UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UM MODELO NÃO PROCEDURAL DE ESPECIFICAÇÃO E IMPLEMENTAÇÃO VOLTADO A SISTEMAS TRANSACIONAIS EM BANCO DE DADOS

por

HUBERT AHLERT

TESE DE DOUTORADO
Tese submetida como requisito parcial para obtenção
do grau de Doutor em Ciência da Computação

Prof. Dr. Carlos Alberto Heuser

Orientador

Porto Alegre, setembro de 1994.

CIP - CATALOGAÇÃO NA PUBLICAÇÃO

Ahlert, Hubert

Um Modelo não Procedural de Especificação e Implementação voltado a Sistemas Transacionais em Banco de Dados / Hubert Ahlert. Porto Alegre, CPGCC da UFRGS, 1994.

1v.

Tese (Doutorado) - Universidade Federal do Rio Grande do Sul, Curso de Pós-Graduação em Ciência da Computação, Porto Alegre, 1994. Orientador: Heuser, C.A.

Tese: Programação Automática, Ferramentas para Construção de Sistemas, Proceduralização de Especificações Declarativas, Tradução de Linguagens, Transações em Banco de Dados.

À minha esposa *GENOVEVA*, filha *JESSICA* e demais familiares, pelo apoio e compreensão recebidos e, principalmente, à minha mãe e ao meu pai que, embora não presentes, **SEMPRE** procuraram incentivar meu estudo e aperfeiçoamento profissional.

AGRADECIMENTOS

Ao professor Carlos Alberto Heuser, incansável orientador deste trabalho, pela motivação inicial na determinação do horizonte de pesquisa e definição do tema de tese, pelas valorosas críticas e sugestões apresentadas no decorrer do trabalho e pela atenção com que acolheu esta tese.

Aos professores José Mauro Volkmer de Castilho, Roberto Tom Price e Cirano Iochpe, pela colaboração e contribuições prestadas durante a defesa da proposta de tese.

Aos professores Paulo Cesar Masiero (ICMSC-USP), Ulrich Schiel (DSC-UFPB) e Cirano Iochpe (CPGCC-UFRGS), membros da Banca Examinadora, pela criteriosa revisão da tese e pelas valiosas sugestões apresentadas.

Ao CPGCC, por oferecer um ambiente de pesquisa que permitisse o desenvolvimento de um trabalho desse tipo.

Aos professores do CPGCC, pela forma amiga de compartilharem seus conhecimentos com os alunos durante as disciplinas que ministraram.

Às bibliotecárias do Instituto de Informática, principalmente à Ida Rossi, pela simpática colaboração prestada na tarefa de levantamento e revisão da bibliografia que embasou a pesquisa consolidada nesta tese.

Ao Centro de Processamento de Dados da UFRGS e à Escola Técnica de Comércio da UFRGS, por permitirem o meu afastamento durante o curso e, principalmente, por colocar à disposição todos os recursos necessários para o desenvolvimento da tese.

A todos que, de alguma forma, colaboraram na elaboração deste trabalho.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE ABREVIATURAS

BD	- Banco(s) de Dados
BNF	- Backus Naur Form
CASE	- Computer-Aided Software Engineering
CPGCC	- Curso de Pós-Graduação em Ciência da Computação
ER	- Entidade-Relacionamento
ER/T	- Entidade-Relacionamento-Transação
SGBD	- Sistema de Gerência de Banco de Dados
SQL	- Structured Query Language
TDF	- Técnica de Descrição Formal
UFRGS	- Universidade Federal do Rio Grande do Sul

RESUMO

Esta tese de doutorado apresenta um modelo de especificação, textual e gráfico, para sistemas transacionais em banco de dados (ER/T^+) e, também, um modelo de implementação desta especificação. Sugere uma técnica de proceduralização de especificações declarativas, usando um grafo de dependência de fluxos de dados para estabelecer a relação de precedência entre os fluxos do diagrama da linguagem gráfica de especificação. Apresenta, também, os mecanismos de execução da linguagem de especificação proposta e as regras de mapeamento da linguagem de especificação, em seus aspectos estruturais (dados) e comportamentais (transações), para correspondentes construções na linguagem de implementação (C e SQL).

Adicionalmente, são discutidos aspectos de otimização de consultas no âmbito da linguagem de especificação de transações e, também, aspectos de aninhamento de consultas para combinar diversos fluxos do diagrama ER/T^+ em expressões complexas de consultas SQL.

PALAVRAS-CHAVE:

Programação Automática, Construção Automática de Sistemas, Ferramentas para Construção de Sistemas, Proceduralização de Especificações Declarativas, Tradução de Linguagens, Transações em Banco de Dados, Otimização de Consultas.

ABSTRACT

TITLE: "A NON-PROCEDURAL MODEL TO SPECIFYING AND IMPLEMENTING DATABASE TRANSACTIONS SYSTEMS"

This Ph.D thesis presents a graphic and textual specification model for database transactions systems (ER/T^+) and, also, an implementation model for this specification. Suggest a proceduralization technique for declarative specifications using a data flow dependency graph to establish a precedence relation between the diagram flows of the graphics specification language. Furthermore it presents the execution mechanism of the proposal specification language and the behavioral and structural rules for mapping the specification language into corresponding implementation language (C and SQL) constructions.

Additionaly, are discussed query optimization aspects for transaction specification language and aspects of nested queries to combine various ER/T^+ diagram flows into complex SQL query expressions.

KEYWORDS:

Automatic Programming, Automatic System Building, Tools for System Building, Proceduralization of Declarative Specifications, Language Translation, Database Transactions, Query Optimization.

1. INTRODUÇÃO

As investigações da engenharia de software, com resultados práticos em produtos como ferramentas CASE e linguagens de quarta geração, vêm aplicando esforços na busca da automação de tarefas de analistas e programadores. No elenco destas investigações, também podem ser incluídas as pesquisas sobre modelos ou paradigmas de construção automática de sistemas.

Um ambiente de construção automática de sistemas busca evitar as traduções manuais realizadas, por um programador, no processo de mapeamento de uma especificação para uma linguagem de programação ou aquelas traduções manuais, entre diferentes niveis de especificação, com refinamentos sucessivos, até chegar ao código executável.

Considerando que uma linguagem de programação é também uma especificação formal, só que com maior nível de detalhe de implementação, e que as atuais técnicas de construção de programas já possuem sedimentados métodos compilativos e interpretativos para a geração de código executável, a questão passa a ser a de extrapolar estes métodos a um nível de abstração maior, ou seja, assegurar que estes métodos também consigam mapear definições não algorítmicas (declarativas) em procedimentos que mostrem a alternativa de implementar estas definições. Nestes termos, foram estudadas as formas de mapear as especificações para correspondentes procedimentos executáveis.

Para a presente tese, o horizonte de pesquisa é o das técnicas de implementação de especificações não procedurais (declarativas). Estas técnicas visam produzir um conjunto de procedimentos que representem as especificações em uma linguagem de programação.

No escopo de investigação da tese são focalizados os sistemas de informação do tipo transacional em banco de dados, abordando o tema sob o prisma das especificações usadas para definir as transações em SGBD relacionais.

O conceito de transação, aqui considerado, é o clássico /GRA 81/ de transição atômica, indivisível, de um estado do sistema para outro e onde nenhum estado intermediário é observado. Por este conceito, uma transação garante a consistência do banco de dados durante a transição de estados. Todas modificações do banco de dados, em uma transição de estados, são feitas coincidentemente. No modelo de transação, também chamado ACID, enfatiza-se as propriedades de **atomicidade** (indivisibilidade da operação; todas operações realizadas ou, então, nenhuma delas), **consistência** (integridade da base de dados ao final da transação), **isolamento** (imunidade a interferências externas) e **durabilidade** (efeito de uma transação sobreviver a falhas e persistir, enquanto não modificada ou desfeita por outra transação).

No presente trabalho são propostos um modelo declarativo de especificação para sistemas transacionais em banco de dados, as técnicas de proceduralização destas especificações e os mecanismos de execução da linguagem de especificação, culminando na obtenção de um modelo de implementação que espelhe estas especificações.

O trabalho inicia com uma definição, no capítulo 2, do perfil desejável para linguagens de especificação destinadas a sistemas de banco de dados. Após, no capítulo 3, avaliando-se algumas linguagens constantes na literatura, a especificação de transações é alvo de análise.

No capítulo 4 é proposta uma linguagem que atende às características desejáveis delineadas no capítulo 2.

No capítulo 5 é discutida a implementação das especificações, segundo o modelo proposto no capítulo 4, onde são apresentados os mecanismos de proceduralização de especificações declarativas, as regras de mapeamento das funções especificadas na descrição do sistema, os detalhes de paralelismos e otimização dos procedimentos gerados, a questão do aninhamento de consultas de banco de dados e o processo de tradução da especificação propriamente dito.

Conclusões e sugestões para futuros trabalhos encerram esta tese.

2. PROPRIEDADES DESEJÁVEIS PARA LINGUAGENS DE ESPECIFICAÇÃO DE TRANSAÇÕES

Nas primeiras publicações sobre organização, projeto /DAT 86a/ e modelagem /CHE 76/ de banco de dados, o enfoque principal se concentrou sobre as formas de realizar a representação de dados. Existia pouca referência à descrição e formalização das propriedades dinâmicas associadas a um sistema de banco de dados e, quando consideradas (por exemplo em metodologias de análise e especificação de requisitos do sistema /CER 83/ /DEM 79/ /ANT 81/), normalmente apareciam como técnicas não integradas com a modelagem de dados.

Já com o surgimento dos modelos semânticos ("survey" aparece em /PEC 88/), que procuram incorporar conceitos da realidade modelada, introduzindo uma maior semântica ao modelo de dados, enfatizava-se principalmente os aspectos ligados a restrições de integridade. Nesta categoria de modelos, alguns exemplos, como TAXIS (/BOR 84/), EVENT (/KIN 84/ /KIN 86/) e SHM+ (/BRO 81/), incorporam a modelagem de transações de banco de dados.

Sob o enfoque da modelagem de transações, neste capítulo são definidas as características desejáveis para uma linguagem de especificação.

Com a intenção de determinar um perfil recomendado para estas linguagens, estabeleceram-se critérios básicos e critérios complementares que viabilizam ou asseguram os critérios básicos. Assim, são propostos:

Critérios Básicos:
- .ser independente de implementação;
- .possuir uma homogeneidade em relação à modelagem de dados;
- .permitir modelagem das dependências entre transações (paralelismo/serialização);
- .permitir tratamento de objetos individuais e de conjuntos de objetos;
- .permitir descrever o tratamento de exceções;
- .permitir descrição precisa e sem ambiguidades;

Critérios Complementares:
.ser declarativa (não procedural);
.permitir modelagem tipo causa/efeito;

Estes critérios foram adotados em vista de alguns fatores determinantes. Para cada critério, os fatores estão a seguir explicitados.

a) Independência de implementação:

Em se tratando da obtenção de um modelo conceitual das transações do sistema, é importante que os detalhes de implementação sejam, ao máximo, abstraídos na fase de modelagem. Esses detalhes, durante a fase de projeto e/ou durante uma tradução da especificação em código executável, devem ser gradativamente adicionados à especificação. Portanto, os detalhes de implementação devem ser postergados para a fase de projeto do sistema.

b) Homogeneidade com modelagem de dados:

Pela definição atual de modelo conceitual (/ISO 82/ /HEU 89/), considera-se este como sendo um modelo no qual são descritas tanto as propriedades estáticas quanto as propriedades dinâmicas do sistema (requisito de totalidade). Com a necessidade de incluir também propriedades dinâmicas neste modelo conceitual, torna-se natural a busca de formas homogêneas de descrever dados e transações de um sistema, culminando com uma integração em um único modelo. A busca dessa integração passa por uma padronização no método de especificação completa do sistema além de perseguir objetivos como uniformidade nos modelos e facilidade na aprendizagem de uma ferramenta de modelagem.

c) Modelagem das dependências entre transações:

Além da modelagem de cada transação individualmente, como uma operação atômica, torna-se importante também a representação da relação de paralelismo/serialização existente entre o conjunto de transações do sistema. Aplica-se o conceito de paralelismo quando duas transações são independentes entre si

e o de serialização quando uma transação depende da outra. Esta modelagem permite a visualização do fluxo de controle dentro do sistema.

d) Tratamento de objetos individuais e de conjuntos de objetos:

As visões que o usuário tem dos dados de um sistema são não normalizadas (não obedecem a primeira forma normal /DAT 86a/), ou seja, o usuário vê os dados como combinação de dados individuais (atômicos) e conjunto de dados (arranjos, grupos repetitivos de dados) em uma mesma visão. Já para o armazenamento destes dados, na base de dados, é interessante que eles sejam submetidos ao processo de normalização /DAT 86a/ para minimizar redundâncias. Portanto, um sistema deve conviver tanto com dados normalizados quanto com dados não normalizados e a linguagem de especificação do sistema deve conter mecanismos que garantam a perfeita descrição de ambos os tipos de dados. Deve permitir, também, a manipulação de elementos individuais do conjunto.

e) Tratamento de exceções:

Uma linguagem de especificação muitas vezes enfatiza somente os mecanismos de descrição do comportamento normal de um sistema, menosprezando as situações de exceção e as alternativas de tratamento destas exceções. Para ser completa, uma linguagem deve incorporar, em seu elenco de construções, também formas de indicar as situações de erro e as ações a serem tomadas para contornar estas situações.

f) Não ambigüidade:

A especificação serve para comunicação entre pessoas e para uma eventual geração automática de implementação. Portanto, não deve ser ambígua. A definição do comportamento do sistema de forma completa, precisa, consistente e sem ambiguidades é conseguida a partir de algum formalismo. Assim, é importante a adoção de técnicas de descrição formal (TDF), ou de uma linguagem de especificação com base formal, para descrever as propriedades dinâmicas do sistema. Esta base formal também pode facilitar uma

posterior transformação automatizada da especificação em código executável.

g) Não proceduralidade:

As linguagens declarativas representam o conhecimento sobre o sistema de forma não algorítmica. Preocupam-se, somente, em indicar quais propriedades o sistema deve satisfazer (resultados esperados) em contraste com as linguagens imperativas que detalham procedimentos para obter estas propriedades. Uma especificação procedural pressupõe o estabelecimento de um conjunto de "passos" ou seqüência determinística de procedimentos a serem executados, enquanto que uma especificação declarativa indica fatos, ações e propriedades do sistema sem estabelecer seqüência de execução. Especificações procedurais forçam a escolha, às vezes arbitrária, de uma seqüência de passos de execução que deveria ser decidida quando da implementação. Isso conflita com a intenção de independência de implementação buscada neste trabalho. Sendo assim, a não proceduralidade na especificação é uma maneira de garantir uma independência de implementação.

h) Modelagem tipo causa/efeito:

Dentre as TDF's, existe uma classe de linguagens de especificação que baseiam-se na representação de transições de estados, descrevendo as pré e pós condições de habilitação de uma transição. Nestas linguagens aparecem claramente delineadas todas as condições que devem estar satisfeitas e todas as operações que devem estar concluídas, com sucesso, antes que seja habilitada a execução de uma transição de estados. Por outro lado, também aparecem especificadas todas as operações que podem ser executadas, como pós-condição, em conseqüência da habilitação dessa transação. A especificação é feita descrevendo as causas da ocorrência de modificações no estado do sistema e os efeitos provocados pela ocorrência dessas modificações. Portanto, é uma classe de linguagem própria para especificar transações em banco

de dados. Este tipo de modelagem é propícia exatamente para uma especificação não procedural.

Buscando linguagens de especificação que preencham o perfil desejado, no âmbito dos critérios aqui mencionados, no próximo capítulo é feita uma avaliação de exemplos de linguagens constantes na literatura.

3. LINGUAGENS DE ESPECIFICAÇÃO EXISTENTES

Da literatura é possível destacar algumas linguagens que permitem a especificação de propriedades dinâmicas em banco de dados.

Para a presente avaliação, estas linguagens não foram escolhidas de forma aleatória e arbitrária, mas por serem representativas de determinadas classes de linguagens para modelagem de propriedades dinâmicas.

Neste capítulo são analisadas estas linguagens a fim de determinar suas principais características e limitações. Isto permite verificar, para cada uma delas, o preenchimento (ou não) das características desejáveis apresentadas no capítulo anterior, e em que condições isto ocorre.

No resto do capítulo, para ilustrar exemplos de modelagem de transações nestas linguagens, será considerado um fragmento de um sistema de informação de biblioteca descrito, aqui, informalmente.

São consideradas as seguintes transações:

- aquisição de obra a partir de sugestão de um leitor;
- reserva de volume;
- empréstimo de volume.

Na transação de aquisição de obra, inicialmente são analisadas as sugestões de compra de obras feitas pelo leitor (sócio), pressupondo que ele já tenha detectado que a obra não exista na biblioteca. É feita uma consulta a catálogos de livrarias que fornecem o tipo de obra sugerida. Após a compra do exemplar, este deve ser cadastrado no sistema (entidades VOLUME e AQUISIÇÃO). Adicionalmente, o leitor que sugeriu a compra recebe, de forma automática, uma reserva para o volume adquirido.

A transação de reserva de volume ocorre quando o livro solicitado estiver emprestado no momento da solicitação. Para que a reserva possa ser efetivada, o livro (volume) e o leitor (sócio) devem estar cadastrados no sistema.

Para a transação de empréstimo do volume, são considerados o credenciamento do sócio no sistema e a disponibilidade do volume para empréstimo, ou seja, o volume deve estar cadastrado e liberado.

Considerando o exemplo acima, em seguida são analisadas linguagens representativas na especificação de transações.

a) ER-R (Entity-Relationship with Rules /TAN 91/)

Alguns autores têm proposto o uso de eventos como conceito principal para modelagem das propriedades dinâmicas do sistema ("survey" em /ANT 81/). O modelo ER-R /TAN 91/ é um exemplo recente que pode ser enquadrado nesta categoria.

O modelo ER-R é uma extensão do modelo ER, acrescentando-lhe regras do tipo situação-ação, baseadas em eventos, o que permite incorporar os aspectos comportamentais (operações) das aplicações. A notação para o diagrama ER-R (ER com regras) é a mesma do modelo ER acrescida, porém, de símbolos (paralelograma) para a representação de uma regra, bem como fluxos de entrada e saída, associados ao paralelograma, para a representação de eventos correspondentes. É um modelo baseado em pré e pós condições, segundo os autores, próprio para modelar SGBD ativos (com regras e gatilhos). Textualmente, o diagrama é especificado por: RULE-NAME, TRIGGERING-EVENT, CONDITION, ACTION e RESULTANT-EVENTS. A Figura 1 ilustra um exemplo de modelagem usando o ER-R.

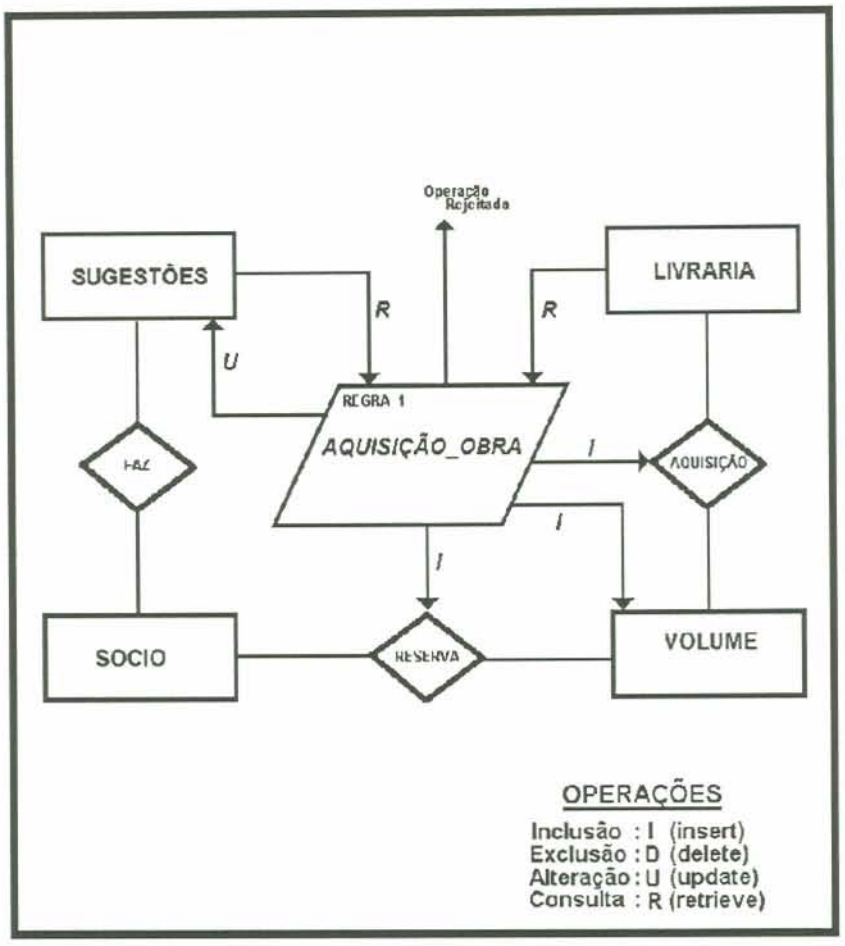


Fig. 1 - Modelagem em ER-R

Textualmente, a regra "AQUISIÇÃO_OBRA" do exemplo pode ser especificada como:

```
Rule_name: Aquisição_obra
  Triggering_event: Não_na_biblioteca
  Condition: SÓCIO (leitor) fez sugestão de compra
     Action: - consulta SUGESTÕES de compra
               (retrieve operation on SUGESTÕES)
             - rejeita operação caso não tenha SUGESTÕES
               (reject_attemped operation)
             - consulta LIVRARIA que fornece TIPO_OBRA
               (retrieve operation on LIVRARIA)
             - inclui obra em VOLUME e AQUISIÇÃO
               (insert operation on VOLUME and AQUISIÇÃO)
             - altera ESTADO_SUGEST em SUGESTÕES
               (update operation on SUGESTÕES)
             - inclui RESERVA para SÓCIO
               (insert operation on RESERVA)
     Resultant_events:{VOLUME inserted,
                       AQUISIÇÃO inserted,
```

SUGESTÕES updated,
RESERVA inserted}

Considerando as características desejáveis para uma linguagem de especificação de transações, o modelo ER-R apresenta as seguintes deficiências:

- proceduralidade encontrada na especificação das ações dos eventos;
- impossibilidade de tratamento de conjuntos;
- impossibilidade de tratamento de exceções (restrito ao tratamento implícito no SGBD);
- dificuldade na definição clara de transações e, com isto, também na modelagem das dependências entre transações (somente possui eventos e gatilhos);

Na área da modelagem semântica de dados também são encontrados alguns exemplos de linguagens de especificação de transações como TAXIS (/PEC 88/,/BOR 84/) e SHM+ (/BRO 81/,/PEC 88/).

b) SHM+(Extended Semantic Hierarchy Model /BRO 81/,/PEC 88/)

O modelo SHM+ permite a modelagem de ações individuais e de transações de banco de dados. Para as ações (ACTION), são especificados os dados de entrada e saída (IN/OUT), as pré e pós condições (PRE-CONDITION/POST CONDITION) e a operação sobre o banco de dados (DB-OPERATION). Para a especificação das transações (TRANSACTION), além dos objetos que a transação acessa (SCOPE), também são indicadas as pré e pós condições de habilitação. As ações individuais alteram um objeto específico do banco de dados a partir das primitivas INSERT, DELETE ou UPDATE. Já uma transação, através de suas ações individuais, pode alterar uma coleção de objetos.

Um exemplo da modelagem em SHM+ pode ser vista na especificação do fragmento do sistema de biblioteca apresentado a seguir.

Ação de Inclusão de Reserva:

```
ACTION: insercao-reserva (v,s)
IN     (v:volume, s:socio, n:nro-res, d:data-res)
OUT    (reserva)
PRE-CONDITION : existe-volume (v) ?
                socio-cadastrado (s) ?
                existe-empréstimo (v) ?
POST-CONDITION: reserva-do-volume (v,s,n,d) ?
DB-OPERATION  : INSERT reserva (v,s,n,d)
```

Ação de Exclusão de Reserva:

```
ACTION: exclusao-reserva (n)
IN     (n:nro-res, v:volume)
OUT    (reserva)
PRE-CONDITION : existe-reserva (n) ?
POST-CONDITION: volume-livre (v) ?
DB-OPERATION  : DELETE reserva (n)
```

Transação de conversão da Reserva em Empréstimo:

```
TRANSACTION: conv-reserva-para-emprestimo (n1)
SCOPE        (n1:nro-res, n2:nro-emp)
PRE-CONDITION : validacao-reserva (n1) ?
                volume-reserva-disponivel (n1) ?
POST-CONDITION: exclusao-reserva (n1) ?
                   /*  reserva-volume-cancelada (n1) ?   */
                inclusao-emprestimo (n2) ?
                   /*  emprestimo-volume-incluido (n2) ? */
```

Na especificação das pré e pós condições são utilizados predicados baseados na sintaxe do PASCAL/R /SCH 79/.

As limitações do modelo SHM+, em relação às características desejáveis para a linguagem de especificação de transações, são:

- proceduralidade na especificação das operações do banco de dados (DB-OPERATION);
- impossibilidade de especificação de tratamento de exceções; no SHM+ há detecção e manipulação de exceção implícita na execução dos predicados, especificados como cláusula de pré e pós condição, não havendo possibilidade de explicitar um tratamento de exceções em validações preventivas de restrições de integridades. Em caso de falha no teste dos predicados, a operação é cancelada e mecanismos do SGBD são acionados para desfazer efeitos de ações incompletas ou mal sucedidas das operações sobre o banco de dados (o tratamento de exceções fica, portanto, sob total responsabilidade do SGBD).

c) TAXIS (/PEC 88/ , /BOR 84/)

Taxis permite a modelagem tanto das propriedades estáticas quanto das dinâmicas e está baseado em conceitos da

abordagem orientada a objetos. Tem características de generalização/especialização, agregação e classificação, mas não tem encapsulamento. A modelagem da dinâmica do sistema é feita via especificação de transações (TRANSACTION <nome> WITH) indicando, além de parâmetros, as pré-condições (PREREQUISITES), as ações (ACTIONS) e o tratamento de exceções (FOR EXCEPTION). Em TAXIS, o tratamento de exceções é feito mediante a técnica de especialização aplicada sobre a descrição das mensagens de erro. Existe também um tratamento implícito nos pre-requisitos da transação, ou seja, sempre que um pré-requisito não for satisfeito, uma exceção é disparada e a execução da transação fica suspensa. Um manipulador de exceções genérico é definido para cada transação, sendo que especializações de exceções também podem ser por ele tratadas.

Um exemplo de modelagem em TAXIS pode ser visto a seguir.

```
dataclass VOLUME with
  attributes
     CODV     : 0...99999;
     TITULO   : string;
     EDITOR   : PUB_EDIT;
     LEITOR   : set of PESSOAS;
          .
          .
          .
end VOLUME;

transaction AQUISICAO with
  parameters
     v: VOLUME;
     b: BIBLIOTECA;
  prerequisites
     Nao_na_biblioteca?: (v not_in b.aquisicoes);
     Nao_ordenado?     : (v not_in b.lista_ord_vol);
  actions
     a1: add v to the b.lista_ord_vol;
          for exception e in AQUISICAO_EXCEPTION
              with vol <- v, bib <- b
          use EX_HANDLER(e);
end AQUISICAO;
```

As restrições de TAXIS, quanto ao perfil desejado para a linguagem de especificação de transações, se situam no âmbito de:

- proceduralidade na especificação das ações (ACTION), indicando uma seqüência determinística das operações;
- a modelagem das dependências entre transações se limita à definição de especializações de transações.

Há algumas linguagens baseadas em redes de Petri que exemplificam especificações do tipo causa/efeito. A rede de Petri, além da capacidade de modelar graficamente as atividades de um sistema a um nível conceitual e de forma declarativa, também permite especificar concorrência e sincronização.

Como exemplos, podem ser citadas as redes RPO (/GAR 90/) e ESDM (/CHA 91/).

d) RPO (Rede de Petri a Objetos /GAR 90/)

No modelo RPO (PNO na versão em inglês), as redes de Petri tipo predicado/transição /GEN 87/ são acrescidas de estruturas de dados e operações do paradigma de orientação a objetos (classes, instâncias, mensagens) a fim de superar as restrições relativas a representação de dados presentes nas redes clássicas. Utilizam as regras de produção como forma natural de representar textualmente o modelo RPO.

Um exemplo de modelagem em RPO pode ser visto na Figura 2.

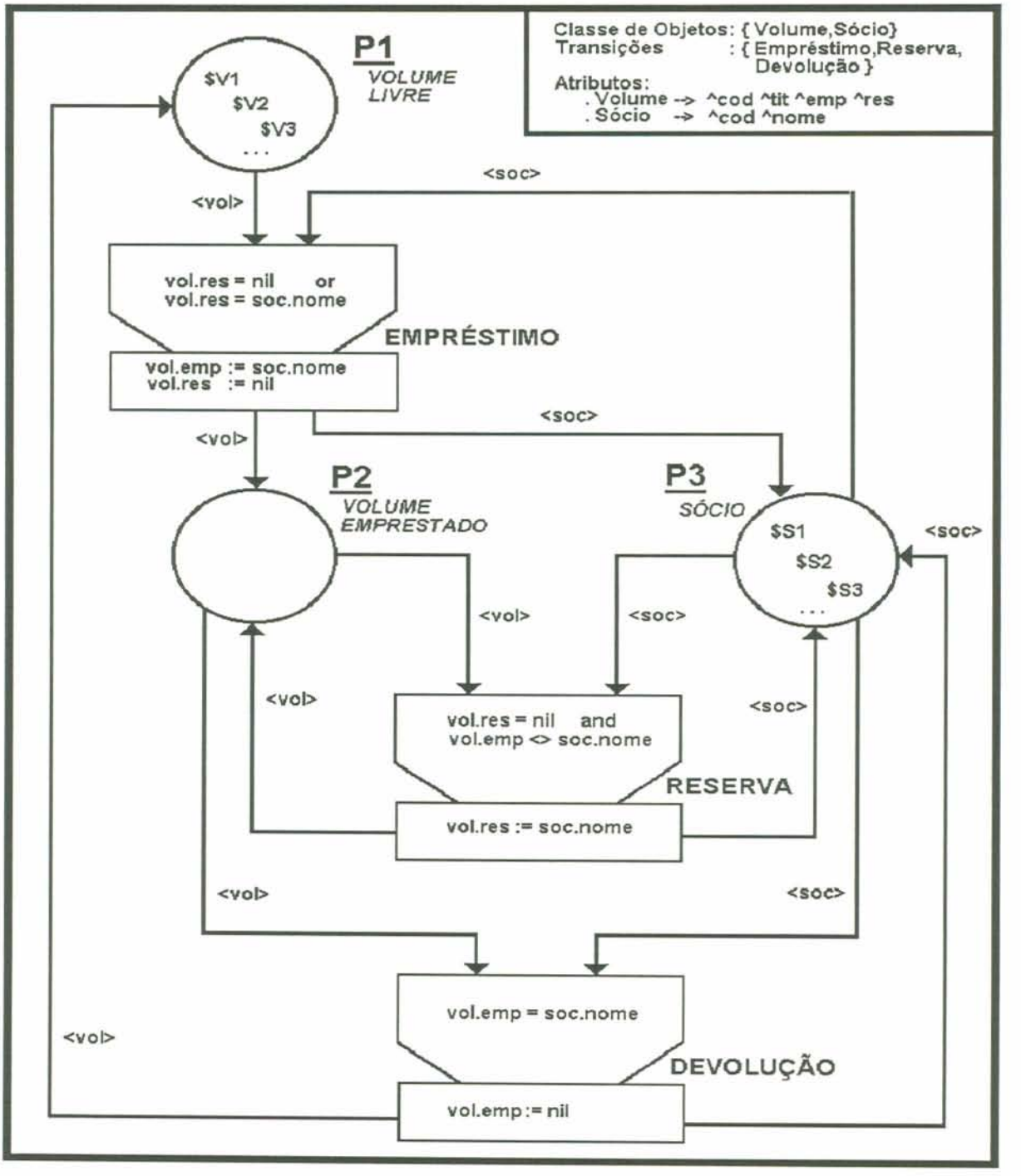


Fig. 2 - Modelagem em RPO

No exemplo, a transição "RESERVA" pode ser representada textualmente através da seguinte regra de produção:

```
(RULE reserva
  IF  Vol    : X
      Soc    : Y
      [X ^lugar:p2 ^res: ()]          (P2)
      [X ^emp:[Y ^lugar:p3 ^nome]]
  THEN
      [X ^lugar:p2]
      [Y ^lugar:p3]
      [X ^res:[Y ^nome]] )
```

Considerando as características desejáveis para uma linguagem de especificação de transações, as limitações no modelo RPO aparecem em:

- não homogeneidade em relação a modelagem de dados (a estrutura de dados(tokens) utilizada é inadequada para uma modelagem conceitual de banco de dados em vista de não permitir descrever relacionamentos entre os dados (atributo com entidade e entidade com entidade));
- impossibilidade de tratamento de conjuntos;
- impossibilidade de tratamento de exceções;
- dificuldade em delinear transações sobre o banco de dados sendo modelado e, com isto, também inviabiliza a modelagem das dependências entre transações.

e) ESDM (Executable Semantic Data Model /CHA 91/)

Em relação ao modelo ESDM, pode-se afirmar que permite a modelagem de aspectos estruturais (dados) e de aspectos comportamentais (funções) de aplicações em banco de dados. Para modelagem estrutural, utiliza uma extensão do modelo ER (com entidades, relacionamentos, atributos, agregações e hierarquia de subset). Para modelagem do comportamento, utiliza o mecanismo das redes de Petri (tipo transições e lugares). Associado a cada transição, pode haver textualmente especificado uma lista de parâmetros (dados de entrada), as pré-condições (condições de dados no banco de dados para permitir a habilitação) e as pós-condições (modificações dos dados do banco de dados após a transição).

A Figura 3 mostra um exemplo de modelagem em ESDM sobre o fragmento do sistema de bibliotecas.

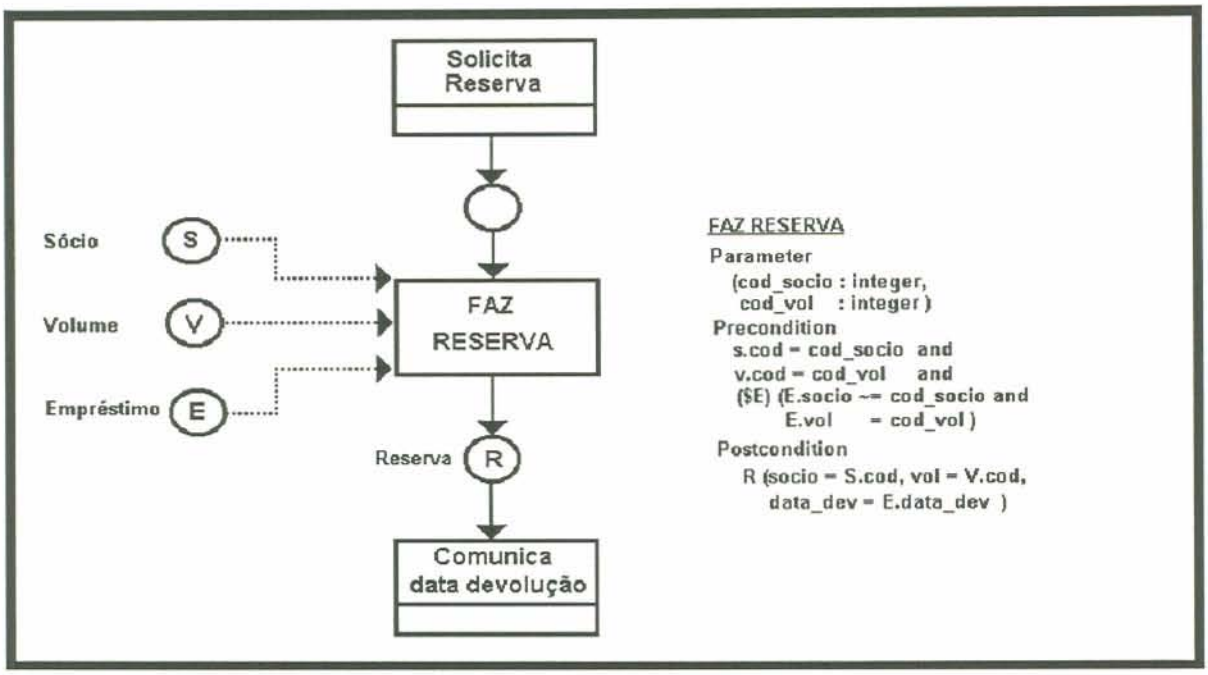


Fig. 3 - Modelagem em ESDM

As limitações constatadas no modelo ESDM, em relação ao perfil desejado para a linguagem de especificação de transações, são:

- impossibilidade de tratamento de conjuntos;
- impossibilidade de tratamento de exceções;
- dificuldade em delinear transações sobre o banco de dados sendo modelado e, com isto, também inviabiliza a modelagem das dependências entre transações.

Avaliando as características das linguagens aqui apresentadas e confrontando com o perfil desejado para a linguagem de especificação de transações, constata-se que as principais limitações situam-se no âmbito da:

- impossibilidade de tratamento de conjuntos presente em ER-R, RPO e ESDM;
- impossibilidade de tratamento de exceções presente em ER-R, SHM+, RPO e ESDM;
- proceduralidade na especificação presente em ER-R, SHM+ e TAXIS;
- dificuldade na modelagem das dependências entre transações presente em ER-R, TAXIS, RPO e ESDM;

- falta de homogeneidade com o modelo de dados presente em RPO.

Estas limitações merecem atenção especial no próximo capítulo, onde é proposta uma linguagem alternativa para os propósitos aqui enumerados.

Na Figura 4 aparece ilustrado um quadro comparativo entre os diversos modelos discutidos neste capítulo. No quadro são apresentadas, de forma mais sistemática, as limitações detectadas nestes modelos no âmbito das caracteristicas desejáveis para uma linguagem de especificação de transações.

	PROPRIEDADES DESEJÁVEIS	MODELOS ANALISADOS				
		ER-R	SHM+	TAXIS	RPO	ESDM
B	INDEPENDÊNCIA DE IMPLEMENTAÇÃO	- Prejudicada pela proceduralidade	- Prejudicada pela proceduralidade	- Prejudicada pela proceduralidade		
B	HOMOGENEIDADE COM MODELAGEM DE DADOS				- Estrutura de dados (tokens) não é adequada para modelagem conceitual de BD	
B	MODELAGEM DAS DEPENDÊNCIAS ENTRE TRANSAÇÕES	- Demarcação de transações não é clara		- Somente especializações de transações	- Demarcação de transações não é clara	- Demarcação de transações não é clara
B	TRATAMENTO DE OBJETOS INDIVIDUAIS E CONJUNTOS	- Não trata conjuntos			- Não trata conjuntos	- Não trata conjuntos
B	TRATAMENTO DE EXCEÇÕES	- Somente implícito ao SGBD	- Somente implícito na execução dos predicados da linguagem		- Não trata exceções	- Não trata exceções
B	NÃO AMBIGUIDADE					
C	NÃO PROCEDURALIDADE (Facilita a independência de implementação)	- Procedural na especificação de ações de eventos	- Procedural na especificação das operações do BD	- Procedural na especificação das ações		
C	MODELAGEM CAUSA/EFEITO (Própria para a não proceduralidade)					

— SATISFAZ
— TEM LIMITAÇÕES

B — CRITÉRIOS BÁSICOS
C — CRITÉRIOS COMPLEMENTARES

Fig. 4 - Quadro comparativo de linguagens de especificação

4. UMA LINGUAGEM DE ESPECIFICAÇÃO DE TRANSAÇÕES BASEADA EM PRÉ E PÓS CONDIÇÕES: ER/T$^+$

Neste capítulo é proposta uma linguagem de especificação de transações em banco de dados que procura representar o comportamento do sistema através de transações especificadas de forma declarativa e dentro de padrões homogêneos com a abordagem ER de modelagem de dados. Ela procura atingir os requisitos desejáveis citados no capítulo 2.

Esta linguagem resulta da compilação das principais idéias encontradas em linguagens como Taxis /BOR 84/, analisadas anteriormente, combinadas com as características de especificação tipo causa/efeito próprias para representação de transações. A proposta aqui apresentada estende e consolida uma linguagem diagramática (ER/T) anteriormente definida em /PER 90/. Para distinguir a linguagem aqui proposta da linguagem original, passa-se a usar a denominação ER/T$^+$.

A linguagem ER/T original foi modificada da seguinte forma:

- os conceitos foram precisados e detalhados;
- foram definidas uma sintaxe gráfica e uma sintaxe textual (ER/T somente possuia sintaxe gráfica, não havia sintaxe da parte textual);
- o tratamento de exceções foi revisto e incrementado;
- o tratamento de conjuntos foi detalhado e refinado.

O ER/T$^+$ possui duas formas de representação: a textual e a gráfica. Ambas permitem a construção de modelos equivalentes. A representação textual é voltada à enumeração de detalhes na especificação do sistema, enquanto que a representação gráfica permite uma fácil visualização do relacionamento entre os dados e as transações que manipulam estes dados.

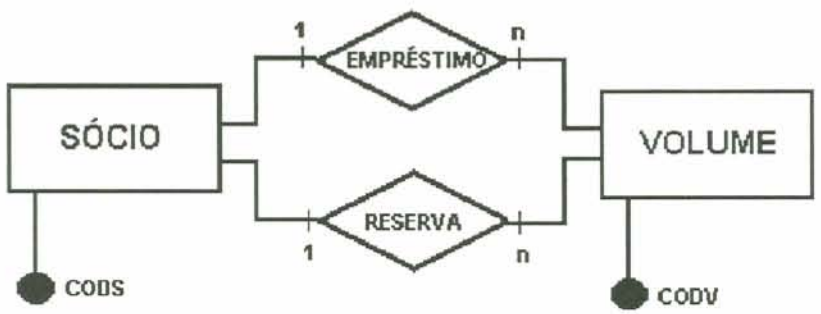
4.1 Visão geral do modelo ER/T+

A abordagem ER/T+ integra os dois aspectos básicos de especificação de sistemas em banco de dados: a do Modelo de Dados e a do Modelo de Transações.

Com o intuito de tornar didática a ilustração de detalhes sobre a linguagem, a sintaxe e semântica do modelo ER/T+ são apresentadas através de um exemplo.

4.1.1 Um exemplo ilustrando sintaxe e semântica

A modelagem de transações é aqui exemplificada sobre um pequeno fragmento de um sistema de automação de bibliotecas. O exemplo propõe a modelagem de uma transação de reserva de um determinado livro (volume) em uma biblioteca. Na realidade modelada nesta transação, o leitor (sócio) só pode realizar a reserva de um volume se este, no momento da solicitação, estiver emprestado e se já não houver uma reserva para o volume por outro leitor (outro sócio). Para esta transação, o fragmento do diagrama ER a ser utilizado é o seguinte (conforme notação de /SET 89/):



A especificação completa das características dos atributos (tipo, máscara de edição, obrigatoriedade, limites e valores válidos, etc.) das entidades e relacionamentos é mantida em um dicionário de dados. Esta especificação é feita, durante o desenho do diagrama ER/T+, numa função de definição de atributos de um editor diagramático (modelagem conceitual).

Considerando as entidades SÓCIO e VOLUME e os relacionamentos EMPRÉSTIMO e RESERVA, vinculando as duas entidades, a execução da transação de reserva inclui as seguintes atividades, não necessariamente nesta ordem:

* código_sócio e código_volume informados por um agente externo (tela) SÓCIO;
* teste de presença do sócio na entidade SÓCIO;
* teste de presença do volume na entidade VOLUME;
* teste de presença de <outro_sócio,volume> no relacionamento EMPRÉSTIMO;
* teste de ausência de <sócio,volume> no relacionamento RESERVA;
* inclusão de <sócio,volume> no relacionamento RESERVA;
* obtenção da data_devolução a partir do atributo <outro_sócio,volume>.data_dev;
* data_devolução retornado ao agente externo SÓCIO;

Neste fragmento do sistema, as estruturas de dados necessárias, modeladas segundo a abordagem ER, podem ser representadas textualmente como:

```
entidade SÓCIO
 ( cods   : num(4);
   nome   : char(30);
      .
      .
      .
 ) chave é cods;

relacionamento EMPRÉSTIMO
 (
   data_dev: num(6);
      .
      .
      .
 ) chave é (cods,codv)
   cardinalidades (1:N);
```

```
entidade VOLUME
 ( codv  : num(5);
   título: char(20);
      .
      .
      .
 ) chave é codv;

relacionamento RESERVA
 (
      .
      .
      .
 ) chave é (cods,codv)
   cardinalidades (1:N);
```

Uma modelagem completa ER/T$^+$ deve possuir uma "parte ER" na qual entidades, relacionamentos e atributos são definidos. Ela não foi aqui especificada por se tratar de conceitos bastante conhecidos (ver, por exemplo, literatura em /CHE 76/, /KOR 93/). Assim, supondo a existência da "parte ER", textual ou gráfica, a transação de reserva, usada como exemplo, pode ser formalmente especificada, em ER/T$^+$ textual, como:

```
transação RESERVA com
 entrada:SÓCIO (código_sócio,código_volume);
 precondição:
    Presença:SÓCIO (sócio);
    Presença:VOLUME (volume);
    Presença:EMPRÉSTIMO (<outro_sócio,volume>);
    Restrição: (sócio.cods  = código_sócio e
                volume.codv = código_volume e
                outro_sócio not = código_sócio);
 efeito:
    Atribuição: (data_devolução =
                      <outro_sócio,volume>.data_dev);
    Inclusão:RESERVA (<sócio,volume>);
    Saída:SÓCIO (data_devolução);
 fim_transação RESERVA;
```

As cláusulas associadas a **"precondição"** e a **"efeito"** são aqui especificadas em uma seqüência arbitrária irrelevante, cabendo a um algoritmo de resolução de dependência de dados (analisado no capítulo 5) determinar a seqüencialização dos procedimentos (proceduralização).

Normalmente, nas operações de atualização de um banco de dados, existe uma verificação prévia da existência (caso de alterações e exclusões) ou inexistência (caso de inclusões) da chave primária correspondente à entidade que está sendo manipulada. Em ER/T$^+$, na declaração de **"precondição"**, já está implícita a verificação da ausência de entidade nas operações de inclusão e a verificação da presença em operações de alteração ou de exclusão. Não é necessária a especificação explícita destes testes. Os mecanismos de tradução da especificação se encarregarão de inserir estas validações como pré-condições de execução da transação.

O parâmetro <outro_sócio> na transação RESERVA representa uma variável livre com um quantificador existencial implícito que, numa especificação ER/T$^+$, é definida a partir do padrão sintático "outro_ <nome_entidade>".

Um possível modelo mental para a execução de uma transação em ER/T$^+$ é:

.executar a entrada (declaração **entrada**);
.avaliar condições, estabelecidas como prerequisitos para habilitação e execução da transação (declaração **precondição**), que podem ser:
    .teste de presença de entidades/relacionamentos (cláusula **Presença**);
    .teste de ausência de entidades/relacionamentos (cláusula **Ausência**);
    .avaliação das restrições adicionais especificadas (cláusula **Restrição**);
    .avaliação das restrições de tempo especificadas (cláusula **Tempo**);
    .sinalização de semáforos para controle de paralelismo/serialização de transações (cláusula **Presença** e **Ausência** sobre semáforo);
.avaliar condições implícitas oriundas de ações de inclusão (gerando teste de ausência) e de exclusão e alteração (gerando teste de presença);
.caso a transação esteja habilitada (prerequisitos válidos), executar as ações especificadas na transação (declaração **efeito**) que podem ser do tipo:
    .inclusão de entidade/relacionamento (cláusula **Inclusão**);
    .alteração de entidade/relacionamento (cláusula **Alteração**);
    .exclusão de entidade/relacionamento (cláusula **Exclusão**);
    .saída de resultados (cláusula **Saída**);
    .atualização de semáforos para controle de paralelismo/serialização de transações (cláusula **Inclusão** e **Exclusão** sobre semáforo);
    .atribuição especificada conforme sintaxe (ver BNF da linguagem) (cláusula **Atribuição**).

Na representação gráfica, os símbolos de ER/T$^+$ usados na construção dos modelos são:

.conjuntos de entidades (retângulo);
.conjuntos de relacionamentos (losângos ligados por linhas às entidades);
.transações (círculos);
.fluxos de dados (setas orientadas ligando elementos da abordagem ER/T);
.agentes externos (2 retângulos semi-sobrepostos);
.semáforos (retângulos pontilhados);
.fluxos de controle (setas orientadas ligando transações a um relógio ou a um semáforo);
.anotações complementares (anotações associadas à transação).

Conforme mostra a Figura 5, a orientação das setas nos fluxos de dados tem significado próprio (inclusão, exclusão, alteração, teste). Esta semântica associada à orientação dos

fluxos foi adaptada a partir dos conceitos de **ramos alteradores** e **ramos restauradores** das redes de Petri descritas em /HEU 89/. Estas redes são do tipo condição/evento compactas (Compact Condition/Event Nets), uma variante das redes de alto nível tipo predicado/transição /GEN 87/, próprias para modelagem conceitual de sistemas.

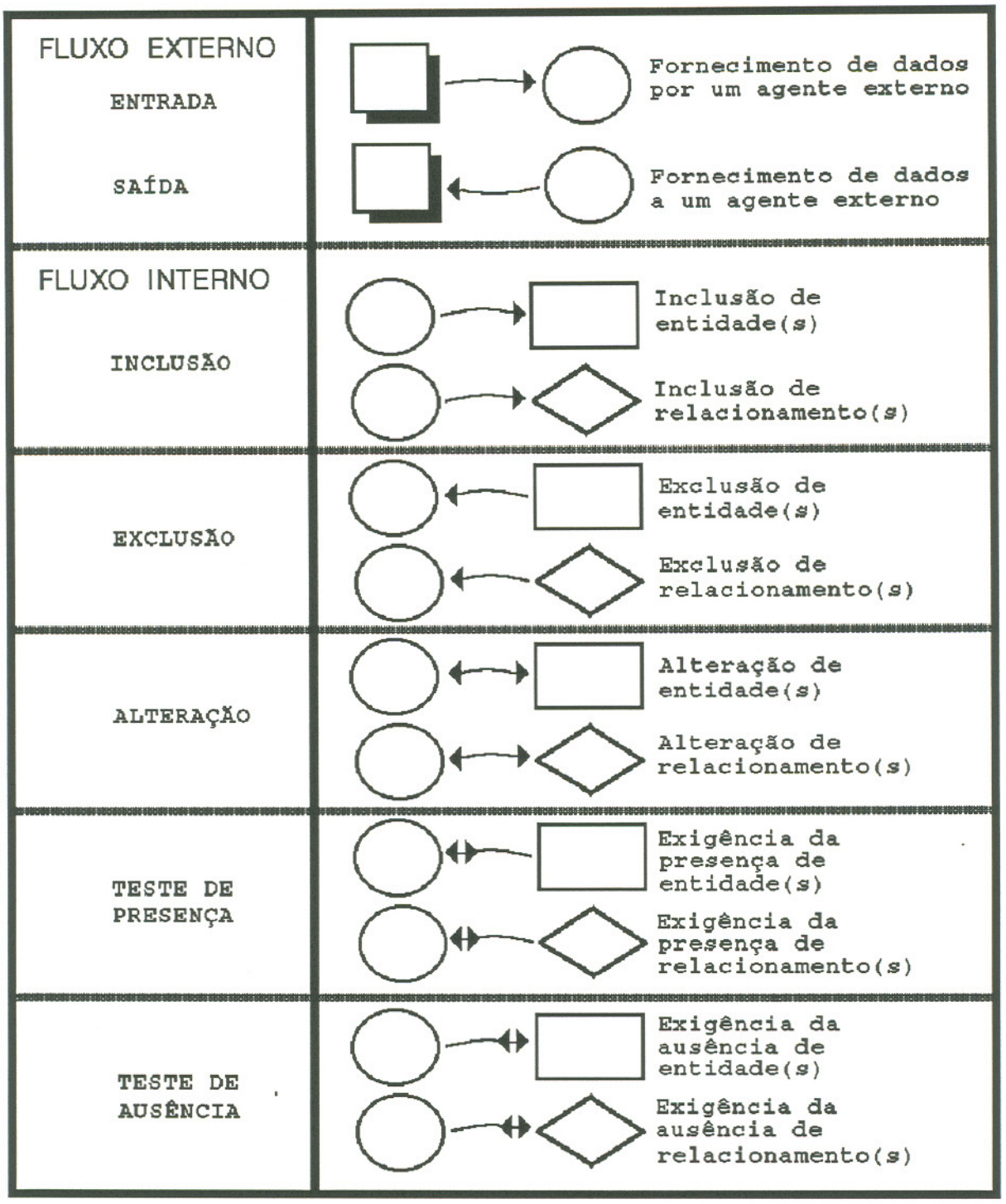


Fig. 5 - Tipos de fluxos da abordagem ER/T
(Extraído de /PER 90/)

Para cada transação é construído um diagrama. Neste diagrama é representado o símbolo da transação e todos os objetos a ele associados (entidades, relacionamentos, agentes externos, etc).

Sintaticamente, a parte do modelo que não é representada diagramaticamente é representada de forma textual. Num diagrama ER/T^+, a descrição textual correspondente à cláusula **Restrição** da declaração **precondição** aparece especificada dentro do círculo que representa a transação. Já a cláusula de **Atribuição** da declaração **efeito**, aparece no rodapé do diagrama (ver Figura 13). As anotações nos fluxos do diagrama aparecem também na sintaxe textual das correspondentes cláusulas (Presença, Ausência, etc.).

As semânticas de ER/T^+ gráfico e de ER/T^+ textual são equivalentes. Esta semântica se fundamenta nas regras de habilitação de redes de Petri de alto nível /HEU 89/.

Transportando o modelo mental de execução de uma transação ER/T^+, anteriormente apresentado, para regras de habilitação de uma transação que regem a semântica ER/T^+, estas podem ser sumarizadas em:

1- Para fluxos de exclusão/alteração/teste de presença: presença das entidades (ou relacionamentos);
2- Para fluxos de inclusão/teste de ausência: ausência das entidades (ou relacionamentos);
3- Dados de entrada presentes nos fluxos externos;
4- Restrições de integridade do E/R (1:1, 1:N, N:M) não violadas;
5- Restrições das anotações das transações não violadas.

Note que um fluxo do tipo inclusão, exclusão, alteração, teste de presença e teste de ausência na representação gráfica corresponde a uma cláusula equivalente na representação textual.

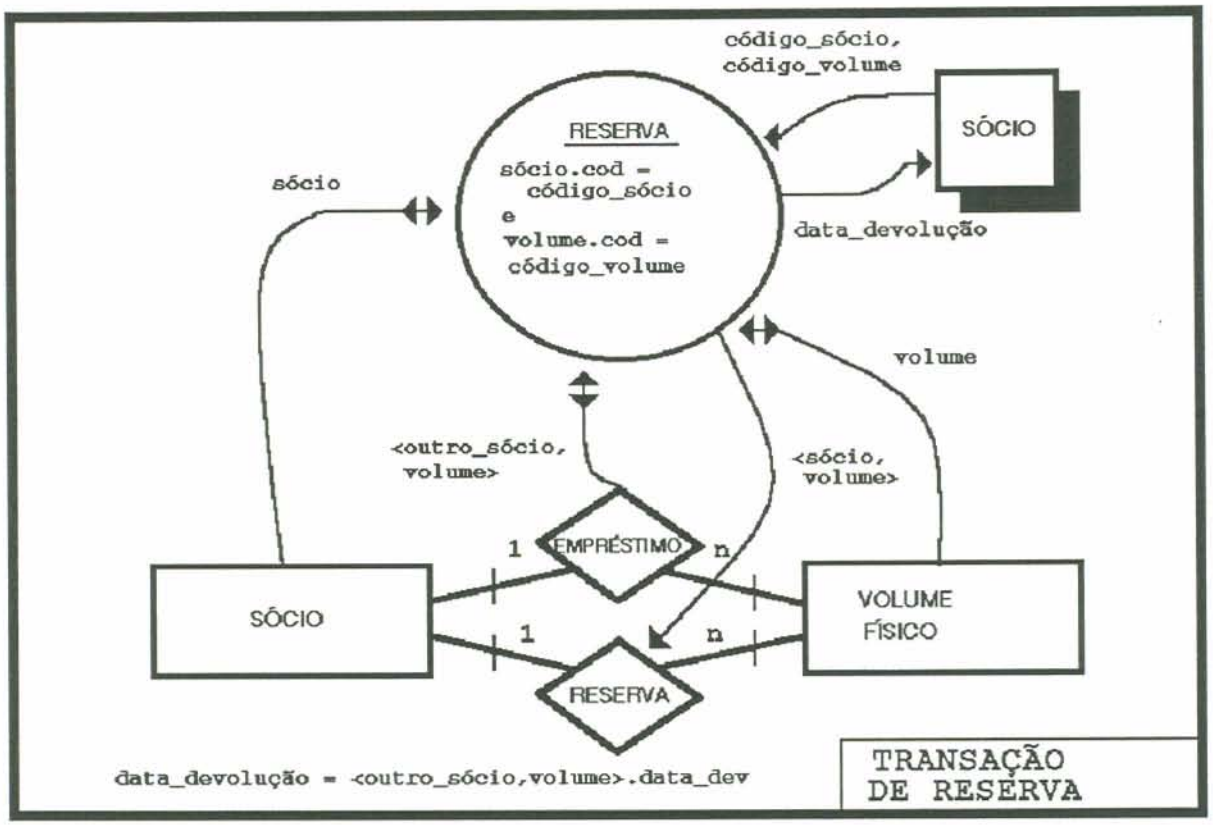


Fig. 6 - Transação de "Reserva" no modelo ER/T$^+$

No ER/T$^+$, também é possível atribuir-se uma característica de opcionalidade a um fluxo, podendo afetar diretamente o mecanismo de habilitação de uma transação. São os chamados fluxos opcionais. Esta característica de um fluxo é especificada a partir da adição de uma variável lógica entre colchetes, prefixando ou posfixando o nome do fluxo. Se colocada antes do nome do fluxo significa que o resultado da habilitação (valor verdadeiro) ou não (valor falso) deste fluxo é atribuída à variável lógica associada. No exemplo da Figura 7, o teste de presença do PRODUTO atribui o valor "verdadeiro" à variável EXISTEPROD, caso localizado o produto, e o valor "falso", caso contrário.

Por outro lado, se a variável lógica estiver posfixando o nome do fluxo, além das regras normais de habilitação dos fluxos, este fluxo também estará sujeito ao teste da variável

lógica associada, obtendo a efetiva habilitação somente se esta estiver valorada como "verdadeira". No exemplo da Figura 7, a habilitação do fluxo TAXA se efetivará somente se o teste de presença resultar em "verdadeiro" e a variável ATRASO também contiver valor "verdadeiro".

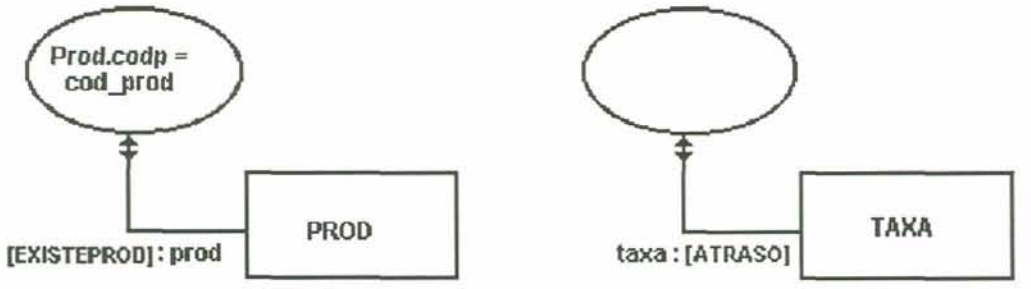


Fig. 7 - Fluxos opcionais no modelo ER/T$^+$

Textualmente, este mesmo exemplo pode ser expresso como segue:

```
transação  T1  com
    ...
 precondição:
    ...
    Presença: PROD ([EXISTEPROD]:prod);
    ...
 fim_transação T1;

transação  T2  com
    ...
 precondição:
    ...
    Presença: TAXA (taxa:[ATRASO]);
    ...
 fim_transação T2;
```

4.1.2 Tratamento de conjuntos em ER/T$^+$

Na manipulação de conjuntos nos fluxos, a linguagem ER/T$^+$ permite a representação dos itens (entidades,

relacionamentos, atributos e dados de entrada/saída) que formam os conjuntos a cada transformação, bem como a referência a cada item individualmente. Para tanto, foram definidos alguns padrões de especificação que permitem, através da combinação de uma especificação de escopo (indicação dos tipos de itens que compõem o conjunto) e de critérios de seleção, referenciar conjuntos e também itens individuais relacionados aos conjuntos. São construções sintáticas como as exemplificadas a seguir:

CONJUNTO sócio **PRESENTES**;

que indica a referência a todos elementos presentes na entidade "sócio";

CONJUNTO <sócio,volume> **PRESENTES**;

que indica a referência a todos elementos presentes no relacionamento "<sócio,volume>";

CONJUNTO volume **de** <sócio,volume> **PRESENTES**;

que indica a referência a todos diferentes valores de chaves de "volume" participantes no relacionato "<sócio,volume>";

CONJUNTO título,assunto **de** volume **PRESENTES**;

que indica uma operação de projeção da álgebra relacional sobre os atributos "título" e "assunto" da entidade "volume";

CONJUNTO título,assunto **de** volume **PRESENTES**
ONDE volume.assunto = "ficção";

que indica uma operação de projeção da álgebra relacional sobre os atributos "título" e "assunto" da entidade "volume", selecionando os elementos que satisfaçam a condição indicada (volume.assunto = "ficção");

CONJUNTO vol **de** <sócio,volume> **PRESENTES**
ONDE vol **PERTENCE A** livros_ficção;

que indica a referência a todos os diferentes valores de chave de "vol" participantes do relacio- "<sócio,volume>" e que satisfaçam a condição de um ítem individual ("vol") estar relacionado com um conjunto ("livros_ficção");

CONJUNTO <sócio,volume> **AUSENTES**
ONDE soc = sócio e
vol **PERTENCE A** volumes_do_pedido;

que indica a referência a todos os elementos ausentes do relacionamento "<sócio,volume>" determinados a partir de um universo de valores conhecidos e especificados em "volumes_do_pedido";

A Figura 8 apresenta exemplos dos padrões de manipulação de conjuntos, denotando fluxos em um diagrama ER/T$^+$. Nesta figura é exemplificada a formulação de conjuntos a partir das entidades presentes.

O exemplo mostra fluxos com todas as entidades presentes (cj_sócios = **CONJUNTO** sócio **PRESENTES**) e com todas as entidades presentes (ou ausentes) que satisfaçam a uma determinada condição (livros_de_ficção = **CONJUNTO** volume **PRESENTES ONDE** volume.assunto = 'ficção'). O nome que segue o termo CONJUNTO é a definição da(s) variável(eis) que identifica(m) quais os tipos de itens (entidades, relacionamentos, atributos) que formam o conjunto que instancia o fluxo. É através desta variável que os elementos do conjunto são referenciados. Estas variáveis são também utilizadas, quando seguem o termo ONDE, para relacionar os elementos do conjunto ao conteúdo de outros fluxos.

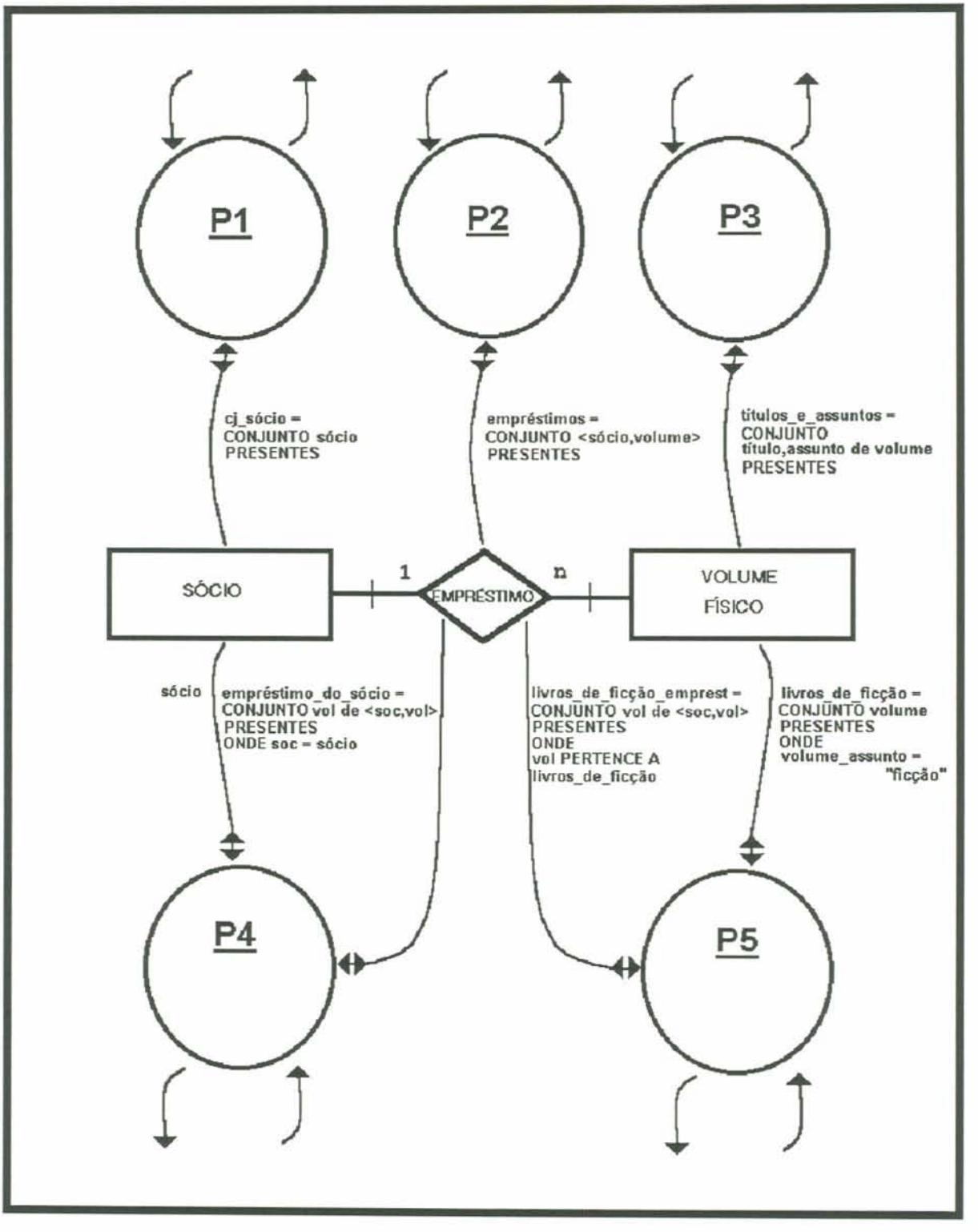


Fig. 8 - Padrões de manipulação de conjuntos no modelo ER/T$^+$ (Adaptado de /PER 90/)

Nos fluxos externos (entrada/saída), eventualmente torna-se necessário representar os dados em uma forma não normalizada. Para tanto, a abordagem ER/T$^+$ permite especificar ítens de dados atômicos combinados com conjuntos na formação destes fluxos. Assim, um fluxo pode ser denotado como mostra o exemplo a seguir.

```
Listagem = código_volume, título, autor,
           1 {leitor, data_empr, data_dev /
             PARA CADA <soc,vol> DE Empréstimo_ok :
                  leitor     = soc.cod  e
                  data_empr  = <soc,vol>.data_empr e
                  data_dev = <soc,vol>.data_dev
           }10
```

No exemplo, o fluxo de saída "Listagem" é constituído por "código_volume", "título" e "autor" de uma obra (como itens de dados atômicos), além de uma lista (como um conjunto representado entre os símbolos { e }) que referencia de 1 a 10 (mínimo/máximo elementos no conjunto) empréstimos realizados e onde cada elemento da lista de empréstimo ("leitor", "data_empr" e "data_dev") tem individualmente uma condição (PARA CADA) associada que, para o caso, é uma atribuição.

Em outro exemplo, um fluxo de entrada "Pedido_empréstimo" é definido por "código_sócio" (como item atômico) e uma lista de "código_volume" (como um conjunto de dados) representando os volumes que o leitor está solicitando para o empréstimo. O exemplo é especificado como:

```
Pedido_empréstimo = código_sócio, {código_volume}
```

4.1.3 Modelagem de sincronização de transações ER/T$^+$

Para modelar a sincronização de transações, em função do tempo (relógio) ou da ocorrência de outras transações (seqüência de transações), são utilizados os fluxos de controle ligados a relógios e a semáforos (depósito de dados de sinalização). Veja exemplos na Figura 9 e Figura 10.

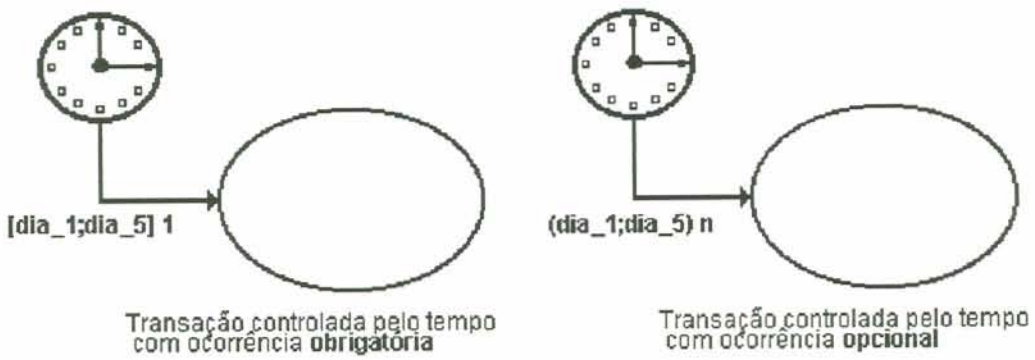


Fig. 9 - Exemplo de fluxo de controle para relógio

O exemplo da Figura 9 mostra transações controladas (liberadas) a partir de intervalos de tempo em ocorrências dos tipos obrigatória e opcional. É um exemplo de uso de tempo para sincronização de transações. A ocorrência obrigatória, especificada mediante o intervalo de tempo colocado entre colchetes e seguido de "1", define que a transação deve ocorrer necessariamente no intervalo de tempo especificado. Já a ocorrência opcional, especificada com o intervalo de tempo colocado entre parênteses e seguido de "n", define que a transação pode ocorrer e, se ocorrer, será dentro do intervalo de tempo especificado.

Na representação textual, o exemplo é especificado da seguinte maneira:

```
transação OBRIGATORIA com
   ...
precondição:
   ...
   Tempo: ( [dia_1;dia_5]1 );
   ...
efeito:
   ...
fim_transação OBRIGATORIA;
```

```
transação OPCIONAL com
    ...
 precondição:
    ...
    Tempo: ( (dia_1;dia_5)n );
    ...
 efeito:
    ...
 fim_transação OPCIONAL;
```

Na modelagem de sincronização de transações em função de outras transações (dependências entre transações) é construído um diagrama mostrando o semáforo (para sinalizar paralelismo/serialização) e as respectivas transações envolvidas (Figura 10).

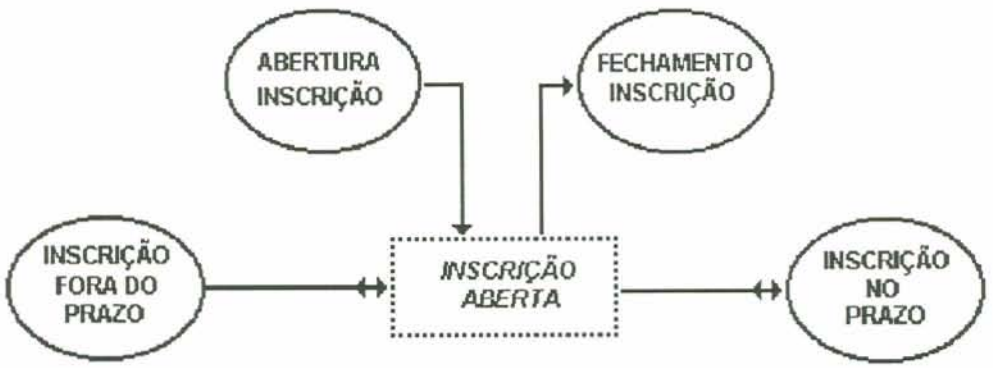


Fig. 10 - Exemplo de fluxo de controle para semáforo

No exemplo é modelada a sincronização das transações "ABERTURA_INSCRIÇÃO", "FECHAMENTO_INSCRIÇÃO", "INSCRIÇÃO_FORA_DO_PRAZO" e "INSCRIÇÃO_NO_PRAZO" em função de um fato de controle "inscrição_aberta" (especificado no semáforo). Estas transações manipulam o fato de controle conforme indicado nos respectivos fluxos de controle (inclusão, exclusão, teste de ausência e teste de presença). É um exemplo de uso de fatos ou sinais de controle para sincronização de transações, viabilizando uma modelagem de paralelismo/serialização.

Textualmente, estas transações são representadas como segue:

```
transação ABERTURA_INSCRIÇÃO com
    ...
 precondição:
    ...
 efeito:
    ...
    Inclusão: SEMÁFORO ("inscrição_aberta");
    ...
 fim_transação ABERTURA_INSCRIÇÃO;

transação FECHAMENTO_INSCRIÇÃO com
    ...
 precondição:
    ...
 efeito:
    ...
    Exclusão: SEMÁFORO ("inscrição_aberta");
    ...
 fim_transação FECHAMENTO_INSCRIÇÃO;

transação INSCRIÇÃO_FORA_DO_PRAZO com
    ...
 precondição:
    ...
    Ausência: SEMÁFORO ("inscrição_aberta");
    ...
 efeito:
    ...
 fim_transação INSCRIÇÃO_FORA_DO_PRAZO;

transação INSCRIÇÃO_NO_PRAZO com
    ...
 precondição:
    ...
    Presença: SEMÁFORO ("inscrição_aberta");
    ...
 efeito:
    ...
 fim_transação INSCRIÇÃO_NO_PRAZO;
```

Nas quatro transações modeladas no exemplo, as cláusulas de **Inclusão**, **Exclusão**, **Ausência** e **Presença** referenciam um sinalizador predefinido, denominado "SEMÁFORO", para realizar a respectiva operação de sincronização e onde string

"inscrição_aberta" é o valor da chave de acesso à entidade "SEMÁFORO".

4.1.4 Tratamento de exceções em ER/T$^+$

Durante a modelagem do sistema, o projetista pode considerar somente as situações de comportamento normal do sistema, deixando que as mensagens de erro, para as diversas situações de inconsistência, sejam derivadas automaticamente.

Para as transações que realizam o consumo dos fluxos de entrada (tratamento de entrada de dados), a abordagem ER/T$^+$ convenciona que o procedimento de tratamento de inconsistência padrão (inclusão para já existente, alteração para inexistente, etc) está implícito na especificação da transação, sendo desnecessária sua representação explícita no diagrama.

Em um diagrama ER/T$^+$ existe uma especificação implícita de tratamento de exceções (/PER 90/,/HEU 91/) configuradas pela não habilitação de uma transação que ocorre em vista de algum (ou vários) de seus fluxos impedirem esta habilitação. Assim, as exceções se habilitam quando:

- a valoração dos agentes externos de entrada não satisfaz as anotações das transações;
- a valoração dos agentes externos de entrada não satisfaz o dicionário ER;
- um objeto de entrada (presente em uma transação consistente) está ausente;
- um objeto de saída (ausente em uma transação consistente) está presente;
- restrições de integridade estáticas expressas pelo diagrama ER (por exemplo, cardinalidades) ou pelo dicionário ER (por exemplo, formatos e domínios) são violadas.

Como efeito da ocorrência de uma exceção implícita, os dados de entrada desaparecem do fluxo de entrada e uma correspondente mensagem de erro associada a exceção aparece no fluxo de saída. O banco de dados não é afetado pela exceção.

O projetista também pode, no entanto, especificar procedimentos de tratamento de exceções (círculo/losango numerado

próximo ao fluxo e detalhamento ao pé do diagrama) associados aos diferentes fluxos do diagrama ER/T$^+$ que modela a transação consistente (ver Figura 11).

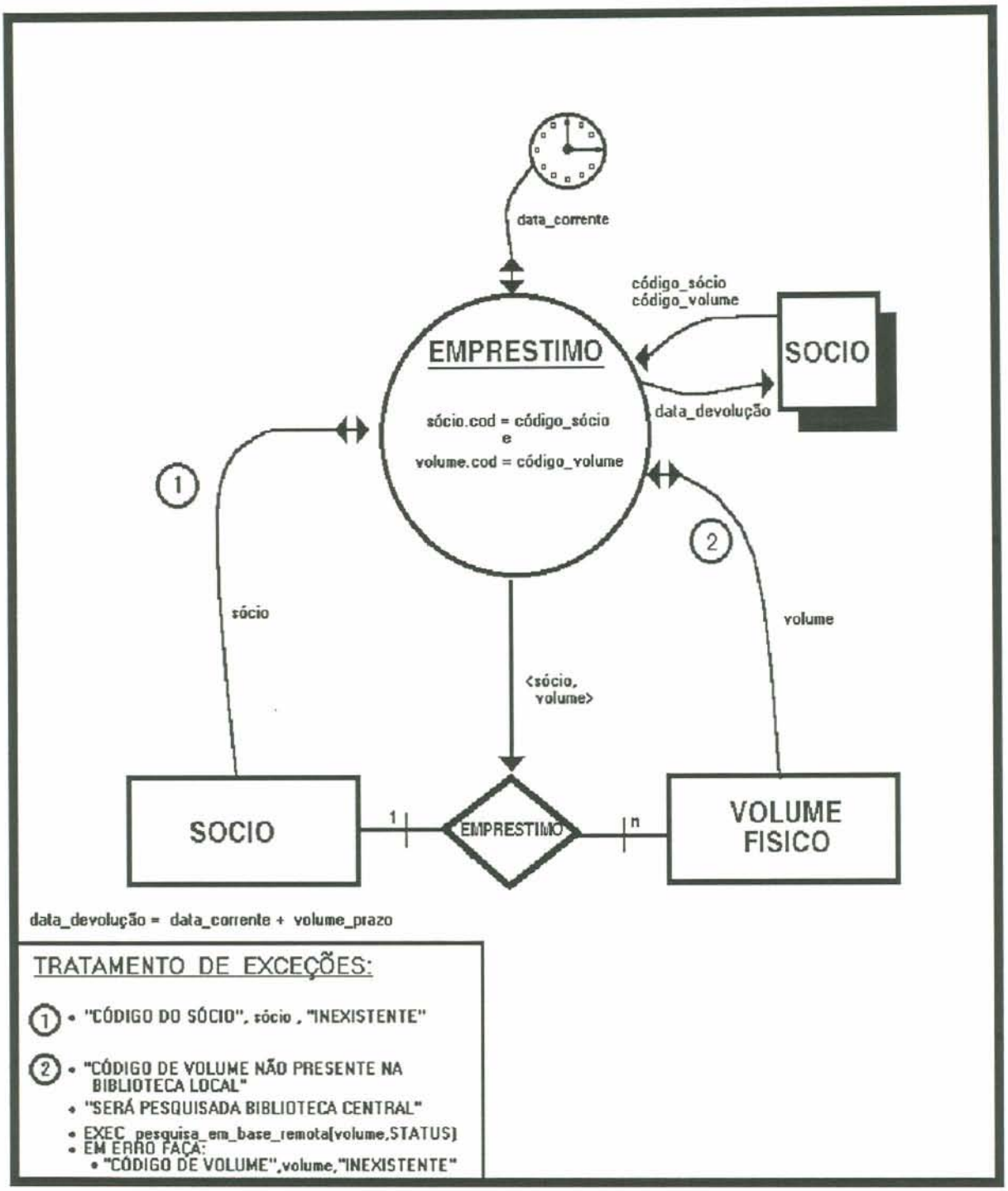


Fig. 11 - Mensagens de inconsistências indicadas na modelagem ER/T$^+$

Na representação textual, o tratamento de exceções para o exemplo pode ser especificado como segue:

```
transação EMPRÉSTIMO com
   ...
precondição:
   ...
   Presença: SÓCIO (sócio,
              EM ERRO FAÇA: "Código do Sócio", sócio,
                            "Inexistente"
                   );
   Presença: VOLUME (volume,
              EM ERRO FAÇA: "Código do Volume não"
                            " presente na Biblioteca"
                            " Local será pesquisada "
                            "Biblioteca Central"
                            EXEC pesquisa_em_base_remota
                                    (volume,STATUS)
                               EM ERRO FAÇA:
                                  "Código de Volume",
                                   volume,
                                  "Inexistente"
                         );
   ...
efeito:
   ...
   Inclusão: EMPRÉSTIMO ( <sócio,volume> );
   ...
fim_transação EMPRÉSTIMO;
```

4.1.5 A sintaxe completa de ER/T+

A sintaxe do ER/T+ está a seguir descrita em BNF (Backus Naur Form). Nessa descrição, os símbolos não terminais são representados entre "<" e ">" e os símbolos e caracteres da linguagem são escritos em negrito. Além disso, são usados os meta-símbolos:

"::="	(é definido por)
"|"	(ou)
"{" e "}"	(elementos opcionais)
"..."	(repetição do símbolo: zero ou mais vezes)

Sintaxe do modelo ER/T$^+$:

```
*------ Construções sintáticas da representação textual

<transação_BD>          ::=  transação <nome_transação> com
                              entrada: <nome_tela>
                                 ( <lista_de_dados> );
                              precondição:
                                 <tipo_cláusula_precond> ...
                              efeito:
                                 <tipo_cláusula_poscond> ...
                             fim_transação <nome_transação> ;

<tipo_cláusula_precond>::= <cláusula_presença>    |
                           <cláusula_ausencia>    |
                           <restrição_de_corpo>   |
                           <restrição_de_tempo>   |
                           <restrição_de_sinal>

<cláusula_presença>     ::=  Presença: <nome_objeto>
                               ( <escopo_e_seleção> );

<cláusula_ausência>     ::=  Ausência: <nome_objeto>
                               ( <escopo_e_seleção> );

<restrição_de_corpo>    ::=  Restrição:
                               ( <anotação_de_corpo> );

<restrição_de_tempo>    ::=  Tempo:
                               ( <nome_fluxo_tempo>  );

<restrição_de_sinal>    ::=  Presença: SEMÁFORO
                               ( <fato_de_controle>  ); |
                             Ausência: SEMÁFORO
                               ( <fato_de_controle>  );

<nome_objeto>           ::=  <nome_entidade>       |
                             <nome_relacionamento>

<escopo_e_seleção>      ::=  <fluxo_e_opcional> |
                             <fluxo_e_opcional> ,EM ERRO FAÇA:
                                          <tratamento_exceção>

<fluxo_e_opcional>       ::= <nome_fluxo_interno>  |
                             <nome_fluxo_opcional>

<tipo_cláusula_poscond>::= <cláusula_inclusão>    |
                           <cláusula_alteração>   |
                           <cláusula_exclusão>    |
                           <cláusula_saída>       |
                           <cláusula_atribuição>  |
                           <atualização_de_sinal>

<cláusula_inclusão>     ::=  Inclusão:  <nome_objeto>
                               ( <escopo_e_seleção> );

<cláusula_alteração>    ::=  Alteração: <nome_objeto>
                               ( <escopo_e_seleção> );

<cláusula_exclusão>     ::=  Exclusão:  <nome_objeto>
                               ( <escopo_e_seleção> );

<cláusula_saída>        ::=  Saída: <nome_tela>
                               ( <lista_de_dados> );

<cláusula_atribuição>   ::=  Atribuição:
                               ( <bloco_atrib> );

<atualização_de_sinal>::=    Inclusão: SEMÁFORO
                               ( <fato_de_controle>  ); |
                             Exclusão: SEMÁFORO
                               ( <fato_de_controle>  );
```

```
*------ Construções sintáticas comuns a represantação textual
*------ e a representação gráfica (anotações no diagrama)

<anotação_de_fluxo>    ::=  <nome_fluxo> {@ <nro_msg_erro>}

<nome_fluxo>           ::=  <nome_fluxo_interno> |
                            <nome_fluxo_externo> |
                            <nome_fluxo_opcional> |
                            <nome_fluxo_tempo>

<nome_fluxo_interno>   ::=  <variável_interna> |
                            <conjunto_interno>

<variável_interna>     ::=  <entidade> | <relacionamento>

<relacionamento>       ::=  < <entidade>, ... <entidade> >

<entidade>             ::=  <nome_entidade> |
                            outro_ <nome_entidade>

<conjunto_interno>     ::=  {<nome_conjunto> = }
                              <definição_presentes> |
                            {<nome_conjunto> = }
                              <definição_ausentes>

<nome_conjunto>        ::=  <entidade_temporaria> |
                            <entidade>

<definição_presentes>  ::=  CONJUNTO <nome_var> PRESENTES
                            { ONDE <condição> }

<nome_var>             ::=  <entidade> |
                            <relacionamento> |
                            <entidade>,... de <relacionamento> |
                            <atributo>,... de <entidade> |
                            <atributo>,... de <relacionamento>

<atributo>             ::=  <nome_atributo>

<condição>             ::=  <atributo> <simb.relação> <var/lit>|
                            <atributo> PERTENCE A
                                       <nome_conjunto>          |
                            <atributo> NÃO PERTENCE A
                                       <nome_conjunto>          |
                            <condição> <simb.lógico> <condição>|
                            ( <condição> )

<simb.relação>         ::=  =   |
                            >   |
                            <   |
                            =>  |
                            =<  |
                            not =

<simb.lógico>          ::=  e   | ou

<definição_ausentes>   ::=  CONJUNTO <nome_var> AUSENTES
                              ONDE <seleção> E
                              <atributo> PERTENCE A
                                         <nome_conjunto>

<seleção>              ::=  <atributo> <simb.relação> <var/lit>

<var/lit>              ::=  <variável> | <literal>

<variável>             ::=  <atributo> | <dado_externo>

<literal>              ::=  <constante>

<nome_fluxo_externo>   ::=  <lista_de_dados>
```

```
<lista_de_dados>        ::=  <variável> |
                             <lista_de_var> |
                             <entidade_temporária> |
                             {<min>}{ <lista_de_var> }{/<max>} |
                               <iteração_individual>
                             <lista_de_dados> , <lista_de_dados>

<lista_de_var>          ::=  <variável>, ... <variável>

<iteração_individual>   ::=  PARA CADA <nome_var> : <bloco_atrib>

<bloco_atrib>           ::=  <atribuição> |
                             <atribuição> e <atribuição>

<atribuição>            ::=  <variável> = <literal> |
                             <variável> = <expressão> |
                             <nome_conjunto> =
                                {<min>}{ <lista_de_var> }{/<max>}
                                <iteração_individual>

<nome_fluxo_opcional>   ::=  <fluxo> : <opcionalidade> |
                             <opcionalidade> : <fluxo>

<fluxo>                 ::=  <nome_fluxo_interno>

<opcionalidade>         ::=  [ <variável lógica> ]

<nome_fluxo_tempo>      ::=  ( <intervalo> )n  |
                             [ <intervalo> ]1

<intervalo>             ::=  <inicio_interv.> ; <fim_interv.>

<início_interv.>        ::=  <tempo>

<fim_interv.>           ::=  <tempo>

<anotação_de_corpo>     ::=  <seleção> |
                             <seleção> <simb.lógico> <seleção>

<anotação_de_rodapé>    ::=  <atribuição> |
                             <definição> |
                             <iteração_individual>

<definição>             ::=  <nro_msg_erro>:<tratamento_exceção>|
                             <nome_conjunto>:<definição_conjunto>

<tratamento_exceção>    ::=  <mensagem_erro> |
                             EXEC <rotina_erro> EM ERRO FAÇA:
                                                <tratamento_exceção>

<definição_conjunto>    ::=  <definição_presentes> |
                             <definição_ausentes>
```

Definições adicionais:

<nome_transação> é o nome atribuído a uma transação;

<nome_tela> é o nome de um agente externo tipo tela ou formulário;

<nome_entidade> é o nome de uma entidade;

<nome_relacionamento> é o nome de um relacionamento;

<dado_externo> é um nome de um item de dado de entrada ou de saída;

<nome_atributo> é o nome de um atributo de uma entidade ou de um relacionamento;

<entidade_temporária> é o nome de uma entidade temporária usada somente durante o processamento;

<constante> é uma cadeia de caracteres numéricos ou uma cadeia de caracteres alfanuméricos delimitados por apóstrofos;

<dado_externo> é um nome de um item de dado de entrada ou de saída;

<expressão> é uma expressão aritmética possuindo, em seu escopo, <variável> (eis) e <literal> (ais);

<variável lógica> é uma variável, escrita em letra maiúscula, que pode assumir os valores 'verdadeiro' ou 'falso';

<nro_msg_erro> é um código de erro que permite associar uma mensagem de inconsistência a um determinado fluxo;

<tempo> é uma expressão de tempo do tipo "data" ou "hora";

<mensagem_erro> é uma mensagem de inconsistência;

<rotina_erro> é uma rotina pre-definida para tratamento de exceções;

<min> é um número inteiro que indica ocorrência mínima.

<max> é um número inteiro que indica ocorrência máxima.

<fato_de_controle> é uma cadeia de caracteres alfanuméricos (string) que indica um fato de sinalização de transações.

As equivalências entre as construções sintáticas das representações textual e gráfica aparecem relacionadas a seguir.

Textual	Gráfica
cláusula entrada	Fluxo externo de entrada
cláusula Saída	Fluxo externo de saída
cláusula Inclusão	Fluxo interno de inclusão
cláusula Alteração	Fluxo interno de alteração
cláusula Exclusão	Fluxo interno de exclusão
cláusula Presença	Fluxo de teste de presença
cláusula Ausência	Fluxo de teste de ausência
cláusula Restrição	Anotações complementares
cláusula Tempo	Fluxo de controle (Relógio)

Numa comparação da representação textual com a gráfica, cabe frisar que as construções sintáticas das diversas cláusulas na representação textual incorporam as anotações textuais feitas no diagrama, os objetos associados à transação (entidades, relacionamentos, etc.), que estão graficamente representados, e a semântica das ações também indicadas graficamente no diagrama.

4.2 ER/T^{+} como linguagem de especificação de banco de dados

A motivação que fundamentou a escolha da abordagem ER/T^{+} nesta tese deve-se, basicamente, às seguintes características:

- .É uma especificação não procedural do tipo causa e efeito, refletindo os estados de um sistema pré e pós transição de estados;
- .preenche o requisito de independência de implementação;
- .permite a manipulação de conjuntos;
- .permite especificar tratamento de exceções;
- .é um modelo completo que permite representar dados e transações de forma homogênea;
- .própria para a especificação de sistemas transacionais;
- .tem base formal o que pode viabilizar uma transformação automatizada para código executável;

.aparece nas modalidades gráfica e textual;

Adicionalmente, a escolha foi reforçada pelo fato da representação gráfica da linguagem ser de relativa simplicidade o que facilita o seu uso, por parte do projetista, e a interação com o usuário. É de fácil assimilação por parte do projetista, pois se fundamenta em conceitos já bastante difundidos (E/R, DFD e redes).

Assim, o modelo ER/T^+ foi escolhido como **modelo de partida** para a realização dos estudos de implementação de especificações.

Cabe salientar que procurou-se com o modelo ER/T^+ incorporar, na linguagem de especificação, algumas características desejáveis para a especificação de transações e, a partir dele, investigar uma metodologia de especificação e de implementação automática de transações em banco de dados.

Não está sendo aqui afirmado que este é um formalismo ideal para a modelagem de sistemas, menosprezando metodologias que vêm sendo estudadas exaustivamente pela comunidade acadêmica como a de orientação a objetos, mas sim que a abordagem ER/T^+ pode ser considerada como representante de especificações não procedurais, para validar as idéias relativas à implementação de especificações. Posteriormente, estas idéias podem ser aplicadas a outros formalismos.

4.3 Conclusão

Neste capítulo foram apresentadas as principais características de uma linguagem gráfica e textual de especificação de transações baseada em pré e pós condições: o modelo ER/T^+.

O modelo proposto resulta da modificação de uma linguagem baseada na semântica de execução de redes de Petri originalmente definida em /PER 90/. A partir da versão original, foram incorporadas melhorias no sentido de atender aos requisitos

desejáveis para uma linguagem de especificação de transações discutidos no capítulo 2.

Assim, investiu-se em:

.refinamento das construções sintáticas para tratamento de conjuntos;
.incremento de funções para tratamento de exceções;
.definição de uma sintaxe textual para a linguagem;
.revisão e detalhamento das facilidades de modelagem de sincronização de transações.

Além da incorporação destas facilidades na linguagem ER/T^+, foi realizada uma descrição completa de sua sintaxe.

O modelo ER/T^+ proposto neste capítulo busca enquadrar-se no perfil de linguagem descrito no capítulo 2, mediante uma série de mecanismos de modelagem e facilidades presentes em suas construções sintáticas. O quadro a seguir ilustra como estas características são viabilizadas no modelo ER/T^+.

	PROPRIEDADES DESEJÁVEIS	PERFIL DO MODELO ER/T+ COMO AS PROPRIEDADES SÃO VIABILIZADAS
B	INDEPENDÊNCIA DE IMPLEMENTAÇÃO	- Através do uso de uma linguagem declarativa permitindo uma modelagem não procedural do sistema.
B	HOMOGENEIDADE COM MODELAGEM DE DADOS	- Através da integração de dados e transações representados de forma homogênea e com semântica equivalente nas representações gráfica e textual.
B	MODELAGEM DAS DEPENDÊNCIAS ENTRE TRANSAÇÕES	- Através das cláusulas de sincronização de transações (fluxo de controle : Relógio e Semáforo).
B	TRATAMENTO DE OBJETOS INDIVIDUAIS E CONJUNTOS	- Através de padrões de especificação associados à cláusula "CONJUNTO" definindo escopo e critérios de seleção.
B	TRATAMENTO DE EXCEÇÕES	- Implícita na não habilitação de uma transação ou explicitada por cláusula "EM ERRO FAÇA: " associada ao fluxo.
B	NÃO AMBIGUIDADE	- Tem base formal fundamentada na semântica das redes de Petri.
C	NÃO PROCEDURALIDADE (Facilita a independência de implementação)	- Construções sintáticas da linguagem ER/T+ são declarativas.
C	MODELAGEM CAUSA/EFEITO (Própria para a não proceduralidade)	- Transação modelada com base em declarações do tipo "PRECONDIÇÃO" e "EFEITO".

B — CRITÉRIOS BÁSICOS
C — CRITÉRIOS COMPLEMENTARES

Fig. 12 - O modelo ER/T^{+} frente às características desejáveis

5. IMPLEMENTAÇÃO DAS ESPECIFICAÇÕES

Neste capítulo são apresentados os mecanismos para a transformação de especificações ER/T$^+$ em uma linguagem de banco de dados.

Em função das características de um SGBD relacional de proporcionar uma visão simples e tabular dos dados e oferecer linguagens de alto nível para manipulação dos dados, tanto na consulta quanto na atualização do banco de dados, e levando em consideração a proximidade com o nível da linguagem de especificação ER/T$^+$ usada nesta tese, a opção de mapear o modelo de especificação de transações, para um correspondente esquema relacional, é considerada neste capítulo. Dentro deste contexto de SGBD relacional, a linguagem de manipulação de dados aqui considerada é SQL, sendo a escolha motivada pelo fato de ser padronizada e bastante popular entre a comunidade acadêmica. A sintaxe SQL utilizada nesta tese é a do padrão ANSI descrita em /DAT 89/.

Considerando que a semântica ER/T$^+$ é equivalente nas representações gráfica e textual e que cada construção sintática gráfica tem uma correspondente representação textual, para facilitar a compreensão do texto, no restante da tese será utilizada a terminologia adotada na representação gráfica (fluxos, anotações, etc.).

Para implementar uma especificação ER/T$^+$, devem ser definidos os seguintes mecanismos:

- Mecanismos de implementação dos aspectos estruturais do banco de dados (hierarquia entre dados, relacionamento entre dados, etc);
- Mecanismos de implementação de restrições de integridade estáticas (chave primária, cardinalidade, exclusão mútua, dependência funcional) com correspondente tratamento de exceções;
- Mecanismos de implementação dos aspectos comportamentais do banco de dados(tratamento de entrada e saída, operações básicas de acesso e atualização, etc.);

5.2., pois, pelo modelo de implementação de transações aqui proposto, validações prévias são necessárias antes da execução das operações sobre o banco de dados.

Para os mecanismos de implementação dos aspectos comportamentais, há uma transformação dos fluxos individuais da especificação ER/T$^+$ em correspondentes construções SQL. Para cada fluxo são usadas instruções SQL equivalentes. Posteriormente, estas instruções SQL são reordenadas e combinadas com instruções da linguagem C para compor o fonte do programa que implementa a

- Mecanismos de seqüencialização de operações para tratar a não proceduralidade da especificação;

Os mecanismos de implementação dos aspectos estruturais já foram largamente estudados na literatura e não serão considerados nesta tese. Entretanto, como são necessários para a compreensão do restante do texto, estão apresentados no Apêndice A.

Nos exemplos SQL ilustrados no texto, para facilitar o entendimento das regras de transformação ER/T$^+$ para SQL, são mantidos em SQL os mesmos nomes de atributos, entidades e relacionamentos especificados no diagrama ER/T$^+$. No momento de efetivamente implementar estas transformações, mecanismos de mapeamento de nomes tratarão das singularidades de representação em cada modelo.

Outro mecanismo que deve ser considerado é o de implementação de restrições de integridade. Além do mapeamento de cada construção individual do diagrama ER em equivalentes construções SQL, também devem ser considerados mecanismos que permitam preservar a correção dos dados para manter o banco de dados em um estado consistente após a execução das operações de atualização. Estes mecanismos, também conhecidos como restrições de integridade semânticas /TUC 82/, tratam da prevenção dos erros feitos através de atualizações autorizadas ou não.

Os mecanismos de implementação de restrições de integridade muitas vezes já estão presentes no próprio SGBD. Todavia, algumas discussões adicionais são realizadas na seção 5.2., pois, pelo modelo de implementação de transações aqui proposto, validações prévias são necessárias antes da execução das operações sobre o banco de dados.

Para os mecanismos de implementação dos aspectos comportamentais, há uma transformação dos fluxos individuais da especificação ER/T$^+$ em correspondentes construções SQL. Para cada fluxo são usadas instruções SQL equivalentes. Posteriormente, estas instruções SQL são reordenadas e combinadas com instruções da linguagem C para compor o fonte do programa que implementa a

especificação. As regras de transformação são apresentadas na seção 5.3.

Como a especificação em ER/T$^+$ não estabelece uma seqüência de execução das construções SQL correspondentes aos fluxos, não basta simplesmente transformá-las em instruções SQL. É necessário definir um mecanismo de seqüencialização das operações a fim de mapear a especificação declarativa para um modelo procedural de transação. Este é um dos principais objetivos perseguidos nesta tese e é amplamente discutido na seção 5.4, onde é tratada a montagem da transação procedural.

Após obtido o modelo procedural da transação, procedimentos de otimização das consultas ao banco de dados são especificados. Esta otimização é aplicada sobre as instruções SQL geradas. O assunto é tratado na seção 5.5.

Porém, antes de descrever os detalhes sobre a implementação das especificações, convém apresentar o modelo de implementação da transação aqui considerado.

5.1 O modelo geral de implementação da transação

O estudo do ER/T$^+$, no capítulo 4, mostrou que este modelo sugere a especificação de uma transação atômica através da construção de um diagrama que detalhe a transação e todos os objetos associados a seus fluxos. Assim, uma transação é completamente especificada mediante o desenho de um único diagrama ER/T$^+$ (ou correspondente especificação textual). Neste diagrama, uma série de fluxos aí representados especificam as ações ou operações individuais que devem ser executadas na transação.

Uma alternativa para implementar a especificação de uma transação ER/T$^+$ é:

- executar seqüencialmente as operações especificadas a partir de uma ordem estabelecida por um mecanismo de

determinação da dependência entre as operações;
- caso, em algum momento, uma exceção for constatada:
  - abortar a transação;
  - o SGBD, utilizando os seus mecanismos de recuperação do banco de dados, mantém a base de dados íntegra, desfazendo as alterações produzidas pela transação.

O inconveniente desta alternativa é a modificação desnecessária sobre o banco de dados em caso da ocorrência de exceções e, em conseqüência, excessivos procedimentos de restauração da base de dados ("Rollback"), por parte do SGBD, retrocedendo o BD para um estado consistente antes da execução da transação mal sucedida.

Outro inconveniente aparece no tratamento de exceções. As mensagens de erro vão sendo liberadas paulatinamente ao usuário a cada exceção constatada. O usuário não tem uma visão completa e simultânea de todos os erros provocados na transação.

Em vista disso, uma outra alternativa foi inicialmente aqui considerada. Um modelo que, baseado em mecanismos de execução das redes de Petri e na propriedade de causa e efeito presente na linguagem de especificação ER/T$^+$, divide a implementação da transação em duas fases: a da habilitação e a da execução.

Na fase de testes de habilitação, que verifica previamente se a transação pode ser executada de forma eficiente, são executadas as operações de:

- entrada de dados;
- validações dos dados contra sua descrição no dicionário de dados;
- validações dos dados contra a base de dados, verificando presença/ausência de tuplas em tabelas.

Nesta fase de testes de habilitação, para a busca de dados no banco de dados, os fluxos de inclusão da especificação ER/T$^+$ são automaticamente convertidos em fluxos de teste de ausência, enquanto que os fluxos de alteração e exclusão são convertidos em fluxos de teste de presença.

A transação somente será efetivamente executada se todos os testes de habilitação forem previamente satisfeitos (testar antes de executar). Os testes de habilitação que não forem satisfeitos geram mensagens ao usuário. Estas mensagens de erro são mostradas ao usuário no final da fase.

Na fase de execução são executadas as operações de:

- atualização do banco de dados (fluxos ER/T$^+$ de inclusão, alteração e exclusão);
- saída de dados.

Uma avaliação deste modelo, em termos das operações realizadas sobre o banco de dados, pode ser resumido como segue:

FLUXOS	FASES	
	HABILITAÇÃO	EXECUÇÃO
Presença	SELECT (LOCK)	---
Ausência	SELECT (LOCK)	---
Inclusão	Teste de Ausência	INSERT
Exclusão	Teste de Presença	DELETE
Alteração	Teste de Presença	UPDATE

Caso não se considere um bloqueio completo da tabela acessada por uma operação sobre o banco de dados, a simples conversão dos fluxos de inclusão em fluxos de teste de ausência, durante a fase de testes de habilitação, não satisfaz plenamente o modelo proposto. O teste de ausência poderia eventualmente resultar em "verdadeiro", indicando tupla não existente e, portanto, apto para a inclusão. No entanto, quando da efetiva inclusão na fase de execução, esta tupla poderia ter sido incluída por outra transação, entre as fases de testes de habilitação e de execução. Pelo conceito de transação ACID, uma tupla detectada como ausente no início da transação deve continuar ausente até o fim da transação, a menos que a própria transação faça a sua inclusão. Isto também deve ser garantido pela operação de teste de ausência.

Outro problema aparece em relação à rotina de avaliação de integridade referencial (seção 5.2), quando for necessário

ativá-la sobre tuplas em dependência funcional entre si e sendo incluídas ou excluídas numa mesma transação. Por exemplo, incluir/excluir uma NOTA FISCAL e todos os seus ITENS relacionados em uma mesma transação. Durante a fase de testes de habilitação, as tuplas ainda não estão com sua atualização efetivada para permitir uma avaliação da integridade referencial.

Estes problemas são corrigidos da seguinte forma:

Para ausência:
- Através de uma inclusão temporária da(s) tupla(s), durante as fases da transação, caso for constatada a sua inexistência no banco de dados. Posteriormente, excluí-la(s) no fim da transação ou, em caso de exceção, no final da fase de teste de habilitação.

Para inclusão:
- Através de uma inclusão prévia da(s) tupla(s) durante a fase de testes de habilitação. Caso alguma exceção ocorrer durante a fase de habilitação, realizar a exclusão desta(s) tupla(s). Caso contrário, na fase de execução, a operação de inclusão torna-se desnecessária (inclusão já feita).

Para exclusão:
- Através de um assinalamento de exclusão lógica da(s) tupla(s) na fase de testes de habilitação. Assim, a rotina de avaliação de integridade referencial não visualizará esta(s) tupla(s). Caso alguma exceção ocorrer durante a fase de testes de habilitação, desmarca-se as exclusões lógicas. Caso contrário, na fase de execução, a operação de exclusão física (DELETE) deve ser providenciada.

Assim, o modelo geral de implementação da transação, numa versão que contempla uma solução para os problemas enunciados, pode ser esquematizado como segue:

FLUXOS	FASES	
	HABILITAÇÃO	CONFIRMAÇÃO
Presença	SELECT (LOCK)	---
Ausência	Se ausente INSERT	DELETE
Inclusão	Teste de Ausência INSERT	Já efetivada na Habilitação
Exclusão	Teste de Presença UPDATE (flag excl.)	DELETE
Alteração	Teste de Presença	UPDATE

Nesta versão melhorada do modelo, a fase de testes de habilitação foi acrescida de operações prévias de atualização do banco de dados para garantir o conceito ACID de transação e também garantir as validações da integridade referencial. Já a fase de execução, que passou a ser denominada de fase de confirmação, é responsável pela confirmação das atualizações prévias da fase de testes de habilitação (inclusão e exclusão lógica) e pela efetivação das alterações sobre o banco de dados. Com isto, as operações executas nas duas fases são:

Na fase de testes de habilitação:

- entrada de dados;
- validações dos dados contra sua descrição no dicionário de dados;
- validações dos dados contra a base de dados, verificando presença/ausência de tuplas em tabelas.
- atualização prévia do banco de dados (inclusões e exclusões lógicas);

Na fase de confirmação:

- atualização efetiva do banco de dados (alterações e exclusões físicas);
- saída de dados.

Note que, no contexto de transação atômica no SGBD relacional, o modelo de implementação de transação em duas fases aqui proposto, a primeira vista poderia apresentar o inconveniente dos testes de habilitação gerarem "locks" de leitura (no SELECT) que deveriam ser promovidos a "locks" de escrita (no INSERT/DELETE/UPDATE) na fase de confirmação e, com isto, se manifestar o fenômeno de "lock upgrade", eventualmente causando um "deadlock" na transação /DAT 88/. Este problema, no entanto, é sanado através de uma estratégia de bloqueio, baseada em bloqueio de duas fases /DAT 88/, discutida no capítulo 6.

O modelo proposto pode eventualmente aumentar o tempo de resposta das transações que terminarem com sucesso. Garante, no entanto, que haja um tratamento de exceções mais adequado, onde o usuário recebe todas as mensagens de erro de uma só vez e não de forma paulatina, a cada exceção produzida pelo SGBD.

Evita, também, os "Rollbacks" de atualizações indevidas em caso de exceção.

Cabe frisar que o diálogo com o usuário na execução de uma transação é extremamente simples onde o usuário fornece os dados e, após a conclusão da transação, é devolvido o resultado (saída dos dados) ou uma lista de erros. A interface de diálogo é determinada durante a modelagem do sistema sendo representada, exclusivamente, através dos agentes externos (e correspondentes fluxos de entrada) desenhados no diagrama. Durante o processamento da transação, não há necessidade de nenhuma intervenção adicional do usuário para um fornecimento interativo de dados como parâmetros de execução.

Para exemplificar a implementação de uma transação, no padrão do modelo aqui proposto e ao nível de código fonte produzido por um programador, é ilustrada a seguir uma rotina codificada manualmente e de forma arbitrária na linguagem C com embutimento de SQL. Esta rotina implementa os procedimentos de uma transação de "Empréstimo" de livros. A Figura 13 apresenta a especificação ER/T$^+$ da transação.

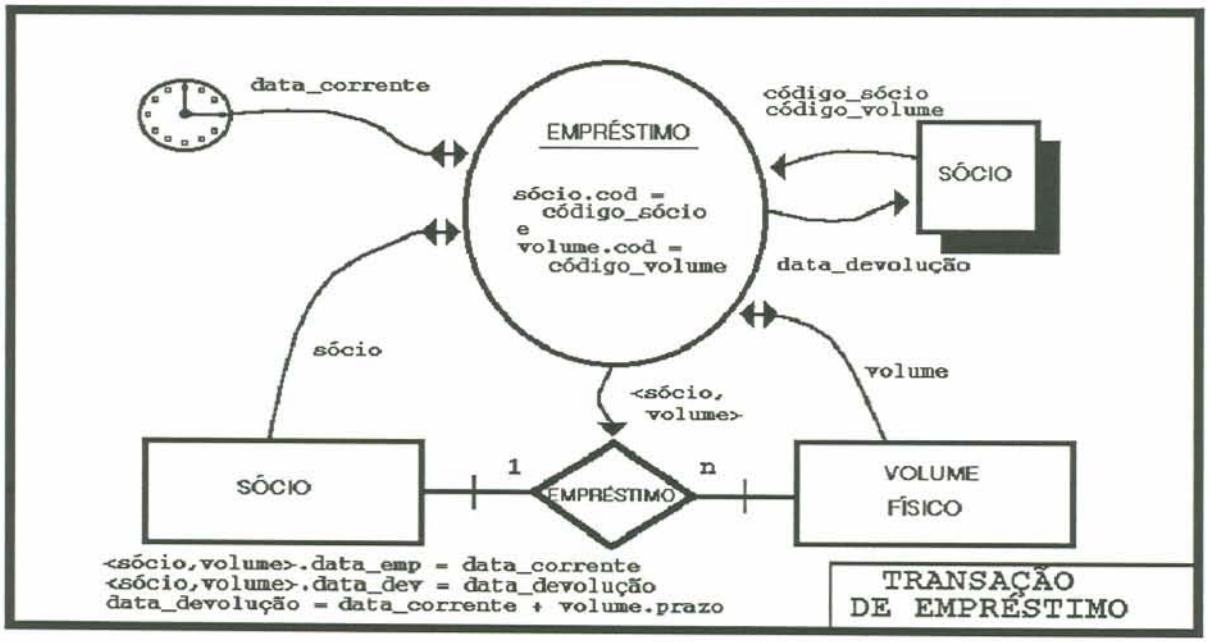


Fig. 13 - Transação de "Empréstimo" no modelo ER/T$^+$

Rotina "Empréstimo":

```
/* FASE DE TESTES DE HABILITAÇÃO                                    */

consiste = FALSE;
while (!consiste)
 {   /* exibe tela cujo layout foi desenhado em um módulo*/
     /* de modelagem do ER/T+                                       */
  solicita_dados(EMPRESTIMO);
     /* lê dados conf. formato declarado no dicionário              */
  entrada(codigo_socio,codigo_volume);
     /* consiste dados conf. domínio def. no dicionário             */
  if (valida_dominio(codigo_socio,codigo_volume) != "OK")
     trata_erro("DOMINIO INVALIDO PARA ENTRADA DE DADOS");
  else
     consiste = TRUE;
 }
data_corrente= time(datasistema); /*recupera data do S.O.*/
habilita = 'OK'; /*pressupõe transação habilitada                   */

/* testa presença da entidade SOCIO                                 */

EXEC SQL SELECT *
         FROM socio
         WHERE socio.cod = :codigo_socio;
if (sqlca.sqlcode == NOT_FOUND)
   presenca = 'NOK';
else
   presenca = 'OK';
if (presenca != 'OK')
   {habilita = 'NOK'; trata_erro("socio NAO CADASTRADO");}

/* testa presença da entidade VOLUME                                */

EXEC SQL SELECT *
         FROM volume
         WHERE volume.cod = :codigo_volume;
if (sqlca.sqlcode == NOT_FOUND)
   presenca = 'NOK';
else
   presenca = 'OK';
if (presenca != 'OK')
   {habilita = 'NOK'; trata_erro("volume NAO CADASTRADO");}

/* testa ausência do EMPRESTIMO para fins de posterior              */
/* inclusão,  onde o relacionamento EMPRESTIMO foi in-               */
/* corporado à relação "volume" (relacionamento 1:N)                */

EXEC SQL SELECT cod,socio
         FROM volume
         WHERE volume.cod    = :codigo_volume;
if (sqlca.sqlcode == NOT_FOUND)
   {habilita = 'NOK';
    trata_erro("volume NAO CADASTRADO");}
else
   {if (volume.socio == NULL)
       presenca = 'NOK'
    else
       presenca = 'OK'; }  /*relacionamento ja existente */
if (presenca == 'OK')
   {habilita = 'NOK';
    trata_erro("INCLUSAO DE emprestimo P/ JA EXISTENTE");}
else
   {
     /* atribuições do corpo e rodapé da transação                  */
    data_devolucao = soma_data(data_corrente,volume.prazo);

     /* processa inclusão do EMPRESTIMO                             */
    EXEC SQL UPDATE volume
             SET  volume.socio     = :codigo_socio ,
                  volume.data_emp  = :data_corrente,
                  volume.data_dev  = :data_devolucao
             WHERE  volume.cod = :codigo_volume ;
   }
```

```
    /* testa a cardinalidade do relacionamento                    */
    if (testa_cardinalidade(EMPRESTIMO,codigo_socio,
                            codigo_volume,
                            1,1,       /*min/max para SOCIO  */
                            0,N        /*min/max para VOLUME */
                           ) != "OK")
      {habilita = 'NOK';
       trata_erro("EXTRAPOLACAO DE CARDINALIDADE:emprestimo");
      }
     /* testa as restrições de integridade referenciais          */
     if (testa_restr_refer( ) ) != "OK")
       {habilita = 'NOK';
        trata_erro("ERRO NAS RESTRICOES REFERENCIAIS");
       }
     /* verifica habilitação / efetiva execução da transação */
     if (habilita == "NOK")
       {
        trata_erro("TRANSAÇÃO NAO HABIL.P/ MOTIVOS ASSINALADOS"
                  );
         /* se erro na transação, desfaz atualização feita   */
         /* sobre o EMPRESTIMO                               */
        EXEC SQL UPDATE volume
                 SET  volume.socio     = NULL ,
                      volume.data_emp  = NULL ,
                      volume.data_dev  = NULL
                 WHERE  volume.cod = :codigo_volume ;

       }
     else
    /* FASE DE CONFIRMAÇÃO                                       */
       {
         /* libera resultados para a saída                      */
         saida(data_devolucao);
       }
```

Sobre esta rotina, cabem os seguintes comentários:

1- A função "solicita_dados" pesquisa, no dicionário de dados, a descrição(layout) da tela de entrada rotulada como "EMPRESTIMO" e a exibe no vídeo. Supõe-se que a definição do layout da tela tenha sido feita anteriormente em uma etapa de definição.

2- Na função "entrada", para cada atributo de entrada, deve ser executado:

- . verificar tipo e tamanho do atributo no dicionário;
- . verificar posição do atributo na tela (layout);
- . posicionar o cursor na tela;
- . montar formato de entrada e executar comando que permite receber o valor do atributo;

3- A validação dos dados de entrada (função "valida_dominio") pressupõe o uso das definições de domínio (obrigatoriedade, lista de valores válidos,limites, etc) mantidas no dicionário de dados.

4- A rotina que implementa fluxos ER/T^+ de teste de presença segue um padrão baseado em:

. comando SELECT com seleção de tupla, tendo como parâmetros:
  .cláusula * : - seleção de tupla completa;
  .cláusula FROM: - nome da tabela (entidade/relac.);
  .cláusula WHERE:- critério de seleção referente a ent/rel. em questão montados a partir da anotação complementar da transação (ver Regras de Mapeamento das Expressões nas Anotações de Fluxo - seção 5.3.1);
. assinalamento de presença válida (ou não) a partir da avaliação do conteúdo da variável 'sqlca.sqlcode' (= a NOT_FOUND).

5- Em parte, a validação das restrições de integridade referencial é garantida pela especificação dos testes de presença e ausência explicitados, no diagrama, durante a modelagem da transação. A rotina "testa_restr_refer" deve garantir a validação da dependência funcional dos atributos ainda não testados pela implementação dos fluxos ER/T^+ de teste de presença/ausência.

6- Situações como a redundância do acesso à relação "volume" para teste de presença de entidade e teste de ausência do relacionato, devem ser tratadas por uma rotina de otimização de consultas.

7- Considerando que, em relacionamentos 1:N, os atributos do relacionamento são colocados na relação que implementa a entidade do lado N (ver Apêndice A), o procedimento de inclusão do EMPRESTIMO é realizado mediante uma operação de alteração (comando UPDATE do SQL) sobre a relação "volume". A valoração do atributo "volume.cod" (atributo chave) é feita a partir dos critérios de seleção nas anotações da transação e a valoração dos atributos "volume.socio", "volume.data_emp" e "volume.data_dev" (atributos não chave), a partir das atribuições contidas nas anotações do rodapé da transação. Este procedimento é realizado já na fase de testes de habilitação, para permitir uma validação das restrições de integridade referencial.

Nesta tese, todos os exemplos de rotinas ilustradas em SQL embutida em C consideram a sintaxe apresentada em /ING 91/.

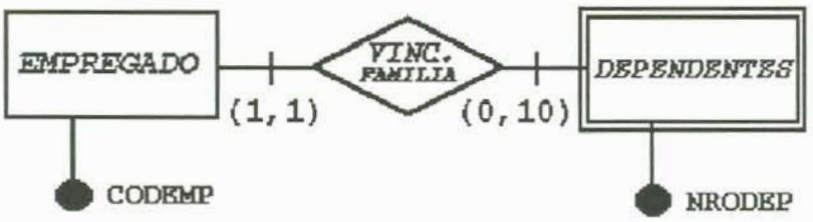
5.2 Validação de Restrições de Integridade

Um SGBD normalmente já verifica a integridade durante as operações de atualização do banco de dados. Entretanto, pelo modelo de transação aqui proposto, na fase de testes de habilitação da transação, torna-se necessário realizar uma validação prévia dos dados, a fim de descartar todas as possibilidades de eventuais erros, antes de confirmar a efetiva

atualização do banco de dados. Devem ser validados os dados confrontando-os com o dicionário de dados e com a base de dados.

Assim, como mecanismos que garantam as restrições de integridade do modelo de dados, são propostas algumas rotinas de validação. São elas:

a) **Avaliação de cardinalidades das entidades no relacionamento**:

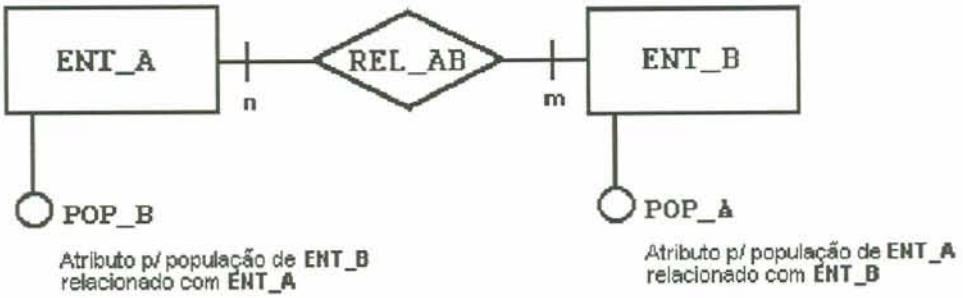


Para aumentar a completeza da especificação, a anotação do modelo no exemplo está indicando os limites (min, max) do relacionamento. Neste exemplo, cada DEPENDENTE tem obrigatoriamente 1 e somente 1 EMPREGADO associado e um EMPREGADO tem, opcionalmente, até no máximo 10 DEPENDENTES.

Esta restrição de integridade deve ser avaliada por uma rotina de teste de cardinalidade ativada sempre que houver:

- uma transação de inclusão na relação que implementa o relacionamento;
- uma transação de exclusão na relação que implementa o relacionamento.

Nesta atualização (inclusão/exclusão), após a série de validações necessárias e entre elas o teste de cardinalidade, deverá ser providenciada também a atualização dos atributos de população do relacionamento nas relações que implementam ambas as entidades envolvidas.



No exemplo, juntamente com a atualização da relação que implementa o relacionamento REL_AB, os atributos POP_B e POP_A das relações, que, respectivamente, representam as entidades ENT_A e ENT_B, deverão ser incrementados (no caso de inclusão do relacionamento) ou decrementados (no caso de exclusão).

Esta alternativa de, adicionalmente, manter um atributo de população nas respectivas relações evita uma recontagem do número de tuplas a cada atualização. Em caso de exclusão de tuplas que possuam atributo que é atributo estrangeiro (chave estrangeira) em outra relação, facilita também a avaliação de restrições de integridade referencial. Facilita no sentido de possibilitar, antes de uma exclusão, a verificação exaustiva da existência destas tuplas de atributo estrangeiro. Mediante uma simples avaliação do conteúdo do atributo população (se população = zero), pode ser determinada a possibilidade de exclusão. Por exemplo, não pode haver exclusão de EMPREGADO se a população de DEPENDENTES para ele não estiver zerada.

Sendo assim, a rotina de teste de cardinalidade é:

Rotina:

- Sejam as tuplas com CHAVE_A, CHAVE_B e os valores de MIN_A, MAX_A, MIN_B, MAX_B especificados no diagrama ER:

- Antes de efetivar a transação, a rotina deve garantir:

a) inclusão de um relacionamento: (Teste de cardinalidade máxima)

- Para inclusão de relacionamento referente às tuplas de CHAVE_A e CHAVE_B, testar os atributos de população do relacionamento nas respectivas tuplas em confronto com MAX_A e MAX_B, ou seja, testar:
  .se POP_A + 1 > MAX_A ou
  POP_B + 1 > MAX_B
  - acusar erro na extrapolação dos limites superiores;

b) exclusão de um relacionamento: (Teste de cardinalidade mínima)

- Para exclusão de relacionamento referente às tuplas de CHAVE_A e CHAVE_B, testar os atributos de população do relacionamento nas respectivas tuplas em confronto com MIN_A e MIN_B, ou seja, testar:
  .se POP_A - 1 < MIN_A ou
  POP_B - 1 < MIN_B
  - acusar erro na extrapolação dos limites inferiores;

Cabe também ressaltar que, quando houver uma inclusão de tupla na relação que implementa a entidade (ENT_B) relacionada com outra entidade (ENT_A), onde esta última tem participação obrigatória (MIN_A > 0) no relacionamento, forçosamente deverá haver uma inclusão deste relacionamento na mesma transação. No exemplo apresentado, não é possível incluir um DEPENDENTE sem relacioná-lo a um determinado EMPREGADO e isto deve ser feito na mesma transação. Esta situação pode ser tratada pelos mecanismos de avaliação de restrições de integridade referencial.

b) **Avaliação de restrições de integridade referencial**:

A rotina de avaliação de dependências funcionais inter-entidades (restrições de integridade referencial /TUC 82/) deve ser ativada sempre que houver uma inclusão ou alteração de tupla que tenha algum atributo estrangeiro (chave estrangeira) ou então uma exclusão ou alteração de tupla que tenha algum atributo que seja atributo estrangeiro em outra relação. Esta rotina pode eventualmente estar incorporada ao SGBD.

Rotina:

- Sejam as relações R1 e R2 do modelo relacional , onde R1 é uma relação com atributos estrangeiros de R2.
- A rotina deve garantir:

Operações sobre R1 (possui atributos estrangeiros)

a) inclusão:
- Para inclusão de tupla t1 na relação R1, deve haver correspondente tupla t2 em R2 tal que:
  t1.atrestr = t2.atr
  Se não tiver correspondência de tuplas, acusar inconsistência (ou incluir tupla t2 em R2).

b) alteração:
- Para alteração de atributo A na tupla t1 de R1:
  . se A for chave primária de R2, acusar inconsistência (alteração proibida); outra possibilidade é adotar a mesma estratégia da inclusão considerando, agora, o novo valor (alteração) do atributo.
  . se A não for chave primária de R2,alterar também o valor do atributo A na tupla t2 correspondente em R2.

Operações sobre R2 (possui atributo que é atributo estrangeiro em outra relação)

a) exclusão:
- Para exclusão de tupla t2 da relação R2, não pode haver correspondente tupla t1 em R1, tal que:
  t2.atr = t1.atrestr
  Se tiver correspondência de tuplas, acusar inconsistência (ou excluir tupla(s) t1 de R1).

b) alteração:
- Para alteração de atributo A na tupla t2 de R2:
  . se A for chave primária de R2, atualizar todas as ocorrências de tuplas t1 em R1 que tenham este valor no atributo A;
  . se A não for chave primária de R2, atualizar a tupla t1 correspondente em R1 com o novo valor de A.

Nesta rotina devem ser desconsideradas as tuplas excluídas logicamente, ou seja, as tuplas que tenham o atributo FLAGEXCL = "S".

Já que, nas implementações de hierarquias de generalização/especialização e de agregações do modelo ER, considera-se o mapeamento das estruturas para o modelo relacional, mediante uma associação entre as relações sendo

mantida através dos atributos estrangeiros (ver Apêndice A), a rotina de avaliação de restrições de integridade referencial garante também a integridade destas estruturas, durante as operações de atualização do banco de dados, através da avaliação das dependências funcionais entre as relações.

Na implementação da especificação ER/T$^+$, esta rotina deve ser incorporada como teste de habilitação para os fluxos de inclusão, alteração ou exclusão caso não tenham sido explicitados, durante a modelagem, via fluxos de teste de presença e de teste de ausência.

c) **Avaliação das restrições de formato e de domínio**:

Com base nas informações sobre os atributos (tipo, máscara de edição, obrigatoriedade, limites e valores válidos, etc.) extraídas do dicionário de dados, a rotina de avaliação de restrições de formato e de domínio é capaz de validar os valores dos atributos nas operações de alteração e inclusão. Os procedimentos desta rotina aparecem ilustrados a seguir.

Rotina:

- Sejam as informações de tipo, obrigatoriedade, limites e enumeração de valores válidos constantes no dicionário de dados.
- A rotina deve garantir os seguintes procedimentos:
  - Testar a valoração do atributo em confronto com o tipo de dado nativo do atributo ("numérico", "alfabético", ...), devolvendo mensagem de erro caso for constatado tipo inválido;
  - Testar se valoração do atributo foi feita (não = branco) em consonância com sua obrigatoriedade ("sempre", "na inclusão", "opcional"), devolvendo mensagem de erro caso for constatado a situação de preenchimento obrigatório não satisfeito;
  - Testar se valoração do atributo está dentro dos limites. Caso não estiver, devolver mensagem de erro: valor fora de limites;
  - Testar se valoração do atributo está contida na enumeração de valores válidos. Caso não estiver, indicar erro de valor inválido.

5.3 Transformação de fluxos individuais ER/T$^{\pm}$ em SQL

As transformações dos fluxos individuais ER/T^{+}, para correspondentes instruções SQL, estão baseadas em uma série de regras de transformação descritas nesta seção.

5.3.1 Regras de Mapeamento das Expressões de Anotações de Fluxos Internos

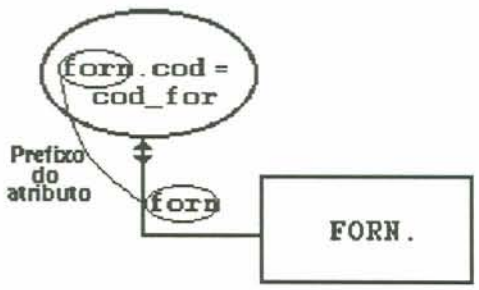
Antes de apresentar as regras específicas aplicadas a cada tipo de fluxo, convém discutir as regras usadas na avaliação das expressões de anotações comuns aos fluxos internos. Estas expressões podem ser encontradas denotando os diversos tipos de fluxos em um diagrama ER/T^{+}.

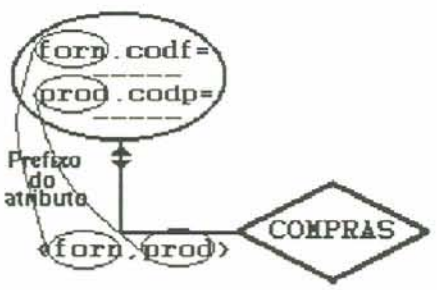
As expressões constantes nas anotações de fluxos internos permitem indicar as entidades/relacionamentos que participam da transação. Cada vez que uma transação ocorre, podem estar presentes, em cada fluxo da transação, uma ou mais entidades/relacionamentos. Estas anotações de fluxo podem indicar:

.variáveis: denotam objetos individuais (entidades/relacionamentos);
.conjuntos: denotam conjunto de objetos.

As regras para o seu mapeamento em instruções ou expressões SQL são:

a) **Variáveis (termos)**:





Regra:

Em um diagrama ER/T$^+$, as variáveis usadas como nomes de fluxos, permitem vincular a entidade/relacionamento com seus atributos referenciados nas anotações complementares da transação. Elas identificam o atributo através da prefixação do nome desse atributo com o nome do fluxo. Por exemplo, a variável "forn", prefixando o nome de um atributo, identifica o atributo "forn.cod" especificado na anotação da transação. Convém frisar que o nome da variável não precisa ser igual ao nome da entidade vinculada. No exemplo, se fosse usada a variável "fornecedor" ao invés de "forn" (para denotar o fluxo) e a expressão "fornecedor.cod" ao invés de "forn.cod" (nas anotações da transação), o efeito seria o mesmo. Durante o processo de tradução da especificação, mecanismos de mapeamento de nomes tratam de realizar as traduções necessárias.

No processo de mapeamento para SQL, as variáveis são usadas no reconhecimento dos atributos que farão parte dos critérios de seleção da cláusula WHERE do comando SELECT. Neste processo, a variável é mapeada para um **<nome_relação>** que identifica a relação que implementa a entidade/relacionamento. Assim, a notação ER/T$^+$ **<nome_var>.<nome_atributo>**, usada para qualificar um atributo, é traduzida para a notação SQL **<nome_relação>.<nome_atributo>**. O reconhecimento das relações do BD a serem pesquisadas (cláusula FROM do SELECT) é feito a partir das regras de mapeamento do modelo de dados (ver apêndice A).

Exemplo:

```
SELECT  *
FROM  FORN
WHERE FORN.cod = cod_for
```

onde a expressão "FORN.cod = cod_for" foi montada a partir da anotação complementar da transação.

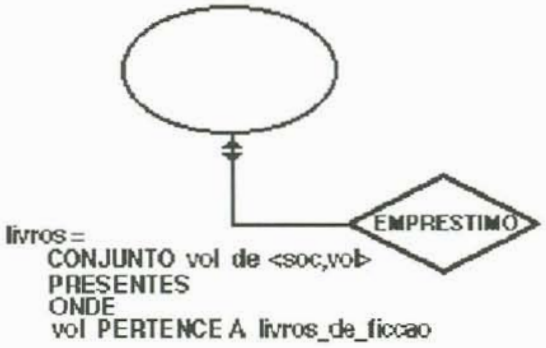
No exemplo, a seleção expressa que o valor do atributo "cod" da entidade "FORN" deve ser igual ao valor de "cod_for", fornecido via algum agente externo.

Exemplo:

```
SELECT  *
FROM  COMPRAS
WHERE COMPRAS.codf = ----  AND
      COMPRAS.codp = ----
```

onde a expressão "COMPRAS.codf = ---- AND ..." foi montada a partir das anotações da transação com a devida substituição do nome das variáveis prefixando os atributos pelo nome da relação que implementa o relacionamento.

b) **Conjuntos**:



Regra:

O uso de conjuntos nos fluxos segue os padrões de manipulação de conjuntos definidos para o modelo ER/T$^+$. Estes padrões foram apresentados no capítulo 4 da tese e estão aqui ilustrados para detalhar as regras de seu mapeamento para SQL. Assim, a notação de conjunto em ER/T$^+$ é:

```
[<nome_conjunto> =] CONJUNTO <nome_var> PRESENTES [ONDE
                                                   <condição>]
                              ou
[<nome_conjunto> =] CONJUNTO <nome_var> AUSENTES
                            ONDE <seleção> E
                            <atributo_ent> PERTENCE A <relação>
```

onde <nome_var> pode assumir os formatos:

<ent>	- indicando a entidade;
<ent1,...,entn>	- indicando o relacionamento;
<enti,... de <...,enti,...>>	- indicando a(s) entidade(s) que participa(m) no relacionamento;
<atr1,... de <ent>>	- indicando o(s) atributo(s) da entidade;

<atr1,... de <ent1,...,entn>>- indicando o(s) atributo(s) do relacionamento.

Estas construções sintáticas são mapeadas para SQL da seguinte forma:

1) **<nome_conjunto>:**

O <nome_conjunto> que denota um conjunto em um fluxo ER/T$^+$ é utilizado, pelos mecanismos de implementação da especificação, para duas finalidades:

a) Orientar o processo de montagem de aninhamento de consultas (discutido na seção 5.4.3), referenciando expressões de subconsultas a serem embutidas. Neste caso, o nome não aparece no fonte SQL gerado.

b) Atribuir um nome a uma tabela intermediária (temporária) que é gerada quando for necessário instanciar consultas SQL(discutido na seção 5.4.3). Neste caso, o nome aparece no fonte SQL gerado.

2) **<ent>:**

A entidade <ent> permite a identificação da relação <rel> a ser pesquisada através do comando SELECT do SQL (cláusula FROM <rel>) e, adicionalmente, especifica que todos os atributos serão tornados disponíveis na consulta ('*' na seleção de atributos do comando SELECT).

Exemplo em ER/T$^+$:

```
cj_socios = CONJUNTO socio PRESENTES
```

Em SQL:

```
SELECT *
FROM socio
```

"cj_socios" não aparece aqui no fonte SQL visto ser interno ao processo de tradução e é usado somente para orientar embutimento de subconsultas.

3) **<ent1,...,entn>:**

O relacionamento <ent1,...,entn> permite a identificação da relação <rel> que deve ser pesquisada através

de um comando SELECT do SQL (cláusula FROM <rel>). Adicionalmente, especifica que todos os atributos serão considerados na consulta ('*' no SELECT).

Exemplo em ER/T$^+$:

```
emprestimos_v = CONJUNTO <socio,volume> PRESENTES
```

Em SQL:

```
CREATE TABLE emprestimos_v AS
       SELECT *
       FROM emprestimo
```

onde "emprestimo" é o nome da relação que implementa o relacionamento <socio,volume> e "emprestimos_v" aparece no fonte SQL para indicar o nome da tabela intermediária a ser gerada.

4) **<enti,... de <...,enti,...> >:**

"enti" indica que, devem ser considerados na consulta, todos os diferentes valores de chaves da relação que implementa "enti" e que aparecem na relação do relacionamento <...,enti,...>. O relacionamento <...,enti,...> permite a identificação da relação a ser pesquisada.

Exemplo em ER/T$^+$:

```
CONJUNTO volume de
      <socio,volume>
PRESENTES
```

Em SQL:

```
SELECT UNIQUE vol
FROM emprestimo
```

onde "emprestimo" é o nome da relação que implementa o relacionamento <socio,volume> e "vol" é o atributo desta relação e também chave da relação que implementa "volume".

5) **<atr1,... de <ent> >:**

"atr1,... de" indica, que devem ser considerados na consulta, todos os atributos da relação que implementa <ent> (lista de atributos no comando SELECT do SQL). Já <ent> permite

identificar o nome da relação a ser pesquisada (cláusula FROM do comando SELECT no SQL).

Exemplo em ER/T$^+$:

```
CONJUNTO titulo, assunto de
      volume
PRESENTES
```

Em SQL:

```
SELECT titulo,assunto
FROM volume
```

6) **<atr1,... de <ent1,...,entn> >:**

Especifica os atributos da relação que implementa o relacionamento <ent1,...,entn> que devem ser considerados na consulta. O nome da relação <rel> (cláusula FROM <rel> do comando SELECT no SQL) deve ser obtido a partir do reconhecimento do relacionamento <ent1,...,entn>.

Exemplo em ER/T$^+$:

```
CONJUNTO vol,soc,data_dev de
      <socio,volume>
PRESENTES
```

Em SQL:

```
SELECT vol,soc,data_dev
FROM emprestimo
```

onde "emprestimo" é o nome da relação que implementa o relacionamento <socio,volume>

7) **<condiçao>:**

Permite uma recuperação qualificada para a consulta. Especifica a condição para a cláusula WHERE do comando SELECT no SQL, onde <condição> é expresso por: <atributo_ent/rel> <símb_relação> <variável/literal>. Também é possível especificar-se combinações lógicas ("and","or") de <condição> e parênteses para alterar a ordem de precedência. Para os símbolos

tradicionais (=,>,<,etc) em <símb_relação>, o mapeamento para critérios de seleção da cláusula WHERE é feita diretamente.

Exemplo em ER/T$^+$:

```
CONJUNTO volume
PRESENTES
ONDE volume.assunto = "ficcao"
```

Em SQL:

```
SELECT *
FROM volume
WHERE volume.assunto = "ficcao"
```

Nas anotações de conjunto ER/T$^+$ é possível utilizar, em <símb_relação>, um símbolo especial "PERTENCE A" ou "NÃO PERTENCE A" definido para relacionar um item individual com um conjunto de itens. Neste caso, <condição> é expressa por: <atributo_ent> PERTENCE A <relação>, onde <relação> é uma tabela temporária do SQL gerada anteriormente ou expressão de subconsulta embutida (com presença garantida por rotina de seqüencialização de operações). O mapeamento para o SQL (cláusula WHERE do comando SELECT) é realizado pela substituição do <símb_relação> "PERTENCE A" por "IN" e "NÃO PERTENCE A" por "NOT IN".

Exemplo em ER/T$^+$:

```
CONJUNTO vol de
      <socio,volume>
PRESENTES
ONDE vol PERTENCE A
         livros_de_ficcao
```

Em SQL:

```
SELECT vol
FROM emprestimo
WHERE vol IN livros_de_ficcao
```

onde "emprestimo" é o nome da relação que implementa o relacionamento <socio,volume>.

Neste exemplo, durante a montagem do aninhamento de consultas, "livros_de_ficcao" pode ser substituído por uma expressão de subconsulta embutida.

8) **AUSENTES ONDE <seleção> E <atributo ent> PERTENCE A <relação>:**

Esta cláusula da linguagem de anotação ER/T$^+$, dentro da especificação "CONJUNTO <nome_conjunto> AUSENTES ...", indica a ausência (não existência), da entidade/relacionamento, de um conjunto de objetos referenciados no fluxo a partir de um universo especificado em <relação>. Para esta cláusula, é necessário especificar um universo de objetos possíveis no contexto da consulta e isto é feito a partir de <relação>.

O mapeamento para SQL é feito por uma substituição pela seguinte subconsulta:

```
WHERE <seleção>
   AND (SELECT  <atributo_ent>  FROM  <relação>
        WHERE <atributo_ent> NOT IN <ent/rel> )
```

onde <ent/rel> é a relação que implementa a entidade ou relacionamento especificado em <nome_conjunto>.

Exemplo em ER/T$^+$:

```
CONJUNTO <socio,volume>
AUSENTES
ONDE soc = socio E
     vol PERTENCE A
         volumes_do_pedido
```

Em SQL:

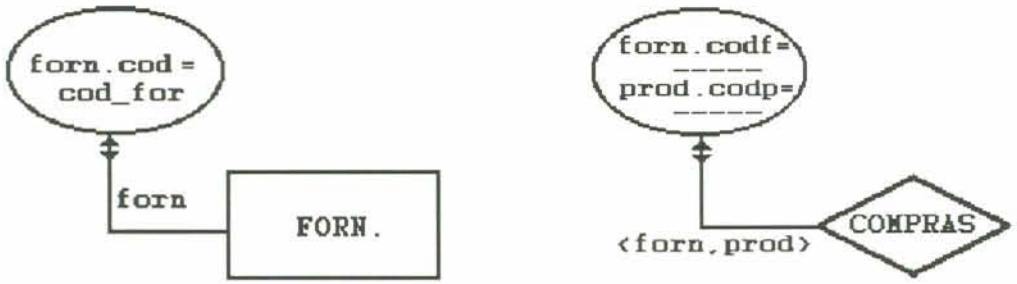
```
SELECT soc,vol
FROM emprestimo
WHERE soc = socio
  AND (SELECT vol FROM volumes_do_pedido
              WHERE vol NOT IN emprestimo )
```

onde "emprestimo" é o nome da relação que implementa o relacionamento <socio,volume>.

5.3.2 Regras de Mapeamento dos Fluxos ER/T$^+$

Os fluxos, segundo os tipos de operações que representam, podem ser expressos em SQL de acordo com as seguintes regras:

a) **Teste de presença**:



Regra:

Para cada fluxo de teste de presença, aplicar as regras de avaliação das expressões de anotações de fluxos internos apresentadas na seção anterior, a fim de montar as cláusulas do comando SELECT do SQL, dentro do escopo dos objetos manipulados pelo fluxo (objetos individuais ou conjuntos). Também deve ser verificado o sucesso (ou não) da consulta SQL resultante mediante uma avaliação do conteúdo da variável de controle do SQL denominada "sqlca.sqlcode". Se o valor de "sqlca.sqlcode" for igual a "NOT-FOUND", assinalar o teste de presença como falso. Caso contrário, assinalar como verdadeiro.

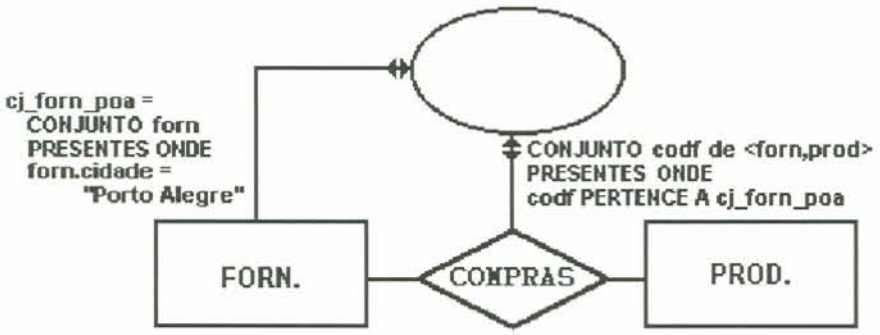
Exemplo:

- montar uma instrução de SQL que recupere, do BD, os dados especificados no fluxo de teste de presença e retorne uma variável booleana, informando a presença ou não dos dados.

Em SQL embutida:

```
EXEC SQL SELECT *
         FROM  FORN          /*nome da entidade*/
         WHERE FORN.cod = :cod_forn;
                             /*seleção montada a partir */
                             /*da anotação da transação */

if (sqlca.sqlcod == NOT-FOUND)
   presença = 'NOK';
else
   presença = 'OK';

return(presença);
```

No caso da manipulação de conjuntos em um fluxo de teste de presença, também devem ser aplicadas as regras de avaliação das expressões de anotações de fluxos internos discutidas na seção anterior. Um exemplo de manipulação de conjuntos e o correspondente mapeamento para SQL aparece ilustrado a seguir.



Em SQL embutida:

```
EXEC SQL SELECT codf
         FROM  COMPRAS       /*nome do relacionamento*/
         WHERE codf IN       /*subconsulta     */
             ( SELECT *
               FROM  FORN  /*nome da entidade*/
               WHERE FORN.cidade = "Porto Alegre");
                             /*seleção obtida da expres-*/
                             /*são de anotação do fluxo */

if (sqlca.sqlcod == NOT-FOUND)
   presença = 'NOK';
else
   presença = 'OK';

return(presença);
```

Regra: (para o caso de relacionamentos 1:N)

No caso de fluxo de teste de presença que envolva relacionamentos 1:N, a consulta SQL deve ser feita sobre a relação que implementa a entidade correspondente ao lado N (ver detalhes de mapeamento do modelo de dados no apêndice A). O critério de seleção (cláusula WHERE do comando SELECT), neste caso, é a composição dos atributos que formam o relacionamento nesta relação (chave da relação do lado N e chave da relação do lado 1 como atributo estrangeiro). Esta composição de atributos é montada a partir da especificação ER/T$^+$ contida na anotação complementar da transação.

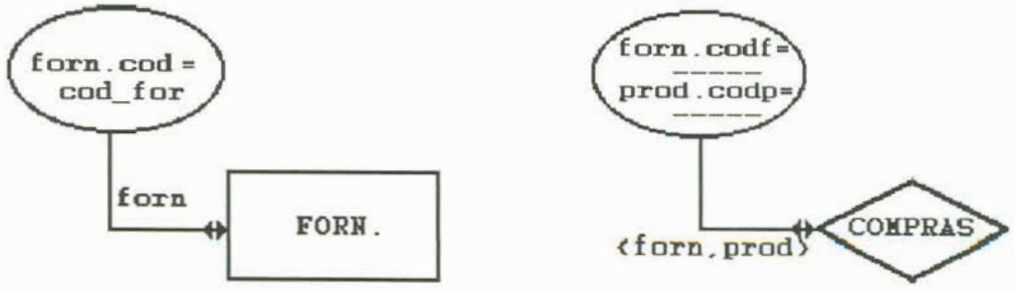
Exemplo:

- supondo que se restrinja o relacionamento COMPRAS a 1 fornecedor (entidade FORN) fornecendo N produtos (entidade PROD); neste caso, o teste de presença deve ser feito sobre a relação PROD, indicando a chave "codp" da relação e o atributo estrangeiro "codf".

Em SQL embutida:

```
EXEC SQL SELECT *
         FROM  PROD
         WHERE PROD.codp = :cod_prod and
               PROD.codf = :cod_forn ;
                           /*seleção montada a partir */
                           /*da anotação da transação */

if (sqlca.sqlcod == NOT-FOUND)
   presença = 'NOK';
else
   presença = 'OK';

return(presença);
```

b) **Teste de ausência**:



Regra:

Para viabilizar o conceito ACID de transação, a implementação do fluxo de teste de ausência deve garantir a ausência da(s) tupla(s) durante toda a transação. Assim, a implementação gera instruções SQL para as duas fases da transação: a)Inicialmente, na fase de testes de habilitação, a especificação modelada no fluxo é mapeada para um comando INSERT do SQL, se o teste de ausência resultar como "verdadeiro". b)Na fase de confirmação da transação, deverá ser realizada, mediante um comando DELETE do SQL, a eliminação da(s) tupla(s) inserida(s) na fase de testes de habilitação.

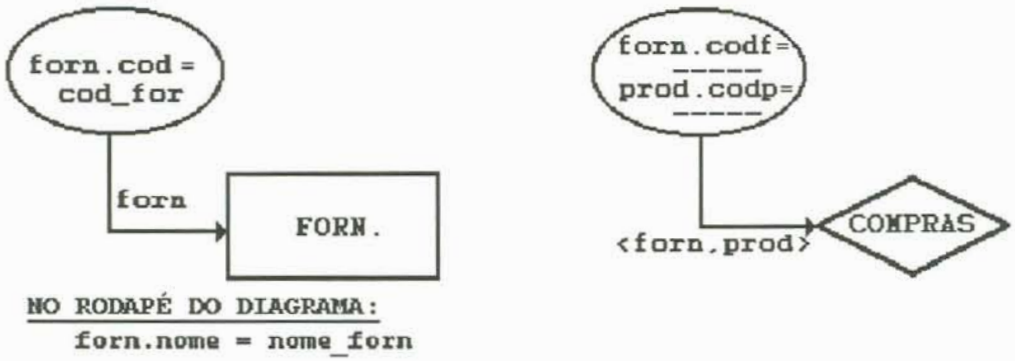
Em SQL, o exemplo resulta nos seguintes procedimentos:

Em SQL embutida:

```
/*  na fase de Testes de Habilitação */

EXEC SQL SELECT *
         FROM  FORN          /*nome da entidade*/
         WHERE FORN.cod = :cod_forn;
                             /*seleção montada a partir */
                             /*da anotação da transação */

if (sqlca.sqlcod == NOT-FOUND)
   ausencia = 'OK';
else
   ausencia = 'NOK';

if (ausencia = 'OK')
   EXEC SQL INSERT
         INTO  FORN (FORN.cod , /* lista de atributos */
                     ...  )
         VALUES     (cod_forn,  /* lista de valores   */
                     ...  );

return(ausencia);
   .
    .

/*  na fase de Confirmação da Transação */

EXEC SQL DELETE
         FROM  FORN          /*nome da entidade*/
         WHERE FORN.cod = :cod_forn;
                             /*seleção montada a partir */
                             /*da anotação da transação */
```

Cuidados especiais devem ser considerados quando a operação for sobre relacionamentos 1:N. Veja detalhes nos fluxos de inclusão e exclusão apresentados a seguir.

c) **Fluxo de Inclusão**:



Regra:

A implementação de um fluxo de inclusão gera uma instrução INSERT do SQL já na fase de testes de habilitação, a fim de garantir a verificação da integridade referencial do banco de dados. A inclusão deverá ser realizada mediante execução do comando SQL INSERT indicando, na cláusula INTO <Ent/Rel> (<lista_de_atributos>), o conjunto de atributos a serem atualizados e, na cláusula VALUES (<lista_de_valores>), a valoração destes atributos.

A valoração dos atributos chave é realizada a partir dos critérios de seleção constantes nas anotações da transação e a valoração dos atributos não chave, normalmente especificada no rodapé do diagrama ER/T^+, é feita a partir das atribuições ai expressas.

Para o exemplo, o fonte SQL gerado é:

Em SQL embutida:

```
/*  na fase de Testes de Habilitação */

EXEC SQL SELECT *
         FROM  FORN          /*nome da entidade*/
         WHERE FORN.cod = :cod_forn;
                             /*seleção montada a partir */
                             /*da anotação da transação */

if (sqlca.sqlcod == NOT-FOUND)
   presenca = 'NOK';
else
   presenca = 'OK';

if (presenca == 'OK')
   {habilita = 'NOK';
    trata_erro("inclusão de ??? para já existente");}
else
   {
    EXEC SQL INSERT
         INTO  FORN (FORN.cod,   /* lista de atributos */
                     FORN.nome,
                     ...  )
         VALUES     (cod_forn,   /* lista de valores   */
                     nome_forn,
                     ...  )
   }
```

Regra: (para o caso de relacionamentos 1:N)

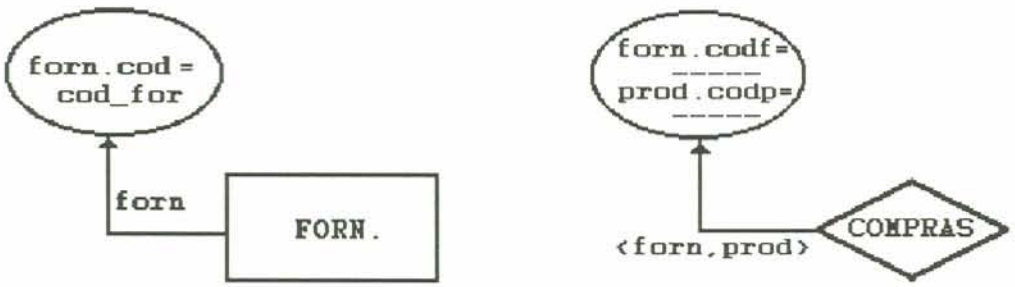
Na implementação de um fluxo de inclusão que envolva relacionamento 1:N, deve ser considerada a peculiaridade de mapeamento do modelo de dados para este tipo de relacionamento (ver apêndice A). Assim, o teste de ausência que, além de considerar a representação do relacionamento 1:N, deve vir acrescido de um teste adicional de ausência do relacionamento com outra tupla da relação. Este teste deve verificar se, na relação do lado N, o atributo estrangeiro que faz referência ao lado 1 está com valor inválido, ou seja, ainda não existe o relacionamento. Além disso, ao invés de uma instrução de inclusão (comando INSERT) sobre a relação, é gerada uma instrução SQL de alteração (comando UPDATE) sobre a relação que implementa a entidade correspondente ao lado N. Nesta relação, são alterados o atributo estrangeiro que vincula a relação com o lado 1 e todos os eventuais atributos do relacionamento.

Assim, o fonte SQL correspondente ao exemplo pode resultar em:

Em SQL embutida:

```
/*  na fase de Testes de Habilitação */

EXEC SQL SELECT codp,codf
         FROM  PROD          /*nome da entidade*/
         WHERE PROD.codp = :cod_prod
                             /*seleção montada a partir */
                             /*da anotação da transação */

if (sqlca.sqlcod == NOT-FOUND)
   {habilita = 'NOK';
    trata_erro(" ??? não cadastrado") ;}
else
   {if (PROD.codf == NULL)
       presenca = 'NOK';
    else
       presenca = 'OK';}    /*relacionamento já existente

if (presenca == 'OK')
   {habilita = 'NOK';
    trata_erro("inclusão de ??? para já existente");}
else
   {
   EXEC SQL UPDATE PROD
         SET    PROD.codf = :cod_forn
                 ...
         WHERE  PROD.codp = :cod_prod ;
   }
```

d) **Fluxo de Exclusão**:



Regra:

Para os fluxos de exclusão, a implementação gera também instruções SQL para as duas fases de execução da transação. Na fase de testes de habilitação da transação, os fluxos de exclusão são convertidos para equivalentes Testes de Presença e aplicadas as regras discutidas anteriormente. Se o teste de presença resultar como "verdadeiro", deverá ser realizada a exclusão lógica para garantir uma posterior verificação da integridade referencial do banco de dados. Isto é feito mediante um comando UPDATE do SQL, valorando o atributo FLAGEXCL com o valor "S".

Depois, na fase de confirmação da transação, com a efetiva execução da operação de exclusão (exclusão física), o fluxo é convertido para um comando SQL DELETE. Nas cláusulas FROM <Ent/Rel> e WHERE <condição>, são indicados os mesmos argumentos anteriormente usados no comando SELECT que materializou o teste de presença.

Assim, o fonte SQL para o exemplo é:

Em SQL embutida:

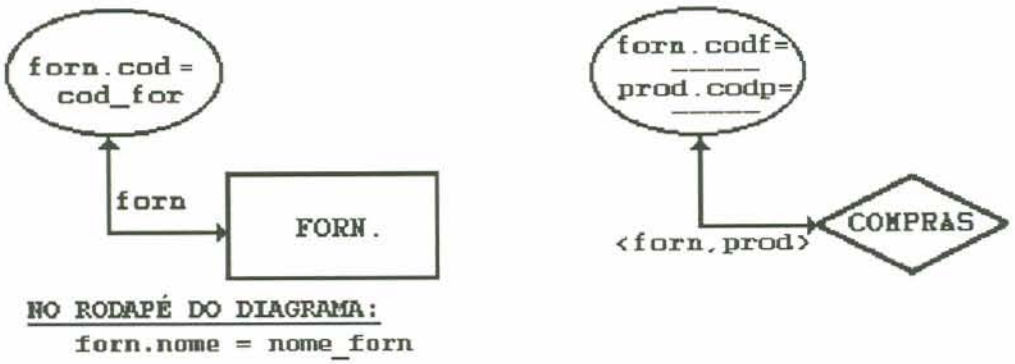
```
/*  na fase de Testes de Habilitação  */

EXEC SQL SELECT *
         FROM  FORN          /*nome da entidade*/
         WHERE FORN.cod = :cod_forn;
                             /*seleção montada a partir */
                             /*da anotação da transação */

if (sqlca.sqlcod == NOT-FOUND)
   presenca = 'NOK';
else
   presenca = 'OK';

if (presenca != 'OK')
   {habilita = 'NOK';
    trata_erro("exclusão de ??? para inexistente");}
else
   {  /* exclusão lógica */
    EXEC SQL UPDATE FORN       /*nome da entidade*/
         SET   FORN.flagexcl = 'S'
         WHERE FORN.cod = :cod_forn;
                             /*seleção montada a partir */
                             /*da anotação da transação */
   }
   .
    .

/*  na fase de Confirmação da Transação */

EXEC SQL DELETE
         FROM  FORN          /*nome da entidade*/
         WHERE FORN.cod = :cod_forn;
                             /*seleção montada a partir */
                             /*da anotação da transação */
```

Regra: (para o caso de relacionamentos 1:N)

Para a fase de testes de habilitação da transação, a regra específica para relacionamentos 1:N apresentada nos fluxos de teste de presença pode aqui ser aplicada. Já para a fase de confirmação da transação, uma instrução SQL de alteração (comando UPDATE) sobre a relação do lado N é usada para invalidar aqueles atributos que compõem o relacionamento (inclusive o

atributo estrangeiro, chave no lado 1). Com isto, o relacionamento é desfeito.

e) **Fluxo de Alteração**:



NO RODAPÉ DO DIAGRAMA:
forn.nome = nome_forn

Regra:

Como nos fluxos de exclusão, para os de alteração deve haver, na fase de testes de habilitação da transação, uma conversão para equivalentes Testes de Presença. Já, na fase de confirmação da transação, o fluxo é traduzido para um comando SQL UPDATE <Ent/Rel>. Neste comando, a cláusula SET <atribuições> deve indicar o conjunto de atributos que terão seus valores alterados e sua respectiva valoração. Na cláusula WHERE <condição>, aparecem os mesmos critérios de seleção anteriormente utilizados no comando SELECT que efetivou o teste de presença.

A valoração dos atributos que sofrerão alteração está expressa pelas atribuições especificadas nas anotações complementares, no rodapé do diagrama ER/T^+.

Os procedimentos a seguir, ilustram o resultado da tradução do exemplo para instruções SQL:

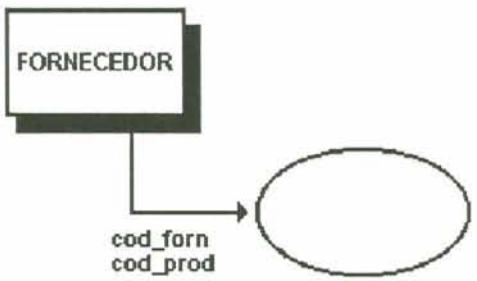
Em SQL embutida:

```
/*  na fase de Testes de Habilitação  */

EXEC SQL SELECT *
         FROM  FORN          /*nome da entidade*/
         WHERE FORN.cod = :cod_forn;
                             /*seleção montada a partir */
                             /*da anotação da transação */

if (sqlca.sqlcod == NOT-FOUND)
   presenca = 'NOK';
else
   presenca = 'OK';

if (presenca != 'OK')
   {habilita = 'NOK';
    trata_erro("alteração de ??? para inexistente");}

   .
    .

/*  na fase de Confirmação da Transação */

EXEC SQL UPDATE FORN         /*nome da entidade*/
         SET   FORN.nome= :nome_forn
         WHERE FORN.cod = :cod_forn;
                             /*seleção montada a partir */
                             /*da anotação da transação */
```

Regra: (para o caso de relacionamentos 1:N)

Neste caso, todos os procedimentos aqui descritos devem ser realizados sobre a relação que implementa a entidade do lado N (ver mapeamento do modelo de dados no apêndice A).

f) **Fluxo de Entrada**:



Regra:

Todo fluxo de entrada está associado a uma tela, cujo layout foi previamente definido em um módulo específico de desenho de telas (telas padrão ou personalizadas). Este layout é mantido no dicionário de dados, juntamente com a descrição dos atributos (formato e domínio) contidos nesta tela.

Assim, a tradução de um fluxo de entrada, em correspondentes comandos SQL ou C, se baseia em um laço de recepção e validação dos dados como segue:

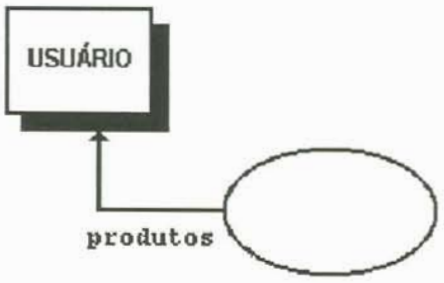
.Enquanto entrada não completa:
  .exibe tela (layout correspondente);
  .lê dados no formato indicado no dicionário de dados;
  .consiste dados conforme domínio;
  .se consistentes
    .completa entrada;
    .habilita fluxo;
  .caso contrário
    .exibe lista de mensagens de erro.

Para o exemplo, o fonte C resultante da aplicação da regra é o seguinte:

Em SQL embutida:

```
consiste = FALSE;
while (!consiste)
 {solicita_dados(FORNECEDOR); /*exibição da tela*/
  entrada(cod_forn,cod_prod); /*leitura dos dados*/
  if (valida_dominio(cod_forn,cod_prod) != 'OK')
                              /*validação       */
     {mostra_tela(ERRO,trata_erro(FORNECEDOR));}
                              /*exibição dos erros*/
  else
     consiste = TRUE;         /*fim de laço se consis-*/
 }                            /* tente                */
```

g)**Fluxo de Saída**:



NO RODAPÉ DO DIAGRAMA:

```
produtos =
  {cod_prod,qtd_prod /
   PARA CADA
     <prod> de temp_prod:
     cod_prod = prod.cod e
     qtd_prod = prod.qtd
```

Regra:

Como nos fluxos de entrada, há para os de saída um layout de tela, onde os resultados são mostrados. Esta tela pode ser uma tela específica (tela "out") ou ser parte integrante do próprio layout de uma tela de entrada (tela "I/O").

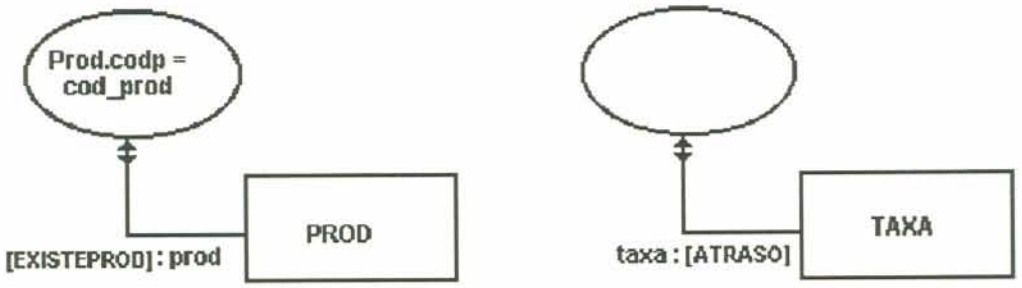
A implementação de um fluxo de saída consiste, basicamente, em exibir o resultado (itens do fluxo) no layout da tela constante no dicionário.

Na manipulação de conjuntos em operações de saída, pode haver uma necessidade de individualização das atribuições de valores aos objetos (modelado via expressão PARA CADA no rodapé do diagrama ER/T$^+$). Esta individualização das atribuições é tratada por um comando de repetição (WHILE,FOR,...) da linguagem hospedeira e pelo mecanismo de cursores do SQL (instruções OPEN, FETCH, CLOSE). Desta forma é possível viabilizar o acesso a tuplas individuais do conjunto. O resultado do mapeamento para SQL e C aparece exemplificado a seguir:

Em SQL embutida:

```
EXEC SQL DECLARE stmt STATEMENT;/*comandos dinâmicos*/
EXEC SQL DECLARE csr CURSOR FOR stmt; /*cursor SQL  */
str= "SELECT ..... ";     /*monta a instrução */
EXEC SQL PREPARE  stmt FROM :str;/*SQL analisa instr.*/
EXEC SQL DESCRIBE stmt INTO :sqlda;
                .
                .
                .
EXEC SQL OPEN csr FOR READONLY; /*abre o cursor     */

/* recupera tuplas individuais e monta a tela       */

tupla = 0;
while (sqlca.sqlcode == 0)
 {EXEC SQL FETCH csr USING DESCRIPTOR :sqlda;
  if (sqlca.sqlcode == 0)
     {tupla ++;
      monta_linha_tela(prod.cod,prod.qtd);

        /* aqui transfere dados para a tela e pode    */
        /* haver um controle de preenchimento da tela */
        /* a partir da variável "tupla" ou utilizar   */
        /* uma opção de tela completa com o artifício */
        /* de folhear páginas via teclas de funções   */
                .
                .
                .
     }
 }
EXEC SQL CLOSE csr;    /* fecha o cursor
mostra_tela(USUARIO);  /* exibe a tela                */
```

h) **Fluxo Opcional**:



Regra:

Este tipo de fluxo se baseia na associação de uma variável lógica ao nome do fluxo e assume uma conotação distinta conforme aparece prefixando ou posfixando este nome.

A semântica ER/T+ para este tipo de fluxo é a seguinte:

.Prefixando no nome do fluxo:
- O resultado da habilitação do fluxo é atribuído à variável lógica associada.

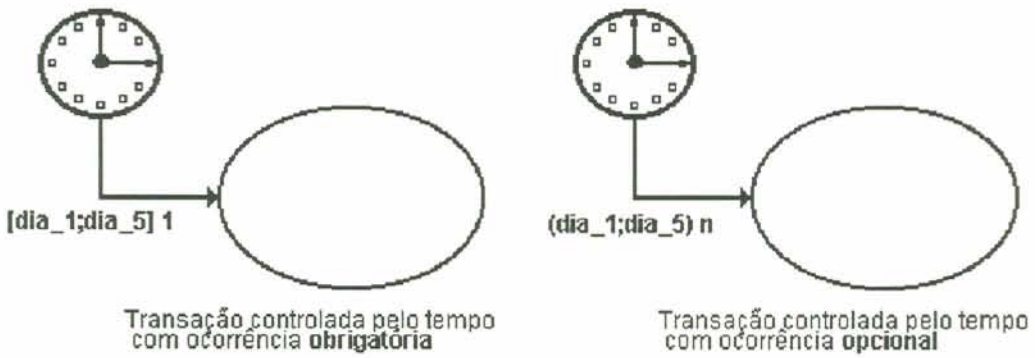
.Posfixando no nome do fluxo:
- Como um teste adicional às regras de habilitação da transação, afeta a habilitação do fluxo de acordo com a valoração (verdadeiro ou falso) da variável lógica associada.

As duas situações estão ilustradas no exemplo. Nele, a variável "EXISTEPROD" recebe o resultado lógico do teste de presença e a variável "ATRASO" é usada como um teste adicional de habilitação. Após a tradução, os procedimentos SQL são os seguintes:

Em SQL embutida:

```
/* para VARIÁVEIS LÓGICAS PREFIXADAS , após o teste */
/* de habilitação do fluxo                           */

if (habilita_fluxo == 'OK')
    existeprod = TRUE;
else
    existeprod = FALSE;
                .
                .
/* para VARIÁVEIS LÓGICAS POSFIXADAS , antes do      */
/* teste de habilitação do fluxo                     */

if (atraso == TRUE)
   /* testa a habilitação do fluxo                   */
                .
                .
                .
else
   habilita_fluxo = 'NOK';
```

i) **Fluxo de Controle (relógio)**:



Regra:

Estes fluxos de controle determinam os intervalos de tempo em que uma transação pode ou deve ocorrer. Portanto, abrangem a liberação ou não de todos os demais fluxos de uma transação, obedecidas as habilitações individuais de cada um . A fase de testes de habilitação da transação pode ou não ser ativada, dependendo do resultado da avaliação dos intervalos de tempos especificados nos fluxos de controle.

No exemplo estão ilustradas duas situações distintas:

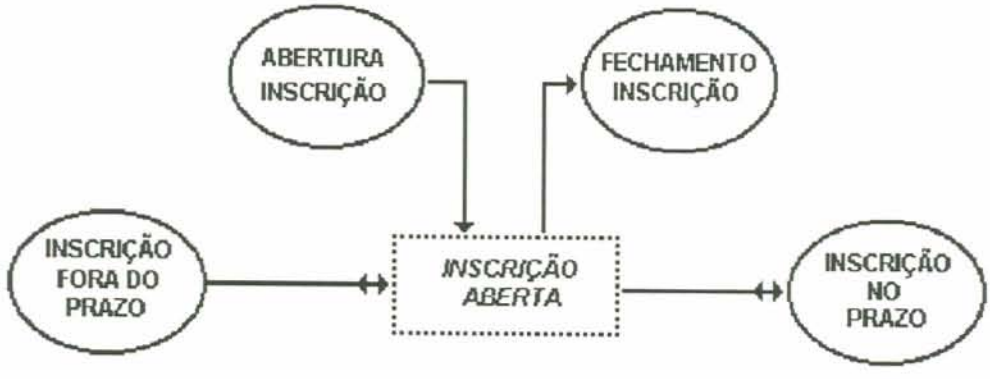
- transação controlada pelo tempo com ocorrência obrigatória: intervalo de tempo entre colchetes e seguido do número "1";
- transação controlada pelo tempo com ocorrência opcional: intervalo de tempo entre parênteses e seguido da letra "n";

Em ambos os casos, as transações definidas somente podem ocorrer dentro do intervalo especificado. Para o caso da ocorrência obrigatória, após a liberação da transação, uma segunda transação deve ser inibida. Esta semântica é representada em SQL como é mostrado a seguir:

Em SQL embutida:

```
/* antes dos testes de habilitação da transação     */
while (controle == FALSE)
{
if ((momento_atual >= intervalo_inf) &&
    (momento_atual <= intervalo_sup))
   if (((ocorrencia == "1") && (execucao_ant == FALSE))
      !! (ocorrencia == "n"))
      {controle = TRUE; proximo = TRUE; }
   else
      {controle = TRUE; proximo = FALSE;}
else
   {controle = FALSE;
    espera(); /* espera por novo evento              */
   }
}
if (proximo == FALSE)
   mostra_tela(ERRO,
         trata_erro("ocorrência obrigatória única  "));
else
   /* ativa fase de testes de habilitação da transação*/
            .
            .
/* após a execução da transação                      */
execuçao_ant = TRUE;
```

j) **Fluxo de Controle (semáforo)**:



Regra:

Para este fluxo de controle, a implementação de sincronização de transações, via um semáforo, está associada à

execução das operações de inclusão, exclusão, teste de presença e teste de ausência realizadas sobre uma relação denominada "semáforo". É uma relação automaticamente incluída no modelo de dados sempre que for constatada a modelagem de uma sincronização de transações, ou seja, é uma relação que não precisa ser especificada pelo usuário no modelo de dados. A estrutura desta relação "semáforo" está descrita no Apêndice A.

A implementação dos fluxos de controle utiliza uma relação do BD, não mantendo simplesmente as informações de sinalização em memória, porque diversas transações manipulam, como um meio de comunicação entre elas, as informações contidas nesta relação. Esta comunicação viabiliza um controle de paralelismo/serialização.

No caso de acesso à relação "semáforo", os procedimentos SQL e C, definidos anteriormente para estas quatro operações, são substituídos pelos que aparecem a seguir descritos. Os demais efeitos sobre a execução da transação (habilitação, tratamento de exceções, etc.) são idênticos aos efeitos normais dos correspondentes fluxos de inclusão, exclusão, teste de presença e teste de ausência.

Assim, a implementação deste fluxo de controle depende do tipo da operação a ser realizada sobre a relação "semáforo".

Inclusão de semáforo:

Considerando uma especificação ER/T$^+$ do tipo "**Inclusão: SEMÁFORO(<fato_controle>)**", ao invés de executar os procedimentos normais que implementam um fluxo de inclusão, executar uma instrução SQL para incluir, na relação "semáforo", uma tupla com atributo chave "rótulo", valorado com o conteúdo especificado por <fato_controle>.

Um exemplo de aplicação desta regra de mapeamento ER/T$^+$ para SQL aparece ilustrado a seguir.

Em ER/T$^+$: Inclusão: SEMÁFORO ("inscrição_aberta")

Em SQL embutida:

```
/*  na fase de Testes de Habilitação */

EXEC SQL SELECT rotulo
         FROM  semaforo
         WHERE semaforo.rotulo = "inscrição_aberta" ;

if (sqlca.sqlcod == NOT-FOUND)
   presenca = 'NOK';
else
   presenca = 'OK';

if (presenca == 'OK')
   {habilita = 'NOK';
    trata_erro("inclusão de SEMÁFORO já existente");}
else
   {
    EXEC SQL INSERT
         INTO  semaforo (semaforo.rotulo )
         VALUES         ("inscrição_aberta")
   }
```

Exclusão de semáforo:

Seja uma especificação ER/T$^+$ do tipo **"Exclusão: SEMÁFORO(<fato_controle>)"**, ao invés de executar os procedimentos normais que implementam um fluxo de exclusão, executar uma instrução SQL para excluir, da relação "semáforo", uma tupla com atributo chave "rótulo", valorado com o conteúdo especificado por <fato_controle>.

No exemplo que aparece a seguir está ilustrada a aplicação desta regra.

Em ER/T$^+$: Exclusão: SEMÁFORO ("inscrição_aberta")

Em SQL embutida:

```
/*  na fase de Testes de Habilitação */

EXEC SQL SELECT rotulo
         FROM  semaforo
         WHERE semaforo.rotulo = "inscrição_aberta" ;

if (sqlca.sqlcod == NOT-FOUND)
   presenca = 'NOK';
else
   presenca = 'OK';

if (presenca != 'OK')
   {habilita = 'NOK';
    trata_erro("exclusão de SEMÁFORO inexistente");}
else
   {  /* exclusão lógica */
    EXEC SQL UPDATE semaforo
         SET   semaforo.flagexcl = 'S'
         WHERE semaforo.rotulo = "inscrição_aberta";
   }
   .
    .

/*  na fase de Confirmação da Transação */

EXEC SQL DELETE
         FROM  semaforo
         WHERE semaforo.rotulo = "inscrição_aberta" ;
```

Teste de presença de semáforo:

Para uma especificação ER/T$^+$ do tipo **"Presença: SEMÁFORO(<fato_controle>)"**, ao invés de executar os procedimentos normais que implementam um fluxo de teste de presença, executar uma instrução SQL para verificar, na relação "semáforo", a existência de uma tupla com atributo chave "rótulo", valorado com o conteúdo especificado por <fato_controle>.

Um exemplo de aplicação desta regra de mapeamento ER/T$^+$ para SQL aparece ilustrado a seguir.

Em ER/T$^+$: Presença: SEMÁFORO ("inscrição_aberta")

Em SQL embutida:

```
EXEC SQL SELECT rotulo
         FROM  semaforo
         WHERE semaforo.rotulo = "inscrição_aberta" ;

if (sqlca.sqlcod == NOT-FOUND)
   presenca = 'NOK';
else
   presenca = 'OK';

return(presenca);
```

Teste de ausência de semáforo:

Para uma especificação ER/T$^+$ do tipo **"Ausência: SEMÁFORO(<fato_controle>)"**, ao invés de executar os procedimentos normais que implementam um fluxo de teste de ausência, executar uma instrução SQL para verificar, na relação "semáforo", a inexistência de uma tupla com atributo chave "rótulo", valorado com o conteúdo especificado por <fato_controle>.

No exemplo a seguir, aparece ilustrado a aplicação desta regra.

Em ER/T$^+$: Ausência: SEMÁFORO ("inscrição_aberta")

Em SQL embutida:

```
/*  na fase de Testes de Habilitação */

EXEC SQL SELECT rotulo
         FROM  semaforo
         WHERE semaforo.rotulo = "inscrição_aberta" ;

if (sqlca.sqlcod == NOT-FOUND)
   ausencia = 'OK';
else
   ausencia = 'NOK';

if (ausencia = 'OK')
   {EXEC SQL INSERT
         INTO  semaforo (semaforo.rotulo,
                         ...  )
         VALUES         ("inscrição_aberta",
                          ...  );
   }
return(ausencia);
   .
    .

/*  na fase de Confirmação da Transação */

EXEC SQL DELETE
         FROM  semaforo
         WHERE semaforo.rotulo = "inscrição_aberta";
```

5.4 Montagem da transação procedural

O propósito da montagem da transação procedural é de obter uma seqüência de operações que implemente uma transação modelada no diagrama ER/T$^+$ e que mantenha equivalência semântica entre os dois modelos: ER/T$^+$ (modelo conceitual na forma declarativa) e transação SQL (modelo de implementação na forma imperativa). Para cada fluxo ER/T$^+$ são geradas instruções correspondentes, em C e/ou SQL, conforme discutido na seção 5.3. É de responsabilidade de um algoritmo de resolução das dependências o estabelecimento da seqüência de execução dos fluxos do diagrama e, por conseguinte, da seqüência das instruções que implementam estes fluxos.

O mecanismo de seqüencialização de operações se baseia no estabelecimento do vínculo de precedência entre as ações modeladas via os fluxos do diagrama ER/T$^+$. A forma de instrumentalizar é a utilização de um grafo de dependência de fluxos de dados.

Os detalhes da seqüencialização de operações estão descritos a seguir e os algoritmos sugeridos, como alternativa de implementação do modelo de transação aqui proposto, estão contidos no Apêndice B desta tese.

5.4.1 Geração do grafo de dependência e seqüencialização das operações

O grafo de dependência é um grafo dirigido /FUR 73/ e representa a dependência entre os fluxos de um diagrama ER/T$^+$. É usado para estabelecer o vínculo de precedência das operações indicadas pelos fluxos. Neste grafo, seus nodos (ou vértices) representam os fluxos e seus elos (ou arcos) representam a dependência entre os fluxos. O elo do grafo é um elo direcionado onde , na extremidade de origem, aparece um nodo representando o fluxo liberador de outro que o sucede no grafo e, na extremidade de destino, aparece um nodo representando o fluxo dependente de outro que o antecede no grafo.

Para uma perfeita compreensão da terminologia aqui utilizada, cabe considerar os seguintes conceitos e as ilustrações da Figura 14:

.Ramo ou grupo:
- grupo de nodos seqüenciais com uma dependência entre eles;

.Nodo liberador:
- nodo da extremidade de **origem do elo** e representa um fluxo liberador no diagrama ER/T^{+};

.Nodo dependente:
- nodo da extremidade de **destino do elo** e representa um fluxo dependente no diagrama ER/T^{+};

.Nodo de convergência:
- nodo cuja habilitação para execução **depende** da conclusão **de 2 ou mais nodos** assíncronos;
- representa um nodo **X** que possui elos adjacentes com extremidade de destino comum em **X**;

.Nodo de divergência:
- nodo que **libera** a execução de **2 ou mais nodos** dependentes dele;
- representa um nodo **Y** que possui elos adjacentes com extremidade de origem comum em **Y**;

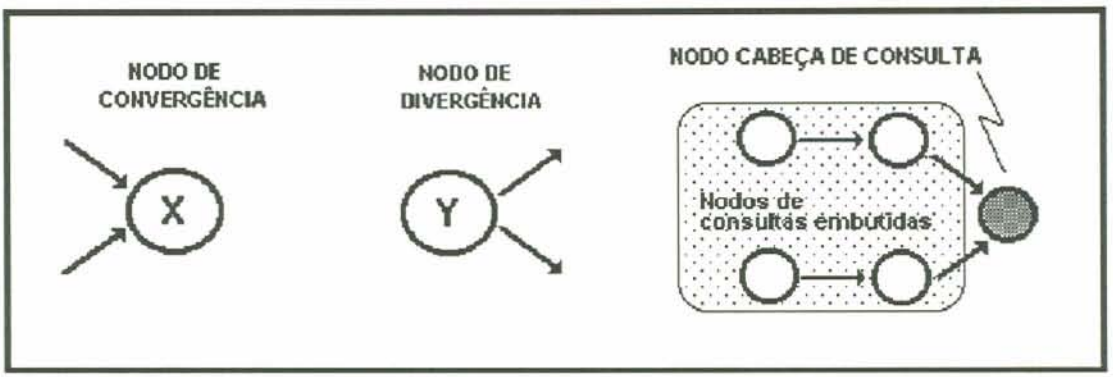


Fig. 14 - Nodos típicos em grafos

.Nodo terminal:
- nodo que **não libera** a execução de nenhum outro nodo, ou seja, **nenhum nodo depende dele**;

.Nodo "cabeça" de consulta:
- nodo de convergência que identifica a consulta(fluxo ER/T^{+}) que tem expressões de subconsultas embutidas e representa a raiz no aninhamento de consultas.

Um exemplo de dependência aparece na Figura 15, onde o fluxo R2.1 somente poderá ser executado após conclusão do fluxo

E2.1 e este, por sua vez, após conclusão do fluxo A1.2. Nesta figura usa-se a seguinte notação para explicar as dependências:

A <-- B

onde **A** indica o fluxo dependente de **B** e
B indica o fluxo liberador de **A**

Os rótulos, denotando os fluxos da Figura 15, têm o seguinte significado:

Xn.m

onde **X** indica o tipo de objeto associado à transação via o fluxo, podendo ser:
A, se for do tipo "agente externo";
E, se for do tipo "entidade";
R, se for do tipo "relacionamento";
H, se for do tipo "relógio".

e onde **n** indica um número seqüencial de objetos dentro de seu tipo e **m** um número seqüencial de itens de dados naquele fluxo.

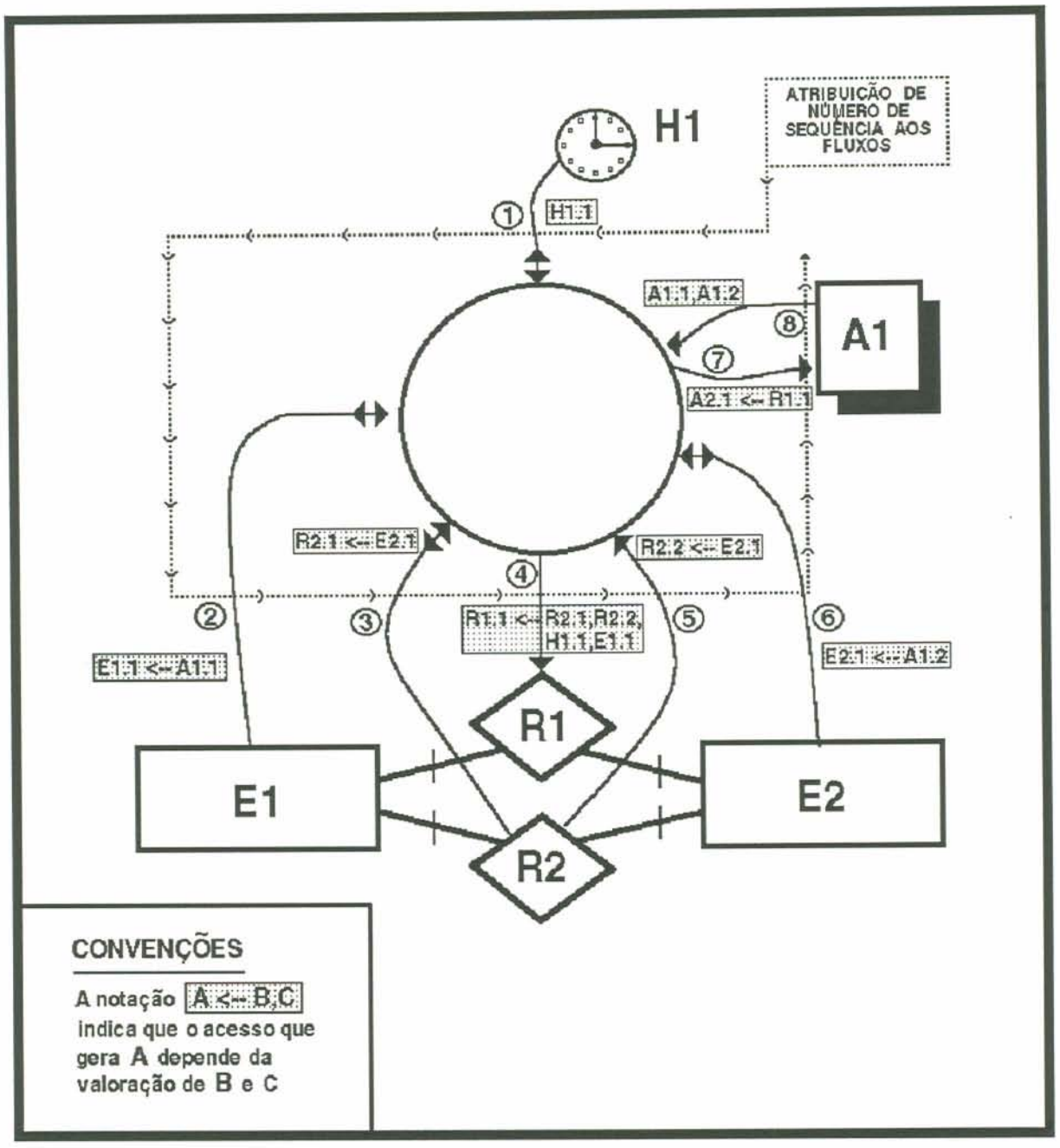


Fig. 15 - Seqüencialização de operações ER/T$^+$

As especificações ER/T$^+$ que geraram a notação usada na rede apresentada na Figura 15 são:

Nro.fluxo	Notação Dependência	Especificação ER/T+
01	H1.1	data_corrente
02	E1.1<-A1.1	sócio
03	R2.1<-E2.1	volumes_com_reserva = CONJUNTO vol de <soc,vol> PRESENTES ONDE vol PERTENCE A volumes_do_pedido
04	R1.1<-R2.1,R2.2, H1.1,E1.1	empréstimos_ok = CONJUNTO <soc,vol> AUSENTES ONDE soc=sócio e (vol NÃO PERTENCE A volumes_com_reserva ou <sócio,vol> PERTENCE A reservas_do_sócio) PARA CADA <sócio,vol> de empréstimos_ok: <sócio,vol>.data_emp = data_corrente
05	R2.2<-E2.1	reservas_do_sócio = CONJUNTO <soc,vol> PRESENTES ONDE soc=sócio e vol PERTENCE A volumes_do_pedido
06	E2.1<-A1.2	volumes_do_pedido = CONJUNTO vol PRESENTES ONDE vol.cod PERTENCE A códigos_volumes
07	A2.1<-R1.1	data_devoluções = {código_volume, data_devolução / PARA CADA <sócio,vol> de empréstimos_ok: código_volume = vol.cod e data_devolução = <sócio,vol>.data_dev}
08	A1.1 A1.2	código_sócio códigos_volumes = {código_volume}

A montagem do grafo de dependência entre os fluxos do diagrama pode ser arbitrada da seguinte forma:

Passo 1 - **Numera os fluxos**
- atribuir, de forma arbitrária e apenas para fins de referência no algoritmo, número de seqüência aos fluxos do diagrama, gerando a seqüência a partir da unidade, no sentido anti-horário (geração dos números arbitrado neste sentido);

Passo 2 - **Gera grafo**
- gerar o grafo de dependência, assinalando as dependências existentes entre os dados (fluxos):
  - . para cada fluxo do diagrama ER/T^+, criar um nodo do grafo iniciando pelo fluxo de número 1;
  - . para cada nodo, assinalar as dependências através de elos direcionados entre os nodos, onde é marcada a origem nos nodos liberadores e o destino nos nodos dependentes.

A Figura 16 ilustra um exemplo de grafo gerado. Este grafo representa a relação de dependência entre os fluxos do diagrama da Figura 15.

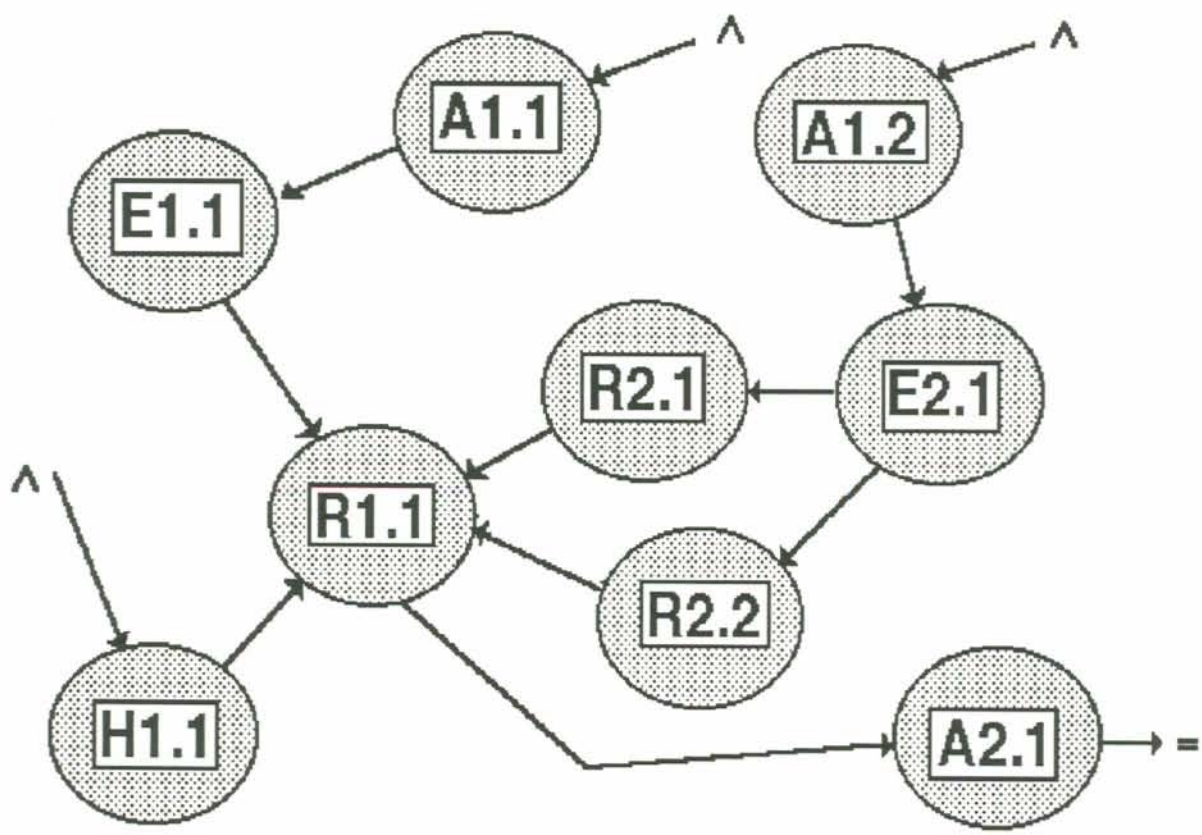


Fig. 16 - Grafo de dependência entre os fluxos ER/T^+

Numa forma tabular, o grafo de dependência gerado pode ser representado como:

Na forma tabular: (a seqüência original)

Nro.fluxo	N o d o	Dependências
1	H1.1	-- // --
2	E1.1	A1.1
3	R2.1	E2.1
4	R1.1	R2.1,R2.2,H1.1,E1.1
5	R2.2	E2.1
6	E2.1	A1.2
7	A2.1	R1.1
8	A1.1	-- // --
8	A1.2	-- // --

Note que, durante a geração do grafo de dependência, para nodos que representam operações de inclusão e exclusão sobre o banco de dados, deve também ser avaliado o modelo de dados para estabelecer um vínculo de precedência entre os fluxos, a partir da dependência funcional entre as tuplas das relações acessadas. Esta dependência funcional é reconhecida pela presença dos atributos estrangeiros nas relações (ver Apêndice A). O vínculo de precedência entre os fluxos de inclusão e os de exclusão torna-se necessário para garantir a integridade referencial do banco de dados (discutido anteriormente na seção 5.2).

Assim, considerando por exemplo um **nodo A**, representando acesso à relação REL_A que possui atributo que é atributo estrangeiro em outra relação, e um **nodo B**, representando acesso à relação REL_B que possui atributo estrangeiro, tem-se:

Para o Modelo de Dados:

```
Relação  : REL_A      Relação  : REL_B
Chave    : A1         Chave    : B1
Atributos: ...        Atributos: ...
           ...                   A1 (Atrib.estrang.c/ REL_A)
```

Precedência na operação de Inclusão:
- primeiro inclui REL_A e depois inclui REL_B;
- a dependência é **B <-- A**, ou seja, **A** liberador de **B**;

Precedência na operação de Exclusão:
- primeiro exclui REL_B e depois exclui REL_A;
- a dependência é **A <-- B**, ou seja, **B** liberador de **A**;

Em se tratando de uma eventual integridade referencial bidirecional, como mostra o exemplo de modelo de dados abaixo:

Modelo de Dados:

```
Relação   : REL_C  (Origem no modelo ER: GENERALIZAÇÃO)
Chave     : C1
Atributos: ...
           D1 (Atrib. estrang.c/ REL_D)

Relação   : REL_D  (Origem no modelo ER: ENTIDADE)
Chave     : D1
Atributos: ...
           C1 (Atrib.estrang.c/ REL_C)
```

A questão de qual tupla incluir primeiro é arbitrada a partir de uma ordem natural de prioridade por tipo de relação, segundo sua origem na especificação do modelo ER. Atribui-se maior prioridade na inclusão a relações que implementam:

```
(+)  1- GENERALIZAÇÃO
     2- ENTIDADE
     3- ENTIDADE FRACA
     4- RELACIONAMENTO
(-)  5- Desempate por antigüidade cronológica de cria-
        ção da relação
```

Assim, considerando o modelo de dados no exemplo apresentado, a tupla da relação REL_C terá prioridade na inclusão sobre a tupla da relação REL_D. Em caso de operações de exclusão, o sentido da prioridade é invertido.

O grafo de dependência é montado durante o processo de tradução da especificação. Para cada diagrama ER/T^+ é gerado um único grafo que será utilizado, durante o processo de execução da transação, tanto na fase de testes de habilitação quanto na fase

de confirmação, respeitando os tipos de operações tratadas em cada fase.

A seqüencialização das operações se propõe a reordenar os fluxos de acordo com a disponibilidade dos dados liberados (habilitação) por fluxos antecedentes, determinando uma seqüência de execução das operações. Esta nova seqüência dos fluxos é obtida a partir do grafo de dependência.

O termo "seqüencialização" não é aqui empregado no contexto de linearização de paralelismos, mas sim no sentido da determinação de dependência e seqüência de passos de execução dos fluxos da especificação, ou seja, no sentido da construção de uma ordem parcial de dependência entre os nodos.

A seqüencialização se configura aqui como o passo 3 na montagem do grafo de dependência. Assim, a seqüencialização é estabelecida por:

Passo 3 - **Resolve as dependências** (fixa a relação de precedência entre fluxos)
- formular seqüência de execução dos procedimentos associados aos fluxos com base na resolução das dependências entre eles:
    . iniciar pelos nodos liberadores sem dependência (ex.:H1.1 (no relógio), A1.1 e A1.2 (na entrada)); se não for observado nenhum nodo sem dependência (todos são dependentes), acusar erro de "falta de dados" para disparar a transação;
    . reordenar o grafo pelas dependências já resolvidas mediante um laço para passagem de todos os nodos de uma lista em seqüência original para uma lista em seqüência ordenada por dependência solucionada (ou, então, realizar um encadeamento entre os nodos da lista na seqüência apropriada); executar o laço até que todos nodos sejam reordenados ou não for constatada nenhuma reordenação em uma iteração completa; caso algum nodo deixar de ser reordenado, acusar erro de "dependência insolúvel";

Como resultado do passo 3 tem-se, na forma tabular, a seguinte seqüência ordenada de fluxos:

Resultado obtido: (a seqüência ordenada)

Nro.fluxo	N o d o	Dependências
1	H1.1	-- // --
8	A1.1	-- // --
8	A1.2	-- // --
2	E1.1	A1.1
6	E2.1	A1.2
3	R2.1	E2.1
5	R2.2	E2.1
4	R1.1	R2.1,R2.2,H1.1,E1.1
7	A2.1	R1.1

O resultado aqui obtido é uma seqüência viável de execução dos fluxos. A solução, no entanto, não está considerando aspectos de otimização dos acessos. O algoritmo aqui proposto considera uma versão inicial na qual somente foi indicada uma possível seqüência de execução dos fluxos. O paralelismo e o aninhamento de fluxos (consultas) são discutidos nas próximas seções, onde são apresentadas versões melhoradas deste algoritmo original.

5.4.2 Paralelismos na execução dos fluxos

A possibilidade de execução paralela dos fluxos do diagrama pode ser considerada mediante uma determinação, no grafo de dependência, de quais os fluxos são funcionalmente independentes. Por exemplo, na Figura 16 é possível identificar uma independência entre os nodos A1.1, A1.2 e H1.1 sendo, portanto, passíveis de execução em paralelo. Através da identificação dos nodos do grafo que representam os fluxos independentes é possível estabelecer-se paralelismo entre nodos.

O reconhecimento dos nodos passíveis de paralelismo é feito durante o processo de seqüencialização dos fluxos. Este reconhecimento é determinado, durante a reordenação dos fluxos, pelo algoritmo de resolução de dependências dos fluxos (ver Passo 3 na seção 5.4.1).

No processo de seqüencialização das operações, como já há uma verificação da dependência entre os fluxos do diagrama ER/T^{+}, basta aí assinalar paralelismo para aqueles que são funcionalmente independentes entre si.

Esta solução de paralelismo entre nodos, quando eles são isoladamente considerados, tem a restrição de não levar em conta que os nodos subseqüentes aos que foram demarcados como paralelos, também eventualmente podem ser executados de forma paralela entre si. Portanto, a solução apresenta deficiências de desempenho, atingindo máximo ganho de paralelismo somente se os tempos de execução de cada fluxo paralelo fossem iguais. Enquanto houver um fluxo paralelo em execução, os demais, que já tiveram a sua execução concluída, permanecem aguardando o término do fluxo pendente.

Em vista desta restrição, torna-se necessário investigar uma alternativa de implementar paralelismos entre ramos do grafo para que, dentro dos ramos, possa continuar a execução seqüencial dos nodos, enquanto outros ramos paralelos continuam sua execução independente. Um exemplo ilustrativo pode ser visto na Figura 17.

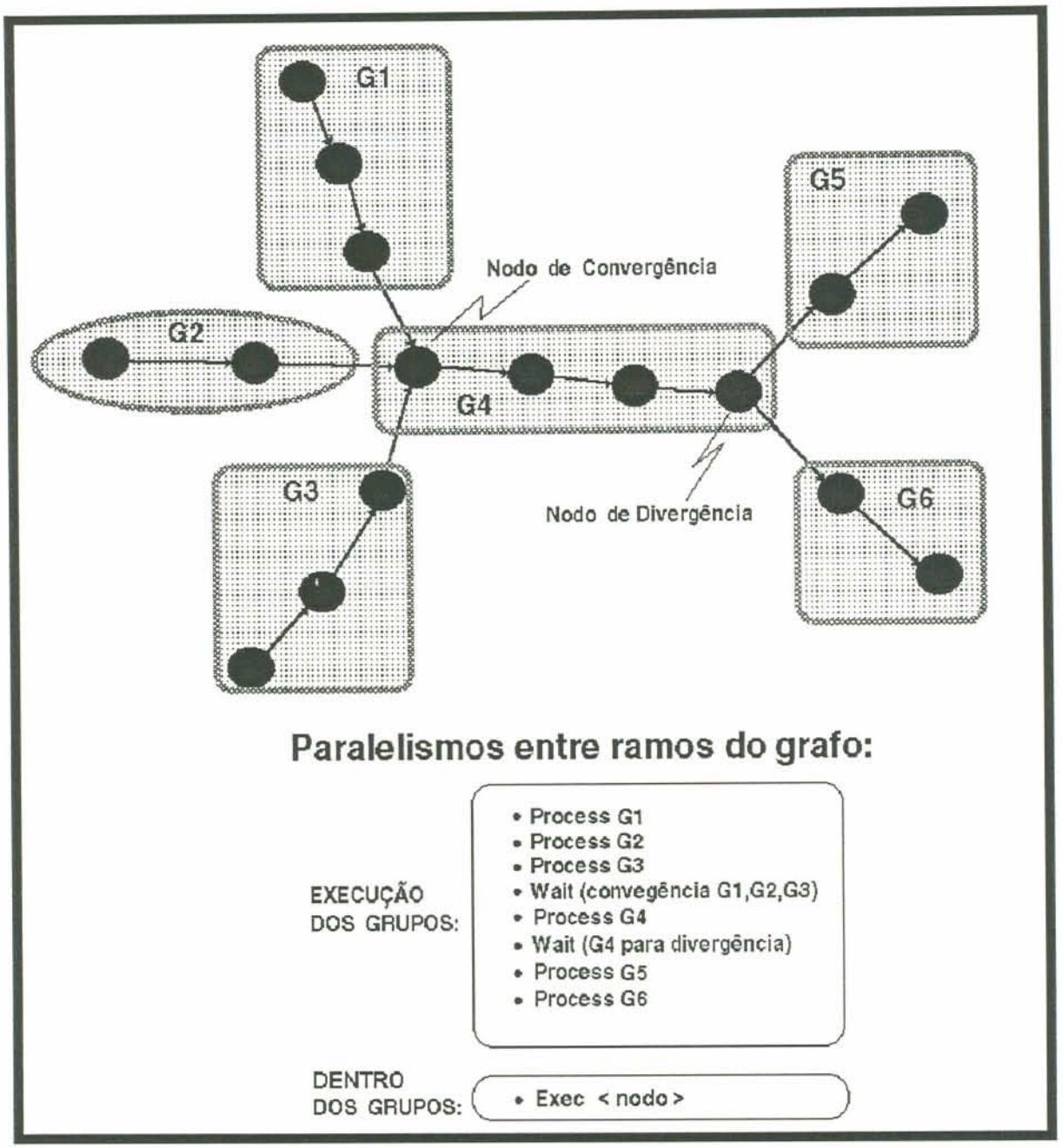


Fig. 17 - Grafo com especificação de ramos paralelos

As operações ilustradas na Figura 17, sobre os grupos paralelos e dentro deles, são baseadas em conceitos de paralelismos e concorrência (/HOL 78/, /CRI 88/, /TAN 92/). Estas operações são definidas como:

.EXEC : operação para execução de tarefas seqüenciais (síncronos);

.PROCESS: operação para execução de tarefas paralelas (assíncronos);
.WAIT : operação de espera pela conclusão de uma ou mais tarefas (sincronismo entre tarefas paralelas).

Por definição, uma operação EXEC <nodo> é semanticamente equivalente a uma operação PROCESS <nodo> seguido de uma operação WAIT <nodo>. Assim, dentro dos grupos paralelos, os nodos são executados seqüencialmente (operação EXEC), não necessitando especificação de operações de espera (WAIT) entre os nodos.

Pela Figura 17, pode ser observado que todos os nodos (fluxos) pertencentes a ramos diferentes são passíveis de execução em paralelo desde que assegurada a sua habilitação por nodos anteriores, ou seja, os ramos entre si podem ser executados de forma paralela. Já dentro de um ramo, o processamento deve ser seqüencial.

Um mecanismo de disparo dos grupos paralelos deve reconhecer os grupos paralelos e, uma vez identificados, ativar de forma assíncrona (PROCESS) todos aqueles cujo primeiro nodo, dentro de cada grupo, não seja dependente de outro(s) grupo(s). Uma operação WAIT deve ser usada para sincronizar os grupos paralelos. A Figura 17 mostra a seqüência de disparo de grupos paralelos (execução dos grupos) para o grafo usado como exemplo na mesma figura.

A versão de paralelismo entre ramos produz um maior grau de paralelismo de execução entre as tarefas. Permite que múltiplas tarefas sejam executadas simultaneamente, exigindo sincronismo somente entre nodos seqüenciais, que aparecem dentro de um ramo ou grupo, e nos nodos de convergência e de divergência.

5.4.3 Aninhamento de consultas

A partir da obtenção do grafo de dependência de fluxos de dados para os fluxos do diagrama ER/T$^+$, é possível investigar

quais consultas podem ser aninhadas e, com isto, realizar um embutimento de subconsultas.

O aninhamento de consultas, ou mecanismo de subconsultas na terminologia adotada em /DAT 86a/, é uma alternativa de implementação que merece atenção, pois permite o aproveitamento do potencial da linguagem de consulta do SGBD.

Analisando a Figura 15 e o grafo de dependência correspondente na Figura 16, constata-se que o fluxo número 4 (R1.1<-R2.1,R2.2,H1.1,E1.1), por exemplo, pode ser executado a partir do argumento H1.1 (relógio) e a partir de um aninhamento com as consultas que irão produzir R2.1 e R2.2 (ambas realizáveis tendo a consulta E2.1 já efetivada) e E1.1 (tendo disponível o dado de entrada A1.1). Em outras palavras, a consulta R1.1 pode ser construída embutindo-se, em sua especificação, as subconsultas R2.1, R2.2 e E1.1 e tendo anteriormente assegurado os dados usados como argumentos para estas três subconsultas e o acesso aos dados fornecidos via fluxo H1.1.

Considere, por exemplo, dois fluxos de um diagrama ER/T$^+$, conforme especificação a seguir:

Fluxo 1:

```
livros_ficcao = CONJUNTO volume
                PRESENTES
                ONDE volume.assunto = "ficcao"
```

Fluxo 2:

```
CONJUNTO vol DE <socio,volume>
                PRESENTES
                ONDE vol PERTENCE A
                         livros_ficcao
```

No exemplo, o fluxo 2 depende da consulta obtida no fluxo 1. Estas consultas, quando mapeadas para a linguagem SQL, podem estar aninhadas como segue:

Em SQL:

```
SELECT vol
       FROM emprestimo
       WHERE vol IN
             (SELECT *
                FROM volume
                WHERE volume.assunto = "ficcao")
```

onde "emprestimo" é o nome da relação que implementa o relacionamento <socio,volume>.

Assim, a subconsulta que representa o fluxo 1 (SELECT * FROM volume ...) está embutida na consulta que representa o fluxo 2 (SELECT vol FROM emprestimo ...).

Para determinar aninhamento de expressões de consultas, devem ser considerados dois tipos básicos de consultas ao banco de dados. São elas:

.Consultas embutidas:
- expressões de subconsultas que estão embutidas em outras consultas;

.Consultas instanciadas:
- expressões de consultas que não podem estar embutidas em outras consultas, pois aparecem representadas em nodos terminais do grafo ou estão presentes em **mais de um fluxo** do diagrama ER/T^+. Podem, no entanto, conter subconsultas embutidas.

Considerando as idéias aqui discutidas e avaliando o vínculo de precedência estabelecido no grafo de dependência, cabe agora generalizar-se uma possível regra de aninhamento de consultas como segue:

Instanciar consulta:
- Instanciar uma consulta quando aparece representada em **nodo terminal** ou em **nodo de divergência**;

Embutir consulta:
- Embutir consultas representadas em nodo liberador como expressão de subconsulta de uma outra representada em nodo dependente. O embutimento pode ser feito sempre que o nodo liberador não for um nodo de divergência.

Na Figura 18 aparece ilustrado o reconhecimento de consultas aninhadas a partir de um grafo de dependência de fluxos de dados.

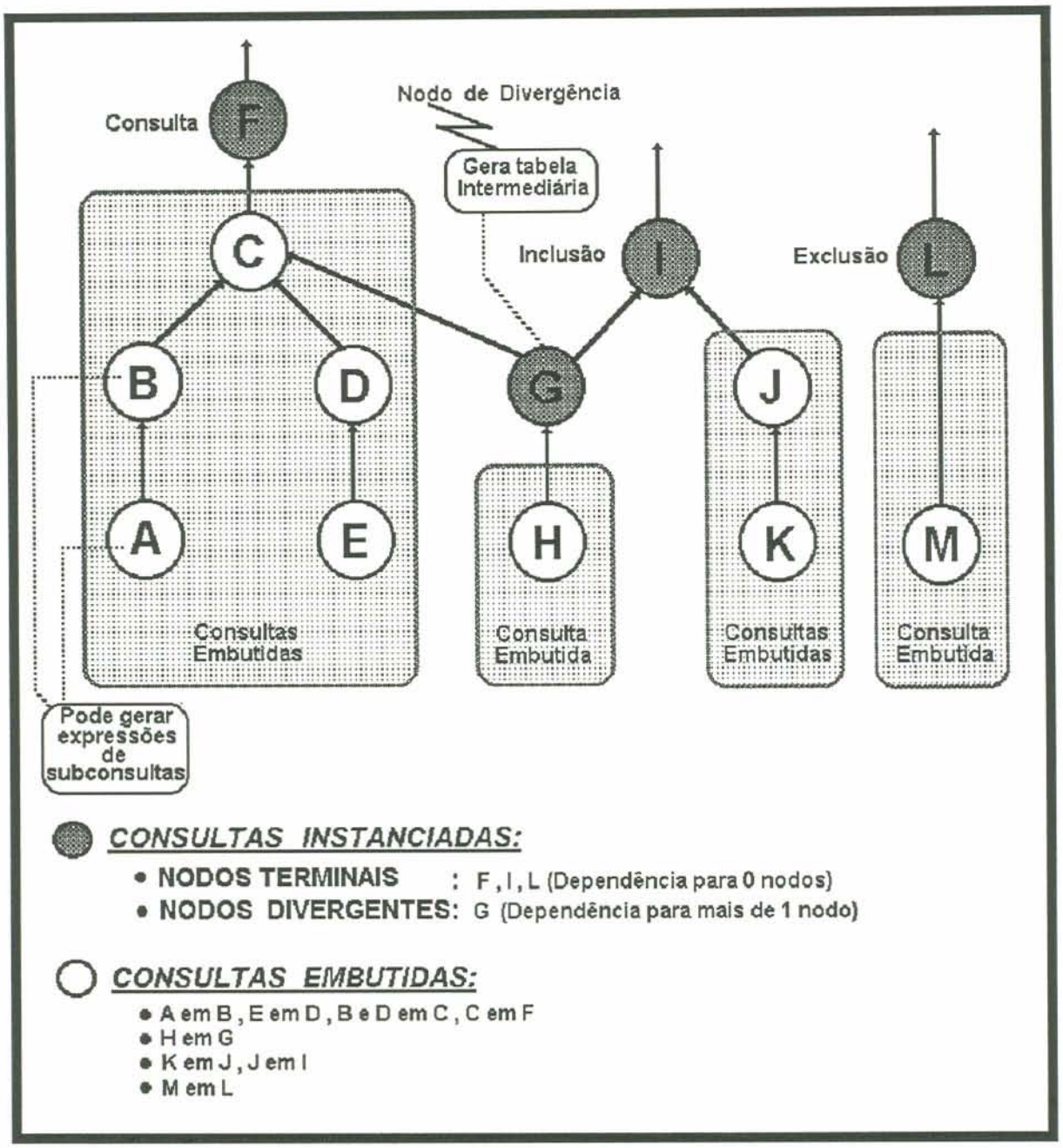


Fig. 18 - Grafo de dependência e o aninhamento de consultas

No exemplo da Figura 18 foi aplicada a regra de aninhamento definida. Convém no entanto salientar que, para facilitar a geração de código SQL correspondente aos fluxos ER/T$^+$ e os procedimentos de tratamento de exceções associados a cada fluxo (discutidos no capítulo 6), o conveniente é orientar o processo de embutimento por tipo de fluxos, ou seja, não embutir,

em um mesmo grupo, tipos de fluxos distintos. Em um grupo de consultas embutidas devem aparecer somente expressões de subconsultas que representam o mesmo tipo de fluxo no diagrama.

Cabe também frisar que todas as consultas representadas por nodos de divergência geram tabelas intermediárias (temporárias), ao invés de subconsultas embutidas. Justifica-se esta medida pelo fato que as consultas geradas em nodos de divergência são consultas instanciadas, sendo utilizadas por mais de um nodo subseqüente. Portanto, necessitam que a consulta tenha sido efetivamente completada para dar continuidade à execução do grafo. Com este procedimento, evitam-se consultas replicadas para cada nodo dependente da habilitação do nodo de divergência.

A avaliação dos grupos de consultas aninhadas é feita pelo algoritmo de resolução das dependências, juntamente com o processo de seqüencialização de operações. Já a montagem do aninhamento de consultas ocorre durante a construção de expressões SQL correspondentes aos fluxos ER/T$^+$ e a execução destas consultas durante a fase de testes de habilitação da transação, pois as consultas ao BD são todas executadas nesta fase.

5.4.4 Aninhamento x paralelismos: embutir ou executar em paralelo

A decisão de estabelecer aninhamentos de consultas SQL, a partir dos fluxos do diagrama ER/T$^+$, pode vir a prejudicar a intenção de gerar paralelismos na execução destes fluxos. O mecanismo de embutimento de subconsultas afeta diretamente a determinação do paralelismo. É conflitante sob o ponto de vista do módulo de tradução de especificações ER/T$^+$ para SQL. Casos, por exemplo, onde seja feita a escolha entre as alternativas: **(a)** obter um maior paralelismo de execução, quando gerenciado pelo algoritmo proposto, através da liberação de um maior número de consultas instanciadas (não embutimento) ao SGBD, ou; **(b)** fazer um agrupamento (embutimento) de um conjunto de consultas em expressões mais complexas de consultas aninhadas e liberando, de uma só vez, todo o grupo de subconsultas ao SGBD. Na alternativa

b), o controle de paralelismo fica por conta do próprio SGBD. Paralelismos entre subconsultas de consultas aninhadas, são tratadas pelo SGBD e, neste caso, pode eventualmente haver um menor paralelismo se comparado com paralelismos gerados pelo algoritmo proposto.

A influência do aninhamento de consultas sobre a determinação de paralelismos pode ser ilustrada analisando-se a Figura 19 e focalizando os nodos R2.1 e R2.2. Pela regra de determinação de paralelismos, podem ser executados de forma independente. Com o aninhamento, passam a fazer parte de um mesmo grupo de consulta, portanto não mais sujeitos a execução em paralelo via o algoritmo proposto anteriormente.

Portanto, é necessário agora compatibilizar o embutimento das consultas com a identificação de paralelismos entre os fluxos. Neste procedimento, o grupo de consultas aninhadas é considerado como se fosse um nodo atômico (ver Figura 19) e a compatibilização pode ser estabelecida a partir do quadro comparativo, a seguir apresentado, onde são ilustradas as características que os diferentes tipos de nodos assumem frente ao paralelismo e ao aninhamento.

Tipo de Nodo	PARALELISMO	ANINHAMENTO
Convergente:	- espera(Wait) conclusão de ramos convergentes. - gera grupo separado dos ramos convergentes.	- pode estar no mesmo grupo de ramos convergentes.
Divergente :	- gera grupo separado dos ramos que dele são divergentes.	- gera instanciamento no nodo de divergência.
De um ramo :	- integra o grupo de nodos seqüenciais dentro do ramo e com paralelismo entre ramos. - se nodo anterior for de divergência, espera(Wait) sua conclusão e abre grupo separado deste.	- pode estar no mesmo grupo de nodos embutíveis. - pode estar no mesmo grupo de ramos convergentes embutíveis.

A avaliação deste quadro mostra que os nodos diretamente afetados, por esta compatibilização, são os de

convergência. Em um paralelismo entre ramos, os nodos de convergência , forçosamente, tinham que compor um grupo separado dos ramos que convergiam para este nodo (ver Figura 17, o nodo de convergência). No caso de anhinhamentos de consultas, se o nodo convergente representa um fluxo de acesso ao banco de dados (considerável, portanto, para embutimento), este nodo pode compor grupo juntamente com seus ramos de convergência (ver Figura 19, nodo R1.1).

O conjunto de nodos dentro de um mesmo ramo também deve receber uma certa atenção. No caso de paralelismos, os nodos do ramo formam, obrigatoriamente, um mesmo grupo que é independente (paralelo) dos demais ramos e cujo sincronismo é estabelecido pelos nodos de convergência e divergência. No caso de aninhamento, dois nodos de um mesmo ramo podem tanto fazer parte de grupos de embutimentos diferentes (caso de embutimentos distintos por tipo de fluxos em um ramo) assim como nodos de ramos diferentes podem eventualmente fazer parte de um mesmo grupo de embutimento (caso de ramos convergentes).

Assim, na busca de paralelismo simultâneo ao aninhamento de consultas, pode-se estabelecer como <u>regra de compatibilização de aninhamento com paralelismo</u>:

- determinar os grupos de consultas aninhadas (embutimentos);
- considerar cada grupo de embutimento como um nodo atômico;
- determinar os paralelismos considerando, agora, a nova situação:
  - . além dos critérios de assinalamento de paralelismo anteriormente discutidos (seção 5.4.2), para o caso de nodos que representam consultas passíveis de um embutimento, marcar grupo de paralelismo para o referido grupo de embutimento, somente no instante de localizar o <u>nodo cabeça de consulta</u> para então, num processo de recursão, marcar os nodos filhos.

A Figura 19 apresenta um exemplo de aplicação desta regra.

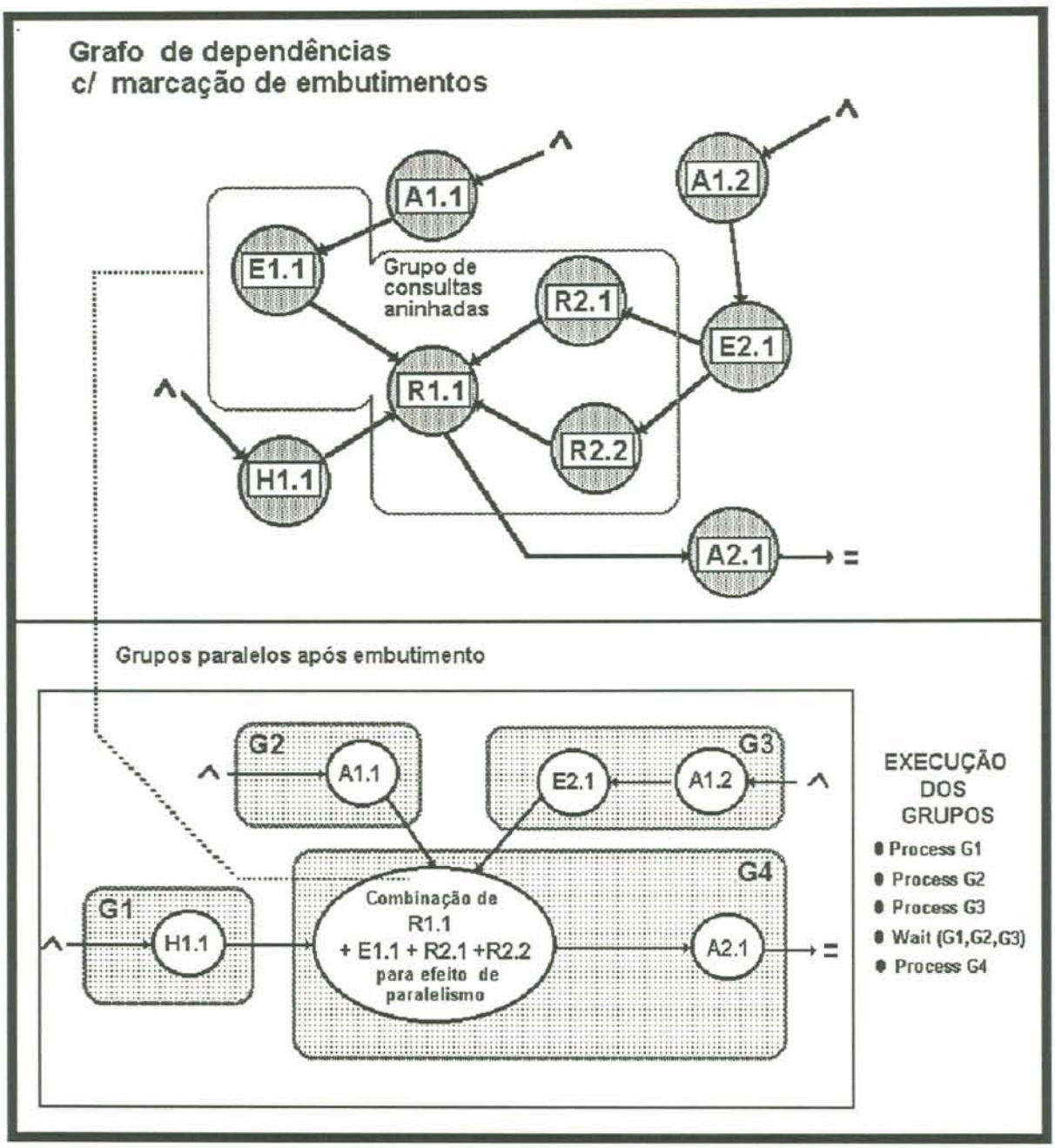


Fig. 19 - Aninhamento x Paralelismo

5.4.5 Compondo o processo de proceduralização

Uma retrospectiva sobre a técnica utilizada para conseguir a proceduralização das especificações mostra que as seguintes etapas foram, até o presente momento, realizadas:

1) Mapeamento das estruturas de dados (APÊNDICE A);
2) Mapeamento de cada fluxo ER/T$^+$ isolado para instruções SQL;
3) Seqüencialização das operações (implementado na versão 1 do algoritmo proposto no APÊNDICE B);
4) Determinação de paralelismos na execução dos fluxos (implementado na versão 3 do algoritmo proposto no APÊNDICE B);
5) Determinação de possíveis aninhamentos de expressões de consultas;
6) Compatibilização entre aninhamento e paralelismo (implementado na versão 4 do algoritmo proposto no APÊNDICE B).

O algoritmo de resolução de dependências dos fluxos foi sucessivamente melhorado, incorporando os procedimentos necessários à implementação de cada uma das funções correspondentes às etapas 3 a 6 anteriormente discutidas. Analisando o algoritmo proposto, constata-se que a técnica aqui apresentada se fundamenta, basicamente, no grafo de dependência de fluxos de dados como instrumento de determinação do vínculo de precedência entre os fluxos do diagrama ER/T$^+$, bem como para a demarcação de aninhamento e paralelismos.

Esta solução encontra respaldo em soluções análogas aplicadas em outras áreas da informática como, por exemplo, a área de arquitetura de computadores, mais especificamente nas arquiteturas super escalares /FER 92/ que exploram paralelismos de baixo nível (em microinstruções).

Na arquitetura de algumas máquinas, a seqüência original do programa fonte sofre modificações, alterando-se a ordem de sua execução. Isto ocorre em vista de mecanismos de otimimização de código ou mecanismos de hardware encarregados pela transferência das operações para as diversas unidades funcionais do processador. Existe uma reorganização do código mantendo-se, no entanto, uma equivalência semântica entre a especificação do programa e o resultado obtido.

A diferença básica, entre a técnica proposta na presente tese e a solução discutida na literatura de arquitetura de computadores, reside no propósito da aplicação (finalidade de uso) de um grafo de dependência, durante a obtenção de uma solução em cada uma das situações. No nível de execução e especificação das instruções, cujo vínculo de precedência está representado neste grafo, aparece outra diferença.

Nas pesquisas da área de arquitetura de computadores, a partir de uma seqüência especificada no fonte do programa (linguagem procedural), é montado um grafo de dependência de dados para avaliar os possíveis paralelismos, ou seja, a partir de uma seqüência de comandos, procura-se determinar quais os que podem ser executados em paralelo e, com isso, na intenção de obter código mais otimizado, permitir uma possível reordenação destes comandos. Já na técnica proposta nesta tese, um grafo de dependência de fluxos de dados é gerado para dar origem a uma possível seqüencialização de uma linguagem declarativa. Adicionalmente, a partir da seqüência gerada, podem ser avaliados os possíveis paralelismos de execução (similar ao uso feito em arquitetura de computadores). O grafo de dependência é aqui usado numa fase de obtenção de uma seqüência, fase esta anterior a detecção de paralelismos. Para a área de arquitetura de computadores, esta seqüência já existe e, portanto, o grafo é utilizado exclusivamente para fins de determinação de paralelismos.

5.4.6 Otimização das consultas ao banco de dados

As consultas a um banco de dados relacional, normalmente, indicam somente o resultado esperado na consulta. Eventualmente, é indicado um caminho a ser usado no seu processamento, porém não é explicitado qual o melhor caminho a ser utilizado para recuperar os dados que a materializam. Encontrar uma estratégia ótima para o processamento de uma consulta, é tarefa das técnicas de otimização.

Alguns trabalhos, como /HAL 76/ e /GOL 80/, propuseram investigações sobre técnicas para melhorar a eficiência de instruções de consultas de BD relacionais em alto nível. Nestes trabalhos, a fim de diminuir o volume de dados manipulados durante a execução completa de uma instrução de consulta ao banco de dados, já se buscavam formas de reordenar as instruções. Transformando a seqüência original da consulta especificada pelo usuário, em uma seqüência alternativa (assegurados os propósitos da consulta), procurava-se diminuir o consumo de recursos necessários para a sua execução e, por conseguinte, otimizando os passos de efetivação desta consulta.

Portanto, abstraindo uma eventual adoção de estruturas auxiliares de acesso ou métodos de classificação prévia de tuplas, o processo de otimização se baseia na combinação de instruções e na alteração da seqüência na qual elas aparecem na expressão.

A idéia básica na transformação da seqüência das consultas gira em torno das seguintes premissas:

a) instruções que contribuem para a diminuição do volume de dados (projeção) devem ser antecipadas;
b) instruções que aumentam o volume de dados (junções) devem ser executadas por último.

A projeção ou estreitamento de tabelas e, por conseguinte, a diminuição de volume de dados, se baseia na diminuição do tamanho das tuplas e, eventualmente, na redução da cardinalidade da tabela, conseguida através da eliminação de tuplas replicadas.

Considerando estas premissas, as transformações propostas foram agrupadas em 3 tipos básicos, a saber:

a) eliminação de instruções redundantes (por exemplo, as instruções que manipulam tabelas vazias) aplicando, como regra de eliminação, as leis de idempotência ($A \cup A = A$; $A \cap A = A$; $A - A = \emptyset$) /HAL76/ da álgebra booleana;
b) combinação de várias instruções em uma só(por exemplo aquelas que manipulam subconjuntos de uma mesma relação);

c) deslocamento de instruções dentro da árvore algébrica de instruções, antecipando aquelas que diminuem o volume de dados envolvidos e retardando as que aumentam este volume.

Para exemplificar o processo de transformação de uma consulta, a Figura 20 ilustra o deslocamento de instruções dentro da árvore de instruções. Nesta figura, para a representação de tipos de consultas, a mesma simbologia usada em /GOL 80/ foi mantida.

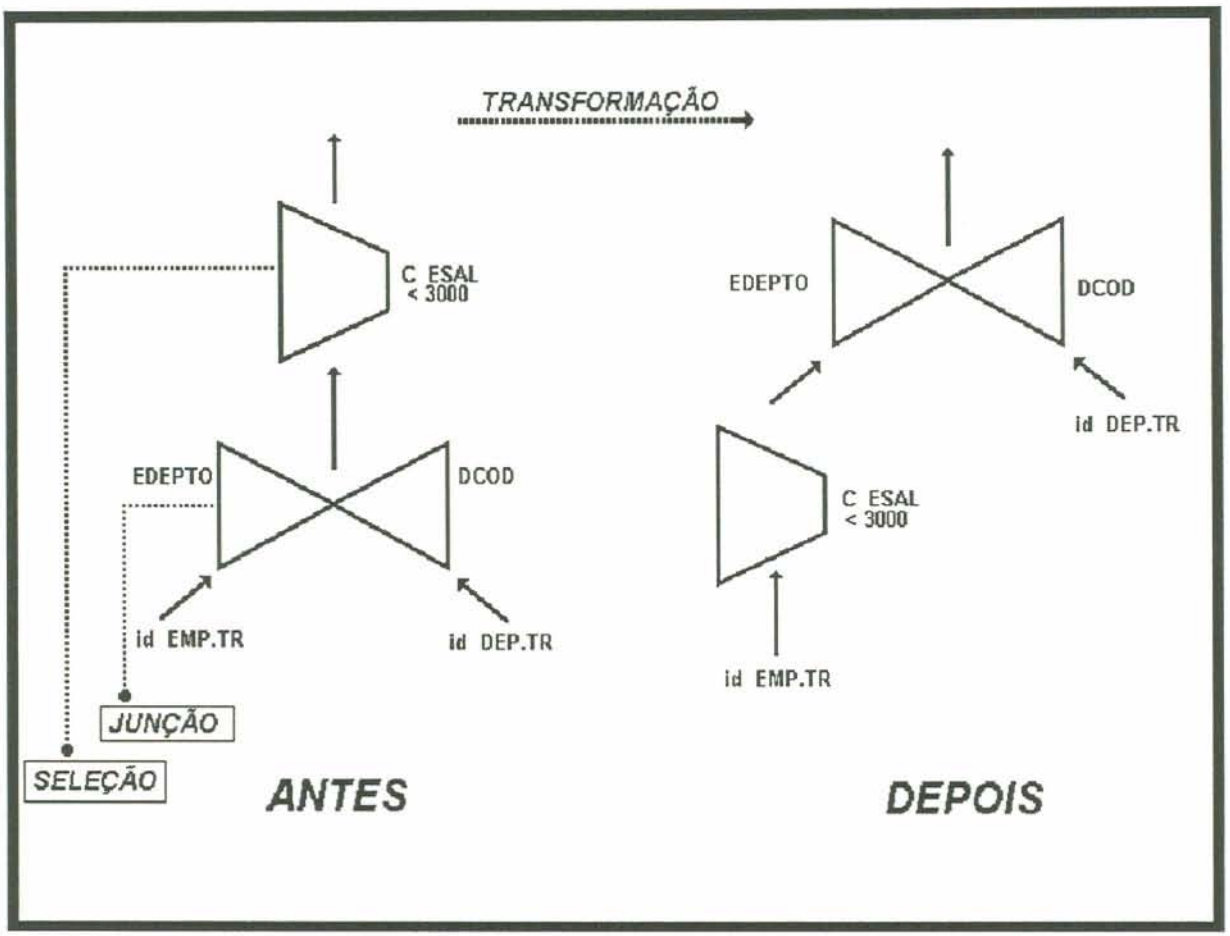


Fig. 20 - Transformação de consulta visando uma otimização
(Adaptado de /GOL 80/)

Neste exemplo, depois da transformação, somente as tuplas que atendem ao critério de seleção (ESAL<3000) serão consideradas para a instrução seguinte que, no exemplo, é uma instrução de Junção.

É intenção também que, a cada passo da transformação de consultas, sejam mantidos somente os atributos da relação que são necessários para as instruções subseqüentes. Isto é conseguido através da antecipação das instruções de projeção.

A ordem de classificação em que as tuplas são liberadas para as consultas subseqüentes, também pode afetar sensivelmente a eficiência do grupo de consultas (conjunto de instruções na árvore de instruções) /GOL 80/. Um trabalho complementar, enfocando este tema, é descrito em /LIM 82/.

Trabalhos mais recentes (/FRE 85/, /FRE 87/, /LOH 87/, /PIR 91/) discutem o tema sob o enfoque de regras de transformação e reescrita de consultas para melhorar o seu desempenho na execução. Em /LOH 87/, por exemplo, o autor propõe que estas regras não sejam simplesmente incorporadas ao código do programa, mas que sejam representadas como dados, numa gramática tipo regras de produção, à semelhança da estratégia usada nos sistemas especialistas /AHL 91b/. Em /GAN 87/ é proposto um método (algoritmo) recursivo para avaliar consultas SQL aninhadas e obter uma versão otimizada das subconsultas embutidas numa expressão SQL.

Adotando-se as idéias sobre otimização de consultas aqui discutidas, o mecanismo a seguir descrito pode aplicá-las ao contexto da tese.

Assim, o processo de otimização das consultas deve ser ativado somente após realizado o mapeamento dos fluxos ER/T^+ para SQL e após a montagem do conjunto de consultas aninhadas. Isto porque os procedimentos de otimização são aplicados sobre expressões SQL já montadas. A otimização se dará sobre as operações SELECT do SQL.

A idéia de dar prioridade à execução de consultas do tipo "projeção", para posteriormente executar as do tipo "junção", não deve conflitar com a seqüência determinada pelo grafo de dependência de fluxos de dados que estabeleceu o grupo de subconsultas aninhadas.

Sendo assim, medidas para prover uma otimização devem centrar-se sobre os seguintes procedimentos:

a) Deslocar, na árvore de expressões de subconsultas (montada a partir do grafo de dependência de fluxos de dados), as "<condições>" de seleção de tuplas para junto dos comandos SQL que acessam diretamente a origem dos dados minimizando, ou até inibindo, a seleção sobre dados que já foram combinados e/ou compilados. Deslocando as condições de seleção para junto da origem dos dados, será naturalmente realizada uma combinação de critérios de seleção especificados em subconsultas que manipulam as mesmas tabelas (MERGE de SELECT /PIR 91/). Obtem-se, assim, um estreitamento das consultas por diminuição da cardinalidade da tabela.

b) Investigar as saídas para avaliar quais os atributos realmente necessários (diretamente ou por composição) e percorrer a árvore de consultas, eliminando aqueles que se originaram a partir de uma agregação durante sucessivas consultas mal especificadas. Assim, os únicos atributos que devem ser mantidos na tupla são os que:
   1- aparecem no resultado final da consulta;
   2- são necessários para processamento de operações subseqüentes.

   Isto permite um estreitamento das consultas por diminuição de tamanho de tuplas manipuladas.

Assim, considerando os embutimentos de subconsultas anteriormente estabelecidos, a otimização das expressões SQL é feita da seguinte forma:

. Dentro dos grupos de consultas aninhadas (árvore de expressões de subconsultas):
  - Prover uma diminuição da cardinalidade das tabelas através da detecção das cláusulas de "<condições>" de seleção de tuplas mal posicionadas nos comandos SQL, deslocando estas cláusulas para junto da origem dos dados. Já que a expressão completa SQL com os comandos SELECT embutidos, é passada ao SGBD, eventualmente este procedimento de otimização pode estar presente nos mecanismos de otimização incorporados no próprio SGBD.

. Na geração de tabelas temporárias que mantem os resultados intermediários:
  - Prover uma diminuição do tamanho de tuplas manipuladas através da eliminação dos atributos não necessários para as consultas subseqüentes representadas no grafo de dependência, ou seja, eliminar aqueles atributos que foram sendo recuperados e agregados durante consultas anteriores e que são desnecessários para as próximas consultas.

5.5 Conclusão

Neste capítulo foram propostos mecanismos de implementação das especificações ER/T$^+$, transformando a linguagem de especificação declarativa em uma correspondente linguagem de implementação procedural.

Foi proposto um modelo geral de implementação da transação, baseado nos mecanismos de execução das redes de Petri e na propriedade de causa e efeito presente no modelo ER/T$^+$, com as devidas adaptações para permitir as validações dos dados contra a base de dados e garantir o conceito ACID de transação.

Com base no modelo de implementação proposto e a partir do formalismo da linguagem ER/T$^+$, foram definidos mecanismos de implementação dos aspectos estruturais (dados) e comportamentais (transações) do banco de dados e das restrições de integridade estáticas.

Em vista da importância do tema frente aos propósitos da tese, atenção especial foi dada ao mecanismo de montagem da transação procedural, responsável pela seqüencialização de operações, tratando a não proceduralidade da especificação mediante seu mapeamento para um modelo procedural. A solução adotada para viabilizar a montagem da transação procedural foi fundamentada em um grafo de dependência que, além de estabelecer o vínculo de precedência entre as ações (fluxos ER/T$^+$) da transação, permitiu também avaliar possíveis paralelismos na execução destas ações. Também permitiu estabelecer um eventual aninhamento de consultas representadas pelos fluxos ER/T$^+$.

Complementando o mecanismo de montagem da transação procedural, foram avaliados aspectos de otimização das consultas ao banco de dados. Para esta otimização, foi proposta uma transformação nas expressões SQL oriundas do processo de aninhamento de consultas, detectando cláusulas de seleção de tuplas mal posicionadas e deslocando-as para junto à origem dos dados, além de prover uma eliminação de atributos não utilizados em consultas subseqüentes.

6. O PROCESSO DE TRADUÇÃO E EXECUÇÃO DA ESPECIFICAÇÃO

As idéias discutidas no capítulo anterior são agora postas em prática para compor-se o processo de tradução e execução da especificação. Este processo pode ser sintetizado por:

.Módulo de tradução:
- procedimentos de tradução da especificação e geração de código.

.Módulo de execução:
- procedimentos de execução do código gerado a partir da tradução.

O módulo de tradução é o responsável pelo mapeamento da especificação ER/T^+ em uma equivalente especificação procedural. O algoritmo de resolução das dependências dos fluxos, proposto anteriormente e usado neste módulo, incorpora a função de geração do grafo de dependência e as funções de seqüencialização de operações, de determinação de possíveis aninhamentos de consultas e de determinação de paralelismos na execução. Todas estas funções são demarcadas usando o grafo de dependência.

Concluída esta fase inicial de tradução, é iniciado o procedimento de construção das expressões SQL correspondentes à especificação ER/T^+. Para esta construção são realizadas as seguintes funções:

.mapeamento dos fluxos para expressões SQL (conforme regras analisadas na seção 5.3), na seqüência estabelecida anteriormente;
.montagem de aninhamento de consultas SQL, a partir dos possíveis aninhamentos marcados no grafo de dependência;
.otimização de consultas.

O módulo de execução é ativado após concluído, com sucesso, a tradução feita sobre a especificação.

A estrutura prevista para o processo de tradução e execução da especificação está ilustrada na Figura 21.

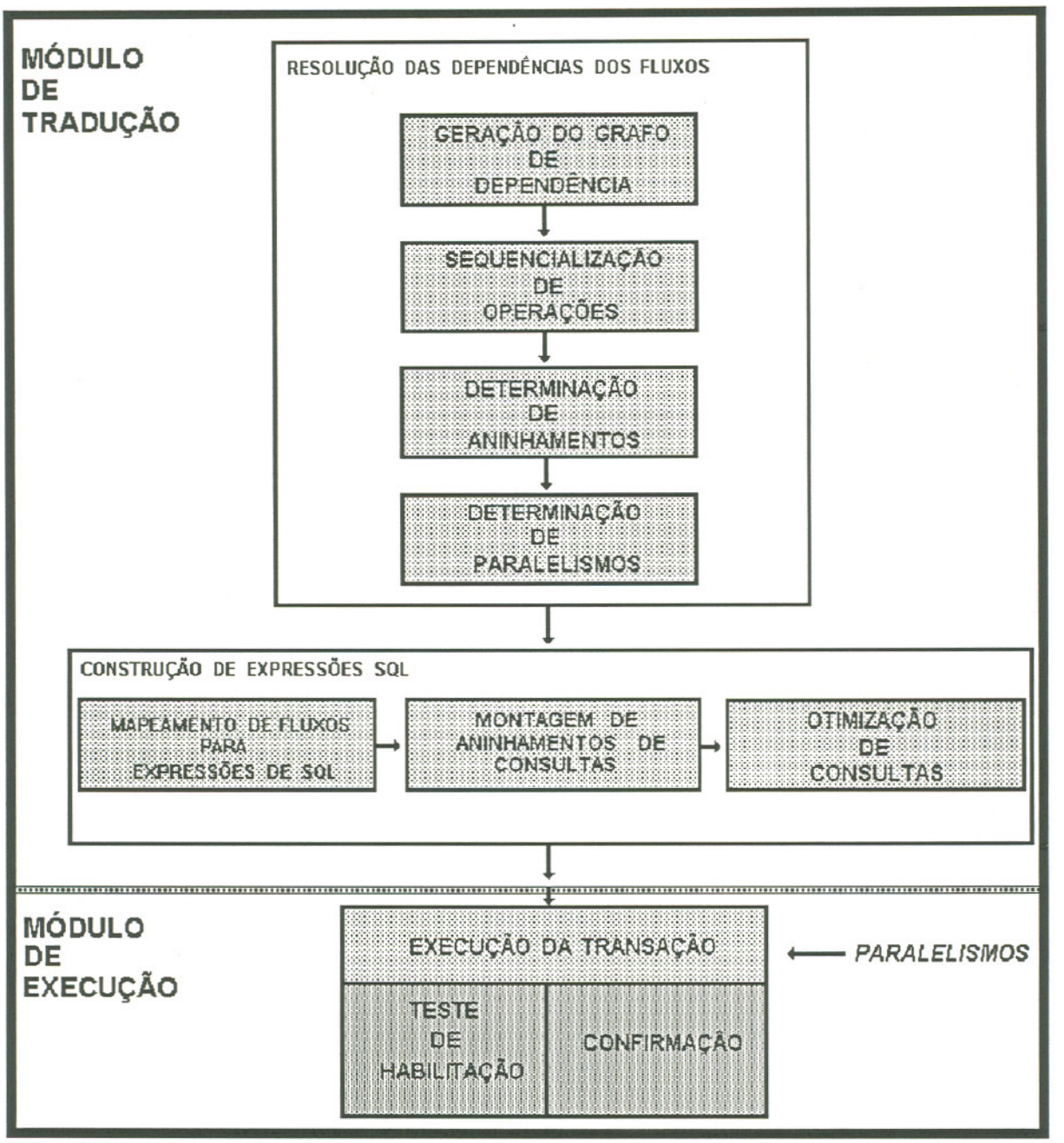


Fig. 21 - Estrutura do Processo de tradução e execução de especificação

O algoritmo que registra os procedimentos de tradução da especificação está a seguir descrito:

MÓDULO DE TRADUÇÃO: Tradução da especificação e geração de código

1.- recebe a especificação textual do diagrama (gerada durante a modelagem e mantida no dicionário de dados);
2.- executa algoritmo de resolução das dependências dos fluxos para avaliar a seqüência de execução destes fluxos(reordena os fluxos por seqüência possível de execução), para estabelecer eventuais aninhamentos de consultas e para determinar paralelismos;
3.- se erro de dependência (originado de um erro na modelagem)
    3.1.- trata erro informando:**"indisponibilidade de dados na seqüencialização dos fluxos"**;
    3.2.- cancela execução do diagrama;
4.- caso contrário (seqüencialização possível)
    4.1.- traduz fluxos ER/T^2 de acesso ao B.D. em expressões SQL;
    4.2.- monta aninhamentos de consultas SQL;
    4.3.- executa algoritmo de otimização de expressões de consulta;

Este algoritmo faz a tradução da especificação e a geração do código.

Uma vez concluída a tradução, o código fonte gerado pode ser submetido a um interpretador SQL ou então entregue a um compilador. Assim, a execução do código pode ser feita de forma compilativa ou interpretativa e se baseia no seguinte algoritmo:

MÓDULO DE EXECUÇÃO: Execução do código

****------- Fase de Testes de Habilitação da Transação ------*****

5.- testa habilitação da transação pelo(s) fluxo(s) de controle:
    5.1.- avalia o(s) fluxo(s) de controle (intervalos de tempo);
    5.2.- se controle(s) não liberado(s) (tempos não válidos):
        5.2.1.- espera por evento de liberação ou período de tempo;
        5.2.2.- volta para 5.1;

6.- executa laço de teste de habilitação de todos fluxos (na seqüência do grafo de dependência) onde:
    para cada fluxo:
    6.1.- testa liberação da execução do fluxo mediante:
        6.1.1.- verificação, no grafo de dependência, se os fluxos liberadores do fluxo corrente foram habilitados, ou seja, as execuções destes fluxos foram bem sucedidas;
        6.1.2.- verificação se variável lógica posfixada (caso fluxo opcional) associada ao fluxo foi valorada como 'verdadeira';
    6.2.- se liberação do fluxo é possível:
        6.2.1.- caso operação no fluxo for:

Entrada:
 .enquanto entrada não completa:
  -exibe tela com layout apropriado;
  -lê dados no formato indicado no D.D.;
  -consiste dados conforme domínio definido no D.D.;
  -se consistente:
   .completa entrada (libera laço de consistência);
   .habilita fluxo;
   .se tratamento de conjuntos na entrada:
    -monta tabela temporária com os dados de entrada;
  -caso contrário:
   .trata erro montando lista de mensagens de inconsistência;
   .exibe tela c/ lista de mensagens;

Teste de Presença:
 .executa teste-de-presença;
 .se presença = 'NOK':
  -desabilita fluxo;
 .caso contrário
  -habilita fluxo;

Teste de Ausência:
 .executa teste-de-ausência-pre;
 .se ausência = 'NOK':
  -desabilita fluxo;
 .caso contrário
  -habilita fluxo;

Inclusão:
 .executa atribuições;
 .executa inclusão; **(*)**
 .se habilita = 'NOK':
  -desabilita fluxo;
 .caso contrário
  -habilita fluxo;
  -se operação sobre relacionamento:
   .testa cardinalidade do relacionamento;

```
                                .se cardinalidade > limite sup.:
                                    -trata erro informando:
                                     "cardinal. > limite superior";
                                    -desabilita fluxo;
                                -se tupla tem atributo estrangeiro:
                                    .testa restrições de integridade
                                     referencial;
                                    .se violação de integridade:
                                        -trata erro informando:
                                         "operação ??? inválida pelo
                                         atributo estrangeiro ???";
                                        -desabilita fluxo;

                            Exclusão:
                              .executa exclusão-pre;
                              .se habilita = 'NOK':
                                  -desabilita fluxo;
                              .caso contrário
                                  -habilita fluxo;
                                  -se operação sobre relacionamento:
                                      .testa cardinalidade do relaciona-
                                       mento;
                                      .se cardinalidade < limite inf.:
                                          -trata erro informando:
                                           "cardinal. < limite inferior";
                                          -desabilita fluxo;
                                  -se tupla tem atributo estrangeiro:
                                      .testa restrições de integridade
                                       referencial;
                                      .se violação de integridade:
                                          -trata erro informando:
                                           "operação ??? inválida pelo
                                           atributo estrangeiro ???";
                                          -desabilita fluxo;

                            Alteração:
                              .executa alteração-pre;
                              .se habilita = 'NOK':
                                  -desabilita fluxo;
                              .caso contrário
                                  -habilita fluxo;
                                  -se tupla tem atributo estrangeiro:
                                      .testa restrições de integridade
                                       referencial;
                                      .se violação de integridade:
                                          -trata erro informando:
                                           "operação ??? inválida pelo
                                           atributo estrangeiro ???";
                                          -desabilita fluxo;

                            Saída:
                              "SEM EFEITO AQUI" (NOP)

        6.2.2.- assinala, no grafo de dependência, a habilitação/desabilitação do fluxo (e também variável lógica prefixada, se fluxo opcional);
    6.3.- caso contrário (não é possível a liberação do fluxo)
        6.3.1.- desabilita fluxo;
        6.3.2.- assinala no grafo de dependência a desabilitação do fluxo(e também variável lógica prefixada, caso fluxo opcional);
        6.3.3.- trata erro informando:"fluxo ??? não habilitado";

7.- se houve desabilitação de fluxos (algum fluxo tenha sido desabilitado, durante os testes de habilitação):
    7.1.- trata erro informando: "transação não habilitada pelos motivos assinalados";
    7.2.- desfaz operações (inclusão, exclusão lógica e teste de ausência);
    7.3.- cancela execução do diagrama;
```

```
**------- Fase de Confirmação da Transação ----------------***
8.- caso contrário (todos fluxos habilitados)
    8.1.- executa laço de execução de todos fluxos (na seqüên-
          cia do grafo de dependência) onde:
          para cada fluxo:
          8.1.1.- caso operação no fluxo for:

                  Entrada ou
                  Teste de Presença:
                   "SEM EFEITO AQUI" (NOP)

                  Teste de Ausência:
                   .executa teste-de-ausência-pos;

                  Inclusão:
                   .se operação sobre relacionamento
                       -atualiza cardinalidade;

                  Alteração:
                   .executa atribuições;
                   .executa alteração-pos;  (*)

                  Exclusão:
                   .executa exclusão-pos;
                   .se operação sobre relacionamento
                       -atualiza cardinalidade;

                  Saída:
                   .exibe tela com layout apropriado indi-
                    cando o resultado;  (*)

Observações:

  (*)  - Na manipulação de conjuntos, durante as operações de
         inclusão, alteração e saída, existe eventualmente a
         necessidade de atribuições a objetos individuais (mo-
         delado via PARA CADA no rodapé da transação). Estas
         atribuições são realizadas dentro das respectivas ro-
         tinas que tratam de cada operação, durante a indivi-
         dualização dos objetos.
```

Os algoritmos de tradução/geração de código (módulo de tradução) e de execução do código (módulo de execução) compõem o processo de tradução e execução da especificação ER/T$^+$. Sobre este processo, cabem os seguintes comentários:

1- O módulo de tradução pressupõe o recebimento da especificação na forma textual. Pressupõe, também, que tenha sido previamente realizada uma análise sintática e semântica da especificação em confronto com as construções da linguagem.

2- O algoritmo de resolução das dependências dos fluxos é aquele proposto na seção 5.4.4 que combina as funções de seqüencialização, aninhamento e paralelismo (no APÊNDICE B, o algoritmo versão 4).

3- Caso alguma inconsistência for detectada na resolução das dependências dos fluxos, o módulo de execução da especificação não é acionado e modificações devem ser realizadas no diagrama via um módulo de modelagem conceitual do sistema.

4- Os procedimentos em SQL, dos respectivos fluxos representados na seção 5.3, são implementados pelas rotinas:

- "executa teste-de-presença"
- "executa teste-de-ausência-pre"
- "executa teste-de-ausência-pos"
- "executa inclusão"
- "executa alteração-pre"
- "executa alteração-pos"
- "executa exclusão-pre"
- "executa exclusão-pos"

Note que, para os fluxos de inclusão, exclusão, teste de presença e teste de ausência, caso sejam considerados para fins de fluxos de controle (semáforos), há também os procedimentos que implementam a sincronização de transações.

5- A rotina "executa atribuições" seleciona, do conjunto de atribuições especificadas no corpo e rodapé do diagrama ER/T$^+$, aquelas que são referentes ao fluxo em questão.

6- A rotina "atualiza cardinalidade" realiza a atualização dos atributos de população nas relações que implementam as entidades envolvidas no relacionamento.

7- A rotina "desfaz operações" reverte as atualizações prévias sobre o banco de dados (inclusão, exclusão lógica e teste de ausência) que foram realizadas na fase de testes de habilitação para garantir o conceito ACID de transação e as validações da integridade referencial.

Nesta estrutura, para máquinas que admitem paralelismos nas operações /MOH 90/ e em banco de dados distribuídos (/WON 83/, /YU 84/), o algoritmo de execução do código deve incorporar estes paralelismos de forma que todas as operações paralelas, demarcadas durante a tradução da especificação, devem agora ser aplicadas nas fases de testes de habilitação e de confirmação da transação, obedecendo os mecanismos de disparo dos grupos paralelos discutido na seção 5.4.2. Os mecanismos de disparo devem considerar os tipos de fluxos válidos em cada uma das fases.

A aplicação do processo de embutimento de consultas também tem reflexos sobre o algoritmo de execução do código. Neste algoritmo, na fase de testes de habilitação, consultas que foram agrupadas pela rotina de aninhamento de consultas são tratadas pelo algoritmo, como se fossem modeladas em um único fluxo.

O grafo de dependência, gerado durante a tradução da especificação, é também utilizado na execução do código. Além de indicar a seqüência da execução dos fluxos, nas fases de testes de habilitação e de confirmação da transação, assume um papel de instrumento auxiliar no estabelecimento do destino dos dados recuperados do banco de dados. O destino dos dados obtidos, durante a fase de testes de habilitação, pelas operações que representam os fluxos de teste (presença e ausência), depende do número de tuplas recuperadas e da sua posterior utilização, avaliada a partir do grafo de dependência. Assim, tem-se:

Uso do grafo na fase de testes de habilitação:

```
.se nodo representa fluxo de teste:
     -se o nodo não é seguido de nodo de  alteração  ou
      de saída:
         .dados recuperados não são mais necessários  (
           são descartáveis);
     -caso contrário:
         .se recuperada uma única tupla:
             -guarda dado na memória;
         .caso contrário (conjunto de tuplas):
             -guarda dados em tabela temporária;
```

Uso do grafo na fase de confirmação:

```
.se nodo representa fluxo de teste:
     -se o nodo não é seguido de nodo de  alteração  ou
      de saída:
         .nada é feito (operação sem efeito);
     -caso contrário:
         .dado está na memória (se 1 tupla) ou em tabe-
          la temporária (se conjunto)  ou, opcionalmen-
          te, deve ser ativada uma reexecução do fluxo.
```

Caso se opte pela reexecução do fluxo (executar novamente a consulta), existe aí uma decisão entre tempo de processamento e espaço de armazenamento. A princípio, uma reexecução do fluxo na fase de confirmação da transação deve ser evitada, pois pode onerar sensivelmente o tempo de execução, havendo uma duplicação de consultas quando executadas em ambas as fases.

Outro propósito do grafo de dependência é o de instrumentalizar uma estratégia de bloqueio das tabelas usadas na transação. Dependendo do tipo de operação representada pelo nodo,

procedimentos específicos de bloqueio devem ser considerados. Estes procedimentos são:

Uso do grafo na fase de testes de habilitação:

.para cada nodo do grafo que represente acesso ao BD:
  -se nodo é de teste e é seguido de algum nodo de atualização(inclusão, exclusão, alteração) no caminho do grafo:
    .bloqueia as tuplas da tabela referenciada no nodo corrente ou no grupo de nodos em caso de aninhamentos (SELECT __ FOR UPDATE OF __);
    .executa operação do nodo;
    .guarda dados em memória ou tabela intermediária (não necessita bloquear pois são dados locais à transação);
    .mantém bloqueio até fim da transação (bloqueio de 2 fases /DAT 88/);
  -caso contrário:
    .se nodo é de atualização:
      -bloqueia as tuplas da tabela referenciada no nodo corrente (SELECT __ FOR UPDATE __ OF);
      -executa operação do nodo;
      -mantém bloqueio até fim da transação (tabela será atualizada na fase de confirmação da transação);
    .caso contrário: (grafo sem nodo de atualização)
      -executa operação do nodo sem necessidade de bloqueio;

Uso do grafo na fase de confirmação:

.todas as tuplas das tabelas, usadas pelas operações de atualização, já foram bloqueadas na fase de testes de habilitação da transação;
.no final da transação(END-TRANSACTION), há uma liberação automática do bloqueio de todas as tabelas.

6.1 Manipulação de Conjuntos

A solução proposta para o processo de tradução e execução da especificação ER/T$^+$ focalizou transações que manipulam objetos individuais nos fluxos do diagrama. Sendo assim, cabe agora avaliar se esta solução se aplica também às transações que manipulam conjuntos nestes fluxos.

Como visto no capítulo 4, a semântica da linguagem ER/T$^+$ permite tanto a especificação de objetos individuais como o

tratamento de conjuntos associados a cada fluxo do diagrama (ver Figura 22). Isto significa que, na modelagem de conjuntos sobre os fluxos, a correspondente implementação desta operação deve garantir o acesso/recuperação de um conjunto de tuplas que satisfazem os critérios de seleção especificados e que, de alguma forma, objetos individuais deste conjunto sejam agrupados, combinados e/ou confrontados com conjuntos ou objetos individuais de outros fluxos.

A forma de tratar estes conjuntos em uma implementação pode considerar as seguintes opções:

Opção 1: Usar o algoritmo de tradução/execução proposto, realizando um tratamento exaustivo do conjunto dentro de cada operação (fluxo). Cada operação (consulta, inclusão, etc.) trata o conjunto completo antes de passar para a avaliação da operação seguinte. Eventuais instanciamentos de objetos individuais ("PARA CADA") são feitos a nível de cada fluxo individual, durante o seu processamento.

Opção 2: Instanciar objetos individuais dentro dos diferentes conjuntos, com uso de "cursores" de ocorrência, e realizar um casamento (matching) entre as operações dos fluxos. Um bloco de controle de execução da transação se encarrega de manipular todos os diferentes conjuntos nos fluxos, como se fossem "n transações de objetos individuais". O instanciamento de objetos individuais é feito por um laço tipo "PARA CADA", mantido a nível deste bloco de controle de execução. Existe, portanto, um tratamento tupla a tupla.

A opção 2 fica prejudicada em vista de não tirar proveito do potencial dos operadores da álgebra relacional (projeção,junção,etc.) para a manipulação de conjuntos. Esta opção pressupõe que as tuplas sejam manipuladas individualmente dentro dos conjuntos pelos diversos operadores da linguagem de implementação, sob o controle de um laço de iteração, explicitamente codificado (tratando individualmente as "n" tuplas selecionadas). Assim, não aproveita as facilidades de manipulação de conjuntos/subconjuntos colocadas a disposição pelos comandos da linguagem SQL. Esta opção também desconsidera a equivalência do nível da linguagem de implementação com a semântica da linguagem de especificação ER/T^+, que possui o tratamento completo de conjuntos associado a cada fluxo (ver Figura 22).

Existe uma equivalência entre o nível da linguagem de especificação e o nível da linguagem de implementação e esta característica deve ser evidenciada ao invés de se desprezá-la. Em vista disso, a opção 2 não está sendo aqui considerada.

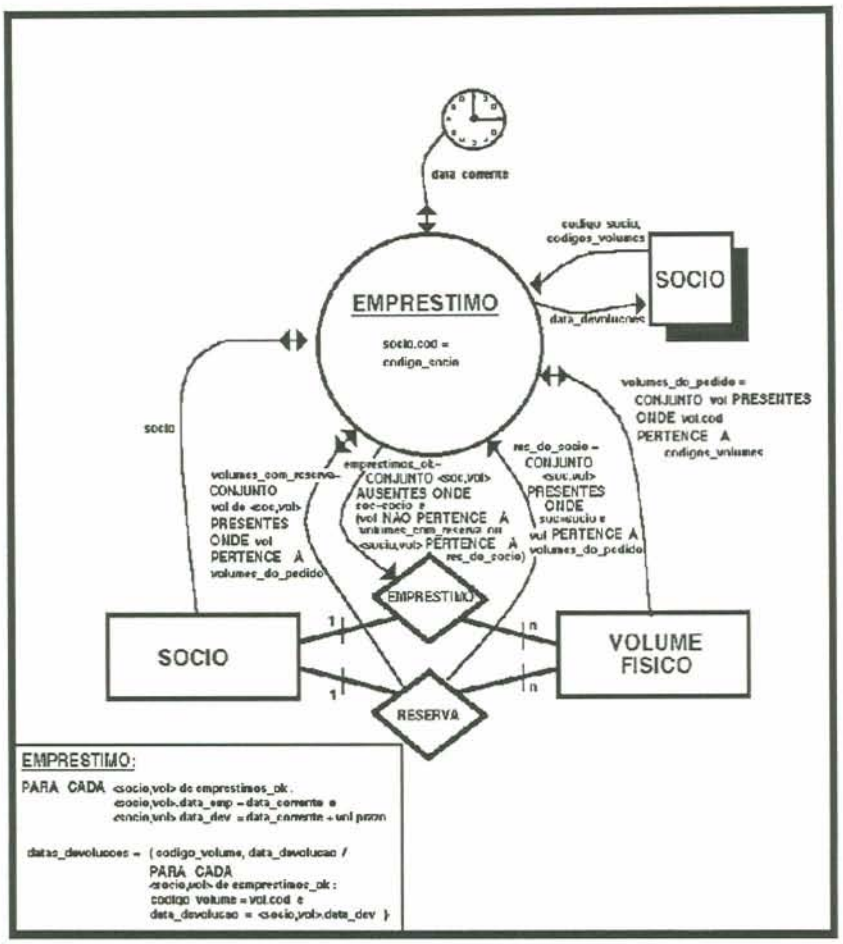


Fig. 22 - Exemplo de manipulação de conjuntos completos nos fluxos

Através da análise superficial da opção 1, é possível afirmar que ela pode ser considerada no escopo da solução de tradução/execução anteriormente apresentada, pois se aplica tanto para a manipulação de objetos individuais quanto para a manipulação de conjuntos.

Existem, entretanto, situações peculiares nas transações com conjuntos que merecem ser mais atentamente

avaliadas. Para transações que manipulam conjuntos nos fluxos, eventualmente podem ocorrer situações particulares, como a ilustrada na Figura 23, onde:

.Para cada objeto individual de uma entidade A:
-processar (consultar, incluir, alterar, excluir) um conjunto de objetos da entidade B;
-processar um conjunto de objetos de C;

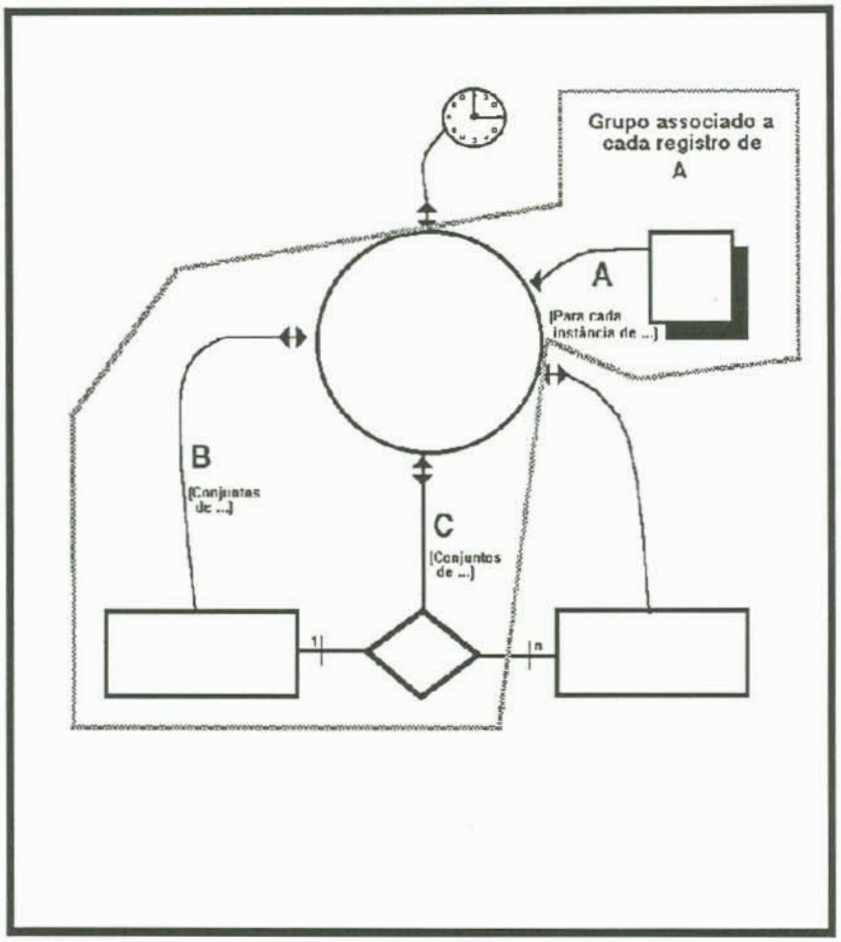


Fig. 23 - Manipulação de conjuntos "1 para n" em Transações

Assim, para este caso, uma implementação deve considerar que exista uma associação tipo "1 para n" entre conjuntos de tuplas de diferentes relações onde, forçosamente, deve haver uma individualização (instanciamento) de tuplas de uma das relações participantes da transação. Para cada tupla

selecionada nesta relação, há necessidade de processar um conjunto de tuplas em outra relação.

Nesta situação, os conjuntos não podem ser exaustivamente manipulados, em cada fluxo isoladamente, pela simples aplicação dos operadores da álgebra relacional.

Uma possível solução, para esta situação, pode considerar os seguintes procedimentos:

- Identificar, no grafo de dependência, nodos em que a relação de precedência apresente grau de dependência "1 para n";
- Subdividir o diagrama em grupos (subconjuntos) de fluxos, que sejam dependentes da individualização de objetos de A, onde cada tupla de A passa a assumir o papel de registro de controle (a exemplo de fluxos de controle) para ativar as fases de testes de habilitação e de confirmação da transação para o grupo de fluxos vinculados. Em outras palavras:
  - se existe tupla A, então:
    - executa testes de habilitação e confirmação da transação considerando os fluxos dependentes de A.
- A transação se completa quando todos as tuplas de A foram consumidas e, eventualmente, outros subgrupos de fluxos tenham sido processados.

Esta solução pode vir a ser insatisfatória se constatados diferentes niveis de hierarquia de conjuntos "1 para n". Isto pode ser observado, por exemplo, quando cada instância de um fluxo A depende de um conjunto de B e cada instância de B depende de um conjunto de D e assim, sucessivamente.

Uma alternativa, para este caso, considera os seguintes procedimentos:

.Estabelecer uma hierarquização no grafo de dependência, dividindo-o em grafos parciais, cada qual correspondendo ao grupo de fluxos (nodos) que modelam a manipulação dos conjuntos relacionados com o fluxo cuja instância individual se vincula a eles. Isto pode levar a uma cadeia hierarquica de dependência "1 para n";
.Nas fases de testes de habilitação e de confirmação da transação do algoritmo de execução do código da especificação, para cada instância do nodo dependente, executar o subgrafo vinculado. Por causa da cadeia hierárquica de dependência, o algoritmo pode ser executado recursivamente, ou seja, os subgrupos podem ser processados de forma recursiva.

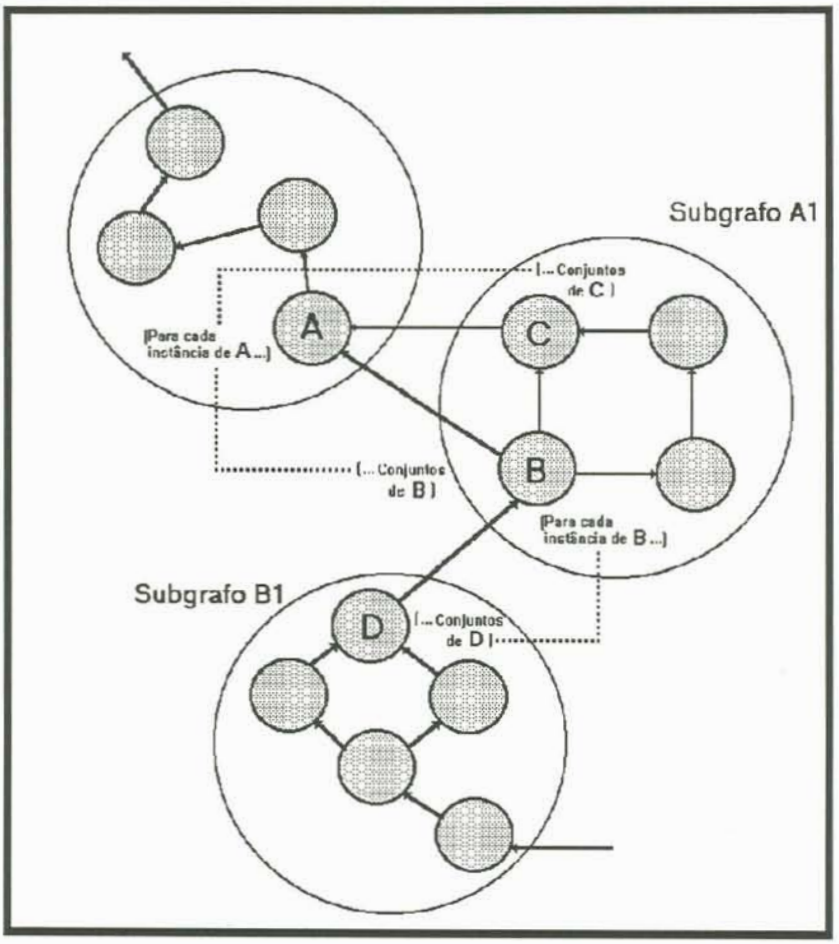


Fig. 24 - Hierarquização do Grafo de Dependência p/ tratamento de transações com conjuntos "1 para n"

A hierarquização do grafo de dependência, para o tratamento com conjuntos "1 para n", pode ser vista na Figura 24 e, em linhas gerais, a estrutura de procedimentos associada está definida no diagrama de ação ilustrado na Figura 25. Este diagrama de ação mostra um esqueleto de procedimentos necessários para adaptar o processo original de tradução/execução da especificação.

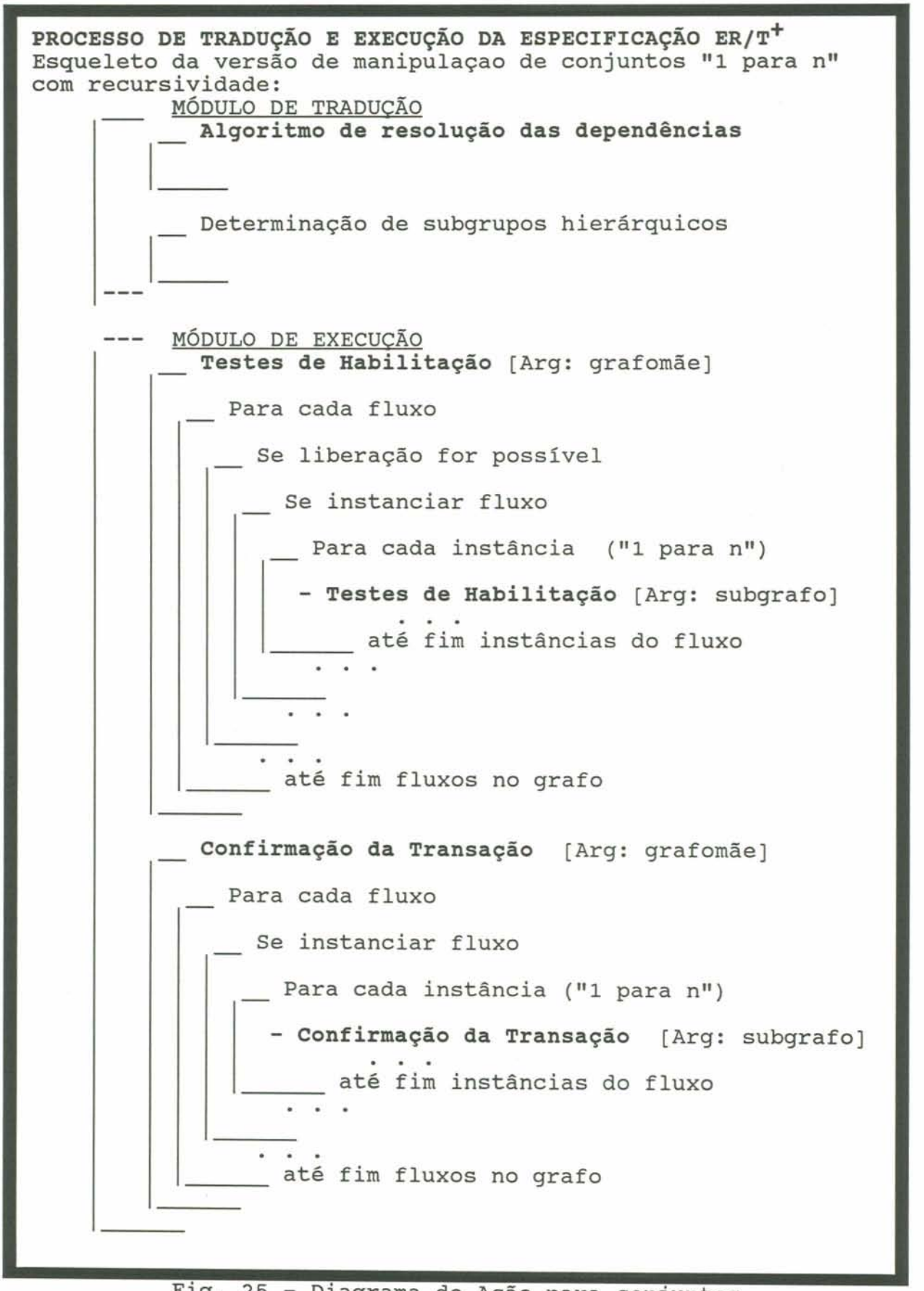


Fig. 25 - Diagrama de Ação para conjuntos "1 para n" em transações c/ recursividade

Neste diagrama de ação cabe frisar que:

1- A determinação de subgrupos hierárquicos assinala, no grafo de dependências, a cadeia hierárquica de dependência "1 para n" para a manipulação de conjuntos.

2- As fases de testes de habilitação e confirmação da transação são executadas recursivamente para cada subgrafo vinculado, sendo que na primeira chamada é indicado, como "grafomãe", o grafo de dependência completo. Para cada fluxo processado, em ambas as fases, obedecendo os tipos de fluxos consideráveis em cada fase (ver algoritmo original de tradução/execução), é avaliado se o referido fluxo se enquadra na situação de dependência "1 para n" e, neste caso, é feita uma chamada recursiva indicando, como argumento, o "subgrafo" vinculado.

3- As fases de testes de habilitação e confirmação da transação, bem como o algoritmo de resolução das dependências, assinalados em negrito, são os procedimentos constantes no algoritmo original.

6.2 Mecanismos de Tratamento de Exceções

Na apresentação das características do modelo ER/T$^+$ (capítulo 4), foi salientado que o projetista do sistema, durante o processo de modelagem, precisa somente explicitar no diagrama o comportamento normal do sistema. Pode deixar que os procedimentos padrões de tratamento de exceções e as respectivas mensagens de erro, para situações de inconsistências, sejam automaticamente derivados. No entanto, pode realizar também uma especificação explícita de algum eventual tratamento de exceções mais complexo associado a determinado fluxo.

Sendo assim, os mecanismos de tratamento de exceções devem acomodar dois tipos de procedimentos para a manipulação das situações de inconsistências:

- .mensagens/procedimentos indicados pelo projetista na modelagem;
- .mensagens/procedimentos incorporados automaticamente pelo algoritmo de tradução e geração de código.

Para o caso das mensagens e procedimentos indicados pelo projetista na modelagem (veja Figura 26), as mensagens (ou procedimentos) associadas a determinado fluxo substituem o tratamento padrão de situação de erro aplicado pelo algoritmo de

tradução e geração de código, quando da implementação deste fluxo. Se uma mensagem for indicada pelo projetista, haverá uma simples troca da mensagem padrão pelo texto indicado no diagrama.

Caso um procedimento (ação de violação) /TUC 82/ mais complexo for indicado, este deve ser traduzido em correspondentes comandos da linguagem hospedeira (C ou SQL) e associados ao tratamento de exceções do respectivo fluxo. Neste último caso, a eventual desabilitação do fluxo deve ser indicada, através dos procedimentos especificados pelo projetista, ao invés de incondicionalmente ser incorporada pelo tratamento de exceções gerado pelo algoritmo.

Uma ação de violação associada ao fluxo pode especificar:

- que a operação representada no fluxo seja rejeitada, o fluxo seja desabilitado e uma mensagem de erro seja emitida;
- que um valor particular seja assumido como resultado da operação;
- que um procedimento alternativo seja ativado para busca de um novo valor.

No exemplo da Figura 26, o tratamento das inconsistências para o fluxo "volume" inclui a execução da rotina pre-definida "pesquisa-em-base-remota" que providencia a busca do "volume" em uma outra base de dados. Neste caso, os procedimentos de tratamento de exceção são indicados durante a modelagem e a cláusula "EM ERRO FAÇA", além de especificar uma mensagem de erro, informa ao algoritmo de tradução e geração de código, que o correspondente fluxo deve ser desabilitado numa situação de erro.

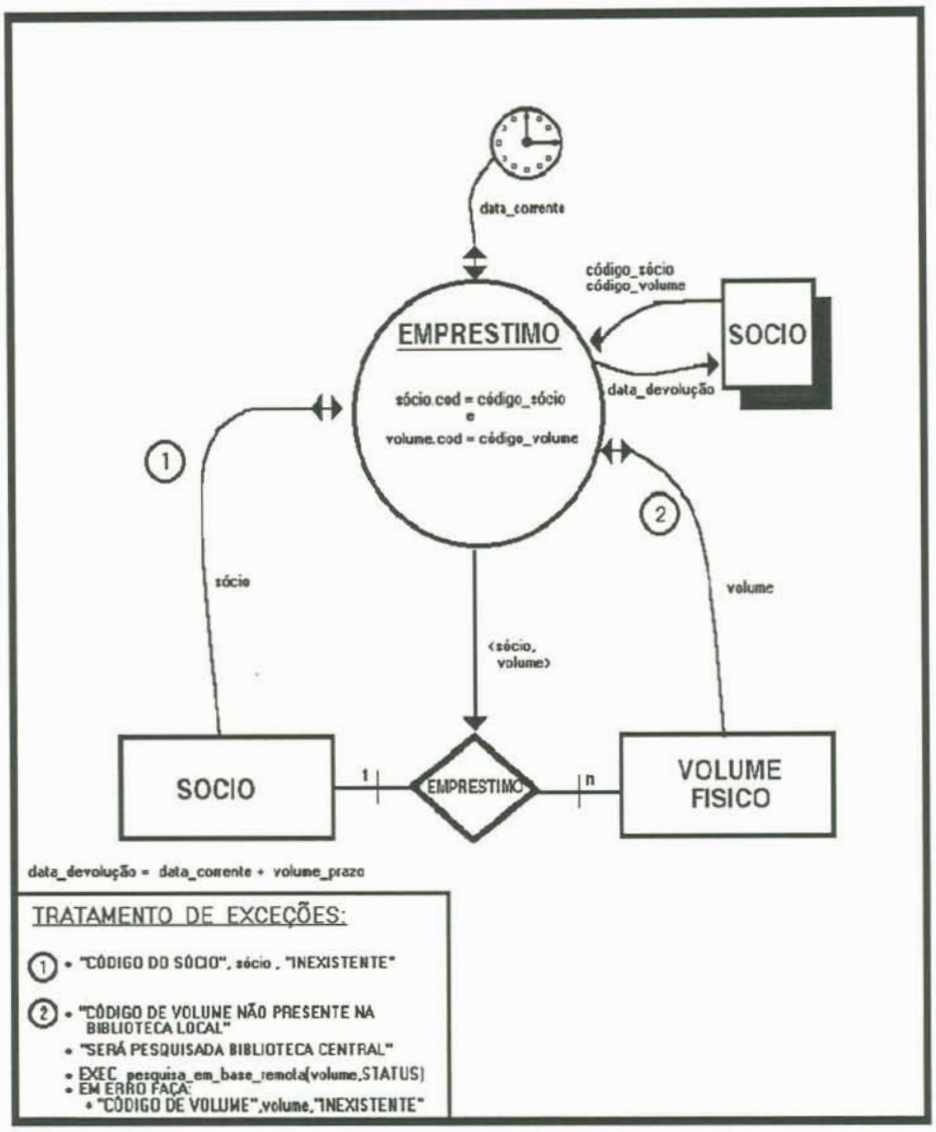


Fig. 26 - Inconsistências indicadas na modelagem

Para o caso das mensagens/procedimentos incorporados automaticamente pelo algoritmo de tradução e geração de código, o tratamento de exceção está implicitamente associado às situações de erro constatadas nas diferentes validações (restrições de formato e de domínio dos dados, restrições de existência, etc.) geradas pelo algoritmo.

O conjunto de validações, com correspondentes mensagens de erro previstas pelo algoritmo, estão enumeradas no quadro a seguir:

Tipo de Validação	Consistências Realizadas	Mensagens de Erro
Restrição de Formato	- verificação de tipo de dado (numérico,alfanum.,data,...); - verificação de máscara de edição(por exemplo, dd/mm/aa para atributos tipo data)	- Tipo Inválido: atributo ??? não ??? (num./alfanum./etc) - Tipo Inválido: atributo ??? fora da máscara de edição ???
Restrição de Domínio	- verificação do preenchimento do atributo em confronto com sua obrigatoriedade: 1-obrigatório sempre 2-obrigatório em inclusões 3-opcional(valores nulos são permitidos e designação de valor por ausência está associado)	- Erro de Preenchimento: atributo ??? de preenchimento ???
	- verificação do valor do atributo em confronto com limites superior e inferior (restrição de intervalo)	- Valor do atributo ??? fora dos limites ??? a ???
	- verificação do valor do atributo em confronto com lista de valores válidos (restrição de enumeração)	- Valor do atributo ??? com enumeração inválida
Restrição de Existência	- verificação da cardinalidade do relacionamento ER	- Extrapolação de cardinalidade :menor que limite inferior - Extrapolação de cardinalidade :maior que limite superior
	- verificação de existência de chave: a)teste de ausência (operação de inclusão)	- Chave ??? já cadastrada em ??? (entidade/relacionamento)
	b)teste de presença (operação de exclusão e alteração)	- Chave ??? não localizada em ??? (entidade/relacionamento)
Restrição Referencial (dependência funcional inter-entidades)	- verificação da dependência entre atributos de entidades (restrição de relacionamento)	- Operação de ??? (alt./incl./excl.) inválida por violação de correspondência de atributo estrangeiro ???
Controle de Execução	- verificação das dependências entre fluxos	- Indisponibilidade de dados na seqüencialização dos fluxos
	- teste de habilitação de fluxo específico	- Fluxo ??? não habilitado
	- teste de habilitação da transação (a partir do conjunto de fluxos)	- Transação não habilitada por motivos assinalados
	- verificação de tempo de espera nos fluxos de controle	- Extrapolação de tempo de espera

Caso seja constatada alguma exceção, cada mensagem gerada (incluída aquela informada pelo projetista) é colocada em uma lista a ser apresentada, ao usuário, no final da fase de testes de habilitação da transação.

No processo de tradução da especificação, a estratégia de realizar um embutimento de consultas, gerando expressões de subconsultas aninhadas, pode ser conflitante com o tratamento de exceções. Em fluxos individualizados, existe um tratamento de exceções implícito (ou especificado pelo projetista) associado a cada fluxo específico. No entanto, quando os fluxos são combinados para formar expressões de subconsultas aninhadas, o tratamento de exceções de cada fluxo deve ser, de alguma forma, considerado no contexto do aninhamento de consultas.

A Figura 27 ilustra esta situação.

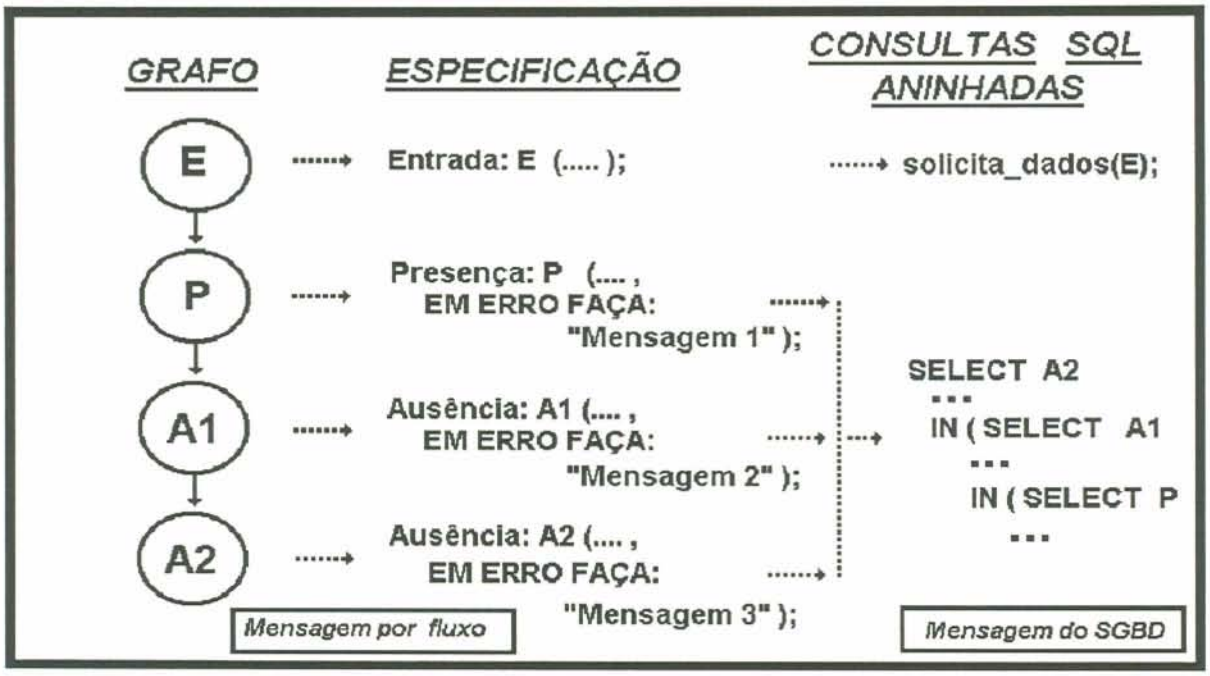


Fig. 27 - Mensagens da especificação x mensagens do SGBD

Uma alternativa pode considerar que mensagens relativas aos diversos fluxos alvos de correspondente embutimento também

devam ser combinadas (uma mensagem única por grupo de subconsultas, por exemplo). Nesta situação, o resultado do tratamento de exceções pode não refletir exatamente a forma como os erros associados às consultas foram modelados. A geração de mensagem única por grupo de subconsultas seria um ônus oriundo da decisão de se realizar aninhamento de consultas.

Entretanto, cabe frisar que um aninhamento de subconsultas pode envolver o acesso a diversas entidades/relacionamentos do banco de dados. Eventualmente, durante a recuperação/atualização de dados, uma das subconsultas de acesso pode produzir uma exceção que afeta a execução de toda a expressão de consulta. É obvio que, para um mecanismo de tratamento de exceções mais aprimorado, é interessante identificar exatamente qual a(s) causa(s) da exceção, qual a(s) entidade(s) envolvida(s) e que trecho da subconsulta produziu o erro.

Para contornar o problema de atribuir mensagem única por grupo de subconsultas, torna-se necessário construir uma rotina que, a partir do reconhecimento das mensagens emitidas pelo SGBD, faça uma individualização de mensagens referente a cada fluxo para identificar as que se relacionam a cada subconsulta (analisar qual o tipo de erro, qual a entidade/relacionamento que está envolvido, etc.).

A rotina deve executar, basicamente, os seguintes procedimentos:

- receber do SGBD os códigos ou mensagens de erros produzidos pelo acesso frustrado ao banco de dados;
- identificar o(s) tipo(s) de erro(s) (categoria, subcategoria);
- identificar a entidade/relacionamento onde foi constatada a exceção;
- analisar a expressão de subconsultas aninhadas para verificar qual(ais) a(s) que produziu(iram) o acesso que originou a referida exceção;
- verificar a(s) mensagem(ens) padrão(ões) ou explicitada(s) pelo projetista associada(s) a esta(s) subconsulta(s) e vinculá-la(s) a exceção;
- realizar uma eventual composição de mensagens através do operador lógico E.

Portanto, associado a cada comando SQL de acesso ao banco de dados, deverá haver uma manipulação explícita de erros como segue:

```
EXEC SQL  <comando de acesso ao BD>
if (sqlca.sqlcode == 00)
   /*  Sucesso na operação          */
else
   /*  Tratamento do erro           */
```

Assim, no tratamento do erro, pode haver um mecanismo para interpretar as mensagens do SGBD, personalizando-as nos padrões do sistema e incluindo-as na lista de erros a ser exibida ao usuário.

Convém também avaliar como o tratamento de exceções se relaciona com os mecanismos de gerenciamento de transações e controle de integridade disponíveis no SGBD. Para este contexto, revendo o processo de tradução e execução da especificação ER/T$^+$, pode ser observado que existe um tratamento de exceções ligado tanto ao módulo de tradução quanto ao módulo de execução.

No módulo de tradução, eventuais inconsistências constatadas, provavelmente são produzidas a partir de erros de modelagem (erro na dependência entre os fluxos) e, portanto, o tratamento de exceções deve remeter o projetista para uma remodelagem do sistema.

Já no módulo de execução, pelas funções enumeradas em cada fase, constata-se que o algoritmo visa, antes de confirmar a efetiva atualização do banco de dados, garantir que todas as consistências e validações tenham sido realizadas durante os testes de habilitação da transação. Sendo assim, o tratamento de exceções gerado pelo algoritmo está fundamentado neste modelo. Já para o tratamento de exceções associado ao SGBD, que requer a demarcação de pontos de reinício e reposicionamento, a especificação de uma transação, como uma operação atômica ("Begin-end Transaction"), deve envolver tanto a fase de testes de habilitação quanto a de confirmação da transação. Com isto, a

manutenção da integridade do banco de dados dentro de uma transação é de responsabilidade do algoritmo que implementa o módulo de execução ER/T$^+$ e da integridade entre transações (para uma eventual reconstrução do banco de dados) é de responsabilidade do SGBD.

6.3 Conclusão

Neste capítulo foi descrita a arquitetura do processo de tradução e execução da especificação construída a partir dos conceitos e idéias apresentadas no capítulo 5. Para esta arquitetura, foi proposto um algoritmo de tradução da especificação e geração de código e também um algoritmo que mostra os procedimentos de execução do código gerado. Estes algoritmos são qualificados também para máquinas com paralelismos nas operações e para banco de dados distribuídos.

Para garantir paralelismos na execução de múltiplas transações, foi defina uma estratégia de bloqueio das tabelas usadas em uma transação. Esta estratégia se baseia no vínculo de precedência entre operações da transação e, para instrumentalizá-la, usou-se novamente o grafo de dependência gerado durante a tradução da especificação. Este mesmo grafo teve utilidade também nos procedimentos de manipulação de conjuntos, onde serviu para estabelecer uma cadeia hierárquica de dependência.

Assim, nas soluções aqui propostas, o grafo de dependência foi usado na determinação de:

.seqüencialização das operações;
.paralelismos na execução das operações;
.aninhamento de consultas;
.destino dos dados recuperados nas consultas;
.bloqueio de tabelas;
.cadeia hierárquica de dependência (subgrafos) para a manipulação de conjuntos.

Por fim, neste capítulo, foram propostos mecanismos de tratamento de exceções incorporados ao modelo de implementação de transações que tiveram, como atribuição principal, a resolução dos conflitos entre mensagens/procedimentos de exceção

especificados durante a modelagem conceitual do sistema e as geradas pelo SGBD durante a execução da transação.

7. CONCLUSÕES E FUTUROS TRABALHOS

Nesta tese foi proposto um modelo de especificação de transações em banco de dados e, também, um modelo de implementação dessas especificações.

Durante as explanações feitas no texto, foram identificadas como características desejáveis de especificações de transações em BD a não proceduralidade, a descrição tipo causa/efeito, a homogeneidade com a modelagem de dados, a representação de objetos individuais interagindo com conjuntos de objetos e a descrição das ações de tratamento de exceção.

Considerando estes aspectos como peculiaridades para a linguagem de especificação, foi desenvolvido um modelo de especificação, com modalidade gráfica e textual, que combina idéias da especificação de transações encontradas em linguagens como TAXIS /BOR 84/ com algumas idéias propostas anteriormente na abordagem ER/T /PER 90/, que utiliza as técnicas de modelagem de dados (diagramas ER) integradas com as técnicas de modelagem de funções (Diagramas de Fluxo de Dados), num formalismo baseado na teoria das redes de Petri.

Defendendo, também, a idéia de um ambiente de programação automática, foi proposto um modelo de implementação das especificações onde são geradas, mediante um mecanismo de proceduralização de especificações, procedimentos de uma linguagem de programação (C com embutimento de SQL), a partir das especificações declarativas.

O trabalho foi motivado a partir da intenção de se construir um modelo completo de especificações de transações em banco de dados que tem o objetivo de evitar sucessivas traduções manuais de especificações, por parte de analistas e programadores, até obter código executável.

Assim, durante as pesquisas, foi desenvolvida uma linguagem de especificação de transações baseada em pré e pós condições, onde o comportamento do sistema é descrito através de

transações especificadas de forma declarativa e integrado com a modelagem de dados. É uma linguagem que originou-se a partir da compilação de técnicas já bastante sedimentadas e utilizadas na prática (ER e DFD), adaptando-as para uma semântica baseada nos mecanismos das redes de Petri.

A partir do formalismo dessa linguagem, foram desenvolvidos os mecanismos de proceduralização das especificações declarativas, onde um algoritmo de seqüencialização das operações se baseou em um grafo de dependência de fluxos de dados para o estabelecimento do vínculo de precedência entre procedimentos. Foram definidas as regras de mapeamento das funções individuais especificadas na descrição do sistema e um algoritmo que, a partir destas regras e da seqüencialização das operações, é responsável pela geração de código executável para a especificação.

No processo de tradução e execução da especificação foram considerados mecanismos de avaliação de restrições de integridade e mecanismos automáticos de dedução de procedimentos implícitos na forma de realizar a especificação (ex.: teste de ausência de tuplas antecedendo a efetivação de operações de inclusão). Na proposta aqui apresentada, a execução da especificação está baseada na semântica de execução das redes de Petri com uma fase de testes de habilitação, onde são tratadas operações de entrada, de testes de presença, de testes de ausência e de atualizações prévias para garantir a validação da integridade referencial, e uma fase de confirmação, onde são consideradas as operações de efetiva atualização do banco de dados e as de emissão dos resultados.

Aspectos de aninhamento de consultas das operações de acesso ao banco de dados, paralelismos de execução das operações e otimização das consultas SQL, também foram analisadas na tese.

Os mecanismos de tratamento de exceção na execução das especificações mereceram uma atenção especial no trabalho. Foram aí salientados, os aspectos de conflitos entre mensagens indicadas pelo projetista na modelagem do sistema e exceções

provocadas quando da execução das expressões de subconsultas construídas pelo algoritmo de tradução e geração de código, durante o processo de embutimento de consultas, e incorporadas automaticamente no fonte executável. Uma mensagem de erro, por exemplo, indicada durante a modelagem, eventualmente pode não refletir exatamente todas as exceções produzidas durante a execução da especificação. Deve haver uma compatibilização entre as exceções geradas pelo SGBD (durante a implementação) e as exceções indicadas na modelagem (durante a especificacão).

No atual estágio, os trabalhos desenvolvidos e previstos como resultado da presente tese apresentam como produtos:

a) Definição do modelo de especificação, detalhando os aspectos sintáticos e semânticos da linguagem de especificação proposta.

b) Definição do modelo de implementação, descrevendo os principais algoritmos necessários para seqüencializar as operações, para detectar e executar paralelismos, para detectar aninhamento de consultas, para compatibilizar paralelismos com embutimentos e para gerenciar o processo de tradução e execução das especificações. Adicionalmente, estão definidas as regras de mapeamento do modelo de dados para correspondentes estruturas de dados de um banco de dados relacional e as regras de mapeamento do modelo de funções para equivalentes construções na linguagem de programação. Estão caracterizados, também, os procedimentos de otimização de consultas de banco de dados resultantes da tradução da especificação e os procedimentos de tratamento de exceções.

As implementações, durante os trabalhos da tese, se restringiram a testar algoritmos propostos e serviram como um instrumento de experimentação e validação das idéias apresentadas e não como resultados práticos oriundos da pesquisa.

Como contribuição central, a tese apresenta um modelo completo de especificação e implementação voltado para sistemas transacionais em banco de dados, enfatizando, principalmente, a proposição de uma técnica de proceduralização de especificações declarativas e apresentando um mecanismo de execução de especificações, baseado na semântica de redes de Petri de alto

nível. Estudos correlatos e complementares /AHL 90a/ /AHL 91a/ /AHL 91b/, que fundamentaram a presente tese, também tiveram sua parcela de contribuição em termos de aprofundamento dos estudos sobre programação automática e ferramentas para construção automática de sistemas.

Dentro das perspectivas para futuros trabalhos, espera-se que a presente tese forneça subsídios para a construção de uma ferramenta CASE para modelagem conceitual do sistema, baseado no modelo de especificação proposto. Uma alternativa, para esta ferramenta, pode estar em um editor gráfico inteligente que, além da análise sintática das especificações e de orientar o projetista em determinadas fases da modelagem (condutor de modelagem: como modelar! o que está faltando!), pode gerar automaticamente algumas especificações (desenho de componentes dinâmicos do sistema) como, por exemplo, os procedimentos para as operações básicas de atualização.

A partir do modelo de implementação aqui definido, pode ser construída uma ferramenta que implemente os algoritmos de tradução/execução da especificação propostos.

Em linhas gerais, as ações para os mecanismos de otimização de consultas foram propostas neste trabalho. Convém, no entanto, realizar estudos adicionais para aprofundar o tema quando da implementação dos algoritmos do processo de tradução/execução da especificação. Estes estudos podem analisar os aspectos de detecção dos caminhos mais eficientes na busca dos dados baseado, por exemplo, na população de tuplas das entidades alvo das subconsultas visando o estabelecimento de uma seqüência ótima de consulta, na classificação prévia de tuplas antes de iniciar uma subconsulta, no uso de índices auxiliares, etc.

Melhorias na linguagem de especificação ER/T^+ podem ser propostas em futuros trabalhos. Atualmente, a sintaxe ER/T^+ pressupõe a existência de uma declaração "Entrada" na especificação de uma transação, tornando obrigatório o fornecimento de parâmetros de entrada para a transação. Esta sintaxe pode ser modificada para aceitar transações sem entrada a

fim de permitir uma aplicação do modelo ER/T^+ em bancos de dados ativos. Neste caso, mecanismos de "gatilhos" também devem ser incorporados no modelo para permitir que transações sem parâmetros sejam ativados, sempre que determinado evento ocorrer (por exemplo, atualização de informações estatísticas sobre o BD).

Considerando que em torno de 3% (três por cento) de transações em um banco de dados efetivamente necessitam realizar um "Rollback" das atualizações (/HAE 83/,/GRA 81/), pode ser avaliada também uma versão alternativa para o modelo proposto na tese, como mostra o esquema abaixo:

FLUXOS	FASES	
	HABILITAÇÃO	CONFIRMAÇÃO
Presença	SELECT (LOCK)	---
Ausência	Se ausente INSERT	DELETE
Inclusão	INSERT	---
Exclusão	DELETE	---
Alteração	UPDATE	---

Este modelo é baseado em "Rollbacks" de transações garantidos por "Aborts" no final da fase de testes de habilitação e provocados em conseqüência de exceções constatadas durante a fase. O modelo alternativo pode ser confrontado com o proposto nesta tese a fim de avaliar desempenho e tempo de resposta na execução da transação.

Por fim, sugere-se um estudo comparativo no confronto entre os paralelismos detectados e executados pelos algoritmos aqui propostos e o nível de embutimento de consultas, onde paralelismos para as expressões de subconsultas ficam a cargo do SGBD. Um estudo que, por exemplo, determine qual o ponto ideal na definição da granularidade do embutimento para obter um maior ganho de desempenho no paralelismo das operações.

BIBLIOGRAFIA

/AHL 90a/ AHLERT, H. **Estudo de metodologias e ferramentas que envolvem suporte à construção automática de sistemas**. Porto Alegre: CPGCC da UFRGS, 1990. (TI, 188).

/AHL 91a/ AHLERT, H. **Estudo comparativo e taxonomia de ferramentas de suporte à construção automática de sistemas**. Porto Alegre: CPGCC da UFRGS, 1991. (RP, 153).

/AHL 91b/ AHLERT, H. **Sistemas especialistas para a engenharia de software**. Porto Alegre: CPGCC da UFRGS, 1991. (RP, 154).

/AHL 92/ AHLERT, H. **Rumo a um modelo de implementação de especificações não procedurais de sistemas transacionais em Banco de Dados**. Porto Alegre: CPGCC da UFRGS, 1992. (RP, 190).

/ALE 89/ ALENCAR, P.S.C.; LUCENA, C.J.P. Métodos formais para o desenvolvimento de programas. In: ESCOLA BRASILEIRO-ARGENTINA DE INFORMÁTICA, 4., 1989, Termas de Rio Hondo, Argentina. **Anais...** Buenos Aires: Kapelusz, 1989.

/ANT 81/ ANTONELLIS, V. de; ZONTA, B. Modeling events in Data Base aplications design. In: VERY LARGE DATA BASE, 7., 1981, Cannes, France. **Proceedings...** New York: IEEE, 1981. p.23-31.

/BAK 90/ BAKO, B.; VALETTE, R.; CARDOSO, J. Implementação de um sistema de regras de produção controlado para a aplicação na supervisão de sistemas flexíveis de manufatura. In: CONGRESSO NACIONAL DE AUTOMAÇÃO INDUSTRIAL, 4., 1990, São Paulo. **Anais...** São Paulo, 1990. p.164-173.

/BAL 81/ BALZER, R. Transformational implementation: An example. **IEEE Transactions on Software Engineering**, New York, v.SE-7, p.3-14, Jan. 1981.

/BAL 82/ BALZER, R.; GOLDMAN, N.; WILE, D. Operational specification as the basis for rapid prototyping. **ACM Sigsoft Software Engineering Notes**, New York, v.7, n.5, p.3-16, Dec. 1982.

/BAL 83a/ BALZER, R.; CHEATHAM Jr., T.E.; GREEN, C. Software technology in the 1990's using a new paradigm. **Computer**, Los Alamitos, CA., v.16, n.11, p.39-45, Nov. 1983.

/BAL 85a/ BALZER, R. A 15 years perspective in automatic programming. **IEEE Transactions on Software Engineering**, New York, v.11, n.11, p.1257-1267, 1985.

/BAL 91/ BALDASSARI, M.; BRUNO, G.; CASTELLA, A. "PROTOB": an object-oriented CASE tool for modelling and prototyping distributed systems. **Software-Practice and Experience**, New York, v.21, n.8, p.823-844, Aug. 1991.

/BAR 77/ BARSTOW, D. A knowledge-based system for automatic program construction. In: INTERNATIONAL JOIN CONF. ARTIFICIAL INTELLIGENCE, 5., 1977, Cambridge, MA. **Proceedings...** [S.l.:s.n.], 1977. p.382-388.

/BAU 76/ BAUER, F.L. Programming as an evolutionary process. In: BAUER, F.L.; SAMELSON, K. (Eds.). **Languages Hierarchies and Interfaces**. Berlin: Springer-verlag, 1976. p.153-182.

/BAU 89/ BAUER, F.L. et al. Formal program construction by transformations - Computer-aided, intuition-guided programming. **IEEE Transactions on Software Engineering**, New York, v.15, n.2, p.165-180, Feb. 1989.

/BEL 86/ BELLIA, M.; LEVI, G. The relation between logic and functional languages: a survey. **The Journal of Logic programming**, v.3, p.217-236, 1986.

/BOK 89/ BOKLIS, V. **Tratamento de Restrições de Integridade em Ambientes de Banco de Dados para Aplicações não Convencionais**. Porto Alegre: CPGCC da UFRGS, 1989. (TI, 129).

/BOR 84/ BORGIDA, A.; MYLOPOULOS, J.; WONG, H.K.T. Generalization/specialization as a basis for software specification. In: BRODIE, M.L.; MYLOPOULOS J.; SCHMIDT, J.W. (Eds.). **On Conceptual Modelling. Perspectives from Artificial Intelligence, Database, and Programming Language**. New York: Springer-Verlag, 1984. p.87-114.

/BRO 81/ BRODIE, M.L. On modelling behavioural semantics of databases. In: VLDB, 7., 1981, Cannes, France. **Proceedings...** New York: IEEE, 1981. p.32-42.

/BRO 84/ BRODIE, M.L.; RIDJANONIC, D. On the Design and Specification of Database Transactions. In: BRODIE, M.L.; MYLOPOULOS J.; SCHMIDT, J.W. (Eds.). **On Conceptual Modelling, Perspectives from Art. Intelligence, Database and Programing Languages**. New York: Springer-Verlag, 1984. p.277-32.

/BRU 86/ BRUNO, G.; MARCHETTO, G. Process translatable Petri nets for the rapid prototyping of process control systems. **IEEE Transactions on Software Engineering**, New York, v.SE-12, n.2, p.346-357, 1986.

/CAM 83/ CAMPOS, F.T. de; SANTOS, C.S. Especificação de Restrições de Integridade em Banco de Dados. In: CONGRESSO NACIONAL DE INFORMÁTICA, 16., 1983, São Paulo. **Anais...** São Paulo: SUCESU, 1983. p.78-81.

/CAN 90/ CANTÚ, E. **Uma abordagem para a representação, simulação e implementação de sistemas baseada na Rede de Petri a Objetos**. Florianópolis: UFSC, 1990. (Dissertação de Mestrado).

/CER 83/ CERI, S. (Ed.). **Methodology and tools for database design**. Amsterdam: North-Holland, 1983. 255p.

/CHA 91/ CHAO, C.M.; KUNG, C. Rapid prototyping of conceptual database design on a relational database management system. In: INTERNATIONAL CONFERENCE ON THE ENTITY RELATIONSHIP APPROACH, 10., 1991. **Proceedings...** [S.l.:s.n.], 1991.

/CHE 72/ CHEATHAM, T.E.; WEGBREIT, B. A laboratory for the study of automatic programming. In: AFIPS SPRING JOIN COMP. CONFERENCE, 1972, Atlantic City, N.J. **Proceedings...** Reston, VA.: AFIPS Press, v.40, p.11-21.

/CHE 76/ CHEN, P.P. The Entity-Relationship Model - toward a unified view of data. **ACM Transactions on Database Systems**, New York, v.1, n.1, p.9-36, 1976.

/CHE 81/ CHEATHAM, T.E.; HOLLOWAY, G.H.; TOWNLEY, J.A. Program refinement by transformation. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 5., 1981, San Diego, California. **Proceedings...** New York: IEEE, 1981. p.430-437.

/CHE 84/ CHEATHAM, T.E. Reusability through program transformation. **IEEE Transactions on Software Engineering**, New York, v.SE-10, n.5, p.589-594, Sept. 1984.

/COH 82/ COHEN, D.; SWARTOUT, W.; BALZER, R. Using symbolic execution to characterize behavior. **ACM Sigsoft Software Engineering Notes**, New York, p.25-32, Dec. 1982.

/CRI 88/ CRICHLOW, J.M. **An introduction to distributed and parallel computing**. Englewood Cliffs: Prentice-Hall, 1988.

/DAT 86a/ DATE, C.J. **Introdução a Sistemas de Banco de Dados**. 4.ed. Rio de Janeiro: Campus, 1986.

/DAT 86b/ DATE, C.J. **Relational Database:** Selected Writings. Reading: Addison-Wesley, 1986. 496p.

/DAT 88/ DATE, C.J. **Banco de Dados:** tópicos avançados. Rio de Janeiro: Campus, 1988.

/DAT 89/ DATE, C.J. **A guide to the SQL Standard**. 2.ed. Reading: Addison-Wesley, 1989.

/DAR 79/ DARLINGTON, J. Program transformation: an introduction and survey. **Comput. Bull.**, p.22-24, Dec. 1979.

/DAY 88/ DAYAL, U.; BUCHMANN, A.P.; MCCARTHY, D.R. Rules are objects too: a knowledge model for an active object-oriented database system. In: DITTRICH, K.R. **Advances in object-oriented database systems**. New York: Springer-Verlag, 1988. p.129-143. (Lecture Notes in Computer Science, v.334).

/DEG 86/ DEGROOT, D.; LINDSTROM, G. **Logic programming:** functions, relations and equations. Englewood Cliffs: Prentice-Hall, 1986.

/DEM 79/ DeMARCO, T. **Structured Analysis and System Specification**. Englewood Cliffs: Prentice-Hall, 1979.

/DIJ 68/ DIJKSTRA, E. A constructive approach to the problem of program correctness. **BIT**, Copenhage, v.8, 1968.

/DIJ 75/ DIJKSTRA, E. Guarded commands, non determinancy and formal derivation of programs. **Communications of the ACM**, New York, v.18, n.8, p.453-457, 1975.

/DIJ 76/ DIJKSTRA, E. **A discipline of programming**. Englewood Cliffs: Prentice-Hall, 1976.

/DIT 88/ DITTRICH, K.R. **Advances in object-oriented database systems**. New York: Springer-Verlag, 1988. (Lecture Notes in Computer Science, v.334).

/FEA 82a/ FEATHER, M. Mappings for rapid prototyping. **ACM Sigsoft Software Engineering Notes**, New York, v.7, n.5, p.17-24, Dec. 1982.

/FEA 82b/ FEATHER, M. A system for assisting program transformation. **ACM Trans. Program. Lang. Syst.**, New York, v.4, n.1, p.1-20, Jan. 1982.

/FER 92/ FERNANDES, E.S.T. **Arquiteturas Super Escalares:** Detecção e Exploração do Paralelismo de Baixo Nível. Gramado: Instituto de Informática da UFRGS, 1992. 135p.

/FIC 85/ FICKAS, S. Automating the transformational development of software. **IEEE Transactions on Software Engineering**, New York, v.11, n.11, p.1268-1277, 1985.

/FRE 85/ FREYTAG, J.C.; GOODMAN, N. **Rule-based translation of relational queries into interative programs.** San Jose, CA.: IBM Almadem Research Center, Dec. 1985. (Research Report, RJ4974(51986)).

/FRE 87/ FREYTAG, J.C. A rule-based view of query optimization. In: ACM-SIGMOD, 1987, San Francisco. **Proceedings...** [S.l.:s.n.], 1987. p.173-180.

/FRI 92/ FRIGERI, S.R. **Um estudo de álgebras e algoritmos para processamento e otimização de consultas em Banco de Dados.** Porto Alegre: CPGCC da UFRGS, 1992. (TI, 261).

/FUR 73/ FURTADO, A.L. **Teoria dos grafos:** algoritmos. Rio de Janeiro: LTC, 1973. 155p.

/GAN 87/ GANSKI, R.A.; WONG, H.K.T. Optimization of nested SQL queries revisited. In: ACM-SIGMOD INTERN. CONF. ON MANAGEMENT OF DATA, 1987, San Francisco. **Proceedings...** [S.l.:s.n.], 1987. p.23-33.

/GAR 89/ GARNOUSSET, H.E. et al. Efficient tools for analysis and implementation of manufacturing systems modelled by Petri net with object: a production rules compilation-based approach, IECON'89. In: ANNUAL CONFERENCE OF THE IEEE INDUSTRIAL SOCIETY, 15., 1989, Philadelphia, Pennsylvania. **Proceedings...** [S.l.:s.n.], 1989. v.3, p.543-549.

/GAR 90/ GARNOUSSET, H.E. et al. A manufacturing system simulator based on the Petri net with objects mode. In: EUROPEAN SIMULATION CONFERENCE - ESM, 1990, Nierenberg. **Proceedings...** [S.l.:s.n.], 1990.

/GEN 87/ GENRICH, H.J. Predicate/transition nets. In: BRAUER, W. et al. (Eds.). **Petri Nets:** Basic Models of Concurrency. Advances in Petri Nets 1986, Part I. Berlin: Springer-Verlag, 1987. p.207-247. (Lecture Notes in Computer Science, v.255).

/GOL 80/ GOLENDZINER, L.G. **Uma metodologia para melhorar a eficiência de instruções de recuperação em Banco de Dados.** Porto Alegre: CPGCC da UFRGS, 1980. (Dissertação de Mestrado).

/GRA 81/ GRAY, J. The transaction concept: virtues and limitations. In: VLDB, 7., 1981, Cannes, France. **Proceedings...** New York: IEEE, 1981. p.144-154.

/HAE 83/ HAERDER, T.; REUTER, A. Principles of Transaction-Oriented Database Recovery. **ACM Computing Surveys**, New York, v.15, n.4, p.287-317, Dec. 1983.

/HAL 76/ HALL, P.A.V. Optimization of Single Expressions in a Relational Data Base System. **IBM Journal of Research and Development**, New York, v.20, n.3, p.244-257, May 1976.

/HAR 86/ HARPER, R.; MITCHELL, K. **Introduction to standard ML.** Edinburgh: University of Edinburgh, 1986. 79p.

/HEU 81/ HEUSER, C.A.; GOLENDZINER, L.G.; OLIVEIRA, J.P.M. Sistema L: uma implementação da linguagem LOBAN. In: SEMISH - SEMINÁRIO INTEGRADO DE SOFTWARE E HARDWARE, 8., 1981, Florianópolis. **Anais...** Florianópolis: UFSC, 1981. p.169-86.

/HEU 87/ HEUSER, C.A. Integrando propriedades estáticas e dinâmicas no modelo conceitual. In: SIMPÓSIO BRASILEIRO DE BANCO DE DADOS, 2., 1987, Porto Alegre. **Anais...** Porto Alegre: UFRGS/SBC, 1987. p.66-80.

/HEU 89/ HEUSER, C.A. **Modelagem conceitual de sistemas.** Buenos Aires: Editorial Kapelusz, 1989.

/HEU 91/ HEUSER, C.A.; PERES, E.M. ER-T diagrams: an approach to specifying database transactions. In: INTERNATIONAL CONFERENCE ON THE ER APPROACH, 10., 1991, San Francisco. **Proceedings...** [S.l.:s.n.], 1991.

/HIL 90/ HILDUM, D.; COHEN, J. A language for specifying program transformations. **IEEE Transactions on Software Engineering**, New York, v.16, n.6, p.630-638, June 1990.

/HOA 69/ HOARE, C.A.R. An axiomatic basis for computer programming. **Communications of the ACM**, New York, v.12, n.10, p.576-580, 1969.

/HOL 78/ HOLT, R.C. **Structured concurrent programing with operating systems applications.** Reading: Addison-Wesley, 1978.

/ING 91/ INGRES. **INGRES/Embedded SQL Companion Guide for C, For the UNIX Operatiing System.** Release 6.4, Dec. 1991.

/INN 90/ INNOCENTI, M.D. et al. "RSF": a formalism for executable requirement specifications. **IEEE Transactions on Software Engineering**, New York, v.16, n.11, p.1235-1246, Nov. 1990.

/ISO 82/ VAN GRIETHUYSEN, J.J. (Ed.). **Concepts and terminology for the conceptual schema and the information base.** New York: ANSI, 1982. (Publication number ISO/TC97/SC5-N695).

/ISO 86/ ISO DP9074. **Estelle:** a formal description technique based on extended state transition model. Out. 1986.

/JAR 84/ JARNE, M.; KOCK, J. Query optimization in Database Systems. **ACM Computing Surveys**, New York, v.16, n.2, p.111-152, June 1984.

/KAN 81/ KANT, E.; BARSTOW, D.R. The refinement paradigm: the interaction of coding and efficiency knowledge in program synthesis. **IEEE Transactions on Software Engineering**, New York, v.SE-7, n.5, p.458-471, Sept. 1981.

/KER 90/ KERSCHBERG, L. Expert database systems: knowledge data management environment for intelligent information system. **Information System**, New York, v.15, n.1, p.151-160, 1990.

/KIN 84/ KING, R.; McLEOD, D. A unified model and methodology for conceptual database design. In: BRODIE, M.L.; MYLOPOULOUS, J.; SCHMIDT, J.W. (Eds.). **On Conceptual Modeling. Perspectives from Artificial Intelligence, Databases and Programming Languages.** New York: Springer-Verlag, 1984, p.313-327.

/KIN 86/ KING, R.; McLEOD, D. The Event database specification model. In: INTERNATIONAL CONFERENCE ON DATABASES: IMPROVING USABILITY AND RESPONSIVENESS, 2., 1986, Jerusalem, Israel. **Proceedings...** [S.l.:s.n.], 1986. p. 299-322.

/KOR 93/ KORTH, H.F.; SILBERSCHATZ, A. **Sistema de Banco de Dados**, São Paulo: Makron Books, 1993.

/KUN 86/ KUNG, C.; SOLVBERG, A. Activity and behavior Modeling. In: OLLE, T. (Ed.). **Information Systems Design Methodologies:** improving the Practice. Amsterdam: North-Holland, 1986. p.145-171.

/LI 89/ LI, P.; VON THUM, M.; DILLON, T.S. Semiautomatic implementation of communication protocols from a Petri net base specification language description. In: INTERNATIONAL CONFERENCE ON FORMAL DESCRIPTION TECHNIQUES, 2., 1989. **Proceedings...** [S.l.:s.n.], 1989.

/LIM 82/ LIMA, V.L.S. **Um estudo sobre resolução de operações de consulta de Banco de Dados.** Porto Alegre: CPGCC da UFRGS, 1982. (Dissertação de Mestrado).

/LIN 86/ LINN Jr., R.J. **The features and facilities of Estelle. Protocol specification, testing and verification.** Amsterdam: North-Holland, 1986. p.271-296.

/LOH 87/ LOHMAN, G.M. **Grammar-like functional rules for representing query optimization alternatives.** San José, CA.: IBM Almadem Research Center, 1987. (Research Report, RJ 5992).

/MAR 91/ MARTIN, J.; McCLURE, C. **Técnicas Estruturadas e CASE.** São Paulo: Makron, McGraw-Hill, 1991.

/MEL 89/ MELO, W.L.M. **Uma proposta de um editor diagramático generalizado.** Porto Alegre: CPGCC da UFRGS, 1989. (Dissertação de Mestrado).

/MOH 90/ MOHANG, C. et al. **Parallelism in Relational Data Base Systems:** Architectural Issues and Design Approaches. San José, CA.: IBM Almadem Research Center, 1990. (Research Report).

/NEL 83/ NELSON, R.A.; HAIBT, L.; SHERIDAN, P.D. Casting Petri nets into programs. **IEEE Transactions on Software Engineering**, New York, v.19, n.5, p.590-602, 1983.

/PAI 83/ PAIGE, R. Transformational programming: aplication to algorithms and systems. In: ACM SYMP. ON PRINCIPLES OF PROGRAMMING LANGUAGES, 10., 1983, Austin, Texas. **Proceedings...** New York: ACM, 1983. p.73-87.

/PAR 83/ PARTSCH, H.; STEINBRUGGEN, R. Program transformation system. **Comput. Surveys**, New York, v.15, n.3, p.199-236, Sept. 1983.

/PEC 88/ PECKHAM, J.; MARYANSKI, F. Semantic data models. **ACM computing surveys**, New York, v.20, n.3, p.153-189, Sept. 1988.

/PER 90/ PERES, E.M. **Abordagem ER/T:** uma proposta para a integração da modelagem conceitual de propriedades estáticas e dinâmicas de sistemas de informação. Porto Alegre: CPGCC da UFRGS, 1990. (Dissertação de Mestrado).

/PIN 88/ PINGALI, K.; EKANADHAM, K. **Accumulators:** new logic variable abstractions for functional languages. Ithaca: Cornell University, 1988. (Technical Report, 88-952).

/PIR 91/ PIRAHESH, H.; HELLERSTEIN, J.M.; HASAN, W. **Extensible/rule based query rewrite optimization in Starburst.** San Jose, CA: IBM Almadem Research Center, 1992. (Research Report).

/RAD 75/ RADUE, J.E.; MULLINS, J.M. Solving synchronization problems using semaphores. **Software-Practice and Experience**, New York, v.5, n.1, p.51-64, Jan./Mar. 1975.

/RED 86/ REDDY, U.S. On the relationship between logic and functions languages. In: DEGROOT, D.; LINDSTROM, G. (Eds.). **Logic Programming:** functions, relations and equations, Englewood Cliffs: Prentice-Hall, 1986. p.3-36.

/RED 87/ REDDY, U.S. Functional logic languages, Part I. In: **Graph Reduction**, Berlin: Springer-Verlag, 1987, p.401-425. (Lecture Notes in Computer Science, v.279).

/RED 89/ REDDY, U.S. Transformational derivation of program using the Focus system. **ACM Sigplan Notices**, New York, v.24, n.2, Feb. 1989.

/REG 90/ REGO, A.M. **Uma linguagem gráfica de definição de dados para um modelo E/R estendido**. Porto Alegre: CPGCC da UFRGS, 1990. (Dissertação de Mestrado).

/REI 87/ REISING, W. Place/transition systems. In: BRAUER, W. (Ed.). **Petri Nets:** basic models of concurrency. Advances in Petri Nets 1986, Part I. Berlin: Springer-Verlag, 1987. p.117-141. (Lecture Notes in Computer Science, v.255).

/SAN 80/ SANTOS, C.S. **Caracterização Sistemática de Restrições de Integridade em Banco de Dados**. Rio de Janeiro: PUC/RJ, 1980. (Tese de Doutorado).

/SAN 82/ SANTOS, C.S. et al. Projeto de Banco de Dados baseado em restrições de integridade. In: CONGRESSO NACIONAL DE INFORMÁTICA, 15., 1982, Rio de Janeiro. **Anais...** Rio de Janeiro: SUCESU, 1982, p.316-323.

/SAK 83/ SAKAI, H. A method for entity-relationship behavior modeling. In: DAVIS, C.; JAJODIA, P.; YEH, R. (Eds.). **Entity-Relationship Approach to Software Engineering**. Amsterdam: North-Holland, 1983. p.111-129.

/SCH 79/ SCHMIDT, J.W. Some high level language constructs for data type relation. **ACM TODS**, v.2, n.3, Sept. 1979.

/SET 89/ SETZER, V.W. **Banco de Dados:** conceitos, modelos, gerenciadores, projeto lógico, projeto físico. São Paulo: Ed. E. Blücher, 1989.

/SIL 90/ PEREIRA E SILVA, R. **Uma proposta para a implementação de modelos baseados em Rede de Petri a Objetos**. Florianópolis: UFSC, 1990. (Dissertação de Mestrado).

/SZW 84/ SZWARCFITER, J.L. **Grafos e Algoritmos Computacionais**. Rio de Janeiro: Campus, 1984.

/TAN 91/ TANAKA, A.K. et al. ER-R: an enhanced ER model with situation-action rules to capture application semantics. In:INTERNATIONAL CONFERENCE ON THE ENTITY RELATIONSHIP APPROACH, 10., 1991. **Proceedings...** [S.l.:s.n.], 1991.

/TAN 92/ TANENBAUM, A.S. **Modern operating systems**. Englewood Cliffs: Prentice-Hall, 1992.

/TUC 82/ TUCHERMAN, L.; FURTADO, A.L. Métodos para garantir a integridade em Banco de Dados. In: CONGRESSO NACIONAL DE INFORMÁTICA, 15., 1982, Rio de Janeiro. **Anais...** [S.l.:s.n.], 1982. p.345-350.

/WEB 83/ WEBER, W. et al. Integrity Checking in Data Base Systems. **Information Systems**, New York, v.8, n.2, p.125-36, Apr. 1983.

/WON 83/ WONG, E.; KATZ, R. Distributing a database for parallelism. In: ACM-SIGMOD, 1983, San José, CA. **Proceedings...** [S.l.:s.n.], 1983, p.23-29.

/YU 84/ YU, C.T.; CHANG, C.C. Distributed Query Processing. **ACM Computing Surveys**, New York, v.16, p.399-433, Dec. 1984.

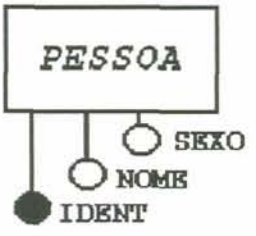
APÊNDICE A - REGRAS DE MAPEAMENTO DO MODELO DE DADOS

Neste apêndice estão especificadas as regras utilizadas no mapeamento do modelo ER para uma correspondente representação no modelo relacional, aqui exemplificado pelo SQL do padrão ANSI /DAT 89/.

Para simplificar as ilustrações dos exemplos em SQL, foram mantidos, nos atributos e relações destes exemplos, os mesmos nomes especificados para os correspondentes atributos e entidades/relacionamentos no diagrama ER/T^+. Durante a efetiva implementação das regras de mapeamento, mecanismos de tradução de nomes devem realizar as transposições necessárias considerando, por exemplo, uma prefixação do nome do atributo da relação com algum mnemônico (ex.: nome da relação, primeiras letras do nome da relação, etc.). Isto permitirá, durante a modelagem do sistema, o uso de nomes iguais para atributos de entidades diferentes e, mesmo assim, evitará a necessidade de qualificação dos atributos quando da referência por instruções SQL que implementam a manipulação dos dados do BD.

As regras são as seguintes:

a) **Entidades**:



Regra:

Para cada entidade E, criar uma relação R incluindo todos os atributos atômicos de E nesta relação. Escolher o atributo(s) chave(s) da entidade E como chave primária da relação R.

Exemplo:

Relação : PESSOA
Chave : IDENT
Atributos: NOME
SEXO

Em SQL:

```
CREATE TABLE Pessoa
       (Ident     DECIMAL(6) NOT NULL,
        Nome      CHAR(30),
        Sexo      CHAR(01),

PRIMARY KEY (Ident)
       )
```

Para os sistemas em que for modelado sincronização de transações em função de outras transações, adicionar no modelo de dados uma relação especial semáforo, onde são mantidos fatos de sinalização para controle de paralelismo/serialização de transações. Nesta relação, incluir o atributo chave "Rótulo".

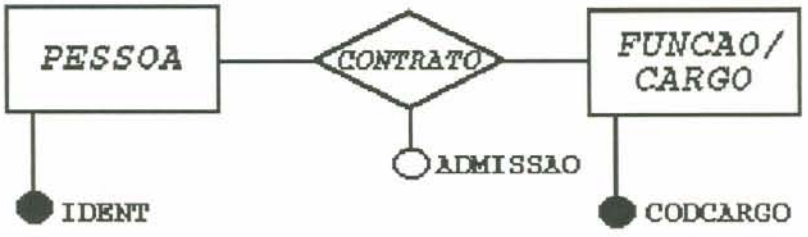
Exemplo:

Relação : SEMÁFORO
Chave : ROTULO

Em SQL:

```
CREATE TABLE Semáforo
       (Rotulo   CHAR(30) NOT NULL,

PRIMARY KEY (Rotulo)
       )
```

b) **Relacionamentos**:



Regra: (Relacionamentos N:N)

Para cada relacionamento S, criar uma relação R incluindo todos os eventuais atributos atômicos de S nesta relação. Como chave primária de R, fazer uma composição de atributos a partir dos atributos chaves das entidades envolvidas no relacionamento.

Exemplo:

```
Relação   : CONTRATO
Chave     : IDENT    (Atrib.estrang. com: PESSOA)
            CODCARGO (Atrib.estrang. com: CARGO)
Atributos: ADMISSAO
```

Em SQL:

```
CREATE TABLE Contrato
       (Ident    DECIMAL(6) NOT NULL,
        Codcargo DECIMAL(6) NOT NULL,
        Admissao CHAR(06),

PRIMARY KEY (Ident, Codcargo),
FOREIGN KEY (Ident)    REFERENCES Pessoa,
FOREIGN KEY (Codcargo) REFERENCES Cargo
       )
```

A regra para relacionamentos N:N pode também ser aplicada para os relacionamentos 1:N e 1:1. No entanto, em uma solução mais racional, é possível aplicar regras específicas para estes dois tipos de relacionamentos. Assim, tem-se:

Regra: (Relacionamentos 1:N)

Adicionar, na relação do lado N, todos os eventuais atributos do relacionamento e, como atributo estrangeiro, a chave da relação do lado 1 (realizando, assim, a vinculação entre as entidades). Evita-se, com isto, a criação de uma nova relação específica para manter o relacionamento.

Exemplo:

```
Relação   : PESSOA
Chave     : IDENT
Atributos: ...

Relação   : CARGO
Chave     : CODCARGO
Atributos: ...
           IDENT (Atrib.estrang. com: PESSOA)
           ADMISSAO
```

Regra: (Relacionamentos 1:1)

Adicionar, em uma das relações que representam as entidades envolvidas no relacionamento, todos os eventuais atributos do relacionamento e, como atributo estrangeiro, a chave da outra relação envolvida neste relacionamento. Também evita-se, aqui, a criação de uma nova relação.

Exemplo:

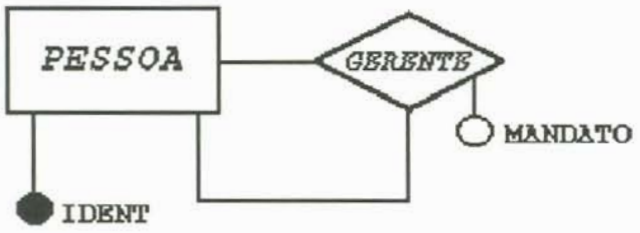
```
Relação   : PESSOA
Chave     : IDENT
Atributos: ...

Relação   : CARGO
Chave     : CODCARGO
Atributos: ...
           IDENT (Atrib.estrang. com: PESSOA)
           ADMISSAO
```

Cabe frisar que relacionamentos 1:1 tem também a característica de permitir juntar as entidades envolvidas, gerando uma entidade única. Sendo assim, outra possibilidade é criar uma relação única que implemente esta união entre as duas entidades. Esta característica, no entanto, não está sendo aqui considerada em vista da idéia de manter-se, na implementação, a intenção do analista, durante a modelagem ER, de não unificar as entidades.

A regra do relacionamento 1:N (inclusão do relacionamento no lado N) pode ser vista como uma especialização da regra do relacionamento 1:1 (inclusão do relacionamento em qualquer lado) e, portanto, mais restrita. Sendo mais restrita, a regra 1:N satisfaz também ao relacionamento 1:1 se considerado, para o caso, o pressuposto lado N (por exemplo, convencionado à direita na representação do diagrama ER) assumindo uma cardinalidade igual a 1. Assim, baseado nesta premissa, para os propósitos dos mecanismos de implementação dos aspectos comportamentais da especificação ER/T^+ apresentados nesta tese, as regras de mapeamento dos relacionamentos se resumem a aplicar:

- regra N:M para relacionamentos N:M;
- regra 1:N para relacionamentos 1:N e 1:1 (caso onde lado N tem cardinalidade igual a 1, convencionado como sendo lado da direita no diagrama).

c) **Auto-relacionamentos**:



Regra:

Aplicar a regra de mapeamento de relacionamentos. Adicionalmente, para a formação da chave primária, renomear o atributo chave da entidade associada como segue:

<chave primária> ::= <atrb>_1 , <atrb>_2

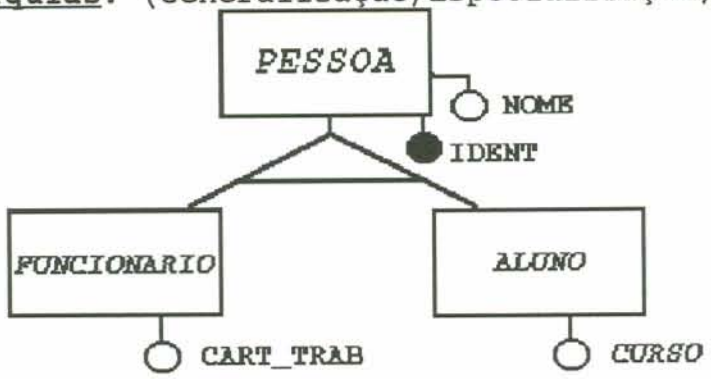
Exemplo:

```
Relação   : GERENTE
Chave     : IDENT_1 (Atrib.estrang. com: PESSOA)
            IDENT_2 (Atrib.estrang. com: PESSOA)
Atributos: MANDATO
```

Em SQL:

```
CREATE TABLE Gerente
       (Ident_1  DECIMAL(6) NOT NULL,
        Ident_2  DECIMAL(6) NOT NULL,
        Mandato  CHAR(06),

PRIMARY KEY (Ident_1, Ident_2),
FOREIGN KEY (Ident_1)  REFERENCES Pessoa,
FOREIGN KEY (Ident_2)  REFERENCES Pessoa
       )
```

d)**Hierarquias**: (Generalização/Especialização, Supertipo/Subtipo)



Regra:

Para cada especialização S (subtipo) da entidade E (supertipo), criar uma relação R incluindo todos os atributos de S na relação. Como chave primária da relação usar a mesma chave da relação correspondente a entidade E.

Exemplo:

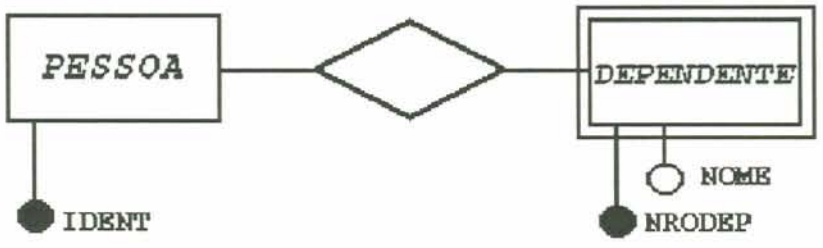
```
Relação   : FUNCIONARIO
Chave     : IDENT    (Atrib.estrang. com: PESSOA)
Atributos: CART_TRAB
```

Em SQL:

```
CREATE TABLE Funcionario
       (Ident     DECIMAL(6) NOT NULL,
        Cart_trab CHAR(06),

PRIMARY KEY (Ident),
FOREIGN KEY (Ident)  REFERENCES Pessoa
       )
```

Desta forma, a associação entre as relações PESSOA (Supertipo) e FUNCIONÁRIO (subtipo) é feita através do atributo chave "Ident", comum às duas relações.

e)**Entidades fracas**: (Existência dependente)



Regra:

Para cada entidade fraca (dependente) F, associada a entidade E (entidade Pai), criar uma relação R com todos atributos atômicos da entidade F. A chave primária de R será composta a partir da combinação do atributo chave de E (atributo estrangeiro) com o atributo chave de F.

Exemplo:

```
Relação   : DEPENDENTE
Chave     : IDENT    (Atrib.estrang. com: PESSOA)
            NRODEP
Atributos: NOME
            DATA_NASC
```

Em SQL:

```
CREATE TABLE Dependente
       (Ident      DECIMAL(6) NOT NULL,
        Nrodep     DECIMAL(2) NOT NULL,
        Nome       CHAR(30),
        Data_nasc  CHAR(06),

PRIMARY KEY (Ident, Nrodep),
FOREIGN KEY (Ident)  REFERENCES Pessoa
       )
```

f)**Atributos multivalorados**: (repetitivos, sujeitos a normalização)

Regra:

Para cada atributo multivalorado M da entidade E, criar uma relação auxiliar R1 contendo, como chave primária, a combinação de atributos a partir da replicação da chave da relação R que mapeia a entidade E acrescida do próprio atributo multivalorado M.

Exemplo:

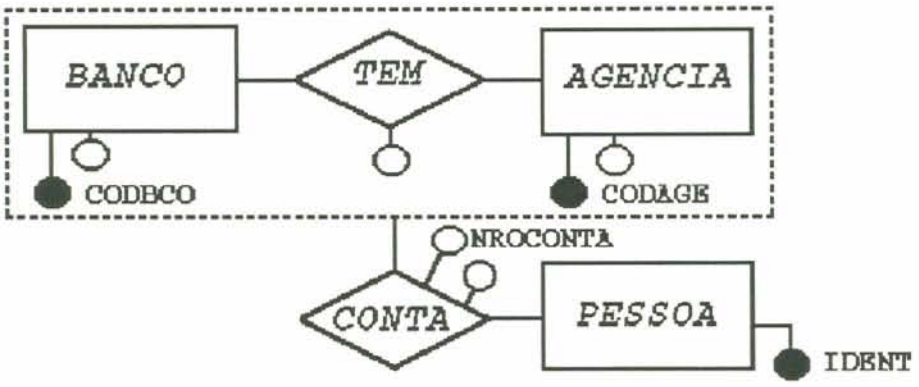
```
Relação   : HABIL_PESS
Chave     : IDENT    (Atrib.estrang. com: PESSOA)
            HABILIT
Atributos: <nenhum>
```

Em SQL:

```
CREATE TABLE Habil_pess
       (Ident     DECIMAL(6) NOT NULL,
        Habilit   DECIMAL(3) NOT NULL,

PRIMARY KEY (Ident, Habilit),
FOREIGN KEY (Ident)  REFERENCES Pessoa
       )
```

Esta regra é também frequentemente aplicada em processos de normalização de estruturas de BD (obtenção de relação com atributos atômicos: primeira forma normal /DAT 86a/, /KOR 93/).

g) **Agregações**:



Regra:

Para cada relacionamento S que tenha vinculado uma agregação A, criar uma relação R incluindo todos os eventuais atributos atômicos do relacionamento. Para compor a chave desta relação, combinar todos os atributos chave da agregação (do relacionamento entre as entidades pertencentes à agregação) com o(s) atributo(s) chave da entidade associada (ou outra agregação)

Exemplo:

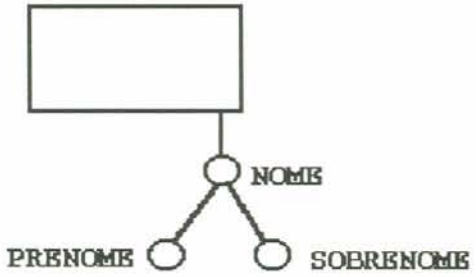
```
Relação   : CONTA
Chave     : CODBCO   (Atrib.estrang. com: TEM)
            CODAGE   (Atrib.estrang. com: TEM)
            IDENT    (Atrib.estrang. com: PESSOA)
```

Atributos: NROCONTA
TIPO
DATA_ABERTURA

Em SQL:

```
CREATE TABLE Conta
       (Codbco          DECIMAL(3) NOT NULL,
        Codage          DECIMAL(4) NOT NULL,
        Ident           DECIMAL(6) NOT NULL,
        Nroconta        DECIMAL(11),
        Tipo            DECIMAL(2),
        Data_abertura   CHAR(6),

PRIMARY KEY (Codbco, Codage, Ident),
FOREIGN KEY (Codbco, Codage) REFERENCES Tem
FOREIGN KEY (Ident)          REFERENCES Pessoa
       )
```

h)**Atributos agrupados**: (compostos)



Regra:

O mapeamento de atributos agrupados se limitará a indicar somente os atributos atômicos que compõem o grupo.

Exemplo:

Relação : -------
Chave : -------
Atributos: PRENOME
SOBRENOME

APÊNDICE B - ALGORITMOS PARA IMPLEMENTAÇÃO DO MODELO DE TRANSAÇÕES

Neste apêndice estão relacionados os algoritmos propostos como alternativa de implementação do modelo de transações a partir dos conceitos discutidos no capítulo 5.

a) **Grafo de dependência**:

Como alternativa de implementação do grafo de dependência, pode-se considerar os seguintes procedimentos:

- especificar uma matriz de adjacência (matriz[n,m]) para representar o grafo de dependência onde "n" é o número de fluxos do diagrama e "m" é o número de diferentes fluxos dependentes.

- considerando "G" o conjunto de arcos (elos) e "(i-j)" um arco indo do vértice (nodo) "i" para o vértice "j", ou seja, no caso do diagrama ER/T^+, o fluxo "i" tem uma dependência do fluxo "j", preencher a matriz de adjacência considerando:
  - .matriz[i,j] = 1, se (i-j) pertence a "G"; (é dependente)
  - .matriz[i,j] = 0, se (i-j) não pertence a "G"; (não é dependente)

- criar uma linha adicional na matriz (matriz[n+1,__]), inicializada com zeros, para assinalamento das dependências já resolvidas.

- criar uma coluna adicional na matriz (matriz[__,m+1]), inicializada com zeros, para assinalamento da seqüência de reordenação ("seq").

Na Figura 28, a primeira matriz ilustra a situação inicial após a implementação do grafo de dependência.

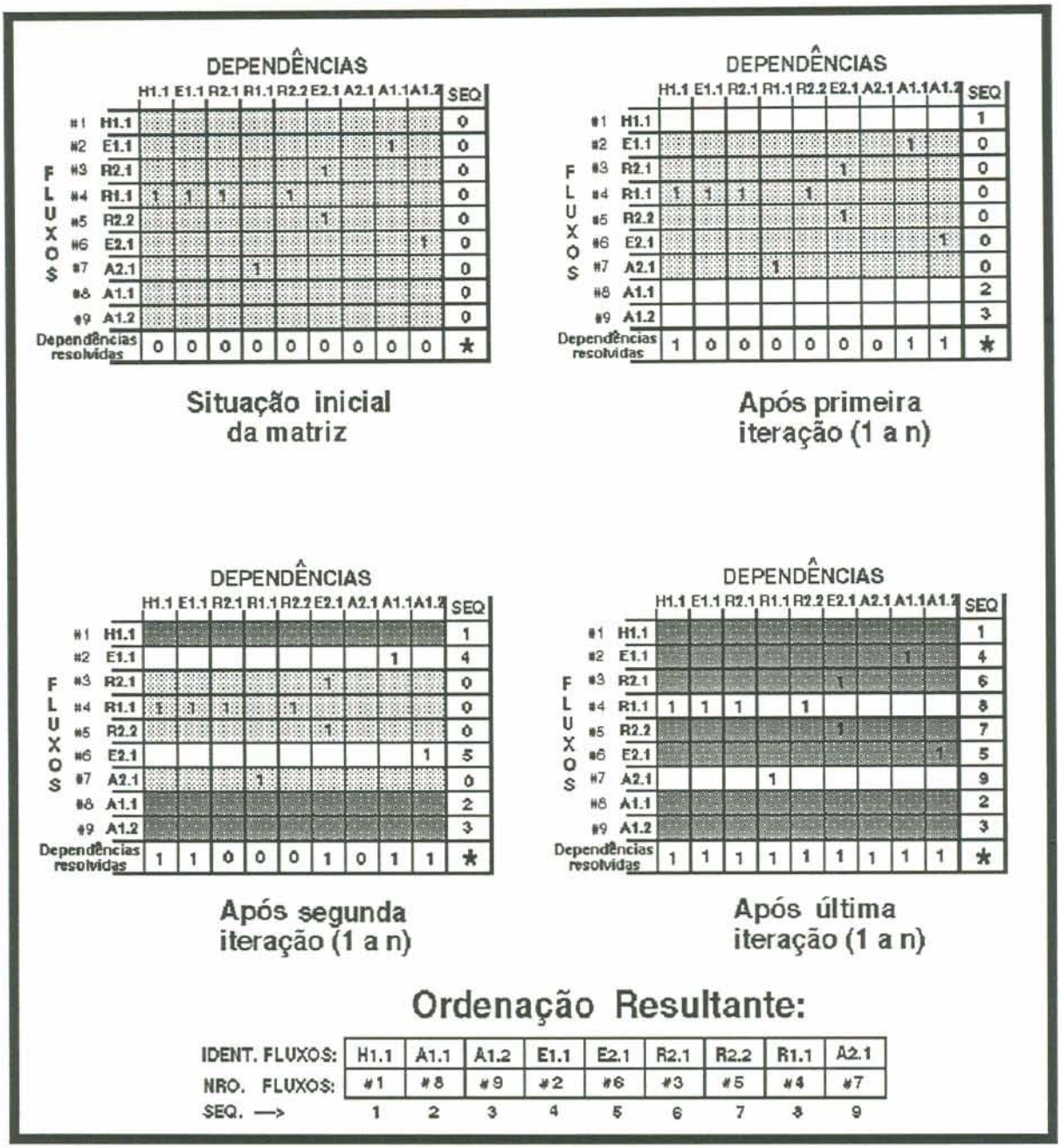


Fig. 28 - Matriz de adjacência para o grafo e a resolução das dependências

b) <u>Seqüencialização das operações</u>:

A seqüencialização das operações, resolvendo as dependências e fixando o vínculo de precedência, pode ser implementada considerando os seguintes procedimentos:

- executar o procedimento de reordenação dos fluxos, representados na matriz[n,m], conforme o seguinte algoritmo:

Algoritmo de resolução das dependências dos fluxos:
(versão 1: versão para seqüencialização das operações)

```
1. INÍCIO
2.   seq=1;
3.   fimreord="FALSO";
4.   ENQUANTO não(fimreord) FAÇA:
5.     INÍCIO
6.       teveseq="FALSO";
7.       PARA i=1 até n FAÇA:
8.         INÍCIO
9.           resolv="VERDADEIRO";
10.          SE matriz[i,m+1] = 0 ENTÃO
11.             /* se fluxo ainda não tenha sido seqüenciado
12.             PARA j=1 até m FAÇA:
13.               /* verifica todas dependências do fluxo
14.               SE matriz[i,j]=1 e matriz[n+1,j]=0 ENTÃO
15.                  /* é dependente e ainda não resolvido
16.                  INÍCIO
17.                    resolv="FALSO";
18.                    j= m+1;  /*se achou, força a parada
19.                  FIM
20.          SE resolv  e matriz[n+1,i] não = 1 ENTÃO
21.             /* se todas dependências do fluxo estão
22.             /* resolvidas mas ainda não assinaladas
23.             INÍCIO
24.               matriz[i,m+1]= seq;
25.               teveseq="VERDADEIRO";
26.               ordena[seq]=i;
27.               seq=seq + 1;
28.               matriz[n+1,i]=1; /* marca depend. resolvida
29.             FIM
30.        FIM /* fim de uma iteração completa na matriz
31.      SE seq > n  ou  não(teveseq) ENTÃO
32.         /* finaliza se todos fluxos foram processados ou
33.         /* não teve seqüencialização numa iteração
34.         fimreord="VERDADEIRO";
35.    FIM /* fim de seqüencialização dos fluxos
36.  SE não(teveseq) ENTÃO
37.     /* não foi possível seqüencializar todos fluxos
38.     trataerro("ERRO: tem dependência insolúvel");
39. FIM
```

Obs.: A seqüência ordenada dos fluxos por dependência está em "ordena[N]"

Na Figura 28, as matrizes 2,3 e 4 ilustram a seqüência de reordenação para resolver as dependências.

c) **Paralelismo**:

Versão paralelismo entre nodos:

Para implementar o reconhecimento de paralelismos, considera-se o algoritmo original de resolução de dependência de fluxos, apresentado anteriormente, e adicionalmente aplicam-se os seguintes procedimentos:

- adicionar uma coluna na matriz de adjacência para assinalar os paralelismos entre nodos; nesta coluna, todos fluxos que podem ser executados em paralelo devem ser rotulados com números iguais;
- modificar o algoritmo de resolução das dependências dos fluxos para, numa iteração completa (i=1 até n) de sequencialização das linhas da matriz, marcar com números iguais (paralelismo) aqueles fluxos que são funcionalmente independentes ou cuja dependência foi liberada (resolvida) na iteração anterior; não permitir, numa mesma iteração , a liberação de 2 fluxos dependentes.

Os procedimentos estão descritos no algoritmo apresentado a seguir. Nele as modificações, a partir do algoritmo original, estão realçadas em negrito/itálico.

Algoritmo de resolução das dependências dos fluxos:
(versão 2:versão com definição de paralelismos entre nodos)

```
 1. INÍCIO
1.1 nroloop=0;
 2.   seq=1;
 3.   fimreord="FALSO";
 4.   ENQUANTO não(fimreord) FAÇA:
 5.     INÍCIO
5.1       nroloop = nroloop + 1;
 6.       teveseq="FALSO";
 7.       PARA i=1 até n FAÇA:
 8.         INÍCIO
 9.           resolv="VERDADEIRO";
10.           SE matriz[i,m+1] = 0 ENTÃO
11.              /* se fluxo ainda não tenha sido seqüenciado
12.              PARA j=1 até m FAÇA:
13.                /* verifica todas dependências do fluxo
14.                SE matriz[i,j] = 1 e matriz[n+1,j] = 0
15.                   /* é dependente e ainda não resolvido
15.1                ou matriz[j,m+2] = nroloop ENTÃO
15.2                  /* ou resolvido nesta iteração (deve ser
15.3                  /* inibida a liberação neste caso)
16.                   INÍCIO
17.                     resolv="FALSO";
18.                     j= m+1;  /*se achou, força a parada
19.                   FIM
20.           SE resolv  e matriz[n+1,i] não = 1 ENTÃO
21.              /* se todas dependências do fluxo estão
22.              /* resolvidas mas ainda não assinaladas
23.              INÍCIO
24.                matriz[i,m+1]= seq;
24.1               matriz[i,m+2]= nroloop;/* marca paralelismo
25.                teveseq="VERDADEIRO";
26.                ordena[seq]=i;
27.                seq=seq + 1;
28.                matriz[n+1,i]=1; /* marca depend. resolvida
29.              FIM
30.         FIM /* fim de uma iteração completa na matriz
31.       SE seq > n   ou  não(teveseq) ENTÃO
32.          /* finaliza se todos fluxos foram processados ou
33.          /* não teve seqüencialização numa iteração
34.          fimreord="VERDADEIRO";
35.     FIM /* fim de seqüencialização dos fluxos
36.   SE não(teveseq) ENTÃO
37.      /* não foi possível seqüencializar todos fluxos
38.      trataerro("ERRO: tem dependência insolúvel");
39. FIM
```

Obs.: Todos fluxos assinalados com números iguais na coluna m+2 da matriz são passíveis de execução em paralelo

A Figura 29 ilustra os procedimentos de marcação dos paralelismos entre nodos do grafo na matriz de adjacência.

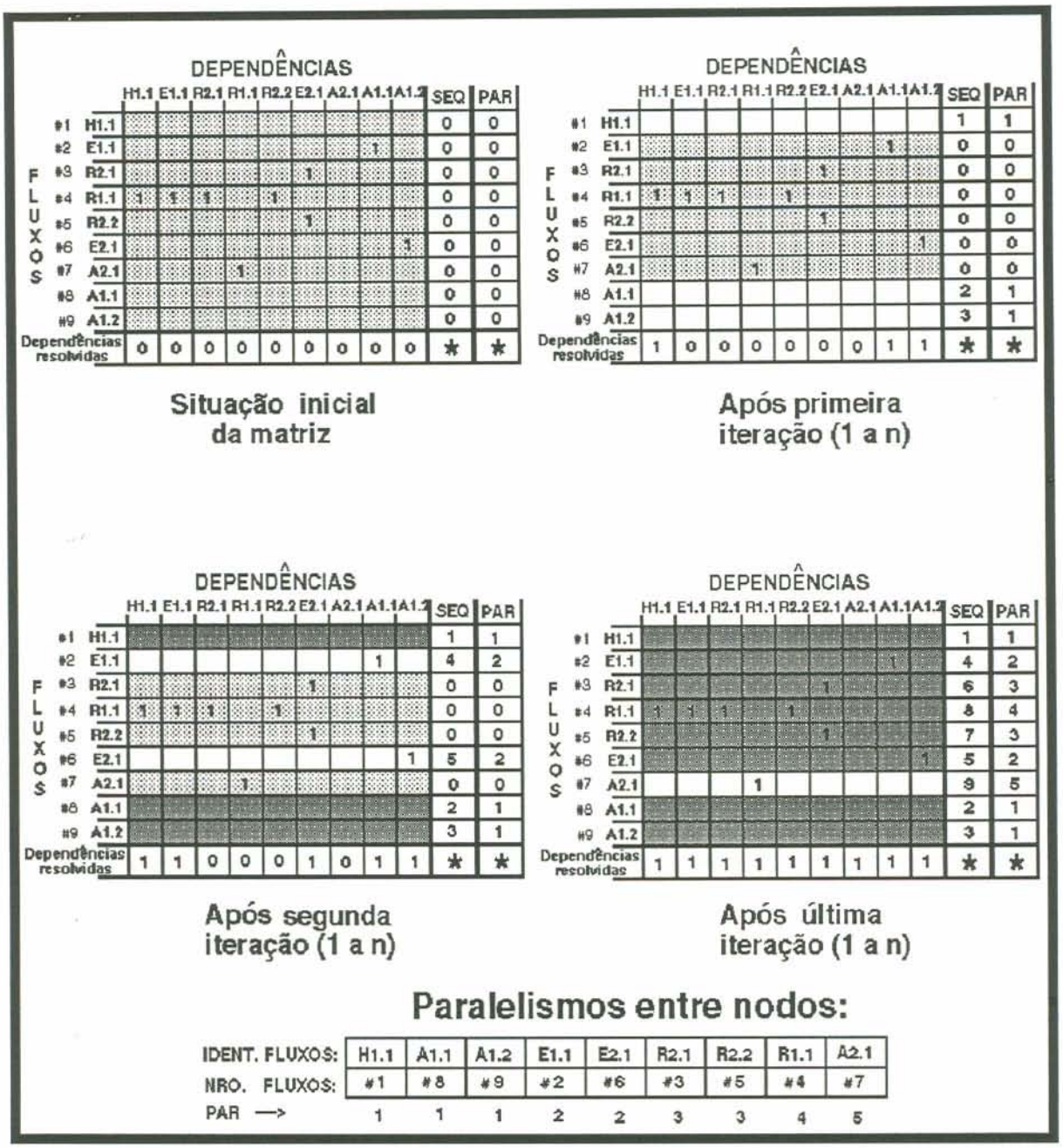


Fig. 29 - Matriz de adjacência para o grafo e a marcação de paralelismos entre nodos

Versão paralelismo entre ramos:

Esta melhoria do grau de paralelismo no grafo também pode ser marcada a partir de uma adaptação do algoritmo original de resolução das dependências dos fluxos. Uma possível

alternativa de implementação pode considerar os procedimentos a seguir.

- adicionar uma coluna na matriz de adjacência para assinalar os paralelismos entre ramos do grafo; nesta coluna, nodos de um mesmo ramo devem receber o mesmo número de grupo;
- adicionar uma linha a matriz de adjacência para assinalar nodos de convergência (1), nodos de divergência (2), nodos de convergência/divergência (3) ou nodos seqüenciais (0);
- modificar o algoritmo de resolução das dependências dos fluxos para:
    a) identificar nodos de convergência (na linha da matriz possuem mais de uma dependência);
    b) identificar nodos de divergência (na coluna da matriz são liberadores de mais de um fluxo dependente deles);
    c) identificar nodos de convergência & divergência;
    d) dentro da iteração de seqüencialização dos fluxos, marcar grupos como segue:
        - se nodo tem dependências e
          nodo não é de convergência e
          nodo liberador não é de divergência
          então:
            - marcar nodo com o número do grupo liberador (mesmo número do nodo liberador)
          caso contrário:
            - iniciar novo grupo com o nodo

No algoritmo proposto a seguir, as modificações a partir do algoritmo original estão realçadas em negrito/itálico.

Algoritmo de resolução das dependências dos fluxos:
(versão 3:versão com definição de paralelismos entre ramos)

```
1. INÍCIO
1.01 /* identifica nodos de convergência/divergência
1.02 PARA i=1 até n FAÇA:
1.03   INÍCIO
1.04     nrodeph=0;
1.05     PARA j=1 até m FAÇA:
1.06       INÍCIO
1.07         SE matriz[i,j] = 1 ENTÃO nrodeph=nrodeph + 1;
1.08         SE i=0 ENTÃO /*na 1 linha vê também as colunas
1.09            INÍCIO
1.10              nrodepv=0;
1.11              PARA k=1 até n FAÇA:
1.12                SE matriz[k,j] = 1 ENTÃO
1.13                   nrodepv=nrodepv + 1;
1.14              SE nrodepv > 1 ENTÃO
1.15                 SE matriz[n+2,j] não = 0 /* ja converg.
1.16                 ENTÃO matriz[n+2,j] = 3; /* ambos
1.17                 CASO CONTRÁRIO
1.18                       matriz[n+2,j] = 2; /* divergência
1.19            FIM
1.20       FIM
1.21         SE nrodeph > 1 ENTÃO
1.22            SE matriz[n+2,i] não = 0 /* ja diverg.
1.23            ENTÃO matriz[n+2,i] = 3; /* ambos
1.24            CASO CONTRÁRIO
1.25                 matriz[n+2,i] = 1; /* convergência
1.26   FIM
1.27 nrogrupo = 1;
2.   seq=1;
3.   fimreord="FALSO";
4.   ENQUANTO não(fimreord) FAÇA:
5.     INÍCIO
6.       teveseq="FALSO";
7.       PARA i=1 até n FAÇA:
8.         INÍCIO
9.           resolv="VERDADEIRO";
9.01         onde = M;  /*inicializa depend. fora dos limites
10.          SE matriz[i,m+1] = 0 ENTÃO
11.             /* se fluxo ainda não tenha sido seqüenciado
12.             PARA j=1 até m FAÇA:
13.               /* verifica todas dependências do fluxo
13.01             SE matriz[i,j]=1 e matriz[n+2,i] não = 1
13.02                e matriz[n+2,i] não = 3 ENTÃO
13.03                /* tem uma e somente uma dependência
13.04                onde = j; /* assinala onde está o seu
13.05                          /* nodo liberador
14.              SE matriz[i,j]=1 e matriz[n+1,j]=0 ENTÃO
15.                 /* é dependente e ainda não resolvido
16.                 INÍCIO
17.                   resolv="FALSO";
18.                   j= m+1;  /*se achou, força a parada
19.                 FIM
20.          SE resolv  e matriz[n+1,i] não = 1 ENTÃO
21.             /* se todas dependências do fluxo estão
22.             /* resolvidas mas ainda não assinaladas
23.             INÍCIO
24.               matriz[i,m+1]= seq;
24.01             /* marca os grupos de paralelismos
24.02             SE onde não = M  e   /* tem dependência
24.03                matriz[n+2,i] não = 1 e
24.04                matriz[n+2,i] não = 3 e /* não é conver.
24.05                matriz[n+2,onde] não = 2 e
24.06                matriz[n+2,onde] não = 3
24.07                                 /* seu nodo liberador
                                      /* não é de divergência
24.08             ENTAO /* marcação dentro do grupo
24.09                matriz[i,m+2]= matriz[onde,m+2];
```

```
24.10                CASO CONTRÁRIO
24.11                  INÍCIO
24.12                    matriz[i,m+2]= nrogrupo;
                           /* marcação próximo grupo
24.13                    nrogrupo= nrogrupo + 1;
24.14                  FIM
25.                  teveseq="VERDADEIRO";
26.                  ordena[seq]=i;
27.                  seq=seq + 1;
28.                  matriz[n+1,i]=1; /* marca depend. resolvida
29.                FIM
30.          FIM /* fim de uma iteração completa na matriz
31.        SE seq > n   ou  não(teveseq) ENTÃO
32.           /* finaliza se todos fluxos foram processados ou
33.           /* não teve seqüencialização numa iteração
34.           fimreord="VERDADEIRO";
35.      FIM /* fim de seqüencialização dos fluxos
36.    SE não(teveseq) ENTÃO
37.       /* não foi possível seqüencializar todos fluxos
38.       trataerro("ERRO: tem dependência insolúvel");
39. FIM
```

Obs.: Assegurada sua habilitação, todos fluxos pertencentes a ramos diferentes(assinalados com números diferentes na coluna m+2 da matriz) são passíveis de execução em paralelo; dentro dos grupos (ramos) o processamento é seqüencial.

A Figura 30 ilustra os procedimentos de marcação dos paralelismos entre ramos do grafo na matriz de adjacência.

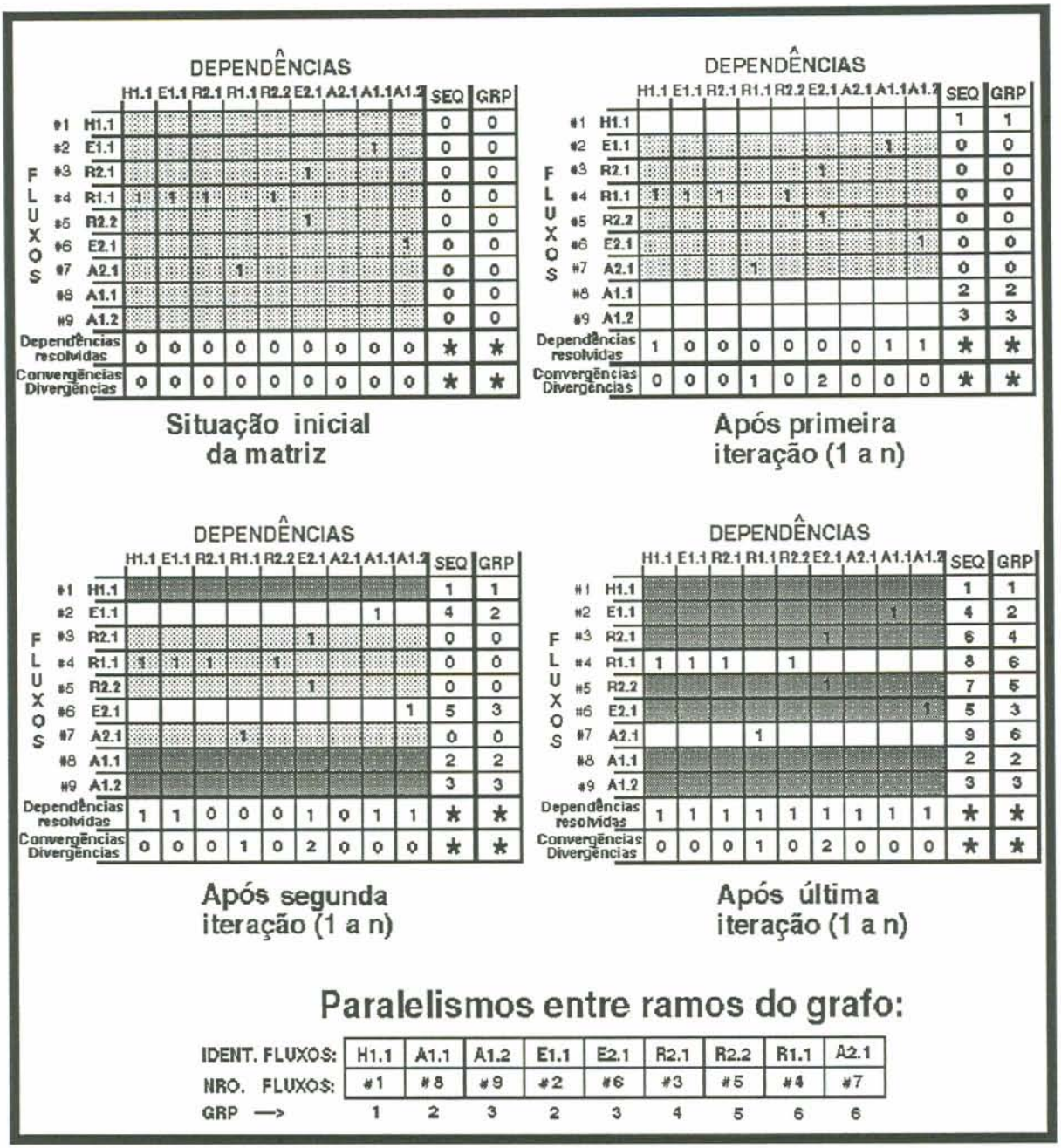


Fig. 30 - Matriz de adjacência para o grafo e a marcação de paralelismos entre ramos

Uma possível montagem do controle de disparo dos grupos paralelos pode considerar os seguintes procedimentos:

- formar grupos com os nodos de mesmo número de grupo (matriz[nodo,m+2]), assegurando a seqüência estabelecida dentro dos grupos (seqüência de operações);

- executar laço de montagem dos disparos dos grupos 1 a n, onde:
  . se primeiro nodo do grupo é dependente de outros grupos, então:
    - esperar conclusão do(s) grupo(s) que habilita(m) este nodo(WAIT G1,...,Gn, onde G1 a Gn são obtidos das dependências (grupos) do nodo (matriz[onde,m+2], com "onde" indexando os diferentes nodos liberadores deste nodo em matriz[nodo,onde]=1));
  . ativar o grupo como bloco de execução paralelo (PROCESS Gn);

Obs.: Dentro dos blocos de execução, os nodos devem ser executados seqüencialmente (EXEC nodo).

d) **Aninhamento de consultas**:

Na avaliação dos grupos de consultas aninhadas, o embutimento pode ser marcado, juntamente com uma compatibilização dos paralelismos, a partir de uma adaptação do algoritmo original de resolução de dependências dos fluxos. Esta adaptação considera os seguintes procedimentos:

- adicionar uma coluna (matriz[__,m+2]) na matriz de adjacência para assinalar os paralelismos entre ramos do grafo; nesta, coluna nodos de um mesmo ramo devem receber o mesmo número de grupo;
- adicionar outra coluna (matriz[__,m+3]) na matriz de adjacência para assinalar os embutimentos de consultas; nesta coluna, é indicado o número do nodo no qual a consulta deverá ser embutida;
- adicionar uma linha (matriz[n+2,__]) a matriz de adjacência para assinalar nodos de convergência (1), nodos de divergência (2), nodos de convergência/divergência (3) ou nodos seqüenciais (0);
- adicionar outra linha (matriz[n+3,__]) a matriz de adjacência para assinalar os tipos de fluxos como segue:

  TIPO = 1 - entrada;
  2 - saída;
  3 - inclusão;
  4 - exclusão;
  5 - alteração;
  6 - teste de presença;
  7 - teste de ausência;
  8 - controle;

  onde os acessos ao BD são realizados via fluxos de tipos 3 a 7;

- modificar o algoritmo de resolução das dependências dos fluxos para:
  a) identificar nodos de convergência (na linha da matriz possuem mais de uma dependência);

b) identificar nodos de divergência (na coluna da matriz são liberadores de mais de um fluxo dependente deles);
c) identificar nodos de convergência & divergência;
d) dentro da iteração de seqüencialização dos fluxos, marcar os embutimentos como segue:
   - se nodo é um fluxo considerável (tipos 3a7) e nodo não é terminal e não de divergência (é nodo liberador para 1 e só 1 nodo "E") e o pressuposto nodo onde embutir (nodo "E") é um fluxo considerável
     então marcar nodo com o número do nodo "E";
e) dentro da mesma iteração de seqüencialização dos fluxos, marcar grupos de paralelismo como segue:
   - se nodo tem dependências e
     nodo não é de convergência e
     nodo liberador não é de divergência
     então:
       - marcar nodo com o número do grupo liberador (mesmo número do nodo liberador);
     caso contrário:
       - iniciar novo grupo com o nodo
   - para nodos pertencentes a grupos de embutimento, marcar o paralelismo somente quando localizar o nodo cabeça de consulta, identificado por (matriz[__,m+3]= -1) e usando um processo de recursão para marcar demais nodos "filhos" do embutimento.

O algoritmo a seguir especificado mostra, em detalhes, os procedimentos que implementam a combinação de embutimentos de subconsultas com a determinação de paralelismos e é um aperfeiçoamento da versão original do algoritmo. As modificações do algoritmo original estão assinaladas em negrito/itálico.

```
Algoritmo de resolução das dependências dos fluxos:
(versão 4:versão com definição de embutimentos de consultas)
(         combinado com definição de paralelismos           )
1. INÍCIO
1.01 /* identifica nodos de convergência/divergência
1.02 PARA i=1 até n FAÇA:
1.03   INÄCIO
1.04     nrodeph=0;
1.05     PARA j=1 até m FAÇA:
1.06       INÍCIO
1.07         SE matriz[i,j] = 1 ENTÃO nrodeph=nrodeph + 1;
1.08         SE i=0 ENTÃO /*na 1 linha vê também as colunas
1.09            INÍCIO
1.10              nrodepv=0;
1.11              PARA k=1 até n FAÇA:
1.12                SE matriz[k,j] = 1 ENTÃO
1.13                   nrodepv=nrodepv + 1;
1.14              SE nrodepv > 1 ENTÃO
1.15                 SE matriz[n+2,j] não = 0 /* ja converg.
1.16                 ENTÃO matriz[n+2,j] = 3; /* ambos
1.17                 CASO CONTRÁRIO
1.18                      matriz[n+2,j] = 2; /* divergência
1.19            FIM
1.20       FIM
1.21         SE nrodeph > 1 ENTÃO
1.22            SE matriz[n+2,i] não = 0 /* ja diverg.
1.23            ENTÃO matriz[n+2,i] = 3; /* ambos
1.24            CASO CONTRÁRIO
1.25                 matriz[n+2,i] = 1; /* convergência
1.26   FIM
1.27 nrogrupo = 1;
2.   seq=1;
3.   fimreord="FALSO";
4.   ENQUANTO não(fimreord) FAÇA:
5.     INÍCIO
6.       teveseq="FALSO";
7.       PARA i=1 até n FAÇA:
8.         INÍCIO
9.           resolv="VERDADEIRO";
9.01         noE = M; /*inicializa depend. fora dos limites
9.02         onde= M; /*inicializa depend. fora dos limites
10.          SE matriz[i,m+1] = 0 ENTÃO
11.             /* se fluxo ainda não tenha sido seqüenciado
12.             PARA j=1 até m FAÇA:
13.             /* verifica todas dependências do fluxo
13.01              /* Embutimento: busca do nodo dependente
13.02              SE matriz[n+2,i] não = 2 e
13.03                 matriz[n+2,i] não = 3 ENTÃO
13.04                 /* nodo não é de divergência (é depend.
13.05                 /* para zero ou um nodo)
13.06                 PARA k=1 até n FAÇA
13.07                  /*procura onde está seu nodo dependente
13.08                   SE matriz[k,i] = 1 ENTÃO
13.09                      INÍCIO
13.10                        noE=k; /* marca onde é embutível
13.11                        k=n+1; /* se achou, força parada
13.12                      FIM
13.13              /* Paralelismo: busca do nodo liberador
13.14              SE matriz[i,j]=1 e matriz[n+2,i] não = 1
13.15                 e matriz[n+2,i] não = 3  ENTÃO
13.16                 /* tem uma e somente uma dependência
13.17                 onde = j; /* assinala onde está o seu
13.18                           /* nodo liberador
14.                SE matriz[i,j]=1 e matriz[n+1,j]=0 ENTÃO
15.                   /* é dependente e ainda não resolvido
16.                   INÍCIO
17.                     resolv="FALSO";
18.                     j= m+1;  /*se achou, força a parada
19.                   FIM
20.          SE resolv  e matriz[n+1,i] não = 1 ENTÃO
21.             /* se todas dependências do fluxo estão
22.             /* resolvidas mas ainda não assinaladas
```

```
23.                INÍCIO
24.                  matriz[i,m+1]= seq;
24.01                /* marca os embutimentos
24.02                SE matriz[n+3,i] >=3 e  /* é fluxo conside-
24.03                   matriz[n+3,i] <=7 e  /* rável (tipo 3a7)
24.04                   noE não = M e /* é depend.p/só 1 nodo"E"
24.05                   matriz[n+3,noE] >=3 e /* nodo "E" também
24.06                   matriz[n+3,noE] <=7   /* é fluxo consi-
24.07                                         /* derável
24.08                ENTÃO  /* assinala onde embutir
24.09                   INÍCIO
24.10                     matriz[i,m+3]= noE;
24.11                     matriz[noE,m+3]= -1; /* e cabeca de
24.12                   FIM                    /* consulta
24.13                /* marca os grupos de paralelismos
24.14                SE matriz[i,m+3]=-1 ou /*cabeca de consulta
24.15                   matriz[i,m+3]= 0 ou /*nao embutimento
24.16                   noE = M             /*nodo terminal
24.17                ENTÃO  /* marca o paralelismo
24.18                SE onde não = M  e   /* tem dependência
24.19                   matriz[n+2,i] não = 1 e
24.20                   matriz[n+2,i] não = 3 e /* não é conver.
24.21                   matriz[n+2,onde] não = 2 e
24.22                   matriz[n+2,onde] não = 3 e
24.23                                   /* seu nodo liberador
                                        /* não é de divergência
24.24                   matriz[i,m+3] não = -1
                                        /* nao e'cab. de cons.
24.25                ENTAO /* marcação dentro do grupo
24.26                   matriz[i,m+2]= matriz[onde,m+2];
24.27                CASO CONTRÁRIO
24.28                  INÍCIO
24.29                    matriz[i,m+2]= nrogrupo;
                           /* marcação próximo grupo
24.30                    nrogrupo= nrogrupo + 1;
24.31                  FIM
24.32                SE matriz[i,m+3]=-1 e  /*cabeca de consulta
24.33                   matriz[i,m+2] não = 0
                                        /*marcou paralelismo
24.34                  INÍCIO
24.35                    marcafilhos(i,matriz[i,m+2]);
                           /* marca paralelismo p/ nodos filhos
24.36                  FIM
25.                  teveseq="VERDADEIRO";
26.                  ordena[seq]=i;
27.                  seq=seq + 1;
28.                  matriz[n+1,i]=1; /* marca depend. resolvida
29.                FIM
30.          FIM /* fim de uma iteração completa na matriz
31.        SE seq > n   ou  não(teveseq) ENTÃO
32.           /* finaliza se todos fluxos foram processados ou
33.           /* não teve seqüencialização numa iteração
34.           fimreord="VERDADEIRO";
35.      FIM /* fim de seqüencialização dos fluxos
36.    SE não(teveseq) ENTÃO
37.       /* não foi possível seqüencializar todos fluxos
38.       trataerro("ERRO: tem dependência insolúvel");
39. FIM
```

Rotina de marcação de paralelismo no grupo de consulta

```
40.marcafilhos(i,nrogrupo);
41.   PARA j=1 até m FAÇA:
42.     INÍCIO
43.       SE matriz[i,j] = 1 e      /* e' dependente deste
44          matriz[j,m+3] não = 0 /* tem embutimento
45.         INÍCIO
46.            marcafilhos(j,nrogrupo); /* recursão p/ netos
47.            matriz[j,m+2] = nrogrupo;
48.         FIM
```

Obs.: A coluna m+3 da matriz assinala o número do nodo onde embutir a referida consulta. O valor -1, nesta coluna, indica o nodo cabeça de consulta.

Na Figura 31 podem ser vistos os embutimentos marcados na matriz de adjacência durante a seqüência de iterações.

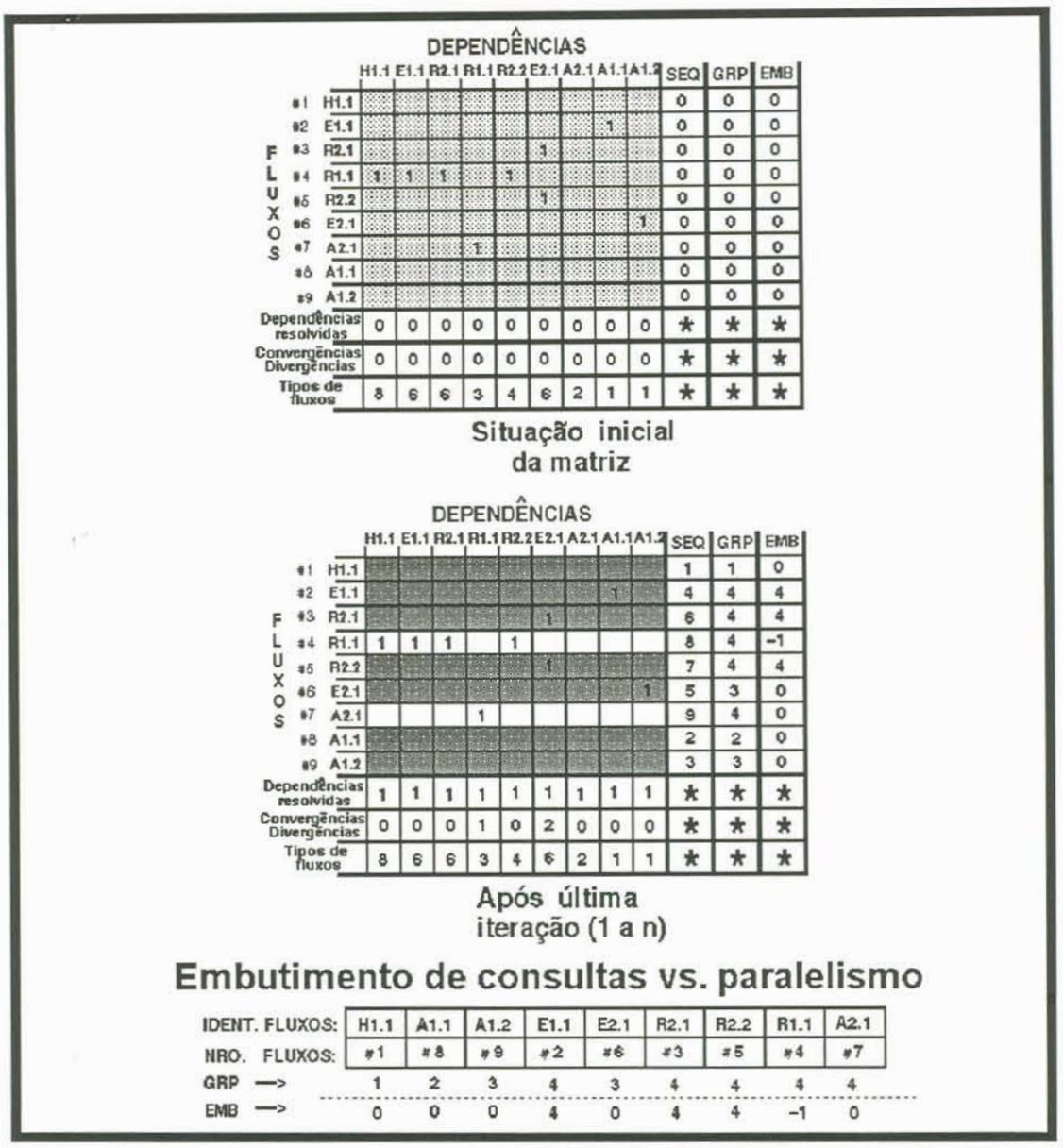


Fig. 31 - Matriz de adjacência para o grafo e a marcação de paralelismos/embutimentos de consultas

Esta última versão do algoritmo incorpora todas as idéias discutidas nas seções anteriores, iniciando com a seqüencialização das operações e passando por determinação de

paralelismos e aninhamento de consultas, até incorporar uma compatibilização final destes conceitos. Portanto, esta é a versão completa do algoritmo que produz a resolução das dependências dos fluxos.

Informática  UFRGS

Um Modelo Não Procedural de Especificação e Implementação Voltado a Sistemas Transacionais em Banco de Dados

Defesa de Tese apresentada aos Senhores:

Prof. Dr. Cirano Iochpe

Prof. Dr. Paulo Cesar Masiero (USP/São Carlos)

Prof. Dr. Ulrich Schiel (UFPb)

Vista e permitida a impressão.
Porto Alegre, 19/10/94.

Prof. Dr. Carlos Alberto Heuser,
Orientador.

Prof. Dr. José Palazzo Moreira de Oliveira,
Coordenador do Curso de Pós-Graduação
em Ciência da Computação.