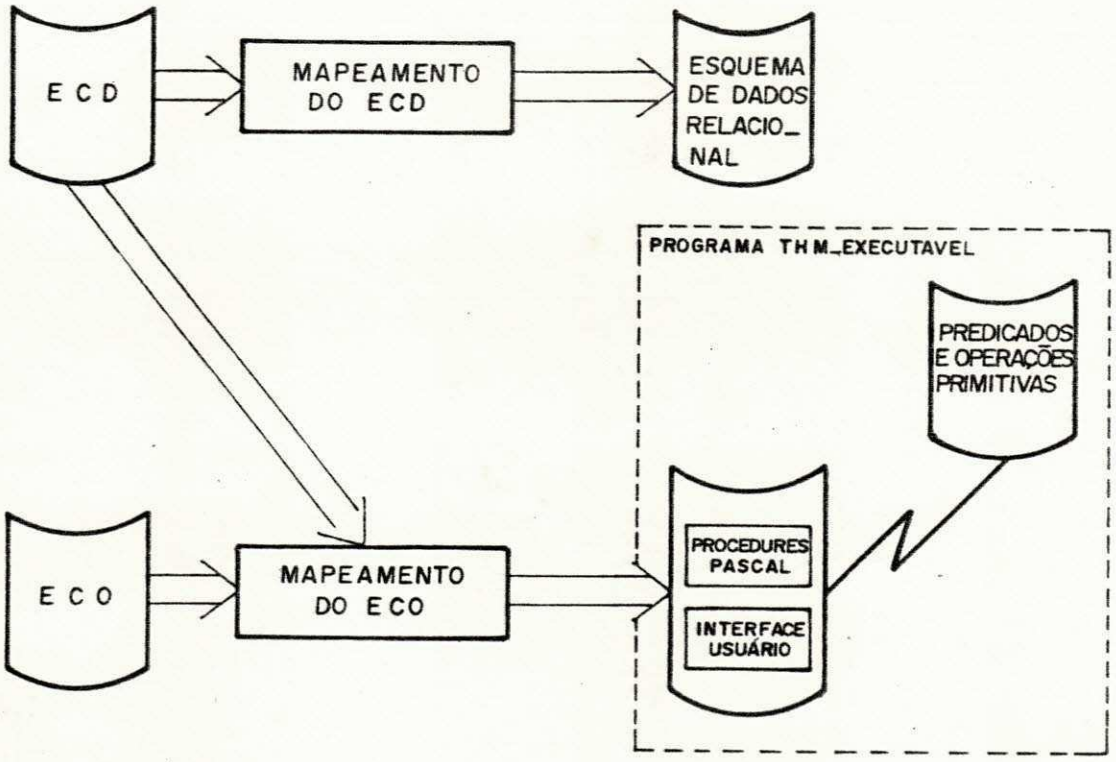


Figura 3.5 : Mapeamento do Esquema Conceitual em Esquema Relacional

No mapeamento do ECD, é aplicado o algoritmo descrito no capítulo 4 que transforma o ECD em um esquema interno relacional equivalente. A saída desta etapa é um arquivo externo contendo a codificação de comandos SQL para criar as relações geradas pelo mapeamento do ECD. Este arquivo deverá ser acessado por um SGDB, que através da linguagem SQL irá criar o esquema relacional definido pelo mapeamento.

No mapeamento do ECO, são aplicados os algorítmos descritos no capítulo 5, onde cada operação-THM definida neste esquema é transformada em uma procedure PASCAL e as operações primitivas que integram as operações-THM são transformadas em chamadas de procedimentos de mesmo nome. Estes procedimentos, por serem primitivos, não são gerados pelo mapeamento, mas pré definidos e codificados em PASCAL com acesso a SQL. Esta ligação com SQL é necessária tendo em vista que as primitivas é que manipulam o banco de dados e, para isto, necessitam de comandos procedurais e de comandos de manipulação de bancos de dados. A codificação destes procedimentos primitivos faz parte de um programa PASCAL que denominaremos THM_EXECUTÁVEL (detalhado na seção 5.1) e é completado pelas procedures geradas pelo mapeamento. O corpo deste programa também é gerado pelo mapeamento e corresponde a uma interface que permite ao usuário ativar as procedures equivalentes às operações do ECO.

A última fase, apresentada na Figura 3.6, diz respeito a operacionalização do sistema, que se processa da seguinte maneira:

1. A interface com SQL permite a criação do banco de dados com os comandos gerados pelo mapeamento do ECO.

2. Compila-se o programa THM-EXECUTÁVEL que contém a diretiva $ I EsqIntOperacoes.Pas para incluir o programa gerado pelo mapeamento do ECO.

3. Deixa-se o módulo objeto do THM-EXECUTÁVEL residente na memória.

4. A interface permite que o módulo fixo do THM-EXECUTÁVEL seja processado.

5. O usuário ativa o THM-EXECUTÁVEL e é apresentada uma tela que permite ao usuário, através das operações-THM previamente definidas no ECO, manipular o banco de dados cujo esquema foi criado em 1.

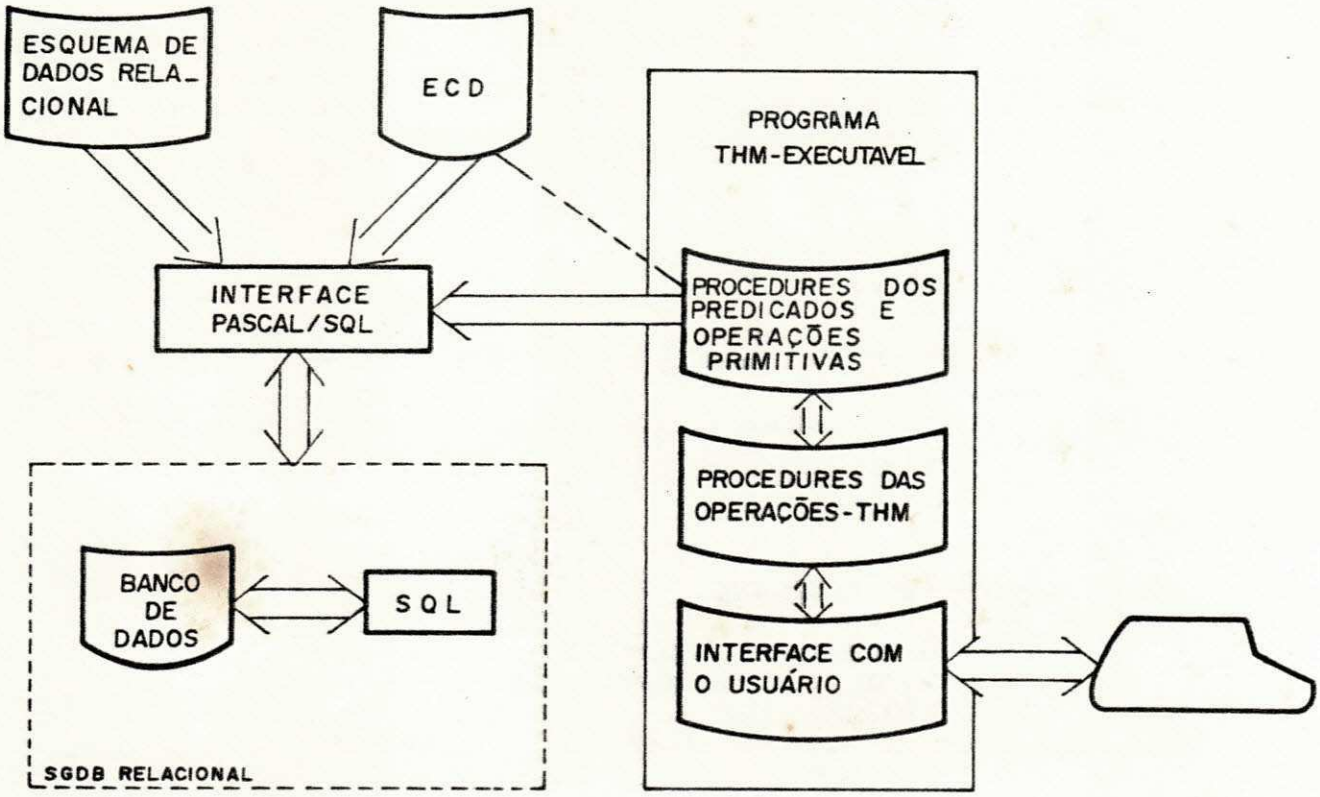


Figura 3.6: Operacionalização do Sistema

4. MAPEAMENTO DO ESQUEMA CONCEITUAL DE DADOS

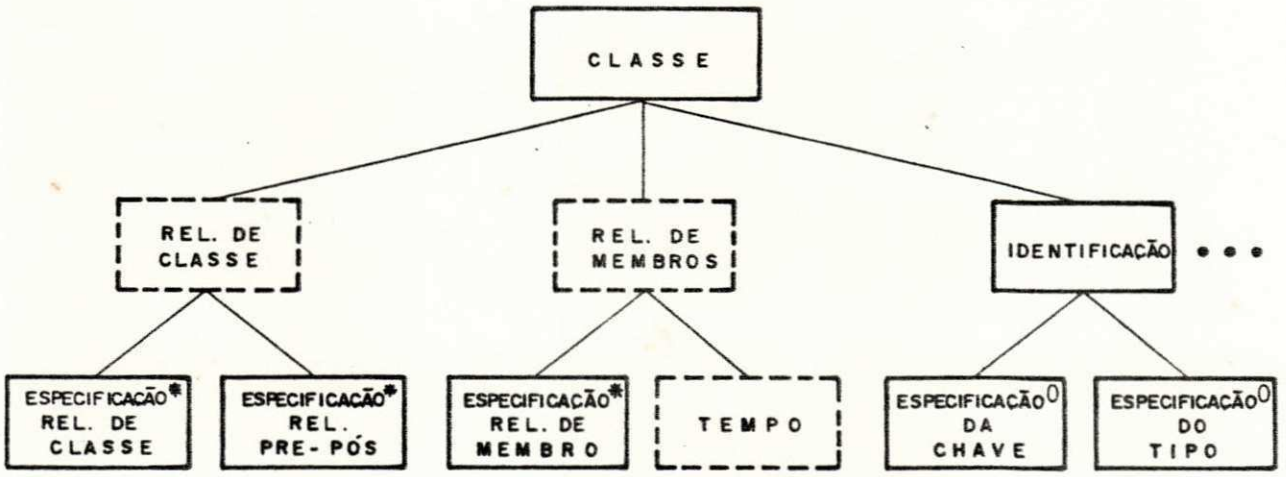
Aplicando-se a metodologia THM descrita em [MS-85, Sc-87], o universo do discurso é modelado e esta modelagem pode ser refinada e formalizada de modo que se torne possível operacionalizar o universo do discurso, mantendo-se a semântica do mesmo. Para isto, é necessário mapear a descrição conceitual até o nível interno, onde é utilizada por um SGDB baseado em um dos modelos tradicionais.

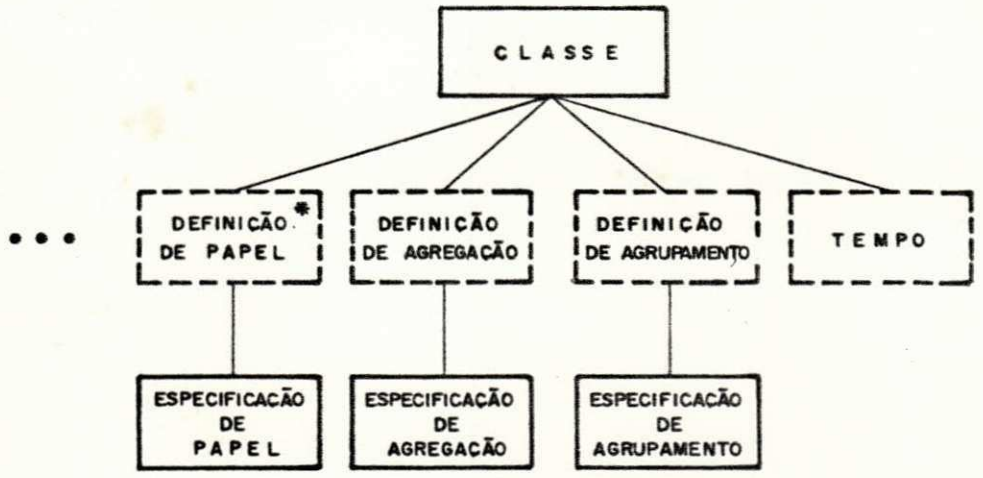
Aqui, o modelo interno considerado é o Modelo Relacional. Para atingi-lo, os conceitos do THM são transformados em conceitos aplicáveis a este modelo. Desta forma, o Esquema Conceitual De Dados (ECD) de cada aplicação é transformado em um esquema relacional.

4.1 Estrutura De Armazenamento Do ECD.

Inicialmente, todos os aspectos estruturais do universo do discurso são codificados na linguagem THM/LDD: nela são definidas todas as classes de uma aplicação e para cada classe são definidos os relacionamentos, os papeis com as generalizações, as agregações e os agrupamentos aplicáveis às entidades da referida classe.

Para darmos uma ideia melhor da estrutura da linguagem, apresentamos a seguir a representação de suas principais estruturas na notação de Jackson [Jc-75]. Maiores detalhes podem ser verificados através da sintaxe descrita no Apendice_A.


CLASSE
REL. DE CLASSE
REL. DE MEMBROS
IDENTIFICAÇÃO
• • •
ESPECIFICAÇÃO* REL. DE CLASSE
ESPECIFICAÇÃO* REL. PRE-PÓS
ESPECIFICAÇÃO* REL. DE MEMBRO
TEMPO
ESPECIFICAÇÃO0 DA CHAVE
ESPECIFICAÇÃO0 DO TIPO


CLASSE
• • •
DEFINIÇÃO* DE PAPEL
DEFINIÇÃO DE AGREGAÇÃO
DEFINIÇÃO DE AGRUPAMENTO
TEMPO
ESPECIFICAÇÃO DE PAPEL
ESPECIFICAÇÃO DE AGREGAÇÃO
ESPECIFICAÇÃO DE AGRUPAMENTO

Com base nesta estrutura, definimos as tabelas que seguem, cuja especificação de seus campos pode ser vista no apêndice-C.

CLASSES (NOME CLASSE, TIPO CLASSE, TIPO ENTIDADE, TEMPO, TEMPO DE VIDA, EVENTO, DELETADA).

RELACIONAMENTOS (NOME RELACIONAMENTO, CAR.MIN., CARD.MAX., CLASSE ORIGEM, RELAC.INVERSO, CARD.MIN.INV., CARD.MAX.INV., CLASSE RELACIONADA, IDENTIF.CHAVE, VAL.ANTER., QUANT.VAL., TIPO RELAC., EVENTO, DELETADO).

PRE-POS (PRECLASSE, POSCLASSE, PRE_POS)

PAPEL (NOME_PAPEL, CLASSE, TIPO, REL.CONJUNTO)

GENERALIZAÇÕES (SUBCLASSE, CLASSE GENERALIZADA, PAPEL)

AGREGAÇÕES (CLASSE COMPONENTE , CLASSE AGREGADA, TIPO)

AGRUPAMENTOS (CLASSE ELEMENTOS, CLASSE GRUPOS, TIPO)

O armazenamento destas tabelas é feito por um programa que interpreta esquemas conceituais de dados descritos em THM/LDD, os transforma nesta estrutura e grava cada tabela em um arquivo externo. Este programa (transformador) foi implementado como projeto da disciplina Compiladores [Fr-87].

Os arquivos gerados pelo transformador são lidos pelo mapeamento que os carrega na memória formando novamente as tabelas. O mapeamento é feito com base na tabela CLASSES que contém a definição de todas as classes do esquema: para cada classe é definida uma relação, cujos atributos são determinados com base nos relacionamentos e na posição hierárquica da referida classe. Este processo corresponde ao algorítmo definido em 4.2.

Durante o processamento do algorítmo, as tabelas vão sendo interpretadas e as relações vão sendo criadas juntamente com seus

atributos, formando duas listas encadeadas contendo respectivamente as relações e os atributos destas. As listas servem de base para a geração dos comandos SQL e têm a estrutura mostrada a seguir, cuja especificação de seus campos pode ser vista no Apêndice-D.

LISTA_RELAÇÕES:

NOME_RELAÇÃO	PONT_ATRIBUTOS	PONT_PROX

LISTA_ATRIBUTOS:

NOME_ATRIBUTO	TP_ATRIBUTO	PONT_PROXIMO

A saída deste mapeamento é um arquivo externo contendo comandos SQL que permitem a criação do esquema relacional equivalente ao ECD.

4.2 Algoritmo Para o Mapeamento do ECD.

O algoritmo para gerar o esquema relacional de dados, se desenvolve em duas etapas:

PRIMEIRA ETAPA:

1. Para cada classe da tabela CLASSES são executados os seguintes passos:
   a) Defina uma relação com o nome da classe.
   b) Se se tratar de uma classe de domínio, defina um atributo com o nome "ID".

c) Se se tratar de uma agregação cujas classes componentes sejam classes de domínio, defina para cada classe componente um atributo com o nome desta. Mas se uma das classes for composta, defina como atributo a identificação desta.

d) Defina para cada relacionamento de membros da classe cuja cardinalidade máxima seja 1 e cujas classes relacionadas não sejam compostas, um atributo com o nome do relacionamento de membros. Se a classe relacionada for composta, defina como atributo a identificação desta.

e) Se a classe for subclasse de uma classe generalizada, defina um atributo com o nome da classe generalizada. Se se tratar de uma classe composta, defina como atributo a identificação desta.

f) Se a classe for uma classe de elementos e os grupos correspondentes forem disjuntos, defina um atributo com o nome da classe de grupos.

g) Se a classe for a classe relacionada em um relacionamento de classe , defina um atributo com o nome do relacionamento de classe.

h) Se forem atribuídos intervalos de tempo a esta classe, defina dois atributos que indiquem o intervalo de tempo (DE e ATE).

2. Defina com auxílio das informações TIPO ENTIDADE da tabela CLASSES, e IDENTIF.CHAVE da tabela RELACIONAMENTOS a chave da relação. Se na relação ocorrerem intervalos de

tempo, os atributos "DE" e "ATE" são acrescentados a chave.

3. Para cada relacionamento n:m crie uma relação com o nome deste relacionamento. Como atributos, coloque as duas classes mencionadas neste relacionamento. Se se tratar de classes compostas, coloque os atributos de identificação das classes. Se a uma destas duas classes estiver agregado um intervalo de tempo, proceda como em (1-h).

4. Para cada relacionamento de membros para o qual na tabela RELACIONAMENTOS o atributo VAL_ANTER foi posto igual a "S" (sim), crie uma relação com o nome deste relacionamento e o prefixo "ANTE". Defina como atributos as classes citadas neste relacionamento. Se se tratar de classes compostas, coloque como atributo a identificação da classe. Além disto, defina os atributos "DE" e "ATE".

5. Crie para cada classe de elementos cujos grupos não sejam disjuntos ou as quais estejam atribuídas intervalos de tempo, uma relação com o nome da classe de elementos e o prefixo "GRP". Os atributos desta relação são a chave da classe de elementos e a chave da classe de grupos.

Em (3) e (5), a chave da relação consiste de todos os atributos, quando a relação não inclui atributos temporais. Caso contrário, o instante "DE" é incluído na chave.

As relações geradas no passo 1.a denominamos de E-Relações por representarem entidades e as geradas no passo 3 denominamos

R-Relações por armazenarem relacionamentos.

SEGUNDA ETAPA: (Opcional);

É dada ao usuário a opção de simplificar o esquema de dados gerado pelo algorítmo. Esta etapa é feita de forma interativa com o usuário, que de acordo com as necessidades de suas aplicações, acata ou não as simplificações sugeridas pelo algorítmo que se baseiam nos seguintes passos.

1. Se desejar, elimine as relações que armazenam classes de domínio, que só contenham o atributo "ID". Este caso acontece quando esta classe de domínio só possue relacionamentos 1:n e 1:1.

2. Cada relacionamento 1:1 é representado duas vezes, cada vez como atributo de uma das E-Relações. Um destes atributos pode ser eliminado ou as duas E-Relações integradas em uma só relação.

3. Para cada relacionamento n:m são geradas 2 R-Relações, uma para cada sentido. Elimine uma.

Este algoritmo garante que não podem ocorrer dependências funcionais entre partes da chave e atributos que não estejam contidos na chave. Assim a chave das relações identificam unicamente uma entidade da classe da qual é derivada o relacionamento, e os atributos são formados apenas pelos relacionamentos n:1 e 1:1 tirados desta classe. Para os relacionamentos problemáticos n:m , são definidas no passo (3)

relações próprias nas quais não podem ocorrer dependências funcionais pela escolha das chaves. Como os relacionamentos n:m são isolados em R-Relações distintas, não podem ocorrer dependências multivaloradas não-triviais, portanto o esquema interno está em 4a. forma normal [Sc-84].

4.3 Aplicação do Algoritmo

Exemplificamos o algoritmo fazendo uma aplicação para o sistema de "Registro de Carros", especificado informalmente em [Gr-82]. Inicialmente apresentamos a representação gráfica do ECD do sistema, seguido da descrição deste em THM/LDD e do armazenamento nas tabelas descritas anteriormente, por último apresentamos o esquema relacional gerado pelo algoritmo.

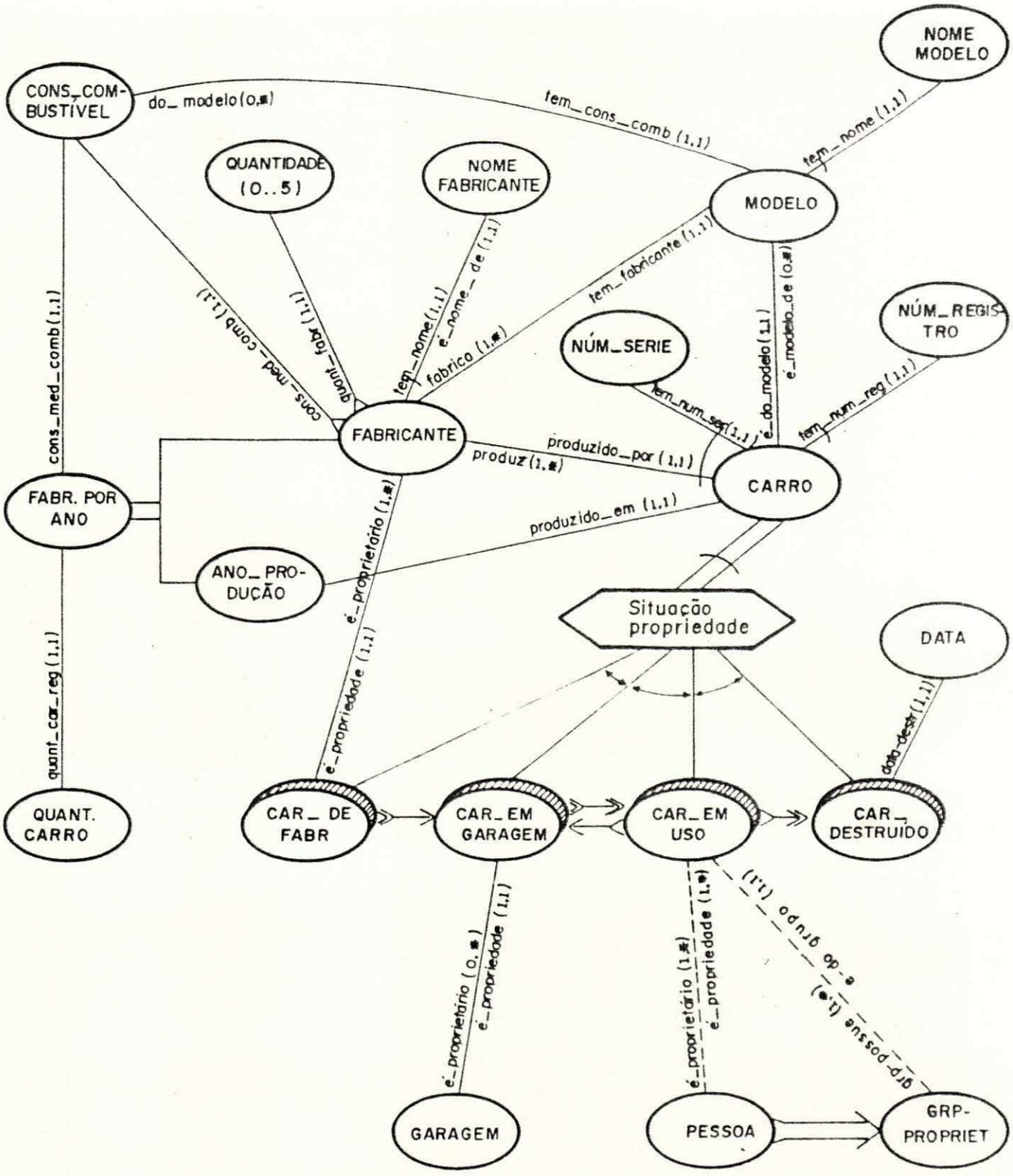


Figura-4.1 - Representação Gráfica Do ECD Do Sistema "Registro de Carros"

4.3.1 Descrição do ECD em THM/LDD

```
class CARRO
    member relationships
        tem_num_reg              : NÚM_REGISTRO (1,1)
        é_do_modelo              : MODELO (1,1)
        produzido_em             : ANO_PRODUÇÃO (1,1)
        produzido_por            : FABRICANTE  (1,1)
        tem_num_ser              : NÚM_SÉRIE (1,1)
    keys are produzido_por, tem_num_ser, tem_num_reg
    with roles situação_propriedade gives subclasses
        CAR_DE_FABR,CAR_EM_GARAGEM,CAR_EM_USO,CAR_DESTRUÍDO
        parameters covering, disjunctive


class FABRICANTE
    class relationships
        cons_med_comb            : CONS_CONBUSTIVEL (1,1)
        quant_fabr               : QUANTIDADE (1,1)
    member relationships
        tem_nome                 : NOME_FABR (1,1)
        produz                   : CARRO (1,*)
        é_proprietário           : CAR_DE_FABR (1,*)
        fabrica                  : MODELO (1,*)
    Keys are tem_nome


class  CAR_DE_FABR
    class_relationships
        post_class     : CAR_EM_GARAGEM exclusive
    member_relationships
        e_propriedade  : FABRICANTE(1,1)
    keys are inherited
    parameters with time and life time 3 year


class CAR_EM_GARAGEM
    class_relationships
        pre_class      : CAR_DE_FABR
        pre_class      : CAR_EM_USO
        post_class     : CAR_EM_USO exclusive
    member_relationships
        é_propriedade  : GARAGEM (1,1)
    keys are inherited
    parameters with time and life time 2 year


class  GARAGEM
    member_relationships
        é_proprietário :  CAR_EM_GARAGEM (0,*)
    type set of string
```

```
class  CAR_EM _USO
    class relationships
         pre_class : CAR_EM_GARAGEM exclusive
         post_class: CAR_DESTRUÍDO, CAR_EM_GARAGEM
    member relationships
         é_propriedade_de  : PESSOA (1,*) with 2 old values
         é_do_grupo        : GRP_PROPRIETÁRIO (1,1)
    keys are inherited
    with time and life time 3 year

class CAR_DESTRUÍDO
    class_relationships
         pre_class : CAR_EM _USO exclusive
    member_relationships
         data_destruição   : DATA (1,1)
    keys are inherited
    with time and life time 3 year

class  PESSOA
    member relationships
         é_proprietário      : CAR_EM USO (1,*)
    type string


class GRP_PROPRIETÁRIO
    member relationships
         grp_possue  :  CAR_EM_USO  ( 1,*) with 2 old values
    type  integer
    grouping of PESSOA using é_proprietário


class MODELO
    member_relationships
         tem_nome        : NOME_MODELO (1,1)
         tem_cons_comb   : CONS_COMBUSTÍVEL (1,1)
         tem_fabricante  : FABRICANTE (1,1)
         é_modelo_de     : CARRO (0,*)
    keys are tem_nome


class FABR_POR_ANO
    member_relationships
         cons_med_comb         : CONS_COMBUSTÍVEL (1,1)
         quant_car_reg         : QUANT_CARRO (1,1)
    Keys are inherited
    agregation of FABRICANTE, ANO_PRODUÇAO


class CONS_COMBUSTÍVEL
    member_relationships
         do_modelo     : MODELO (0,*)
    type set of integer
```

```
class NÚM_REGISTRO
    type integer

class ANO_PRODUÇÃO
    type set of integer

class DATA
    type array [1..3] of integer
    format 99/99/99

class NÚM_SÉRIE
    type integer

class NOME_MODELO
    type set of string

class  NOME_FABR
    type set of string

class QUANTIDADE
    type (0..5) of integer

class QUANT_CARRO
    type integer
```

4.3.2 Armazenamento do ECD Em Tabelas.
(Descrição no Apêndice-C)

CLASSES

NOME CLASSE	TIPO CLASSE	TIPO ENTIDADE	TEMPO	TEMPO VIDA	EVENTO	DELETADA
CARRO	C	M	N	-	N	N
FABRICANTE	C	M	N	-	N	N
CAR_DE_FABR	C	G	S	3 anos	N	N
CAR_EM_GARAGEM	C	G	S	2 anos	N	N
CAR_EM_USO	C	G	S	3 anos	N	N
PESSOA	S	S	N	-	N	S
GRP_PROPRIETÁRIO	S	I	N	-	N	N
CAR_DESTRUÍDO	C	G	S	3 anos	N	N
DATA	S	I	N	-	N	S
NÚM_REGISTRO	S	I	N	-	N	S
NÚM_SÉRIE	S	I	N	-	N	S
NOME_MODELO	D	S	N	-	N	S
NOME_FABR	D	S	N	-	N	S
MODELO	C	M	N	-	N	N
CONS_COMBUSTÍVEL	S	I	N	-	N	N
QUANTIDADE	S	I	N	-	N	N
ANO_PRODUÇÃO	D	I	N	-	N	S
FABR_POR_ANO	C	A	N	-	N	N
QUANT_CARRO	S	I	N	-	N	S
GARAGEM	D	I	N	-	N	S

RELACIONAMENTOS

NOME RELACIONAMENTO	CARD MIN.	CARD MAX.	CLASSE ORIGEM	RELAC. INVERSO	CARD MIN. INV.	CARD MAX. INV.	CLASSE RELACIONADA	IDEN-TIF. CHAVE	VAL. ANTER.	QUANT. VAL	TIPO REL	EVENTO	DELE-TADO
tem_num_reg	1	1	CARRO	inv_tem_num_reg	1	1	NUM_REGISTRO	1	n	-	m	n	n
eh_do_modelo	1	1	CARRO	eh_modelo_de	0	*	MODELO	0	n	-	m	n	n
produzido_em	1	1	CARRO	inv_produzido_em	0	*	ANO_PRODUÇÃO	2	n	-	m	n	n
produzido_por	1	1	CARRO	produz	1	*	FABRICANTE	2	n	-	m	n	n
tem_num_ser	1	1	CARRO	inv_tem_num_ser	1	1	NUM_SERIE	0	n	-	m	n	n
tem_nome	1	1	FABRICANTE	eh_nome_de	1	1	NOME_FABR	1	n	-	m	n	n
produz	1	*	FABRICANTE	produzido	1	1	CARRO	0	n	-	m	n	n
é_proprietário	1	*	FABRICANTE	é-propriedade	1	1	CAR_DE_FABR	0	n	-	m	n	n
fabrica	1	*	FABRICANTE	tem_fabr	1	1	MODELO	0	n	-	m	n	n
quant_fabr	1	1	FABRICANTE	inv_quant_fabr	1	1	QUANTIDADE	0	n	-	c	n	n
é_propriedade	1	1	CAR_EM_GARAGEM	é_proprietário	1	*	GARAGEM	0	n	-	m	n	n
é_propriedade	1	*	CAR_EM_USO	é_proprietário	1	*	PESSOA	0	s	2	m	n	s
é_do_grupo	1	1	CAR_EM_USO	grp_possue	1	*	GRP_PROPRIETARIO	0	s	2	m	n	s
é_proprietário	1	*	PESSOA	e_propriedade	1	*	CAR_EM_USO	0	n	-	m	n	n
grp_possue	1	*	GRP_PROPRIETÁRIO	e_do_grupo	1	1	CAR_EM_USO	0	s	2	m	n	n
data_destr.	1	1	CAR_DESTRUÍDO	inv_data_destr.	0	*	DATA	0	n	-	m	n	n
tem_nome	1	1	MODELO	inv_tem_nome	1	1	NOME_MODELO	1	n	-	m	n	n
tem_cons_comb	1	1	MODELO	do_modelo	0	*	CONS-COMBUSTIVEL	0	n	-	m	n	n
tem_fabricante	1	1	MODELO	fabrica	1	*	FABRICANTE	0	n	-	m	n	n
é_modelo_de	0	*	MODELO	é_do_modelo	1	1	CARRO	0	n	-	m	n	n
do_modelo	0	*	CONS_COMBUSTIVEL	tem_cons_comb	1	1	MODELO	0	n	-	m	n	n
cons_med_comb	1	1	FABR_POR_ANO	inv_cons_med_comb	0	*	CONS_COMBUSTIVEL	0	n	-	m	n	n
quant_car_reg	1	1	FABR_POR_ANO	inv_quant_car_reg	0	*	QUANT_CARRRO	0	n	-	m	n	n
é_propriedade	1	1	CAR_DE_FABR	é_proprietário	1	*	FABRICANTE	0	n	-	m	n	n
cons_med_comb	1	1	FABRICANTE	inv_cons_max	0	*	CONS_COMBUSTIVEL	0	n	-	c	n	n

RELACIONAMENTOS PRE_POS

PRE-CLASSE	POS-CLASSE	PRE_POS
CAR_DE_FABR	CAR_EM_GARAGEM	es
CAR_EM_GARAGEM	CAR_EM_USO	ee
CAR_EM_USO	CAR_DESTRUÍDO	se
CAR_EM_USO	CAR_EM_GARAGEM	ss

PAPÉIS

NOME PAPEL	CLASSE	TIPO	REL. CONJUNTO
situacao_propriedade	CARRO	E	B

GENERALIZAÇÕES

SUBCLASSE	CLASSE GENERALIZADA	PAPEL
CAR_DE_FABR	CARRO	situacao_propriedade
CAR_EM_GARAGEM	CARRO	situacao_propriedade
CAR_EM_USO	CARRO	situacao_propriedade
CAR_DESTRUÍDO	CARRO	situacao_propriedade

AGREGAÇÕES

CLASSE COMPONENTE	CLASSE AGREGADA	TIPO
FABRICANTE	FABR_POR_ANO	E
ANO_PRODUÇÃO	FABR_POR_ANO	E

AGRUPAMENTOS

CLASSE ELEMENTOS	CLASSE GRUPO	TIPO
PESSOA	GRP_PESSOA	E

4.3.3 Esquema Relacional Gerado Pelo Algoritmo

O esquema relacional gerado pela execução da 1a. etapa do algoritmo é o que segue:

CARRO (tem_num_reg, tem_num_ser, produzido_em,
tem_nome(fabricante), tem_nome(modelo))

FABRICANTE (tem_nome)

CAR_DE_FABR (tem_num_reg, de, até, tem_nome(fabricante))

CAR_EM GARAGEM (é_propriedade_de(garagem), tem_num_reg, de, até)

GARAGEM (id)

CAR_EM_USO (é_do grupo, tem_num_reg, de, até)

PESSOA (id)

GRP_PROPRIETÁRIO (id)

CAR_DESTRUÍDO (tem_num_reg, de, até, data_destruição)

DATA (id)

NÚM_REGISTRO (id)

NÚM_SÉRIE (id)

NOME_MODELO (id)

NOME_FABR (id)

MODELO (tem_nome(modelo), tem_cons_comb, tem_nome(fabricante))

CONS_COMBUSTÍVEL (id, cons_med_comb)

QUANTIDADE (id, quant_fabr)

ANO_PRODUÇÃO (id)

FABR_POR_ANO (tem_nome(fabricante), ano_produção, cons_med_comb,
quant_car_reg)

QUANT_CARRO (id)

é_propriedade (tem_num_reg, PESSOA, de, até)

é_proprietário (PESSOA, tem_num_reg, de, até)

ante_é_propriedade (tem_num_reg, PESSOA, de, até)

ante_é_do_grupo (tem_num_reg, GRP_PROPRIETÁRIO, de, até)

ante_grp_possue (GRP_PROPRIETÁRIO, tem_num_reg, de, até)

GRP_PESSOA (PESSOA, GRP_PROPRIETÁRIO)

Como os passos da 2a. etapa são opcionais, um dos possíveis esquema final, gerado pelo algoritmo, pode ser o que segue:

CARRO (tem_num_reg, tem_num_ser, produzido_em,
tem_nome(fabricante), tem_nome(modelo))

FABRICANTE (tem_nome)

CAR_DE_FABR (tem_num_reg, de, até, tem_nome(fabricante))

CAR_EM GARAGEM (é_propriedade_de(garagem), tem_num_reg, de, até)

CAR_EM_USO (é_do grupo, tem_num_reg, de, até)

CAR_DESTRUÍDO (tem_num_reg, de, até, data_destruição)

MODELO (tem_nome(modelo), tem_cons_comb, tem_nome(fabricante))

CONS_COMBUSTÍVEL (id, cons_med_comb)

QUANTIDADE (id, quant_fabr)

FABR_POR_ANO (tem_nome(fabricante), ano_produção, cons_med_comb,
quant_car_reg)

é_proprietário (PESSOA, tem_num_reg, de, até)

ante_é_propriedade (tem_num_reg, PESSOA, de, até)

ante_grp_possue (GRP_PROPRIETÁRIO, tem_num_reg, de, até)

GRP_PESSOA (PESSOA, GRP_PROPRIETÁRIO)

5. MAPEAMENTO DO ESQUEMA CONCEITUAL DE OPERAÇÕES:

O Esquema Conceitual de Operações (ECO) corresponde às ocorrências do mundo real que causam modificações no banco de dados. O THM dispõe de três ferramentas para registrar estas ocorrências: os eventos que detectam ocorrências no banco de dados ou fatores relacionados com o tempo; as operações triggers que são ativadas automaticamente em decorrência de eventos; e as operações de manipulação do banco de dados.

Neste trabalho nos limitamos a mapear as operações de manipulação do banco de dados, e para evitar ambiguidade com as operações primitivas as chamaremos de operações-THM. Os eventos e triggers têm estrutura semelhante e fazem parte de outro trabalho [Tr-87].

As operações_THM são codificadas em THM/LMD, são definidas em nível conceitual e consequentemente obedecem à semântica definida no ECD. E para que a semântica seja mantida, sua própria estrutura faz com que a operação só seja executada se a semântica e a integridade dos dados forem mantidas. Isto é verificado pelas pré e pós-condições na seguinte forma :

"pré-condições ---> corpo da operação ---> pós-condições",

apresentando a seguinte estrutura:

```
operation nome-operação
    (input-parameters
        lista dos parâmetros de entrada )
    (output-parameters
        lista dos parâmetros de saída)
    (pre-conditions
        lista das pré-condições necessárias
        para a operação poder ser executada )
     body
        lista de declarações (operações
        primitivas e/ou declarações call)    em
        estrutura conjuntiva, disjuntiva ou iterativa.
    (pos-conditions
        lista dos predicados que verificam se
        a operação não afetou a integridade
        do banco de dados )
```

onde os parâmetros de entrada são entidades fornecidas pelo usuário ou obtidas do Banco de Dados (BD), os de saída são novas entidades criadas pela operação. As pré-condições são uma conjunção de predicados que são testados quando a operação é ativada e cada predicado tem uma opção **otherwise** que especifica a ação a ser tomada caso o predicado falhe:

warning - é dada uma advertência e a execução prossegue sem causar problemas;

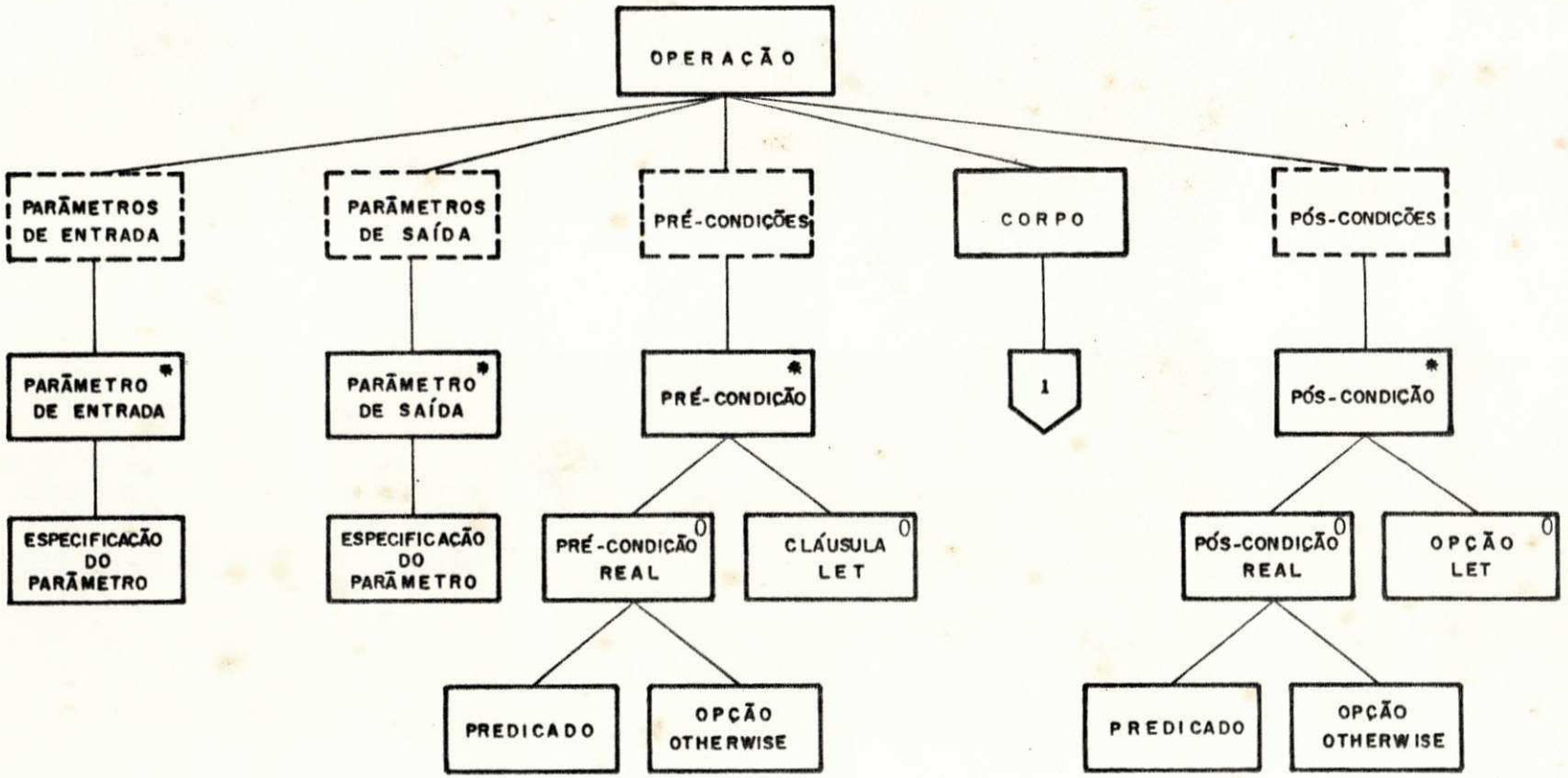
cancel - a operação será cancelada;

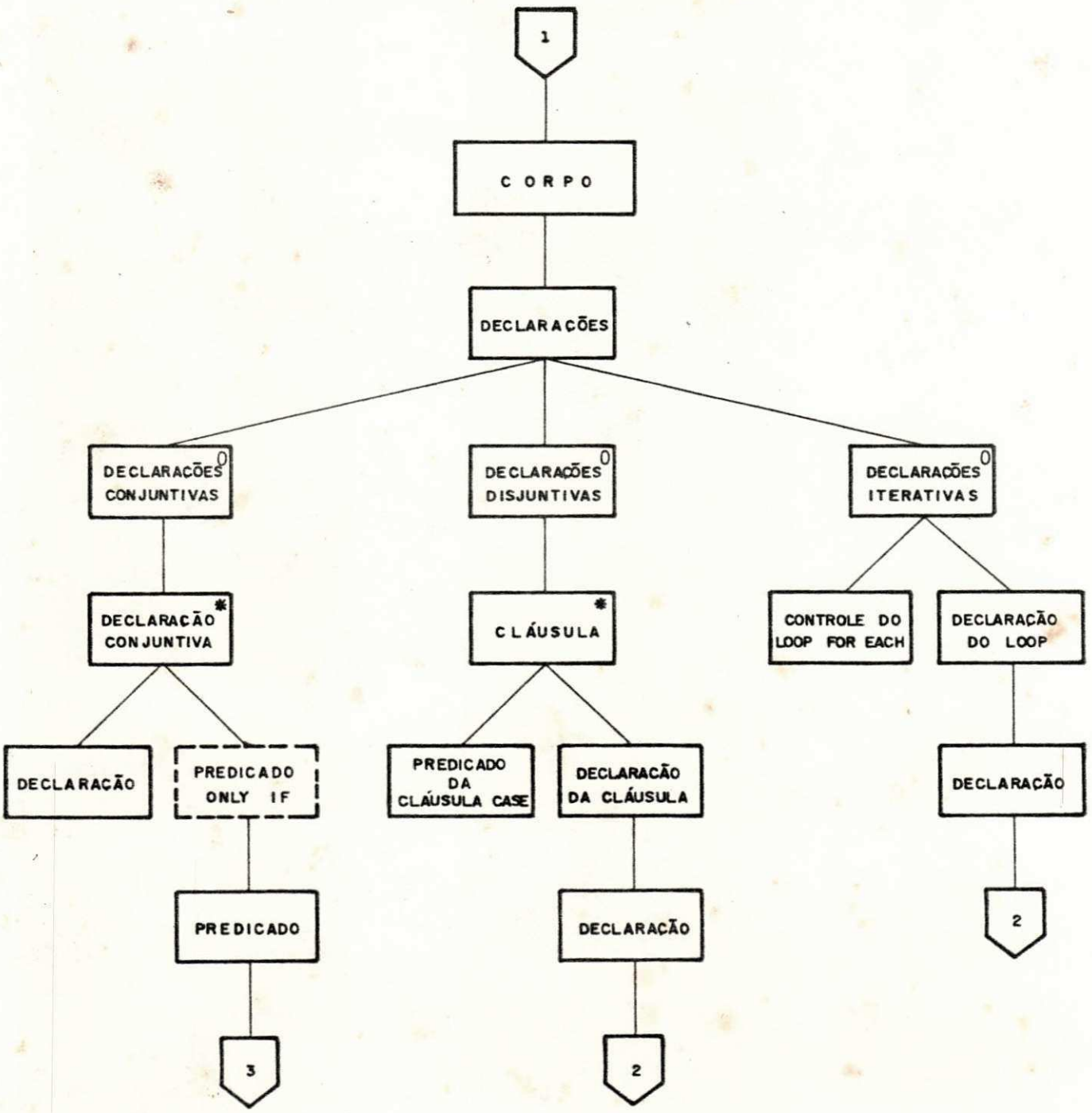
error - o erro exige que a operação e todo processo que a envolve seja cancelado;

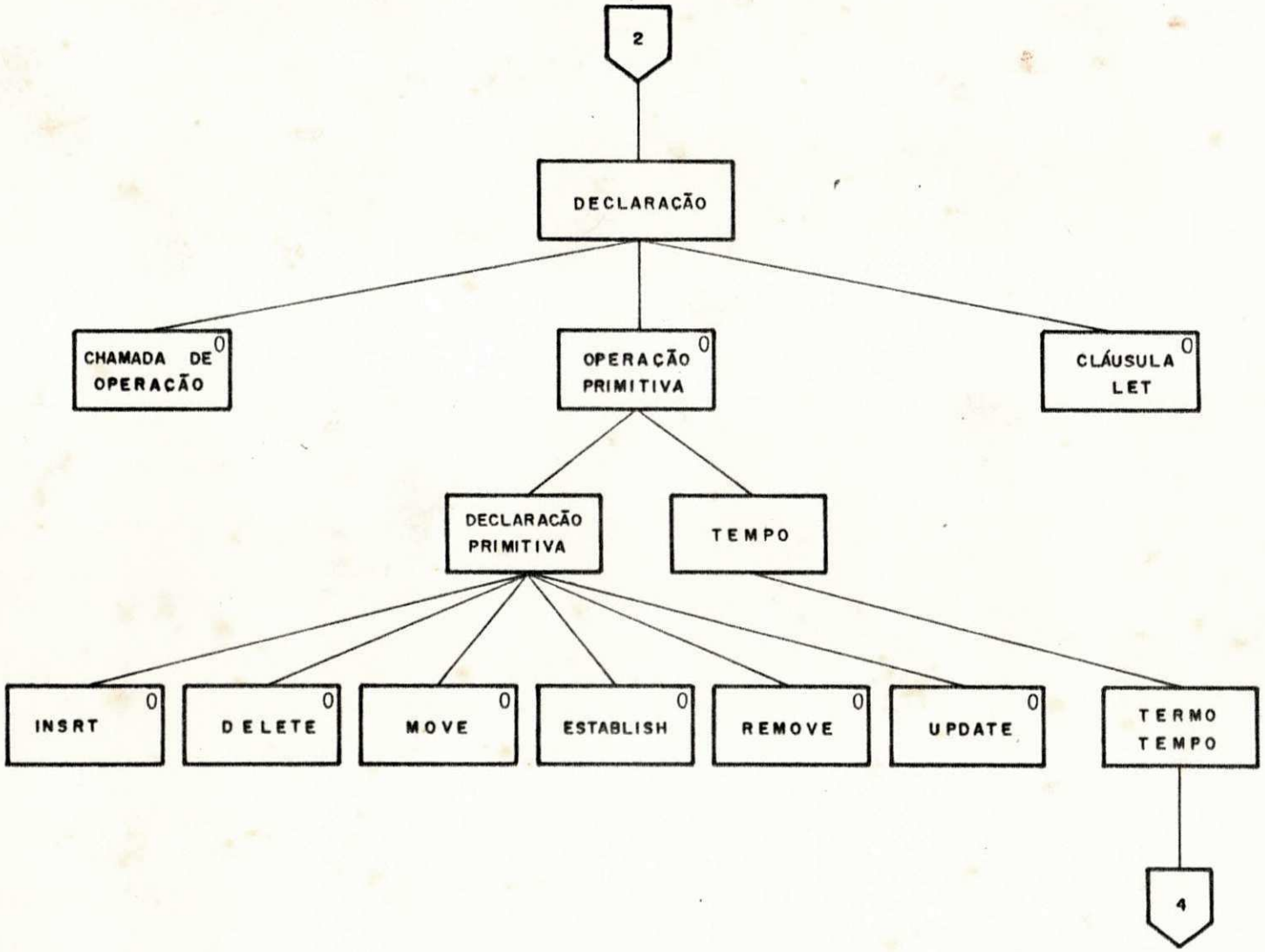
O corpo da operação é formado por um conjunto de declarações submetidas a comandos de controle que determinam se as declarações devem ser executadas de forma conjuntiva, disjuntiva ou iterativa. As pós-condições reforçam a integridade, verificando se depois da execução o banco de dados continuará consistente.

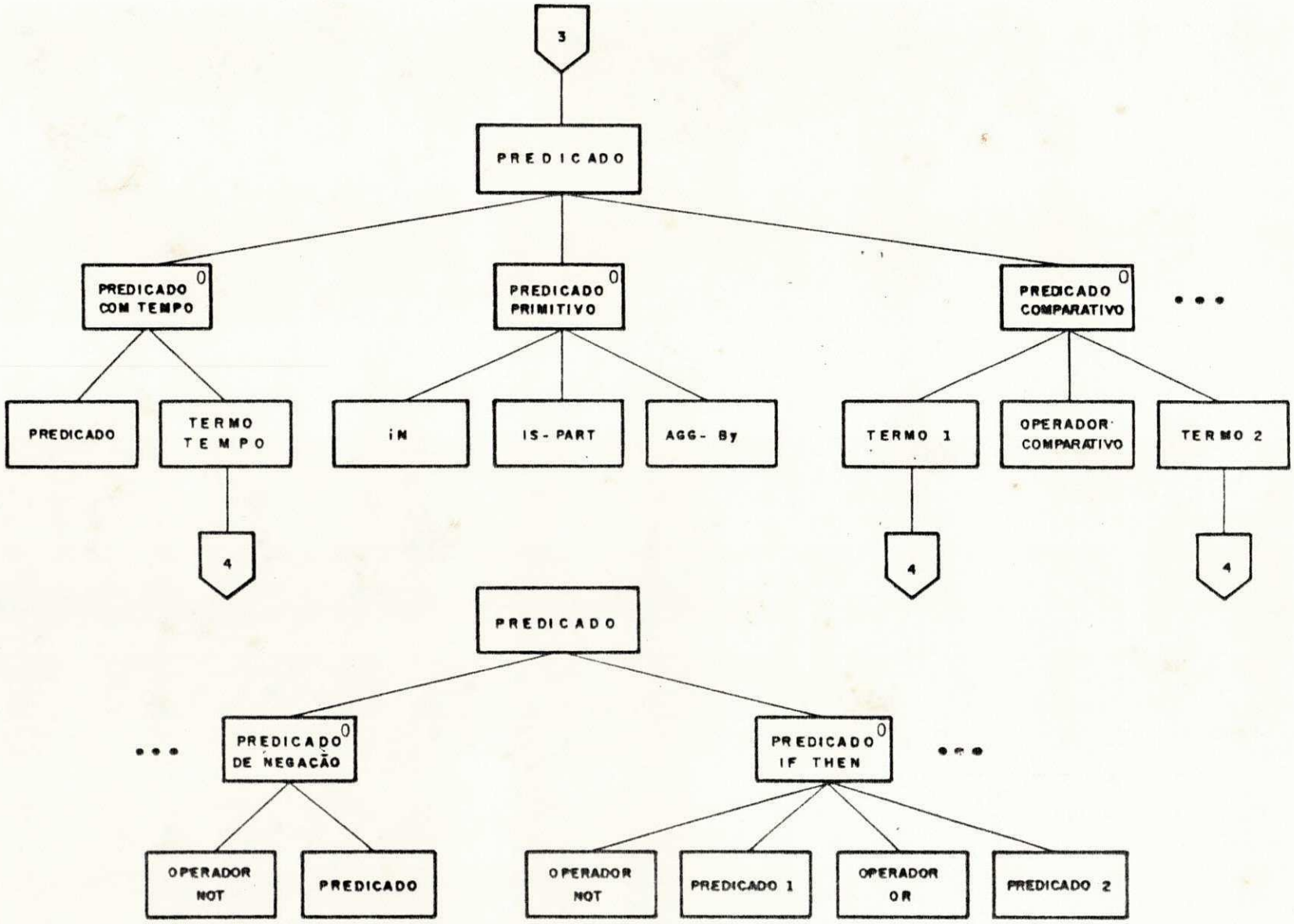
5.1. Estrutura De Armazenamento Do ECO.

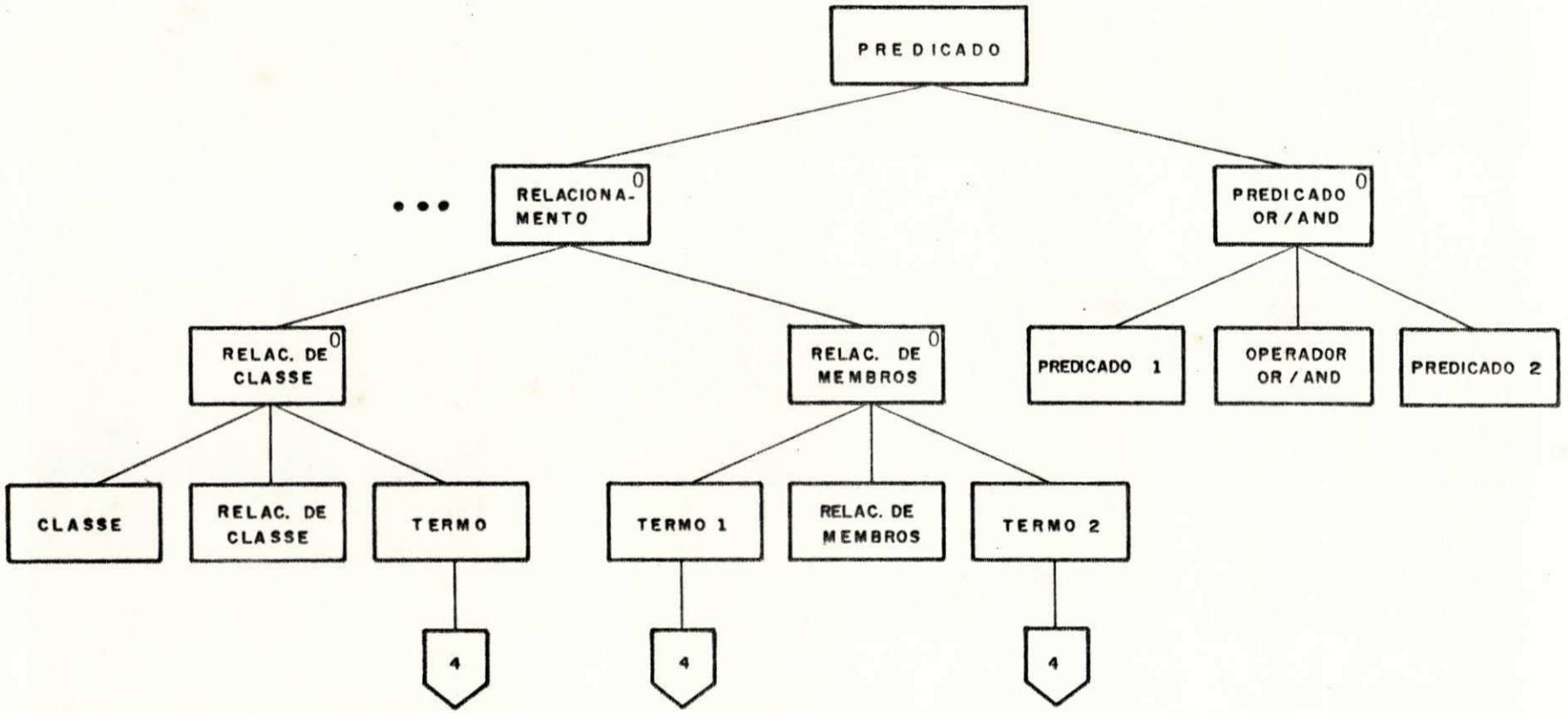
Para o mapeamento do ECO, além da descrição das operações que compõem este esquema, precisamos também da definição do ECD, portanto as tabelas equivalentes ao ECD, definidas anteriormente, farão parte da entrada para este mapeamento. Quanto ao ECO, é representado por um conjunto de tabelas baseado na estrutura da linguagem THM/LMD, a qual representamos a seguir na notação de Jackson [Jc-75]. Maiores detalhes sobre a linguagem, podem ser verificados através da sintaxe descrita no Apêndice-B.

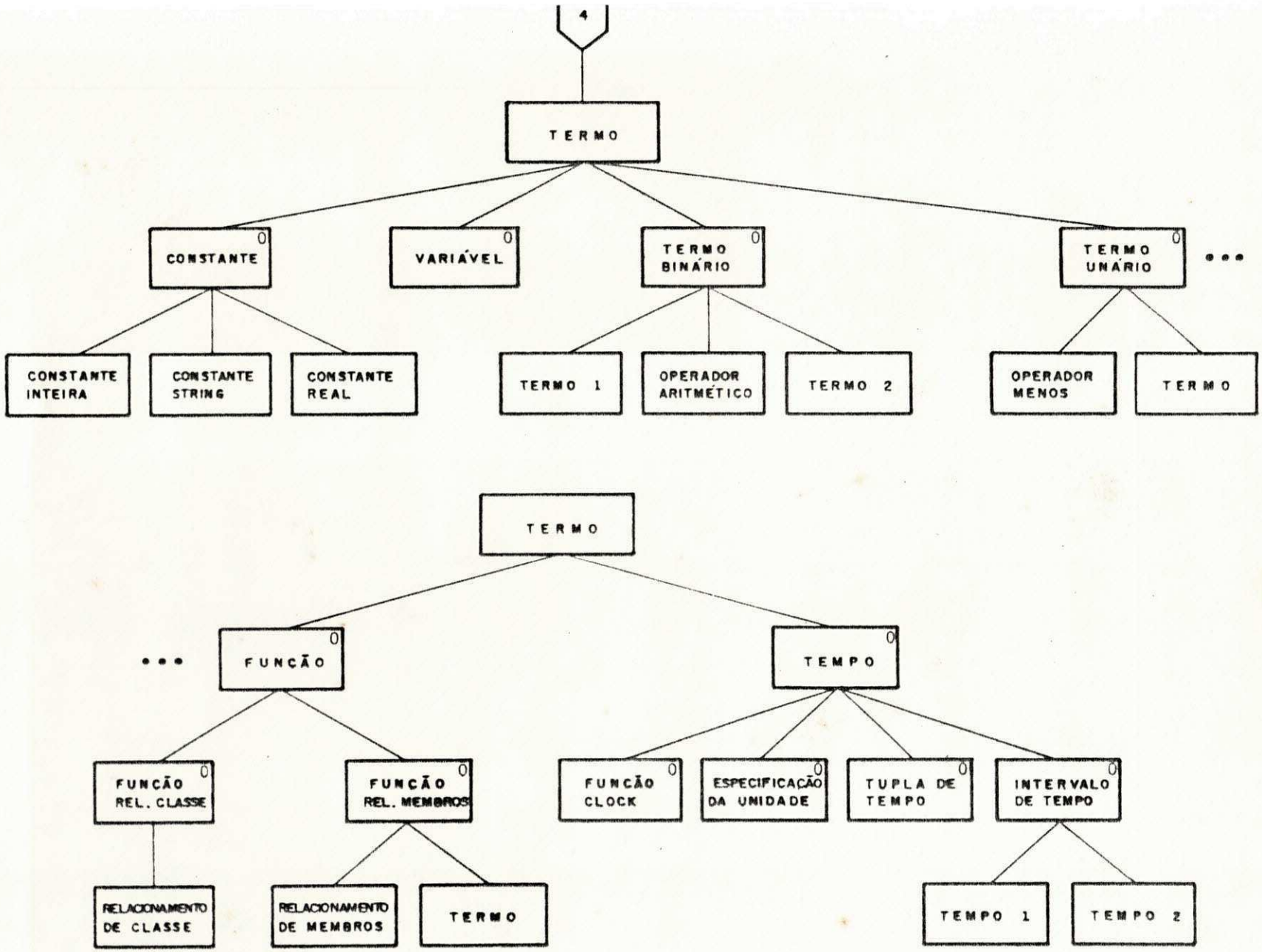

OPERAÇÃO
PARÂMETROS DE ENTRADA
PARÂMETROS DE SAÍDA
PRÉ-CONDIÇÕES
CORPO
PÓS-CONDIÇÕES
PARÂMETRO DE ENTRADA
*
PARÂMETRO DE SAÍDA
*
PRÉ-CONDIÇÃO
*
1
PÓS-CONDIÇÃO
*
ESPECIFICAÇÃO DO PARÂMETRO
ESPECIFICAÇÃO DO PARÂMETRO
PRÉ-CONDIÇÃO REAL
0
CLÁUSULA LET
0
PÓS-CONDIÇÃO REAL
0
OPÇÃO LET
0
PREDICADO
OPÇÃO OTHERWISE
PREDICADO
OPÇÃO OTHERWISE


1
CORPO
DECLARAÇÕES
DECLARAÇÕES CONJUNTIVAS
0
DECLARAÇÕES DISJUNTIVAS
0
DECLARAÇÕES ITERATIVAS
0
DECLARAÇÃO CONJUNTIVA
*
CLÁUSULA
*
CONTROLE DO LOOP FOR EACH
DECLARAÇÃO DO LOOP
DECLARAÇÃO
PREDICADO ONLY IF
PREDICADO DA CLÁUSULA CASE
DECLARAÇÃO DA CLÁUSULA
DECLARAÇÃO
PREDICADO
DECLARAÇÃO
2
3
2


2
DECLARAÇÃO
CHAMADA DE OPERAÇÃO
0
OPERAÇÃO PRIMITIVA
0
CLÁUSULA LET
0
DECLARAÇÃO PRIMITIVA
TEMPO
INSRT
0
DELETE
0
MOVE
0
ESTABLISH
0
REMOVE
0
UPDATE
0
TERMO TEMPO
4


3
PREDICADO
PREDICADO COM TEMPO
PREDICADO PRIMITIVO
PREDICADO COMPARATIVO
PREDICADO
TERMO TEMPO
IN
IS-PART
AGG-By
TERMO 1
OPERADOR COMPARATIVO
TERMO 2
4
4
4
PREDICADO
PREDICADO DE NEGACÃO
PREDICADO IF THEN
OPERADOR NOT
PREDICADO
OPERADOR NOT
PREDICADO 1
OPERADOR OR
PREDICADO 2


PREDICADO
RELACIONA-MENTO
0
PREDICADO OR/AND
0
RELAC. DE CLASSE
0
RELAC. DE MEMBROS
0
PREDICADO 1
OPERADOR OR/AND
PREDICADO 2
CLASSE
RELAC. DE CLASSE
TERMO
TERMO 1
RELAC. DE MEMBROS
TERMO 2
4
4
4


4
TERMO
CONSTANTE
VARIÁVEL
TERMO BINÁRIO
TERMO UNÁRIO
CONSTANTE INTEIRA
CONSTANTE STRING
CONSTANTE REAL
TERMO 1
OPERADOR ARITMÉTICO
TERMO 2
OPERADOR MENOS
TERMO
TERMO
FUNÇÃO
TEMPO
FUNÇÃO REL. CLASSE
FUNÇÃO REL. MEMBROS
FUNÇÃO CLOCK
ESPECIFICAÇÃO DA UNIDADE
TUPLA DE TEMPO
INTERVALO DE TEMPO
RELACIONAMENTO DE CLASSE
RELACIONAMENTO DE MEMBROS
TERMO
TEMPO 1
TEMPO 2

Com base nesta estrutura definimos as seguintes tabelas, cuja especificação de seus campos pode ser vista no apêndice-E.

OPERAÇÕES : (NOME_OPERAÇÃO, PONT_PARAM (entrada), PONT_PARAM (saída), PONT_PRÉ_CONDIÇÕES, TP_CORPO,PONT_TP_CORPO)

PARAM_OPERAÇÕES (POS, PONT_VARIÁVEL (nome_param), PONT_CLASSE, ORIGEM_OU_DESTINO, PONT_CHAVES, TP_DEFAULT, PONT_TP_DEFAULT, PONT_PROX)

PRÉ-CONDIÇÕES: (POS, TP_PRE_COND, PONT_TP_PRE_COND, OTHERWISE, PONT_MENSAGEM, PONT_PROX)

CONJUNTIVAS (POS, TP_DECL, PONT_TP_DECL, ONLY_IF, PONT_PREDICADO, PONT_PROXIMO).

DISJUNTIVAS (POS, PONT_PREDICADO, TP_DECL, PONT_TP_DECL, PROX)

ITERATIVAS (POS, VAR_LIVRE, PONT_COND_ITER, PONT_CONJUNTIVAS)

COND_ITERATIVA (POS, TP_COND, X_in, ENTD_GRUPO, PONT_PREDICADO)

PREDICADOS (POS, TP_PREDICADO, SIMB_PREDICATIVO, PONT_TERMOS, AT_TIME, PONT_TEMPO, NOT, OR_AND, PONT_PROX_OR_AND)

CLÁUSULAS_ LET (POS, PONT_VARIAVEL, PONT_TERMO)

IF_THEN (POS, PONT_PREDICADO1, PONT_PREDICADO2)

FUNÇÕES (POS, TP_FUNÇÃO, SIM_FUNCIONAL, PONT-TERMOS)

PARAM_ATUAIS (POS, ENTR/SAIDA, PONT-TERMO, PONT_PROXIMO)

OPERAÇÕES PRIMITIVAS (POS, OPER_PRIMITIVA, SIMB_AÇÃO, PONT_TERMO, AT TIME, PONT_TEMPO).

OPERAÇÕES_CALL (POS, NOME_OPER, PONT_PARAM_ATUAIS)

CHAVES (POS, NOME_CHAVE, PONT_PROX)

MENSAGENS (POS, LINHA_MENSAGEM, PONT_PROX).

TEMPO (POS, TP_TEMPO , PONT_TP_TEMPO)

CLOCK (POS, UNIDADE , OPER_ARITM, VALOR)

UNIDADE_EXPLICITA (POS, UNIDADE, VALOR)

INTERVALO_TEMPO (POS, PONT_TEMPO_INICIAL, PONT_TEMPO_FINAL)

TUPLA_TEMPO (POS, ANO, MES, DIA, HORA, MINUTO, SEGUNDO)

TERMO_BINARIO (POS, TIPO_TERMO, PONT_TERMO, PONT_PROXIMO)

CLASSES (POS, NOME_CLASSE)

ENTIDADES (POS, NOME_ENTD, PONT_PROX_AGR)

VARIAVEIS (POS, NOME_VAR)

CONST_INTEIRAS (POS, VALOR)

CONST_STRING (POS, VALOR)

O preenchimento destas tabelas é feito por um programa que interpreta esquemas conceituais de operações descritos em THM/LMD, os transforma nesta estrutura e grava cada tabela em um arquivo externo. Este programa (transformador) também foi implementado como projeto da disciplina Compiladores [Ab-87].

Os arquivos gerados pelo transformador são lidos pelo mapeamento de operações que os carrega na memória formando novamente as tabelas. De acordo com a estrutura das tabelas, o armazenamento de cada operação forma uma árvore e o mapeamento de cada operação inicia-se na tabela OPERAÇÕES pelo nome da operação. A medida que cada operação vai sendo interpretada, informações adicionais sobre classes e relacionamentos vão sendo coletadas do ECD e, em paralelo, vão sendo construídas procedures PASCAL com o mesmo nome e mesma função das operações. As procedures vão sendo armazenadas gradativamente numa estrutura de listas encadeadas apresentadas a seguir, cuja especificação de cada campo pode ser vista no apêndice-F.

LISTA_PROCEDURES:

NUM_PROCD	NOME_PROCD	É_FORWARD	PONT_TIPOS	....

PONT_PARAM_ENTR	PONT_PARAM_SAIDA	PONT_VARIAVEIS	.....

PONT_ATRIBUIÇÕES	PONT_IF's	PONT_COMANDOS	PROX_PROCEDURE

LISTA_TIPOS_REGISTROS:

NOME_REGISTRO	PONT_CAMPOS	PROX_TIPO

LISTA_CAMPOS_REGISTRO:

CAMPO	TIPO	PROX_CAMPO

LISTA_VARIÁVEIS:

NOME_VARIÁVEL	TIPO_VARIÁVEL	CLASSE	PROX_VARIAVEL

LISTA_ATRIBUIÇÕES:

ATRIBUIÇÃO	PROX_ATRIBUIÇÃO

LISTA_IF's:

COMANDO_IF	PROX_IF

LISTA_COMANDOS:

COMANDO	PROX_COMANDO

Depois de todas as operações terem sido interpretadas, a lista se completa e como todas as procedures com seus respectivos parâmetros são conhecidas, é gerada uma interface (descrita na seção 5.3) que permite ao usuário ativar estas procedures, o que equivale a executar as operações-THM.

A saída deste mapeamento é um arquivo tipo texto contendo as procedures equivalentes ao ECO e uma interface que permite ao usuário ativar estas operações. Este arquivo é um programa PASCAL que será acoplado a outro programa PASCAL que contém a codificação das operações primitivas. Esta junção forma o programa THM_EXECUTÁVEL, que fica composto de duas partes : a parte fixa e a parte variável.

A parte fixa é formada pela definição de tipos, variáveis globais e pelas procedures que representam os predicados e operações primitivas definidas pelo THM. Estas procedures são codificadas em PASCAL e SQL, mas no momento não estão implementadas e para os testes de execução, estão sendo simuladas via terminal.

A parte variável, específica para cada aplicação, é a que é gerada pelo mapeamento do ECO. Esta parte é concatenada à parte fixa através de diretivas de inclusão, completando assim o programa THM-EXECUTÁVEL que fica composto dos seguintes módulos:

Tipos e variáveis globais
Funções que acessam o relógio do sistema (utilizadas pelos predicados e operações primitivas)
Funções e procedures equivalentes aos predicados e operações primitivas definidas pelo THM. (São codificadas em PASCAL e SQL. Só estas procedures manipulam o BD gerado pelo mapeamento do ECD)
Procedures geradas pelo mapeamento do ECO. (Cada operação gerou uma procedure, onde cada predicado e operação primitiva transformou-se em chamada aos procedimentos codificados no módulo anterior)
Interface gerada pelo mapeamento do ECO. (Corresponde ao programa principal. Permite ao usuário ativar as procedures do módulo anterior)

Depois de mapeado o ECO, o programa THM-EXECUTÁVEL fica completo e é submetido a compilação. O código objeto é deixado permanentemente na memória de forma que as operações-THM possam ser executadas sempre que necessário.

5.2. Algoritmo Para o Mapeamento das Operações-THM

Cada operação-THM descrita no ECO é transformada em uma procedure PASCAL por um algoritmo composto dos seguintes passos:

1. O nome da operação passa a ser o nome da procedure.

2. Cada parâmetro da operação é definido como parâmetro da procedure e seu tipo é obtido do ECD através da informação tipo das entidades da classe a qual o parâmetro pertence.

3. Cada cláusula **let** é transformada em um comando de atribuição juntamente com a definição da variável especificada na cláusula.

4. Para as pré-condições são criadas as variáveis lógicas PRECONDICOES, ERRO, CANCELAMENTO; E cada pré-condição é transformada em um If...Then.. no seguinte formato:

```
If Not (pre -condição) Then
  Begin
    Write (mensagem da opção otherwise);
    PRECONDICOES := False;
    ERRO := True;  ( Ou CANCEL := True se a  opção
                     especificada  for otherwise cancel)
  End;
```

5. O corpo da operação é submetido ao seguinte teste:

```
If PRECONDICOES Then
  Begin
    ... corpo da operação
  End;
```

e é mapeado de acordo com o tipo da estrutura: conjuntiva, disjuntiva ou iterativa.

conjuntiva - tem a estrutura [<declaração>] + e é transformada em uma estrutura de comandos seqüenciais

disjuntiva - tem a estrutura

[case <predicado> do <declaração>] +

e é transformada na seguinte estrutura condicional

```
[ If <predicado>
  Then <declaração>; ]+
```

iterativa - tem a estrutura

```
for each (<variável_entd> in <entd_grupo>) |
         (<variável_entd> in <classe>) |
         (class <var_classe>)
such that <predicado>
do [<declaracao>] +
```

Esta estrutura é mapeada em uma estrutura repetitiva e, o comando **for each** é transformado em uma chamada de procedure de acordo com a condição especificada pelo **for each**:

Procedure ForEachEntidadeInGrupo: Armazena no arquivo resultante (arq. de consultas) todas as entidades de um grupo.

Procedure ForEachEntidadeInClasse: Armazena no arquivo resultante

todas as entidades de uma classe.

Procedure ForEachClasse: Armazena no arquivo resultante o nome de todas as classes existentes.

Logo, a estrutura iterativa fica mapeada da seguinte maneira:

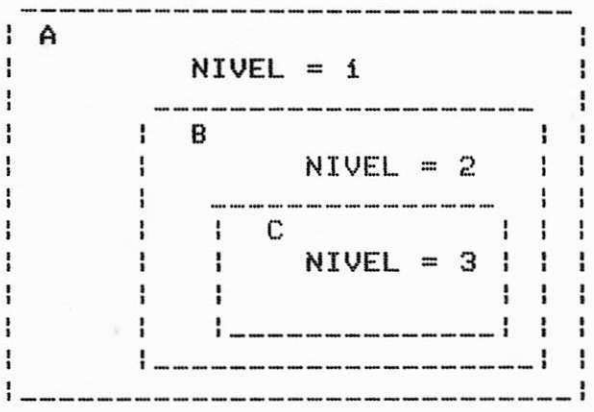
```
Chamada da Procedure ForEach equivalente
While Not EOF(ArqResultante) Do
  Begin
    Read(ArqResultante,Registro);
    { armazena o registro na váriavel livre
      do comando for each }
    If (predicado) Then
      Begin
        [(declaração)]+
      End
  End;
```

ó. Cada operação é encerrada com um teste que permite ou não a operação ser de fato executada.

```
If Not CANCEL And Not ERRO And (NIVEL = 1)
  Then Execute (NomeOperação)
  Else If NIVEL = 1
         Then NaoExecute (NomeOperação)
         Else NIVEL = NIVEL - 1;
```

onde a variável lógica ERRO é global a todas as procedures (operações) e faz com que se houver um ERRO em uma das operações, todo o processo que a envolve será cancelado. A variável CANCEL é local e permite que só esta operação seja cancelada de forma que o processo continue. A variável NIVEL representa em que nível de execução a operação foi solicitada. Por exemplo, se o usuário solicita a operação A e esta chama B e B chama C, então

temos 3 níveis de execução:



Só no nível 1 podemos ou não liberar a execução da operação.

A procedure Execute, chamada no final da operação de NIVEL = 1, permite que todas as alterações solicitadas pela operação sejam de fato executadas (Realize). E a procedure NaoExecute ignora todas as alterações deixando o banco de dados inalterado. Desta forma cada operação_THM chamada externamente é tratada como uma transação.

O algoritmo é executado para cada operação definida no ECO a ser mapeado.

Uma observação importante em relação ao algoritmo é que cada predicado e operação primitiva contida na operação-THM, será transformada numa chamada a um procedimento de mesmo nome seguido dos parâmetros atuais. Além dos parâmetros previstos pelas primitivas, são acrescentados os seguintes parâmetros:

- Tupla de tempo formada por ANO,MES,DIA,HORA,MIN e SEG
- Tipo da entidade. Por questões de compatibilidade de tipo na passagem de parâmetros, todas as entidades

são passadas como string, para isto é acrescentado um parâmetro que indica o tipo daquela entidade.

5.3. Algoritmo Para Geração Da Interface Com O Usuário

Depois de executado o algoritmo anterior, a estrutura de listas está completa e todas as procedures geradas são conhecidas juntamente com seus parâmetros. Então, é possível gerar comandos PASCAL que permitam uma iteração do sistema com o usuário. Esta interface corresponde ao programa principal do THM-EXECUTÁVEL que é composto de comandos que formam a seguinte tela:

```
+-------------------------------------------------------------+
|                                                             |
|   QUE SISTEMA QUER EXECUTAR? ....x.....                     |
|   PARA O SISTEMA   x   TEMOS AS SEGUINTES OPERAÇÕES         |
|                                                             |
|   1: Operação 1     2: Operação 2   3: .....                |
|       .                                                     |
|       .                                                     |
|   n: Operação n                                             |
+-------------------------------------------------------------+
|                                                             |
|   DIGITE O NÚMERO DA OPERAÇÃO QUE DESEJA EXECUTAR: ...y...  |
|                                                             |
|   PARA EXECUTAR A OPERAÇÃO y FORNEÇA OS SEGUINTES           |
|   PARÂMETROS:                                               |
|                                                             |
|   Parâmetro-1 (tipo-1) : .........................          |
|                                                             |
|   Parâmetro-2 (tipo-2) : .........................          |
|   .                 .                .                      |
|   .                 .                .                      |
|   .                 .                .                      |
|   Parâmetro-n (tipo-n) : .........................          |
|                                                             |
|   DESEJA EXECUTAR MAIS ALGUMA OPERAÇÃO (S/N) ?.....         |
|                                                             |
+-------------------------------------------------------------+
```

O nome das operações e os parâmetros com seus tipos são obtidos respectivamente da LISTA DE PROCEDURES, pelo NOME_PROC, e

da LISTA DE VARIÁVEIS através de PONT_PARAM_ENTRADA.

O algoritmo é basicamente composto dos seguintes passos:

1 - Para cada operação é criado um código inteiro que repesentará a procedure.

2 - Os códigos das operações são submetidos a um comando **Case** que ativará a procedure (operação) solicitada pelo usuário.

3 - Para cada código de procedure submetido ao **Case** são gerados comandos **Write** e **Read** para que o usuáio forneça os parâmetros necessários a execução da procedure correspondente a operação-THM solicitada.

4 - O comando **Case** fica submetido a um **WHILE** que permite ao usuário ativar sucessivamente mais de uma operação.

5.4 Aplicação dos Algoritmos

Nesta seção, exemplificamos o mapeamento de três operações que fazem parte do ECO do "Sistema de Carros" descrito na seção 4.3. Inicialmente, apresentamos a descrição destas em THM/LMD e em seguida o resultado do mapeamento (procedures e interface na linguagem PASCAL).

5.4.1 Descrição Do ECO Em THM/LMD.

Apresentamos a codificação em THM/LMD de operações aplicáveis ao "Sistema de Carros":

```
operation define_modelo              *  Operação Conjuntiva *
  input_parameters
    m : MODELO from DB by key tem_nome
    c : CONS_COMBUSTIVEL from DB
    f : FABRICANTE from DB by key tem_nome
  pre_conditions
    not(m in MODELO)
    c in CONS_COMBUSTIVEL
  body
    insert m into MODEL
    establish m tem_cons_comb c
    establish m tem_fabricante f



operation venda_exclusiva            * Operação Iterativa *
  input_parameters
    f : FABRICANTE from db by key tem_nome
    g : NOME_GARAGEM from DB
  body
    for each X in CARRO such that X produzido_por f do
      delete X from CAR_DE_FABR  at clock.now
      insert X into CAR_EM_GARAGEM   at clock.now
      establish  X  é_propriedade  g



Operation venda_garagem               *Operação Disjuntiva *
     input parameters
         c: CARRO_EM_GARAGEM from DB
         p: SUPER_CLASSE
     pre-conditions
         c in CAR_EM_GARAGEM at month: january to clock.now
         p in  PESSOA Or  p  in  GR_PESSOA
     body
         Case
              p in PESSOA
                   establish c é_propriedade p
              p in GR_PESSOA
                   establish c é_do_grupo g at day: 01
```

5.4.2 Procedures Geradas Pelo Algoritmo.

Aplicando-se o algoritmo 5.2, estas operações são transformadas nas seguintes procedures PASCAL:

```
PROCEDURE define_modelo(m:Tipostring;c:integer;f:Tipostring);
Var
  Cancel,PreCondicoes:Boolean;
Begin
  Nivel := Nivel + 1;
  Cancel := False;
  Precondicoes := False;
  If Not(Not(PrmtvIn(m,'modelo',s,0,0,0,0,0,0,0,0,0,0,0,0))) Then
    Begin
      Cancel := True;
       Writeln('**CANCELAMENTO** ');
    End;
  If Not(PrmtvIn(StrI(c),'cons_combustivel',i,0,0,
                0,0,0,0,0,0,0,0,0,0,0)) Then
    Begin
      Cancel := True;
       Writeln('**CANCELAMENTO** ');
    End;
  If Erro Or Cancel
    Then Writeln('Operacao nao pode ser executada')
    Else PreCondicoes := True;
  If Precondicoes Then
    Begin
      insert(m,'modelo',s,0,0,0,0,0,0);
      EstablishRelM('tem_cons_comb',m,StrI(c),s,i,0,0,0,0,0,0);
      EstablishRelM('tem_fabricante',m,f,s,s,0,0,0,0,0,0);
    End;
  If Not Erro And (Nivel = 1) And Not Cancel
    Then
      Execute('define_modelo')
    Else
      If Nivel = 1
        Then NaoExecute('define_modelo')
        Else Nivel := Nivel - 1;
End;
```

```
PROCEDURE venda_exclusiva(f:Tipostring;g:Tipostring);
Var
  K:Integer;
  X:integer;
Begin
  Nivel := Nivel + 1;
      ForeachEntdClasse('carro');
      Reset(ArqResultante);
      For K:= 1 To Filesize(ArqResultante) Do
        Begin
          Read(ArqResultante,RegResultante);
          X := ValI(RegResultante);
          If RelMembros('produzido_por',StrI(X),f,i,s,0,0,0,0,0,
                        0,0,0,0,0,0,0,0) Then
            Begin
              delete(StrI(X),'car_de_fabr',i,Clk('year'),
                     Clk('mounth'),Clk('day'),Clk('hour'),
                     Clk('minute'),Clk('second'));
              insert(StrI(X),'car_em_garagem',i,Clk('year'),
                     Clk('mounth'),Clk('day'),Clk('hour'),
                     Clk('minute'),Clk('second'));
              EstablishRelM('e_propriedade',StrI(X),g,i,s,
                     0,0,0,0,0,0);
            End
        End; { For }
      Close(ArqResultante);
  If  Not Erro And (Nivel = 1)
    Then
      Execute('venda_exclusiva')
    Else
      If Nivel = 1
        Then NaoExecute('venda_exclusiva')
        Else Nivel := Nivel - 1;
End;
```

```
PROCEDURE venda_garagem(c:integer;p:Tipostring);
Var
  Cancel,PreCondicoes:Boolean;
Begin
  Nivel := Nivel + 1;
  Cancel := False;
  Precondicoes := False;
  If Not(PrmtvIn(StrI(c),'car_em_garagem',i,Clk('year'),01,0,0,
          0,0,Clk('year'),Clk('mounth'),Clk('day'),Clk('hour'),
          Clk('minute'),Clk('second'))) Then
    Begin
      Cancel := True;
      Writeln('**CANCELAMENTO** ');
    End;
  If Not((PrmtvIn(p,'pessoa',s,0,0,0,0,0,0,0,0,0,0,0,0)) Or
     (PrmtvIn(p,'grp_pessoa',s,0,0,0,0,0,0,0,0,0,0,0,0))) Then
    Begin
      Cancel := True;
      Writeln('**CANCELAMENTO** ');
    End;
  If Erro Or Cancel
    Then Writeln('Operacao nao pode ser executada')
    Else PreCondicoes := True;
  If Precondicoes Then
    Begin
      If PrmtvIn(p,'pessoa',s,0,0,0,0,0,0,0,0,0,0,0,0)
        Then EstablishRelM('é_propriedade',StrI(c),p,i,s,
                           0,0,0,0,0,0);
      If PrmtvIn(p,'grp_pessoa',s,0,0,0,0,0,0,0,0,0,0,0,0)
        Then EstablishRelM('é_do_grupo',StrI(c),p,i,s,
                    Clk('year'),Clk('mounth'),01,0,0,0);
    End;
  If Not Erro And (Nivel = 1) And Not Cancel
    Then
      Execute('venda_garagem')
    Else
      If Nivel = 1
        Then NaoExecute('venda_garagem')
        Else Nivel := Nivel - 1;
End;
```

5.4.3 Interface Com O Usuário Gerada Pelo Algoritmo.

Apresentamos o programa principal do THM-EXECUTÁTAVEL gerado pela execução do algoritmo da seção 5.3, que equivale a interface que permite ao usuário executar as operações-THM definidas no ECO.

```
{  P R O G R A M A   P R I N C I P A L   }
 Begin
   Assign(ArqResultante,'Consult.THM');
   Erro := False;
   Nivel := 0;
   Continua := 's';
   While (Continua = 's') Or (Continua = 'S') Do
     Begin
        Clrscr;
      Gotoxy(27, 2); Write(' 1:define_modelo');
      Gotoxy( 1, 6); Write(' 2:venda_garagem');
      Gotoxy(53, 8); Write(' 3:venda_exclusiva');
      Writeln;
      Write('DIGITE O NUMERO DA OPERACAO QUE DESEJA EXECUTAR : ');
      Readln(NumProced);
      Case NumProced Of
      1 : Begin
          Writeln('PARA EXECUTAR A OPERACAO define_modelo
                   FORNECA OS SEGUINTES DADOS ');
          Write('modelo (Tipostring): '); Readln(ValStr1);
          Write('cons_combustivel (integer): ');Readln(ValInt1);
          Write('fabricante (Tipostring): '); Readln(ValStr2);
          define_modelo(ValStr1,ValInt1,ValStr2);
          End;
      2 : Begin
          Writeln('PARA EXECUTAR A OPERACAO venda_garagem
                   FORNECA OS SEGUINTES DADOS ');
          Write('car_em_garagem (integer): '); Readln(ValInt1);
          Write('pessoa (Tipostring): '); Readln(ValStr1);
          venda_garagem(ValInt1,ValStr1);
          End;
      3 : Begin
          Writeln('PARA EXECUTAR A OPERACAO venda_exclusiva
                   FORNECA OS SEGUINTES DADOS ');
          Write('fabricante (Tipostring): '); Readln(ValStr1);
          Write('nome_garagem (Tipostring): ');Readln(ValStr2);
          venda_exclusiva(ValStr1,ValStr2);
          End;
        End; { Case }
      Write('DESEJA EXECUTAR MAIS ALGUMA OPERACAO (S/N) ? ');
      Readln(Continua);
   End; { While }
  End.
```

5.5. Codificação de Predicados e Operações Primitivas

Note-se que no mapeamento proposto, os predicados e operações primitivas são representados por procedimentos

codificados em PASCAL/SQL e fazem parte do módulo fixo do programa THM-EXECUTÁVEL. Cada predicado é codificado em uma função de mesmo nome e cada operação numa procedure. Estes procedimentos necessitam dos recursos de linguagens procedurais e de recursos para manipulação de Banco de Dados. Como até agora não dispomos de um software desta natureza, não foi possível implementá-los, apenas os simulamos de forma que as consultas SQL fossem respondidas pelo usuário. Isto foi feito no intuito de testarmos as procedures e interface geradas pelo mapeamento.

No entanto, fornecemos o esboço da codificação de alguns predicados e operações primitivas. Observa-se que são chamadas duas funções de identificação, a **IdentfEntdClasse** (e,C), que retorna uma variável RECORD contendo informações sobre a identificação de uma entidade e na classe **C** e, a **IdentfRel** (r) que retorna uma variável RECORD contendo informações sobre as classes envolvidas no relacionamento r. Estas informações são obtidas do ECD.

Os registros têm a seguinte estrutura PASCAL:

```
Type
 Id_entd = record
             Valor :  string  {  valor  que  identifica   a
                                 entidade }
             id_rel:  string   {  nome  do  relacionamento
                                 identificador}
             id_classe:string  {classe de identificação}
           end;

 Id_rel = record
             cl_origem: string;
             card         : (min, max: integer);
                          {cardinalidade  mínima e máxima  do
                          relacionamento }
             cl_relacionada: string;
```

```
        card_inversa : (min, max: integer);
                        { cardinalidade do relacionamento
                          inverso }
    end;
```

5.5.1 Predicados Primitivos

Os resultados das consultas são armazenados numa relação temporal RESULT. Se RESULT for vazia (COUNT (val)= 0), a consulta é FALSE, se não, a consulta é TRUE.

a) Predicado e in C (at time)

```
Function PrimitivaIn (Entidade, Classe: TipoString;
                      IntervaloTempo: TuplaTempo): Boolean;
  Var
    Idtf: Id_entd;
    Cont: Integer:
  Begin
    Idtf:= IdentfEntdClasse (Entidade, Classe);
    * CREATE TABLE Resultado (Valor: string)
    * INSERT INTO Resultado
        SELECT Idtf.id_rel FROM Idtf.id_classe
          WHERE Idtf.id_rel = Entidade AND
                De_Ate = IntervaloTempo
    * TRANSMIT (SELECT Count (*) FROM Resultado ) to cont
    If cont < > 0
      Then PrimitvIn := True
      Else PrimitivIn:= False;
    * DROP TABLE RESULTADO
  End;
```

b) Predicado c rel d (at time)

```
Function RelMemb (c,d, rel: Tipostring;
                  Tempo: TuplaTempo) Boolean;
  Var
     Idtf_c, Idtf_d: Id_entd;
     Idtf_rel: id_rel;
  Begin
     Idtf_rel:= IdentfRel (rel)
     Idtf_c:= IdentEntdClasse (c, Idtf_rel.cl_origem);
     Idff_d:= IdentEntdClasse (d, Idtf_rel.cl_relacionada);
     *CREATE Table Resultado (Valor: Tipostring)
     If Idtf_rel.card_max = 1
         (  Classe  origem  gerou uma E_RELAÇÃO e um
            dos atributos eh rel)
     Then
        *INSERT INTO Resultado
           SELECT Idtf_c.id_rel FROM Idtf_rel.cl_origem
             WHERE (Idtf_c.id_rel = c AND Rel = d ) AND
                   (EM = Tempo);

        ELSE
          If Idtf_rel. card_max_inv = 1
             ( classe relacionada gerou uma E_RELAÇÃO
               e um dos atributos é rel_inv)
          Then
             *INSERT INTO Resultado
                SELECT Idtf_d.id_rel FROM Idtf_rel.cl_relacionada
                  Where (Idtf_d.id_rel = d
                         AND Idtf_rel_inv = c )
                         AND ( EM = Tempo )
          Else ( o relacionamento é n:m e foi gerada uma
               R_RELAÇÃO com o mesmo nome)
             *INSERT INTO Resultado
               SELECT Idtf_rel.cl_origem FROM rel
                 WHERE ((Idtf_rel.cl_origem = c) AND
                        (Idtf_rel.cl_relacionada = d)) AND
                        (EM = Tempo);
             If Resultado = EMPTY
                Then RelMemb: = False
                Else RelMemb: = True
     End;
```

5.5.2 Operações Primitivas

Seguem duas operações primitivas

a) insert e into C (at time)

```
Procedure insert (e,c: Tipostring; Tempo : TuplaTempo);
  Var
    Tipo_Classe: (Composta,Dinamica,Estatica);
  Begin
    Tipo_Classe:= Identf_TipoClasse (C)
    Case Tipo_Classe of
         Dinamica:
              INSERT INTO C: (e);
         Composta
              Begin
                Completa_Tupla (e,C);
                INSERT INTO C: (Tupla);
              End;
   End;
```

b) establish a rel b (at time)

```
Procedure Establish (a, b, rel: Tipostring;
                     Tempo: TuplaTempo);
        Var
          Idtf_a, Idtf_b: Id_entd;
          Idtf_rel: Id_entd;
        Begin
          Idtf_a : = IdentfEntdClasse (a);
          Idtf_b : = IdentfEntdClasse (b);
          Idtf_rel := IdentfRel (rel);
          If Idtf_rel.card_max = 1
             Then
                *UPDATE IN Idtf_rel.cl_origem: rel BY b
                    WHERE Idtf_a.id_rel = a
             Else
              If Idtf_rel.card_max_inv = 1
                Then
              * UPDATE IN Idtf_rel.cl_relacionada: rel BY a
                    WHERE Idtf_b.id_rel = b
              Else
                * INSERT INTO rel: (a,b);
        Teste_Cardinalidade (a,b);
              (se  a cardinalidade for excedida,  um  outro
               relacionamento será removido)
  End;
```

6 - CONCLUSÕES E SUGESTÕES

O principal mérito deste trabalho é partir da formalização de sistemas de informações modelados do mundo real por uma metodologia aplicável a um modelo semântico de dados, no caso o THM, e transferir este formalismo para o nível de implementação. Como vimos, o universo do discurso é modelado gerando o chamado esquema conceitual e, a partir deste esquema inicia-se a fase de implementação que deve manter toda a semântica do sistema de informação.

O mapeamento de modelos semânticos para esquema interno de dados não tem sido muito explorado, apesar de terem surgido muitos modelos nos últimos anos. Sabemos que, quanto mais rico em semântica for o modelo, mais distante está do nível de implementação e mais espinhoso será este mapeamento. Pela quantidade de fatos do mundo real tanto estruturais como operacionais e temporais que o THM pode modelar, admitimos que seja um modelo bastante completo e complexo, porém muitos esforços têm sido utilizados nesta direção. Como exemplo a implementação do sistema de efeitos colaterais para as operações-THM [Fr-87], a implementação do sistema de eventos e triggers aplicável ao THM [Tr-87] e um sistema especialista em THM para a geração de esquemas conceituais (em fase inicial).

O mapeamento aqui proposto abre caminhos para fechar este cerco, deixando o esquema conceitual de dados e o esquema de operações definidos no nível interno, ou seja, torna o THM

executável, faltando apenas o desenvolvimento de uma interface com um SGDB existente. O objetivo desta interface é tornar possível a execução dos predicados primitivos e operações primitivas usadas nas operações-THM. Estas primitivas é que manipulam o banco de dados e são tratadas como procedimentos PASCAL contendo comandos SQL. Estes procedimentos não são gerados pelo mapeamento, e sim definidos no módulo fixo do programa THM_EXECUTÁVEL.

Como continuação deste trabalho, além da interface com o SGBD deve-se pensar na integração deste com os sistemas citados no parágrafo anterior. Outro trabalho mais adiante pode ser o desenvolvimento de uma linguagem de consultas (THM/QL) para usuários esporádicos que não modificam o banco de dados. Esta linguagem, além das consultas convencionais, deverá fornecer condições do usuário explorar o histórico das entidades e sua hierarquia, através dos conceitos temporais e hierárquicos abordados pelo THM.

Com a conclusão deste trabalho e dos propostos poderemos formalizar um sistema de informação pela metodologia-THM, modelá-lo com o modelo semântico-THM, criar e manipular o banco de dados equivalente ao sistema de informação, o qual sempre se manterá consistente com a especificação.

REFERÊNCIAS BIBLIOGRÁFICAS

[Ab-74] - Abrial, J.R - "Data Semantics, em Data Base Management", Klimbie e Koffeman (eds.), North Holland, 1974.

[Ch-76] - Chen, P.P - "The Entity Relationship Model: Towards a Unified View of Data", ACM-TODS Vol.1, Num.1, 1976.

[Gr-82] - Griethuysen,J.J. (ed.) - " Concepts and Terminology for the Conceptual Schemma and Information Base", ISO TC 97/SC5/WG3, 1982.

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

[Sc-84] - Schiel. U. - "A Semantic Data Model and its Mapping to an Internal Relational Model", em "Databases: Role and Structure", Stocker, Atkinson e Gray (eds.), Cambridge Univ. Press, 1984.

[Sc-84] - Schiel, U. - "Um Modelo Semântico de Dados e seu Mapeamento para um Esquema Relacional Interno: Tese de Doutorado, Univ. Stuttugart (e alemão).

[Pr-84] - Priemsch, J. - "Transformação de um Esquema Conceitual em um Esquema Relacional Interno" - Dissertação de Diploma, Univ. Stuttgart, 1984 (em alemão).

[HS-85] - Horndasch, A, e Schiel, U. - "THM/CSL: A Language for

Conceptual Schema Specification", Bericht 5/83, Univ. Stuttgart, 1983.

[HSY-85] - A.Horndasch, R. Studer e R. Yasdi - "An Approach to Conceptual Schema Design of Information Systems", in "Information Systems: Theoretical and Formal Aspects". A. Sernadas, J.Bubenko e A. Olive (eds), North Holland, 1985.

[MS-85] - Medeiros, C.L.G. e Schiel,U. - "Projeto do Esquema Conceitual para Sistemas de Informação : Uma Abordagem Usando a Metolologia THM", Anais do SEMISH, Porto Alegre 1985.

[Sc-86] - Schiel, U. - "Projeto do Esquema Conceitual : Metodologia THM - Parte II", Informazonia, São Luis 1986.

[Tr-87] - Turazi, A. - "Sistema de Eventos e Triggers do Modelo THM", Dissertação de Mestrado, Departamento de Sistemas e Computação UFPb, 1987.

[Fr-87] - Ferreira, A. - "Sistema de Efeitos Colaterais do Modelo THM", Dissertação de Mestrado, Departamento de Sistemas e Computação UFPb, 1987.

[Fs-87] - Farias, F.J. - " Transformação Da Linguagem THM/LDD Em Uma Estrutura De Tabelas Proposta" , Projeto da disciplina Compiladores, período 87.2.

[Ab-87] - Abud, F.A. - " Transformação Da Linguagem THM/LMD Em Uma Estrutura De Tabelas Proposta", Projeto da Disciplina Compiladores, período 87.2.

[Jc-75] - Jackson,M. - "Principles of Program Design", Prentice Hall, 1975.

APÊNDICE A : SINTAXE DA THM/LDD

```
class (nome da classe )
     (class relationships
         [(nome rel.) ":"(cl.relacionada)
             '('(card_min),(card_max)')']*
         [pre-classe  ":" (cl.relacionada)(exclusive)]*
         [post-classe ":" (cl.relacionada)(exclusive)]*)
     (member relationships
         [(rel-name) : (related class) '(' (card_min) (card_max)')'
         (with n old values)]+)
     ((keys are ([(lista de chaves)]+ ! inherited) !
       (type (tipo) (format (especificação)))
     [(def. papel)]*
     [(def.agrupamento)]
     [(def.agregação)]
     (parameters: with time (and lifetime (constante) :
         (unidade de tempo))
(define papel)::=
     with role (nome do papel)
         gives subclasses ((lista de classe) (explicit) !
              by predicate(predicado)!
              using(relacionamento)as index
         parameters: (disjunctive ! covering)
(def. agregação)::=
     agregation of  ((lista de classes) (all ! explicit)!
         (classe-1, classe-2) by
         (relacionamento)] (exclusive))
(def.agrupamento) ::=
     grouping of ((nome da classe) (all ! explicit) !
         by predicate (predicado) !
         using (relacionamento))
```

Notação da sintaxe:

```
texto em negrito - símbolo terminal
; : ,             - símbolos terminais
'string'         - o string é um terminal
[nome]           - não terminal
(texto)          - texto ocorre 0 ou 1 vez
[texto]*         - texto ocorre 0 ou mais vezes
[texto]+         - texto ocorre 1 ou mais vezes
!                - ou
```

APÊNDICE B: SINTAXE DA THM/LMD

```
operation ⟨nome operação⟩
[input parameters
     [⟨nome parâmetro⟩ : ⟨nome classe⟩ (external|
          from DB [by Key ⟨lista chaves⟩]+)
[output-parameters
     [⟨nome parâmetro⟩ : ⟨nome classe⟩ (external|
          to DB [by Key ⟨lista chaves⟩]+)
[pre_conditions
     [([⟨predicado⟩[at⟨tempo⟩] [otherwise (warning| cancel| error)
          [⟨mensagem⟩]]) | ⟨cláusula let⟩)]+)
body
     ⟨declaração conjutiva⟩|
     ⟨declaração disjuntiva⟩|
     ⟨declaração iterativa⟩
[pos_conditions
     idem pre_conditions      ]
⟨declaração conjuntiva⟩ ::=
     [⟨declaração⟩ [only if ⟨predicado⟩]]+
⟨declaração disjuntiva⟩ ::=
     case [⟨predicado⟩ : ⟨declaração⟩]+
⟨declaração iterativa⟩ ::=
     for each ⟨condição⟩ do ⟨declaração conjuntiva⟩
⟨condição⟩ ::= ((⟨var-entidade⟩ in (⟨entidade de grupo⟩|⟨classe⟩))|
     class ⟨var-classe⟩) [such that ⟨ predicado⟩]
⟨declaração⟩ ::=  ⟨operação primitiva⟩ |⟨operação call⟩|
     ⟨cláusula let⟩
⟨operação primitiva⟩ ::= (⟨insert⟩|⟨delete⟩|⟨gr-insert⟩|
     ⟨gr-delete⟩|⟨nome⟩|⟨establish⟩|⟨remove⟩ |
     ⟨update⟩) [at⟨tempo⟩]
⟨operação call⟩ ::= ⟨nome operação⟩ "("⟨lista parâmetros entrada⟩
     ";" ⟨lista parâmetros saida⟩ ")"
⟨cláusula let⟩ ::=  let ⟨variável⟩ be ⟨termo⟩
⟨insert⟩ ::=  insert ⟨termo⟩ into ⟨classe⟩
⟨delete⟩ ::=  delete [⟨termo⟩ from ⟨classe⟩
⟨gr-insert⟩ ::=  gr-insert ⟨termo⟩ into ⟨entidade-de-grupo⟩
⟨gr-delete⟩ ::=  gr-delete ⟨termo⟩ from ⟨entidade-de-grupo⟩
⟨move⟩ ::=  move ⟨termo⟩ from ⟨classe1⟩ to ⟨classe2⟩
⟨establish⟩ ::=  establish ([⟨termo1⟩|⟨classe⟩)⟨relacionamento⟩
          ⟨termo2⟩
⟨remove⟩ ::=  remove ([⟨termo1⟩|⟨classe⟩)⟨relacionamento⟩ ⟨termo2⟩
⟨update⟩ ::=  update ([⟨termo1⟩|⟨classe⟩)⟨relacionamento⟩ ⟨termo2⟩
          to ⟨termo3⟩
⟨tempo⟩ ::=  (clock. ⟨unidade⟩[operador aritmético⟩
          ⟨valor numérico⟩)|⟨especificação unidade⟩|
          ⟨tupla de tempo⟩|⟨intervalo de tempo⟩
⟨unidade⟩ ::=  "day" | "month" | "year" | "hour" |
          "minute" | "second"
⟨especificação da unidade⟩  ::=  ⟨unidade⟩ ":" ⟨valor⟩
⟨intervalo de tempo⟩ ::=  "⟨" ⟨tempo⟩ to ⟨tempo⟩ "⟩"
⟨tupla de tempo⟩ ::=  ⟨ano⟩⟨mês⟩⟨dia⟩⟨hora⟩⟨minuto⟩⟨segundo⟩
⟨predicado⟩ ::= ⟨predicado primitivo⟩ |
        ⟨termo1⟩⟨operador relacional⟩⟨termo2⟩| not ⟨predicado⟩|
```

```
          if⟨predicado1⟩ then ⟨predicado2⟩
          ⟨predicado1⟩ or  ⟨predicado2⟩ |
          ⟨relacionamento de membros⟩"("⟨termo1⟩,⟨termo2⟩")"
⟨termo⟩    ::=   ⟨constante⟩    |   ⟨variável⟩  |   ⟨função⟩   |
          ⟨termo1⟩⟨operador aritmético⟩⟨termo2⟩ |- ⟨termo⟩
⟨função⟩  ::=   ⟨relacionamento membros⟩ "("⟨termos⟩")" |
          ⟨relacionamento classe⟩ "("  ")"
⟨predicado primitivo⟩ ::= in| is-rel | is-a | role-comp.....
⟨in⟩  ::=       ⟨termo⟩ in   ⟨classe⟩
⟨is-part⟩ ::=  is-part  "("⟨entidade   simples⟩,   ⟨entidade
          agregada⟩")"
⟨nome operação⟩ ::=  ⟨Identificador⟩
⟨nome parametro⟩  ::=  ⟨Identificador⟩
⟨nome classe⟩     ::=  ⟨Identificador⟩
⟨lista chaves⟩    ::=  [⟨Identificador⟩]+
⟨variável⟩ ::= ⟨Identificador⟩
⟨constante⟩ ::= ⟨numero⟩|⟨string⟩
⟨entidade⟩ ::= ⟨var-entidade⟩| constante
⟨var-entidade⟩ ::= "X" |"Y" | "Z"
⟨entidade de grupo⟩ ::=  ⟨Identificador⟩| ⟨Constante⟩
⟨lista param entrada⟩ ::= ⟨Identificador⟩|⟨Constante⟩
          [","⟨lista param⟩]*
⟨lista param saída⟩  ::=  Idem param entrada
⟨relacionamento⟩ ::= ⟨relacionamento de membros⟩ | ⟨relacionamento
          de classe⟩
⟨op.aritmético⟩ ::= + | - | * | /
⟨valor numérico⟩ ::=  ⟨dígito⟩ [⟨dígito⟩]*
⟨op. relacional⟩    ::=  < | < | = | <= | >=
⟨relacionamentos de membros⟩ ::=  ⟨termo1⟩ ⟨rel. memb⟩ ⟨termo2⟩
⟨relacionamento de classe⟩   ::=  ⟨classe⟩ ⟨rel. classe⟩ ⟨termo⟩
⟨rel. memb⟩ := ⟨Identificador⟩
⟨rel. classe⟩ := ⟨Identificador⟩
⟨entidade agregada⟩ ::= "⟨" ⟨termo⟩ [","  ⟨termo⟩]*.
⟨classe⟩ ::=  ⟨Identificador⟩
⟨classe agregada⟩ ::=   ⟨Identificador⟩
⟨classe componente⟩ ::= ⟨Identificador⟩
⟨classe de grupo⟩   ::= ⟨Identificador⟩
⟨classe elemento⟩ ::=  ⟨Identificador⟩
⟨classe generalizada⟩ ::= ⟨Identificador⟩
⟨subclasse⟩ ::= ⟨Identificador⟩
⟨identificador⟩ ::= ⟨letra⟩ [letra|dígito]*
⟨letra⟩    ::=  A,B.... Z
⟨dígito⟩   ::=  0,1.....9
```

Notação da sintaxe : Descrita no Apêndice A.

APÊNDICE C: DESCRIÇÃO DAS TABELAS QUE ARMAZENAM O ECD.

A estrutura destas tabelas é baseada na THM/LMD.

A primeira tabela é chamada CLASSES, Nela são incluídas informações necessárias a descrição das classes.

CLASSES (NOME CLASSE, TIPO CLASSE, TIPO ENTIDADE, TEMPO, TEMPO DE VIDA, EVENTO, DELETADA).

NOME CLASSE - indica o nome da classe.

TIPO CLASSE - indica se a classe é uma classe de domínio (S=estática ou D = dinâmica) ou uma classe geral (C = composta).

TIPO ENTIDADE - indica o tipo da entidade. Além dos três tipos padrão Inteiro, Real e String - podemos ter:
"A" para entidade agregada
"M" para uma entidade identificada por relacionamentos de membros.
"G" para a identificação de uma entidade de subclasse gerada por uma classe generalizada.

TEMPO - indica se é uma classe com intervalo de tempo (s/n).

TEMPO DE VIDA - indica o tempo de vida para entidades desta classe.

EVENTO - informa se para esta classe existem eventos da forma "on insert" ou "on delete" (S/N).

DELETADA - mostra se existe para a classe uma relação própria no banco de dados (S/N). A princípio toda classe gera uma E_Relação, porém na segunda etapa do algoritmo esta E_Relação pode ser eliminada, se isto ocorrer este campo passará a ser "S"

Na tabela RELACIONAMENTOS são depositadas informações sobre relacionamentos de membros e relacionamentos de classe.

RELACIONAMENTOS (NOME RELACIONAMENTO, CAR.MIN., CARD.MAX., CLASSE ORIGEM, RELAC.INVERSO, CARD.MIN.INV., CARD.MAX.INV., CLASSE RELACIONADA, IDENTIF.CHAVE, VAL.ANTER., QUANT.VAL., TIPO RELAC., EVENTO, DELETADO).

NOME RELACIONAMENTO - indica o nome do relacionamento.

CARD.MIN, CARD.MAX. - indicam respectivamente, a cardinalidade mínima e a cardinalidade máxima do relacionamento.

CLASSE ORIGEM - indica a classe que dá origem ao relacionamento.

RELAC.INVERSO - indica o relacionamento inverso. No caso de não existir um relacionamento inverso, este fica representado colocando-se o prefixo INV antes do nome do relacionamento.

CARD.MIN.INV e CARD.MAX.INV - indicam respectivamente a cardinalidade mínima e máxima do relacionamento inverso.

CLASSE RELACIONADA - indica a classe para a qual existe o relacionamento

IDENTIF.CHAVE - indica se o relacionamento é ou não chave da classe origem, pode assumir os seguintes valores:
0 - o relacionamento não é chave nem parte da chave.
1 - o relacionamento é chave principal das entidades da CLASSE ORIGEM.
2 - o relacionamento é parte da chave principal da CLASSE ORIGEM.

VAL.ANTER. - indica se para o relacionamento ficam registrados valores anteriores (S/N).

QUANT.VAL. - indica a quantidade de valores anteriores que devem ser conservados (valor inteiro)

TIPO RELAC. - distingue se o relacionamento é de classe ou de membros

EVENTO - indica se existem para o relacionamento, eventos da forma "on establish" ou "on remove" (S/N).

DELETADO - indica se o relacionamento inverso está armazenado no esquema (S/N). A princípio todos os relacionamentos 1:1 e n:1 passa a ser atributo, mas pela segunda etapa do algoritmo pode ser eliminado. Se isto ocorrer este campo passará a ter o valor "s"

Os relacionamentos pré-pós são armazenados na tabela PREPOS.

PREPOS (PRECLASSE, POSCLASSE, PRE_POS)

PRECLASSE - dá o nome da classe da qual sai o relacionamento temporal.

POSCLASSE - indica a classe para a qual existe o relacionamento temporal.

PRE_POS - pode ter quatro valores:
"ss" - o relacionamento é Pré-simples/Pós-simples.
"se" - o relacionamento é Pré-simples/Pós-exclusivo.
"es" - o relacionamento é Pré-exclusivo/Pós-simples.
"ee" - o relacionamento é Pré-exclusivo/Pós-exclusivo.

Nas duas tabelas seguintes são armazenadas a descrição das generalizações.

PAPEL (NOME_PAPEL, CLASSE, TIPO, REL.CONJUNTO)

NOME PAPEL - indica o nome do papel.

CLASSE - indica o nome da classe na qual o papel é aplicado.

TIPO - pode assumir os seguintes valores:
"E" - os elementos das subclasses resultantes são indicados explicitadamente.
"P" - as subclasses são determinadas por um predicado.
"R" - as subclasses são determinadas por um relacionamento de membros.

REL.CONJUNTO - indica como a generalização está relacionada com a teoria dos conjuntos:
"D" - as subclasses devem ser disjuntas.
"C" - a união das subclasses devem dar a classe generalizada.
"B" - "D" e "C".
"-" - não especificado.

GENERALIZAÇÕES (SUBCLASSE, CLASSE GENERALIZADA, PAPEL)

SUBCLASSE - indica o nome da classe gerada pela aplicação do PAPEL

CLASSE GENERALIZADA - indica o nome da classe genralizada correspondente.

PAPEL - indica o papel aplicado a genralização.

Na penúltima tabela são armazenadas informações sobre as agregações.

AGREGAÇÕES (CLASSE COMPONENTE , CLASSE AGREGADA, TIPO,

CLASSE COMPONENTE - dá o nome de uma classe componente dentro de uma agregação.

CLASSE AGREGADA - dá o nome da classe agregada correspondente.

TIPO - pode ter um dos seguintes valores:
"A" - é formado todo o produto carteziano sobre as classes componentes.
"X" - é deixada ao usuário a decisão de quais entidades das classes componentes ele reunirá em uma entidade agregada (tupla).

A última tabela contém a descrição dos agrupamentos.

AGRUPAMENTOS (CLASSE ELEMENTOS, CLASSE GRUPOS, TIPO, REL.CONJUNTO)

CLASSE ELEMENTOS - dá o nome da classe de cujas entidades são formados os grupos.

CLASSE GRUPOS - nome da classe de grupos

TIPO - pode ter os seguintes valores:
"A" - identifica a classe de grupo como conjunto da classe de elementos.
"E" - indica que os grupos devem ser formados explicitamente pelo usuário.
"R" - os grupos são formados com base em um relacionamento de membros.
"P" - indica que o agrupamento é feito com base num predicado "by predicate".

APÊNDICE-D : ESTRUTURA DAS LISTAS ENCADEADAS UTILIZADAS NO MAPEAMENTO DO ESQUEMA DE DADOS.

LISTA_RELAÇÕES: Armazena informações sobre as relações geradas.

NOME_RELAÇÃO: Equivale ao nome das classes ou relacionamentos que geraram respectivamente E_Relação ou R_Relação.

PONT_ATRIBUTOS: Apontador para LISTA_ATRIBUTOS, que armazena os atributos de cada relação.

PONT_PROX : Apontador para a próxima relação.

LISTA_ATRIBUTOS : Armazena os atributos das relações.

NOME_ATRIBUTO: Corresponde ao nome dos relacionamentos da classe que gerou a E_Relação ou ao nome das classes relacionadas no relacionamento que gerou a R_Relação.

TP_ATRIBUTO: Equivale ao tipo de cada atributo da relação.

PONT_PROXIMO: Encadeamento para o próximo atributo.

APENDICE-E : DESCRIÇÃO DAS TABELAS QUE ARMAZENAM O ECO.

A estrutura destas tabelas retrata a estrutura da THM/LMD e armazena cada parte das operações-THM, descrita nesta linguagem, em uma tabela específica.

A primeira tabela é a OPERAÇÕES e especifica a estrutura geral das operações.

OPERAÇÕES : (NOME_OPERAÇÃO, PONT_PARAM (entrada), PONT_PARAM (saída), PONT_PRÉ_CONDIÇÕES, TP_CORPO,PONT_TP_CORPO)

NOME_OPERAÇÃO: Nome da operação

PONT_PARAM (entrada) : Apontador para a tabela PARAM_OPERAÇÕES, para especificar os parâmetros de entrada da operação.

PONT_PARAM (saída) : Idem atributo anterior, para os parâmetros de saída.

PONT_PRÉ_CONDIÇÕES : Apontador para tabela PRÉ_CONDIÇÕES, que armazena as pré-condições da operação.

TP_CORPO: Especifica o tipo do corpo de uma operação_THM, que pode ser formado por:
"c" - declarações conjuntivas,
"d" - declarações disjuntivas,
"i" - declarações iterativas,

PONT_TP_CORPO: Apontador para tabela CONJUNTIVAS, ou DISJUNTIVAS ou ITERATIVAS, conforme o tipo especificado no atributo anterior.

A tabela PARÂMETROS determina os parâmetros de entrada e os de saída das operações_THM.

PARAM_OPERAÇÕES (POS, PONT_VARIÁVEL (nome_param), PONT_CLASSE, ORIGEM_OU_DESTINO, PONT_CHAVES, TP_DEFAULT, PONT_TP_DEFAULT, PONT_PROX)

POS : Linha da tabela

PONT_VARIAVEL: Apontador para a tabela VARIÁVEIS para armazenar o nome do parâmetro

PONT_CLASSE : Apontador para tabela CLASSES para armazenar

a classe a que o parâmetro pertence.

ORIGEM_OU_DESTINO: Refere-se a condição do parâmetro ser externo ao bando de dados ou interno.
"bd" - o parâmetro originou-se/destina-se ao BD
"ex" - o paramêtro será transferido para o usuário.

PONT_CHAVES : Apontador para tabela CHAVES quando o parâmetro tem a opção **by key**.

TP_DEFAULT : Refere-se a opção **with default** para os parâmetros de entrada.
"n": Não tem a opção default
"f": O parâmetro tem uma função como default

PONT_TP_DEFAULT: Apontador para tabela FUNÇÕES ou CONSTANTES, conforme TP_DEFAULT.

PONT_TP_PROX Encadeamento para o próximo parâmetro.

A tabela PRÉ-CONDIÇÕES especifica as pré-condições das operações.

PRÉ-CONDIÇÕES: (POS, TP_PRE_COND, PONT_TP_PRE_COND, OTHERWISE, PONT_MENSAGEM, PONT_PROX)

POS : linha tabela

TP_PRE_COND: Tipo da pré-condição
"p" : Predicado
"l" : Cláusula Let
"i" : If <predicado1> then <predicado2>

PONT_TP_PRE_COND : Apontador para as tabelas PREDICADOS, CLÁUSULA_LET ou IF_THEN de acordo com TP_PRE_COND.

OTHERWISE: Diz respeito a opção otherwise
"n" : a pré-condição não tem a opção otherwise,
"c" : foi usada a opção **otherwise cancel**
"w" : foi usada a opção **otherwise warning**
"e" : foi usada a opção **otherwise erro**

PONT_MENSAGEM : Apontador para tabela MENSAGENS

PONT_PROXIMO : Encadeamento para próxima pré-condição.

A tabela CONJUNTIVAS descreve o corpo das operações conjuntivas.

CONJUNTIVAS (POS, TP_DECL, PONT_TP_DECL, ONLY IF, PONT_PREDICADO, PONT_PROXIMO).

POS : Linha da tabela

TP_DECL : Especifica o tipo da declaração:
"p" - operação primitiva
"c" - chamada de uma operação definida pelo usuário
"l" - cláusula Let

PONT_TP_DECL : Apontador para tabela OPERAÇÕES PRIMITIVAS, OPERAÇÕES_CALL ou cláusula LET de acordo com TP_DECL

ONLY_ IF : se a declaração possue a opção ONLY IF (s/n).

PONT_PREDICADO: Apontador para tabela PRDICADOS se existe a opção ONLY IF

PONT_PROX : Encadeamento para próxima declaração conjuntiva.

A tabela DISJUNTIVAS descreve o corpo das operações disjuntivas.

DISJUNTIVAS (POS, PONT_PREDICADO, TP_DECL, PONT_TP_DECL, PROX)

POS : Linha da tabela

PONT_PREDICADO: Apontador para PREDICADO do case

TP_DECL, PONT_TP_DECL e PROX, idem tabela anterior.

A tabela ITERATIVAS descreve o corpo das operações iterativas;

ITERATIVAS (POS, VAR_LIVRE, PONT_COND_ITER, PONT_CONJUNTIVAS)

POS : Linha da tabela

VAR_LIVRE : "X", "Y" ou "Z" variáveis que assumirão qualquer valor durante o processamento.

PONT_COND_ITER: Apontador para tabela COND_ITERATIVA

PONT_CONJUNTIVAS: Apontador para tabela CONJUNTIVAS.

A tabela COND_ITERATIVA armazena informações sobre as condições usadas nas operações iterativas.

COND_ITERATIVA (POS, TP_COND, X_in, ENTD_GRUPO, PONTPREDICADO)

POS : Linha de tabela

TP_COND : o tipo da condição de acordo com a sintaxe.
"1" : for each (var_entid) in (entid. de grupo)
"2" : for each (var_entid) in (classe)
"3" : for each classe (classe)

X_in : identificação de uma entid_de_grupo ou de uma classe conforme TP_COND:
(entd de grupo) se TP_COND = "1"
(classe) se TP_COND = "2"
espaço em branco se TP-COND = "3"

ENTD_GRUPO : identificação de uma entidade de grupo quando TP_COND = 1, nos demais casos é espaço em branco.

PONT-PREDICADO: Aponta para tabela PREDICADOS quando a opção **such that** é especificada.

A tabela PREDICADOS especifica todos os predicados que aparecem nas operações.

PREDICADOS, (POS, TP_PREDICADO, SIMB_PREDICATIVO, PONT_TERMOS, AT_TIME, PONT_TEMPO, NOT, OR_AND, PONT_PROX_OR_AND)

POS : Linha da tabela

TP_PREDICADO: Um predicado pode ser de um dos tipos:
"p" : Predicado primitivo
"c" : Comparativo (termos ligados por operador relacional)
"r" : Um relacionamento de membros.

SIMB_PREDICATIVO: Armazena o símbolo predicativo de acordo com TP_PREDICADO. Se o TP_PREDICADO = "p" ou "r" o SIMB_PREDICATIVO é o nome do relacionamento, Se o TP_PREDICADO = "c" o SIMB_PREDICATIVO è o operador relacional.

PONT_TERMOS: Apontador para tabela TERMOS, para especificar os termos do predicado.

AT_TIME : Determina se o predicado tem a opção at time (S/N)

PONT_TEMPO : Apontador para tabela TEMPO quando o predicado

dispõe da opção AT TIME.

NOT : Determina se o predicado é precedido por um NOT (S/N)

OR_AND : Determina se há outro predicado conectado a este por um OR ou AND. (s/n)

PONT_PROX_OR_AND: se o ítem anterior for "s", aponta para própria tabela de PREDICADOS para armazenar outro predicado conectado por OR ou AND.

A tabela TERMOS especifica todos os termos que aparecem nas operações .

TERMOS (POS, TP_TERMO, PONT_TP_TERMO, PONT_PROX)

POS : Linha da tabela

TP_TERMO : Um termo pode ser de um dos tipos:
I, R, S: Uma constante inteira, real ou string
V : uma variável
F : uma função
B : um termo binário (expressão aritmética)
C : uma classe
E : uma entidade

PONT_TP_TERMO : Apontador para a tabela equivalente ao tipo especificado por TP_TERMO.

PONT_PROX : Encadeamento para o próximo termo;

A tabela CLÁUSULAS_LET descreve as cláusulas Let das pré-condições e do corpo das operações.

CLÁUSULAS_ LET (POS, PONT_VARIAVEL, PONT_TERMO)

POS : Linha de tabela

PONT_VARIÁVEL: Apontador para tabela VARIÁVEIS para armazenar o nome da variável que vai referenciar o termo.

PONT_TERMO : Apontador para tabela TERMOS

A tabela IF_THEN faz referência aos predicados das pré-condicões do tipo IF P1 THEN P2.

IF_THEN (POS, PONT_PREDICADO1, PONT_PREDICADO2)

POS : Linha da tabela

PONT_PREDICADO1 : Apontador para tabela PREDICADOS para armazenar o primeiro predicado.

PONT_PREDICADO2 : Idem anterior, sendo para o segundo predicado.

A tabela FUNÇÕES armazena as funções de relacionamentos de membros e as de relacionamentos de classes que aparecem no contexto das operações.

FUNÇÕES (POS, TP_FUNÇÃO, SIM_FUNCIONAL, PONT-TERMOS)

POS : Linha de tabela

TP_FUNÇÃO : Uma função pode ser de um dos tipos:
M : função de relacionamento de membros r (t)
C : função de relacionamento de classe r ( )

SIM_FUNCIONAL : Armazenam o nome da função.

PONT_TERMOS: Apontador para tabela TERMOS para armazenar os parâmetros das funções.

A tabela PRAM_ATUAIS armazena os parâmetros das chamadas de operações (declarações do tipo call).

PARAM_ATUAIS (POS, ENTR/SAIDA, PONT-TERMO, PONT_PROXIMO)

POS : Linha da tabela

ENTR/SAIDA: Determina se o parâmetro é de entrada ou de saída (E/S)

PONT_TERMO: Apontador para tabela termos, para armazenar o parâmetro

A tabela OPERAÇÕES-PRIMITIVAS armazena as operações primitivas usadas nas operações_THM.

OPERAÇÕES PRIMITIVAS (POS, OPER_PRIMITIVA, SIMB_AÇÃO, PONT_TERMO, AT TIME, PONT_TEMPO).

POS : Linha da tabela

OPER_PRIMITIVA : Nome da operação primitiva

SIMB_AÇÃO: Armazena a ação de cada operação primitiva
Ex: insert "João" into NOME
operação : insert
ação : "joão into NOME"

PONT_TERMO: Apontador para tabela TERMOS, para armazenar os termos de cada ação.

AT_TIME : Determina se a opção AT TIME foi especificada na operação primitiva (S/N)

PONT_TEMPO : Apontador para tabela TEMPO para armazenar o fator tempo quando associado a operação

A tabela OPERAÇÕES_CALL armazena as declarações call da operação que está sendo descrita.

OPERAÇÕES_CALL (POS, NOME_OPER, PONT_PARAM_ATUAIS)

POS : Linha da tabela

NOME_OPER: Nome da operação que está sendo chamada

PONT_PARAM_ATUAIS: Apontador para tabela PARAM_ATUAIS, para armazenar os parâmetros com os quais a operação será executada.

A tabela CHAVES armazena a chave dos parâmetros de entrada, quando a opção **from DB by Key (<chaves>)** é especificada.

CHAVES (POS, NOME_CHAVE, PONT_PROX)

POS : Linha da tabela

NOME CHAVE : corresponde a um relacionamento de membros que é chave de identificação de uma classe.

PONT_PROX : Encadeamento para próxima chave. Isto ocorre quando a chave é composta.

A tabela MENSAGENS armazena as mensagens da opção otherwise.

MENSAGENS (POS, LINHA_MENSAGEM, PONT_PROX).

POS : Linha da tabela

LINHA_MENSAGEM : Mensagem de advertência

PONT_PROX : Encadeamento para linha de continuação da mensagem

As cinco tabelas que seguem armazenam informações sobre períodos de tempo especificados nas pré-condições e nas operações primitivas.

TEMPO (POS, TP_TEMPO , PONT_TP_TEMPO)

POS : Linha da tabela

TP_TEMPO refere-se a forma que o tempo foi referenciado
C : ⟨clock⟩ ⟨unidade⟩
E : ⟨especificação explícita da unidade⟩
T : ⟨ tupla de tempo ⟩
I : ⟨ intervalo de tempo⟩

PONT_TP_TEMPO : Apontador para uma das tabelas descritas abaixo de acordo com TP_TEMPO.

CLOCK (POS, UNIDADE , OPER_ARITM, VALOR)

POS : Linha da tabela

UNIDADE : Unidade referenciada. Ex: year, month, etc.

OPER_ARITM: Um dos operadores aritméticos

VALOR : valor inteiro

UNIDADE_EXPLICITA (POS, UNIDADE, VALOR)

POS : Linha da tabela

UNIDADE : Uma das unidades de tempo

VALOR : Armazena o valor da unidade especificada

INTERVALO_TEMPO (POS, PONT_TEMPO_INICIAL, PONT_TEMPO_FINAL)

POS : Linha da tabela

PONT_TEMPO_INICIAL e

PONT_TEMPO_ FINAL : Apontam para a tabela TEMPO

TUPLA TEMPO (POS, ANO, MES, DIA, HORA, MINUTO, SEGUNDO)

Armazena tuplas de tempo

A tabela TERMO_BINÁRIO armazena as expressões aritméticas.

TERMO_BINARIO (POS, TIPO_TERMO, PONT_TERMO, PONT_PROXIMO)

POS : Linha da tabela

TIPO_TERMO : Refere-se a cada componente da expressão.
"t" - se for um termo
"+" , "-", "*" ou "/" - se for um operador aritmético.

A tabela CLASSES armazena as classes referenciadas na operação.

CLASSES (POS, NOME_CLASSE)

A tabela ENTIDADES armazena as entidades referenciadas no decorrer das operações

ENTIDADES (POS, NOME_ENTD, PONT_PROX_AGR)

POS : Linha da tabela

NOME_ENTD : Nome da entidade

PONT_PROX_AGR: Apontador para próxima entidade que compõe uma agregação.

As três últimas tabelas armazenam as variáveis e

constantes referenciadas nas operações.

VARIAVEIS (POS, NOME_VAR)

CONST_INTEIRAS (POS, VALOR)

CONST_STRING (POS, VALOR)

APÊNDICE - F: ESTRUTURA DAS LISTAS ENCADEADAS UTILIZADAS NO MAPEAMENTO DO ESQUEMA CONCEITUAL DE OPERAÇÕES.

LISTA_PROCEDURES: Armazena informações sobre a estrutura das procedures que representam as operações_THM

NUM_PROCD: Código gerado para procedure. É utilizado pela interface.

NOME_PROCD: mesmo nome da operação_THM

É_FORWARD: (s/n)- Determina se esta procedure (operação) é ou não chamada por outra. Se for, precisará ser declarada com a opção FORWARD.

PONT_TIPOS: Apontador para LISTA TIPOS_REGISTROS, determinando se é necessário declarar um novo tipo de dado para a procedure

PONT_PARAM_ENTR: Apontador para LISTA VARIAVEIS, para especificar a declaração dos parâmetros de entrada da procedure

PONT_PARAM_SAIDA : Idem PONT_PARAM_ENTRADA, considerando a saída

PONT_VARIAVEIS: Apontador para a LISTA VARIAVEIS, para especificar as variáveis locais a procedure.

PONT_ATRIBUIÇÕES: Apontador para LISTA ATRIBUIÇÕES, que especifica os comandos de atribuições gerados pela cláusula LET;

PONT_IF's: Apontador para LISTA_IF's, que especifica os comandos IF's gerados pelas Pré-condições.

PONT_COMANDOS: Apontador para LISTA COMANDOS, que armazena o corpo das operações_THM.

PROX_PROCEDURE - Encadeamento para a próxima procedure.

LISTA_TIPOS_REGISTROS: Armazena os regitros criados para chaves compostas e agregações.

NOME_REGISTRO: Nome da chave ou da agregação.

PONT_CAMPOS: Apontador para LISTA_CAMPOS_REGISTRO, que armazena os campos do registro.

PROX_TIPO: Encadeamento para o próximo tipo de registro.

LISTA_CAMPOS_REGISTRO: Armazena os campos dos registros criados.

CAMPO: O nome do campo equivale a classe componente de uma agregação ou, a chave componente de uma chave composta.

TIPO: Tipo do campo definido

PROX_CAMPO: Encadeamento para o próximo campo

LISTA_VARIÁVEIS: Armazena informações para a declaração dos parâmetros e das variáveis locais à procedure.

NOME_VARIÁVEL: mesmo nome especificado na operação_THM.

TIPO_VARIÁVEL: Para os parâmetros. O tipo equivale ao tipo das entidades da classe que o parâmetro pertence. Para as variáveis locais, que são geradas pela cláusula LET, o tipo será equivalente ao termo especificado pela cláusula.

CLASSE: Armazena a classe a que pertence o parâmetro, deixado em branco para as variáveis locais.

PROX_VARIAVEL: Encadeamento para a próxima variável.

LISTA_ATRIBUIÇÕES: Armazena os comandos de atribuições equivalentes a cláusula LET.

ATRIBUIÇÃO: Armazena o comando propriamente dito.

PROX_ATRIBUIÇÃO: Encadeamento para a próxima atribuição.

LISTA_IF's: Armazena os comandos IF's originados pelas pré-condições.

COMANDO_IF : Armazena o comando completo

PROX_IF: Encadeamento para o próximo IF.

LISTA_COMANDOS: Armazena os comandos equivalentes ao corpo da operação_THM (body).

COMANDO: Armazena vários tipos de comandos.

PROX_COMANDO: Encadeamento para o próximo comando.