UNIVERSIDADE-ESTADUAL DE CAMPINAS
INSTITUTO DE MATEMÁTICA, ESTATÍSTICA E
CIÊNCIA DA COMPUTAÇÃO
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

TRANSPARÊNCIA DE MODELOS EM SISTEMAS DE BANCOS DE DADOS HETEROGÊNEOS

por

Ronaldo Lopes de Oliveira

11 de agosto de 1993

BC

NADA: 1241

Ex.

BC/ 17710

261/93

☐ D [X]

CR$ 800,00

02/09/93

PD

M-00048075-2

Transparência de Modelos em Sistemas de Bancos de Dados Heterogêneos

Este exemplar corresponde a redação final da tese devidamente corrigida e defendida pelo Sr. Ronaldo Lopes de Oliveira e aprovada pela Comissão Julgadora.

Campinas, 11 de agosto de 1993.

Prof. Dr. _____

Geovane Cayres Magalhães OK

Dissertação apresentada ao Instituto de Matemática, Estatística e Ciência da Computação, UNICAMP, como requisito parcial para a obtenção do Título de MESTRE em Ciência da Computação.

Banca Examinadora:

Dr. Geovane Cayres Magalhães — DCC–UNICAMP — Orientador

Dra. Claudia Maria Bauzer Medeiros — DCC–UNICAMP

Dr. Marco Antônio Casanova — Centro Científico–IBM–Brasil

Dr. Jacques Wainer — DCC–UNICAMP — Suplente

"O nosso ideal, devemos colocá-lo nas estrelas, ainda que fiquemos no meio do caminho."

Tolstoi

Dedico este trabalho especialmente a minha esposa, Zezinha, e a minha filha, Ana Carolina, que me deram o incentivo para conseguir alcançar mais esse objetivo. Dedico-o também aos meus pais, Urbano e Nadir, que me ensinaram a valorizar os estudos e a lutar pelos meus ideais.

Abstract

Heterogeneous Database Systems (HDBSs) integrate, in a cooperative environment, autonomous and heterogeneous database systems (DBSs). Model transparency in HDBSs is an important property that allows the users to deal with global data using a single model and database language.

This work proposes and discusses solutions to support such property in HDBSs built through the integration of network DBSs and relational DBSs. The solutions presented include methodologies for schema conversion and, architectures and algorithms for command transformation.

The approach used in this work differs from others published in two main points. First, it assumes that each user will manipulate global data using the data model and database language he was supposed to use before the HDBS exist. Second, it proposes mechanisms to support access to HDBS's data through application programs instead of *ad-hoc* transactions.

Sumário

Sistemas de Bancos de Dados Heterogêneos (SBDHs) integram, em um ambiente cooperativo, Sistemas de Bancos de Dados (SBDs) autônomos e heterogêneos entre si, particularmente no que concerne aos modelos de dados utilizados. A transparência de modelos em SBDHs é a propriedade que permite que o usuário visualize as informações segundo um único modelo de dados e manipule esses dados usando uma única linguagem.

Esta dissertação propõe e discute soluções para suportar tal propriedade em SBDHs construídos a partir da integração de SBDs que seguem o modelo relacional ou o modelo rede. Essas soluções incluem metodologias para conversão de esquemas, e arquiteturas e algoritmos para transformação de operações.

A abordagem adotada neste trabalho difere da maioria dos trabalhos publicados em dois aspectos. Primeiro, parte do pressuposto que cada usuário vai manipular os dados do ambiente integrado usando a linguagem associada ao modelo de dados utilizado por ele antes da construção do SBDH. Segundo, propõe soluções para permitir que o usuário manipule os dados do SBDH através de programas de aplicação construídos em uma linguagem de propósito geral e não apenas através de operações *ad-hoc*.

Conteúdo

Lista de Figuras

Capítulo 1

Introdução

1.1 Evolução dos Sistemas de Bancos de Dados

Os chamados **Sistemas de Bancos de Dados (SBDs)** surgiram com o objetivo de prover o armazenamento, acesso e manipulação de dados de uma forma integrada, compartilhada e confiável. De fato, o uso da filosofia de bancos de dados oferece uma série de vantagens em relação aos sistemas tradicionais de arquivos separados por aplicações [Dat86] como: diminuição da redundância de dados com a conseqüente diminuição da probabilidade de inconsistências, compartilhamento dos dados, maior controle sobre a integridade dos dados, maior segurança e maior independência das aplicações em relação à organização física dos dados.

Um SBD é formado por um **Sistema Gerenciador de Banco de Dados (SGBD)** e pela(s) base(s) de dados por ele gerenciada(s). O SGBD é a camada de software responsável pelo controle e operação dos dados armazenados. Em geral um SGBD controla o acesso aos dados armazenados, garante a segurança e proteção dos dados contra operações não autorizadas, gerencia as transações concorrentes e mantém regras que estabelecem restrições de integridade sobre os dados armazenados.

Para representar as informações contidas no mundo real, os SBDs precisam recorrer a modelos lógicos que consigam, através de construtores específicos, capturar, da melhor maneira possível, a semântica das informações e torná-la visível aos usuários. A representação da estrutura lógica dos dados, de acordo com os padrões de um modelo particular, recebe o nome de **esquema**.

Os primeiros modelos de dados utilizados foram modelos orientados a registros, isto é, modelos que estavam fortemente relacionados à estrutura de registro em que os da-

dos são observados como uma seqüência fixa de campos valorados. Os dados nesses modelos assumem uma homogeneidade horizontal e vertical, ou seja, cada tipo de registro tem os mesmos tipos de campos (atributos) e cada campo tem o mesmo tipo de informação em todos os registros [Ken79]. O modelo hierárquico, o modelo rede (ou modelo CODASYL/DBTG[1]) e o modelo relacional são exemplos de modelos orientados a registros.

Apesar de serem utilizados de forma eficiente em uma enorme gama de aplicações, os modelos orientados a registros apresentam certas limitações do ponto de vista semântico [Ken79]. Para tentar superar essas limitações foram propostos alguns modelos que possuem construtores para modelar abstrações mais poderosas dos relacionamentos entre entidades, permitindo uma visão mais natural e realista [HK87]. Os exemplos mais conhecidos de modelos semânticos são o modelo entidade-relacionamento (MER), o modelo funcional e o SDM (semantic data model).

Uma terceira linha de modelos de dados que tem merecido uma atenção muito grande dos pesquisadores em bancos de dados é a dos modelos orientados a objetos (modelos o.o). Na verdade, assim como os modelos semânticos, os modelos o.o pretendem representar de uma forma mais apropriada o mundo real, principalmente no caso de aplicações não convencionais, como CAD/CAM, que não são suportadas eficientemente pelos modelos tradicionais. A grande diferença entre os modelos semânticos e os modelos o.o é que os primeiros enfatizam aspectos estruturais dos objetos enquanto que os últimos se preocupam principalmente com os aspectos comportamentais desses objetos. Alguns conceitos importantes dentro do contexto de modelos o.o são: objetos complexos, identificadores de objetos, encapsulamento, tipos e classes de objetos, hierarquias de tipo e classe e sobreposição e acoplamento tardio de métodos. A descrição desses conceitos pode ser vista em [ABD+89].

Além de estarem evoluindo em relação aos modelos de dados utilizados, os SBDs têm evoluído também em relação a outros parâmetros como a distribuição e a integração dos dados.

Nos primeiros SBDs, tanto o SGBD como as bases de dados por ele gerenciadas se encontravam em uma única localidade física, isto é, em um mesmo computador. Esses sistemas eram baseados em uma arquitetura de três níveis:

1. **nível interno:** corresponde ao nível mais baixo do SBD e está associado à estrutura física dos dados;

[1]Por razões históricas o modelo de dados rede é também conhecido como modelo de dados CODASYL/DBTG, já que o grupo de trabalho CODASYL/DBTG foi o primeiro a definir os construtores e linguagens utilizados no modelo.

2. **nível conceitual:** corresponde ao nível lógico dos dados, isto é, organiza logicamente os dados para representar as informações do mundo real;

3. **nível externo:** corresponde às visões particulares que um usuário ou um grupo de usuários têm do universo lógico representado.

À medida que as pesquisas na área de bancos de dados e redes de comunicação foram se desenvolvendo, começaram a surgir novas idéias e propostas sugerindo a distribuição das bases de dados em vários pontos de uma rede de computadores. Essa distribuição visa aumentar a disponibilidade e a segurança dos dados e melhorar o desempenho das aplicações, aproveitando as vantagens do processamento descentralizado e diminuindo a sobrecarga do sistema como um todo.

Para conseguir tais resultados muitos problemas técnicos oriundos da distribuição e replicação dos dados precisam ser resolvidos. SBDs distribuídos (SBDDs) têm que garantir ao usuário uma série de propriedades [Sto88]: transparência de localização dos dados, transparência de desempenho, transparência de replicação, transparência de transações e transparência de fragmentação. Esta tarefa não é nada simples, justificando o fato de não existirem SBDDs comercialmente disponíveis contemplando todas essas características. Apesar disso, o entendimento desses problemas tem sido cada vez maior, antevendo a proximidade de uma solução ideal.

Mais do que a simples distribuição dos dados, os pesquisadores em banco de dados têm demonstrado um grande interesse na construção de sistemas que possibilitem a interligação lógica de SBDs isolados. Nesse ambiente cooperativo, cada SBD existente recebe o nome de **SBD componente**. A integração dos dados nesses sistemas complexos deve ser feita respeitando a autonomia dos SBDs componentes e permitindo que usuários possam operar sobre os dados do sistema como um todo, a despeito das possíveis diferenças existentes entre os SBDs componentes em relação aos modelos de dados adotados, às linguagens de manipulação dos dados utilizadas, e às características dos seus SGBDs. Várias nomenclaturas diferentes, às vezes conflitantes, têm sido utilizadas na literatura para designar tais sistemas, de acordo com a visão particular que cada autor tem do problema e das suas soluções: *Sistemas de Bancos de Dados Heterogêneos* (SBDHs) [HK89, EP90], *Sistemas de Bancos de Dados Federados* (SBDFs) [HM85, SL90] ou *Multidatabases* [LA86, HB91]. Uma proposta de taxonomia para esses sistemas aparece em [BHP92]. Nesta dissertação será usado o termo Sistemas de Bancos de Dados Heterogêneos (SBDHs).

O interesse em SBDHs se justifica por uma conjunção de fatores tecnológicos e sócio-econômicos. O grande desenvolvimento da área de bancos de dados e a popularização do uso de SBDs em diferentes atividades permitiram a proliferação de SBDs dos mais diferentes tipos no mercado, cada qual identificado com certos tipos de aplicação. Por outro lado, existe uma crescente necessidade de intercâmbio de informações dentro das

empresas e entre empresas diferentes, causada pela complexidade e interdependência das atividades econômicas atuais. Além disso, a interligação de vários sistemas de informação pode ser feita através de redes de comunicação de dados que se tornam cada vez mais rápidas, confiáveis e baratas.

Tanto SBDDs como SBDHs apresentam como característica a **distribuição** geográfica dos dados. Entretanto, existem diferenças marcantes entre esses dois tipos de sistemas. Nos primeiros existe uma distribuição física dos dados, mas o controle sobre esses dados é feito por um único SGBD que monitora todas as bases de dados e transações existentes, sejam elas locais (submetidas por um usuário em um ponto da rede e operando exclusivamente sobre os dados de sua base local) ou globais (transações que envolvem dados não locais). Em outras palavras, pode-se dizer que nos SBDDs existe uma submissão dos processadores de transações e gerenciadores de dados locais a uma autoridade única. Além disso, o SGBD "enxerga" os dados sob uma mesma ótica (mesmo modelo de dados) e gerencia os dados e transações de acordo com os mesmos critérios. Os SBDHs, por outro lado, além da característica da distribuição, têm duas outras dimensões que complicam o quadro: a **heterogeneidade** e a **autonomia** dos SBDs componentes.

As três dimensões existentes nos SBDHs serão vistas na próxima seção.

1.2 Caracterização dos SBDHs: Distribuição, Heterogeneidade e Autonomia

Como foi visto na seção anterior, a distribuição de dados é uma característica comum entre os SBDDs e os SBDHs. Portanto, todos os problemas já existentes no contexto de SBDDs devem ser levados em consideração nos SBDHs.

Em termos ideais, o usuário de um SBDH não deve se preocupar em saber onde o dado está localizado ou qual SBD componente o possui (transparência de localização). Da mesma forma, independentemente do local de origem de uma operação do usuário, o tempo de resposta deve ser praticamente o mesmo (transparência de desempenho). Mais, as transações do usuário envolvendo mais de um SBD componente devem ser atômicas e manter a consistência do sistema como um todo, como se fossem transações locais (transparência de transações). Essas propriedades são difíceis de serem garantidas em SBDDs, mesmo considerando a possibilidade de se distribuir inicialmente os dados de uma forma mais conveniente. No caso de SBDHs, essa dificuldade é ainda maior porque a distribuição dos dados nos SBDs componentes não deve ser mudada (pelo princípio da autonomia dos SBDs componentes) [SL90].

A heterogeneidade dos SBDs componentes pode ser vista sob dois aspectos: diferenças relacionadas aos SGBDs componentes e diferenças relacionadas à semântica dos dados contidos em cada SBD componente [SL90].

Os SGBDs componentes podem ser diferentes em relação ao modelo de dados adotado, às restrições de integridade suportadas, às linguagens de manipulação utilizadas, aos mecanismos para gerenciamento de transações (primitivas, algoritmos para controle de concorrência e recuperação de falhas). Além disso, os SGBDs podem estar localizados em equipamentos distintos, com sistemas operacionais, hardware e protocolos de comunicação diferentes. Por outro lado, mesmo desconsiderando as diferenças concretas dos SGBDs componentes, ainda é preciso lidar com diferenças mais sutis relacionadas às diferentes semânticas dos dados nos diversos SBDs componentes. Essa heterogeneidade se deve principalmente às diferentes, até mesmo conflitantes, percepções que as pessoas têm do mundo real [BLN86]. Outros fatores que contribuem para a heterogeneidade semântica são a variedade de construtores associados aos modelos de dados existentes e a incapacidade desses modelos de capturar toda a semântica existente no mundo real.

A última dimensão que caracteriza os SBDHs é a autonomia dos SBDs componentes. A princípio, os SBDHs devem preservar e garantir quatro níveis de autonomia para os SBDs componentes: autonomia de projeto, autonomia de comunicação, autonomia de associação e autonomia de execução [SL90].

Segundo a autonomia de projeto, o SBD componente tem a liberdade de definir a organização dos seus dados segundo os seus próprios critérios, estabelecendo entre outras coisas o modelo de dados, a linguagem de manipulação dos dados, a interpretação semântica dos dados, as operações suportadas e as características de implementação como, por exemplo, as estruturas de dados e os algoritmos para controle de concorrência. A autonomia de comunicação se refere à capacidade de um SGBD componente de definir quando e como ele vai se comunicar com outros SGBDs componentes. A autonomia de associação permite ao SBD componente decidir quando e como participar de um SBDH. Por último, a autonomia de execução garante aos SBDs componentes a possibilidade de executar as transações sobre os seus dados sem nenhuma interferência ou submissão a controles externos. Dessa forma, o gerenciador de transações locais pode determinar a ordem de execução das transações e abortar transações, independentemente do fato dessas transações fazerem parte de transações globais. Mais do que isso, os SGBDs locais não precisam informar ao gerenciador global a ordem em que são executadas as transações locais que compõem uma transação global.

Todos esses fatos associados à autonomia contribuem para a heterogeneidade do sistema e dificultam a manutenção e operacionalização de um SBDH [PLC91]. No caso de gerenciamento de transações, por exemplo, os algoritmos de controle de concorrência usados tradicionalmente em SBDDs para garantir a atomicidade das transações não po-

dem ser usados diretamente em SBDHs, porque esses algoritmos exigem o monitoramento das transações locais através da troca de informações entre os gerenciadores de transação locais e o gerenciador de transação global. Além disso, protocolos como o protocolo de validação em duas fases exigem que os gerenciadores locais forneçam certas primitivas que podem não ser suportadas por todos os SGBDs componentes [MR91].

Resumindo o que foi visto até aqui, pode-se perceber que os problemas para se construir e manter um SBDH são influenciados por três características fundamentais: distribuição, heterogeneidade e autonomia. A figura 1.1 representa graficamente essas dimensões (quanto maior a distância da origem, maior é a complexidade do problema).

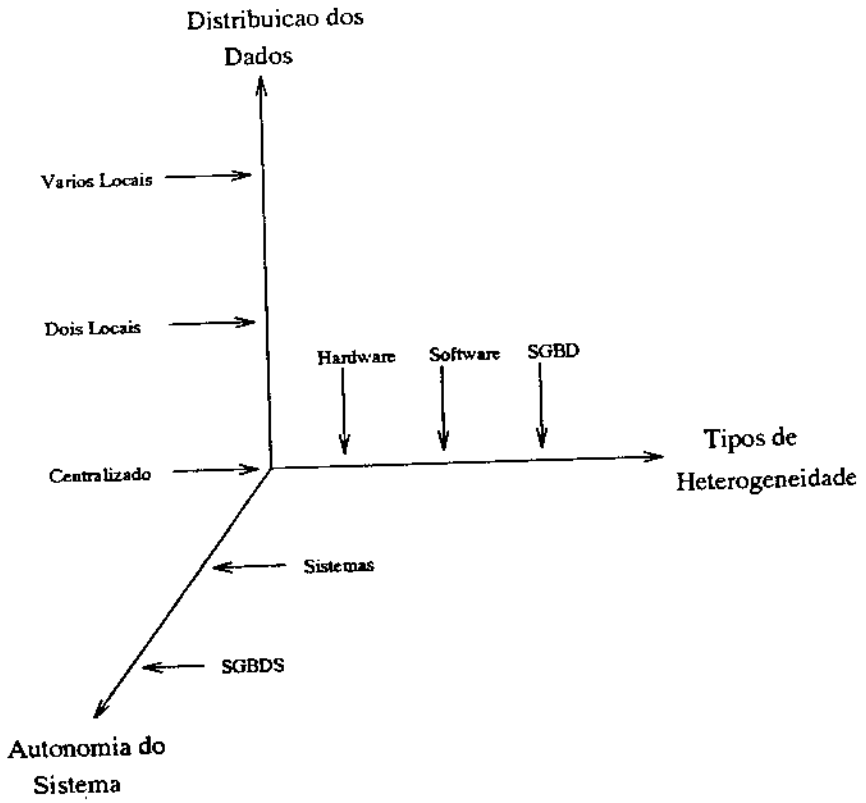

Figura 1.1: Dimensões que caracterizam os SBDHs

Problemas envolvidos com essas três características devem ser resolvidos para se obter SBDHs eficientes e confiáveis. Pode-se dizer que os SBDHs representam uma sofisticação dos SBDDs causada pela heterogeneidade e autonomia dos SBDs componentes. Dificilmente essas características podem ser totalmente suportadas. As pesquisas reali-

zadas mostram que é preciso muitas vezes estabelecer um compromisso entre o ideal e o factível, relaxando certas condições de heterogeneidade e, principalmente, de autonomia [SL90, PLC91].

1.3 Abordagens para construção de SBDHs

Existem basicamente duas abordagens para construção de SBDHs.

A primeira abordagem, ilustrada na figura 1.2, é adotada por pesquisadores como Litwin e Abdellatif [LA87] e argumenta que o principal objetivo a ser perseguido na construção de SBDHs não é a integração de vários SBDs, mas sim, conseguir a sua interoperabilidade. Esse tipo de sistema é conhecido na literatura como **multidatabase** [LA87] ou ainda por **sistema de bancos de dados federados fracamente acoplados** [SL90]. Nesses sistemas a transparência de localização e distribuição dos dados não é tão importante, isto é, o usuário pode estar consciente do fato de que ele está usando dados localizados em vários SBDs distintos. Isto não é problema, desde que ele consiga acessar os dados da forma desejada. A interoperabilidade pode ser conseguida através de **esquemas importados** construídos a partir dos dados que ele deseja obter nos diversos SBDs que compõem o sistema. Esses esquemas são de certa forma análogos às visões em SBDs isolados. Uma maneira de se construir essas visões seria através de operações formuladas em uma linguagem especial que suportasse uma série de funções para tratamento das diferenças semânticas entre os dados dos diversos SBDs componentes. Exemplos desse tipo de linguagem podem ser vistos em [LA87], [CRE87] e [DAT87]. Em [LA87] é descrita a linguagem MDSL usada no sistema MRDSM que incorpora uma série de funcionalidades adicionais como o tratamento de consultas sobre vários bancos de dados e a construção de atributos dinâmicos a partir de atributos já existentes. Em [CRE87] e [DAT87] são apresentadas linguagens especiais, baseadas na álgebra relacional, que possuem operadores para tratamento de incompatibilidades existentes entre os esquemas dos SBDs componentes.

A segunda abordagem defende a construção de SBDHs a partir da integração de esquemas de SBDs já existentes, como mostra a figura 1.3.

Em um processo de negociação, normalmente estática, entre os DBAs[2] locais e o(s) DBA(s) do sistema integrado, são definidos os dados existentes nos SBDs componentes que serão integrados, gerando esquemas globais que poderão ser utilizados por todos os usuários. Sheth e Larson [SL90] chamam esse tipo de sistema de **sistema de bancos de dados federados fortemente acoplados**.

[2] essa sigla será usada para significar administrador de banco de dados

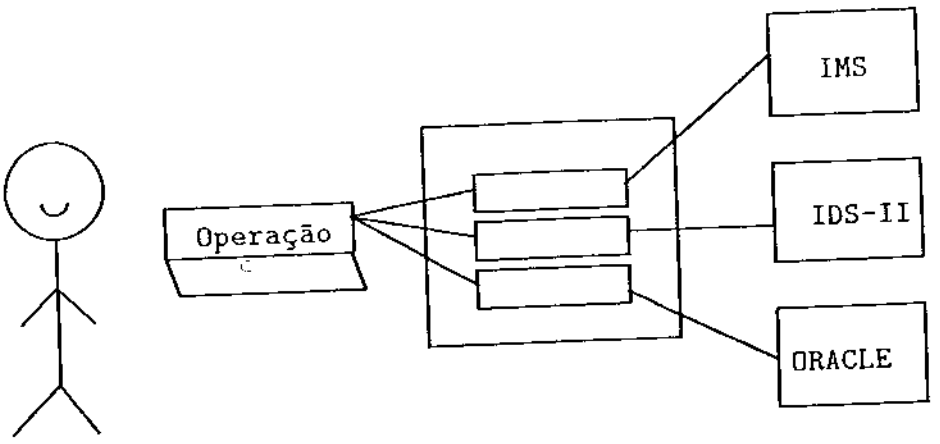

Figura 1.2: Enfoque Multi-BD

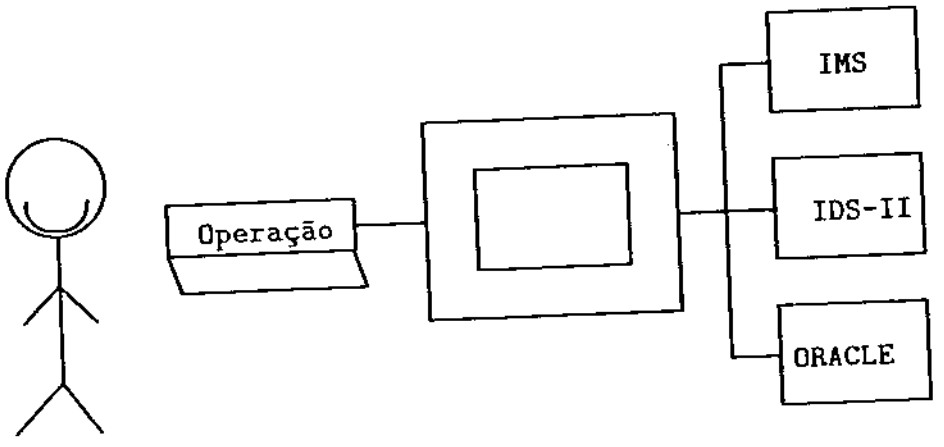

Figura 1.3: Enfoque Integrado

Alguns projetos foram desenvolvidos usando o conceito de SBDs federados fortemente acoplados, como por exemplo, o sistema MERMAID e o sistema MULTIBASE. Uma descrição sucinta das características desses sistemas pode ser encontrada em [TTC+90].

Existe uma grande discussão entre os defensores das duas abordagens com relação às possíveis vantagens e desvantagens de cada uma. Alguns autores afirmam que SBDs federados fortemente acoplados não são uma boa solução porque tornam muito difícil a administração do sistema que é, por sua própria natureza, dinâmico e de grandes dimensões. Esses mesmos autores argumentam ainda que a integração de esquemas é uma tarefa inglória devido aos problemas semânticos envolvidos e por comprometer, em alguns casos críticos, a autonomia dos SBDs componentes [LA87, LMR90]. Desse modo, uma grande vantagem dos SBDs federados fracamente acoplados é a sua capacidade de tratar dados heterogêneos do ponto de vista semântico, de acordo com os parâmetros definidos pelo próprio usuário através de uma linguagem de consulta especial.

Por outro lado, os defensores dos SBDs federados fortemente acoplados afirmam que o outro tipo de abordagem, apesar de interessante para certos tipos de aplicações, é inadequada para as aplicações tradicionais em bancos de dados que existem nas empresas. Essas aplicações exigem controle mais rígido e são normalmente utilizadas por usuários pouco solisticados que dificilmente conseguiriam construir suas próprias visões a partir dos dados dos SBDs componentes. Além disso, SBDs federados fracamente acoplados não conseguiriam suportar adequadamente operações de atualização, porque já que o usuário é livre para definir suas operações de acordo com a sua própria interpretação da semântica dos dados, a integridade dos mesmos ficaria vulnerável à operações mal definidas, ocasionadas pela falta de informação sobre algum dos SBDs componentes ou pela interpretação inadequada da semântica dos dados.

Como se pode observar, ambas as propostas têm pontos positivos e negativos. Entretanto, esta dissertação irá se concentrar na segunda abordagem. A partir desse ponto, SBDs federados fortemente acoplados serão referenciados simplesmente como SBDHs.

1.4 Arquitetura de referência para SBDHs

A arquitetura ANSI/SPARC de três níveis para sistemas de bancos de dados tradicionais não é adequada para SBDHs porque não leva em consideração as características de heterogeneidade, distribuição e autonomia desses sistemas.

Vários projetos de pesquisa [Car87, TBC+87, DL87, LR82] propuseram arquiteturas alternativas para SBDHs. Sheth e Larson [SL90] propuseram uma arquitetura de referência para SBDHs baseada nos pontos comuns que haviam entre várias alternativas de

arquitetura sugeridas na literatura. Essa arquitetura composta de esquemas e processadores é ilustrada na figura 1.4.

O *esquema local* representa o esquema conceitual do SBD componente expresso segundo o modelo de dados usado pelo SGBD local. Portanto diferentes esquemas locais podem estar descritos segundo diferentes modelos de dados.

O *esquema componente* representa o esquema local descrito segundo um modelo de dados comum (MDC). A tradução dos esquemas locais para um único modelo de dados permite a uniformização da representação dos conceitos modelados por cada esquema. Essa uniformidade é essencial para a integração dos esquemas locais formando esquemas conceituais mais abrangentes. O processador de transformação responsável pela tradução de esquemas locais em esquemas componentes gera mapeamentos dos objetos representados nos esquemas locais para os objetos representados nos esquemas componentes. Esses mapeamentos são usados a seu turno para permitir que operações sobre um esquema no modelo de dados comum possam ser transformadas em operações no modelo de dados nativo do SGBD local.

O *esquema exportável* é obtido através de um processador de filtragem que permite o acesso de usuários não locais somente aos dados autorizados pelo(s) DBA(s) do SBD componente. Dessa forma, o esquema exportável representa um subconjunto dos dados existentes no esquema componente. Os esquemas exportáveis e os processadores de filtragem associados contribuem para a manutenção da autonomia de associação dos SBDs componentes que devem decidir quais dados serão compartilhados por eles com usuários não locais.

O *esquema federado* é resultado da integração de esquemas exportáveis. Essa integração é feita por um processador de construção que gera um esquema conceitual virtual a partir dos esquemas exportáveis. A integração de esquemas é um processo bastante complexo devido às divergências estruturais e semânticas que podem existir entre os diversos esquemas a serem integrados [dO92]. Durante a integração dos esquemas, o processador de construção deve gerar mapeamentos que associem cada objeto do esquema integrado com os objetos dos esquemas originais. Dessa maneira, serão geradas informações relativas à distribuição dos dados que permitirão que transações globais envolvendo mais de um SBD componente possam ser desmembradas em transações locais sobre os dados físicos. Da mesma forma, esses mapeamentos serão utilizados na junção do resultado das subtransações enviadas aos SBDs componentes. Portanto, os esquemas federados e os processadores de construção suportam a distribuição dos dados em SBDHs.

O último nível da arquitetura é representado pelos esquemas externos. O *esquema externo* é usado para criar uma visão particular dos dados nos esquemas federados para uma classes de usuários/aplicações. Através dos esquemas externos podem ser criadas

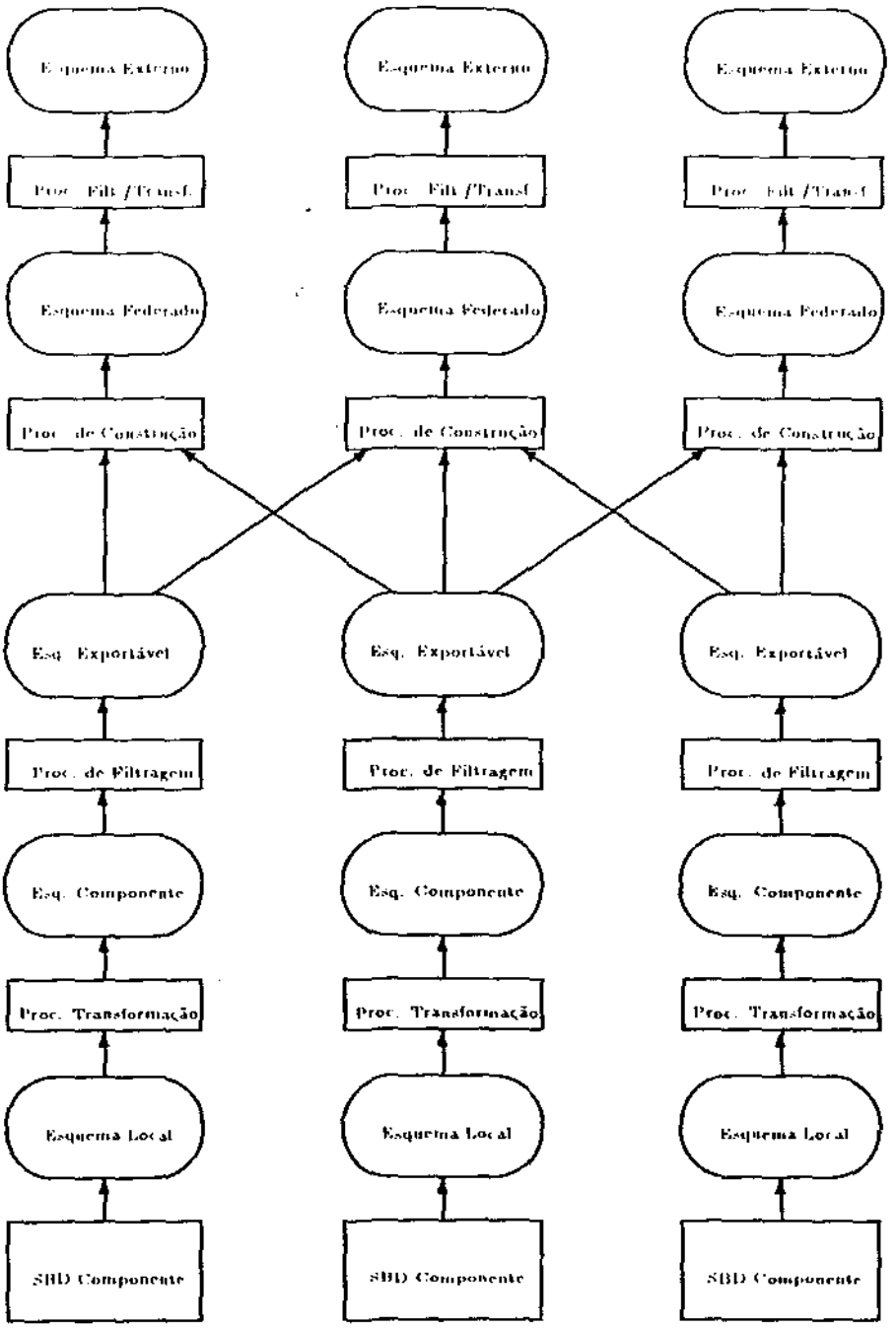

Figura 1.1: Arquitetura de referência para SBDHs

restrições de integridade e segurança adicionais que permitam uma utilização mais específica e controlada dos esquemas federados. O processador de filtragem existente entre o esquema federado e o esquema externo deverá garantir a manutenção dessas restrições através de uma verificação nas operações executadas sobre o esquema externo.

Uma outra utilidade dos esquemas externos é possibilitar que os usuários/aplicações possam ter uma visão do esquema federado construída segundo um modelo de dados diferente do modelo de dados comum. Dessa maneira, os usuários utilizam as linguagens de manipulação de dados já disponíveis nos seus SBDs nativos ao invés de serem obrigados a aprender uma linguagem associada ao MDC. Essa facilidade permitiria uma maior transparência do sistema como um todo, apesar de aumentar sua complexidade devido ao maior número de mapeamentos necessários. Vários projetos em SBDHS [Chu90, ADD+91, BDK+88, TBC+87, DL87, LA86, LR82, DPSM93] adotam a abordagem de acesso uniforme através do MDC por questões de simplicidade e desempenho. Entretanto, alguns trabalhos [Car87, DH87, DH88] defendem o acesso através de diversos modelos e linguagens de manipulação de dados (LMD).

O processador de transformação, responsável pela tradução de esquemas federados no MDC para esquemas externos em outro modelo de dados, deve gerar mapeamentos similares àqueles construídos na tradução de esquemas locais em esquemas componentes. Todavia, os objetivos de uns e de outros são, de certa maneira, contrários. O mapeamento entre esquemas federados e externos visa permitir que operações formuladas sobre modelos de dados diversos possam ser convertidas em operações sobre um mesmo modelo de dados, enquanto que o mapeamento entre esquemas locais e esquemas componentes visa permitir que operações no MDC possam ser transformadas em operações nos modelos de dados dos SBDs componentes.

1.5 Transparência de modelos de dados em SBDHs

O primeiro objetivo que se tem em mente quando se pensa em SBDHs é permitir que um usuário/aplicação opere sobre os dados de SBDs que usam modelos de dados e linguagens de manipulação de dados diferentes. Tudo isso deve ser feito de forma transparente ao usuário.

Esta transparência pode ser conseguida através de processadores que mapeiam esquemas locais para esquemas componentes e que mapeiam esquemas federados para esquemas externos. A tarefa desses processadores pode ser dividida em duas partes. A primeira parte é tipicamente estrutural e objetiva traduzir esquemas expressos em um modelo de dados M1 para esquemas equivalentes em um modelo de dados M2. A segunda parte é essencialmente operacional e utiliza os mapeamentos estruturais para permitir que operações

expressas em uma linguagem de manipulação utilizada no modelo M2 sejam transformadas em operações em uma linguagem de manipulação de dados usada no modelo M1. A figura 1.5, apresentada em [SL90], ilustra essa idéia.

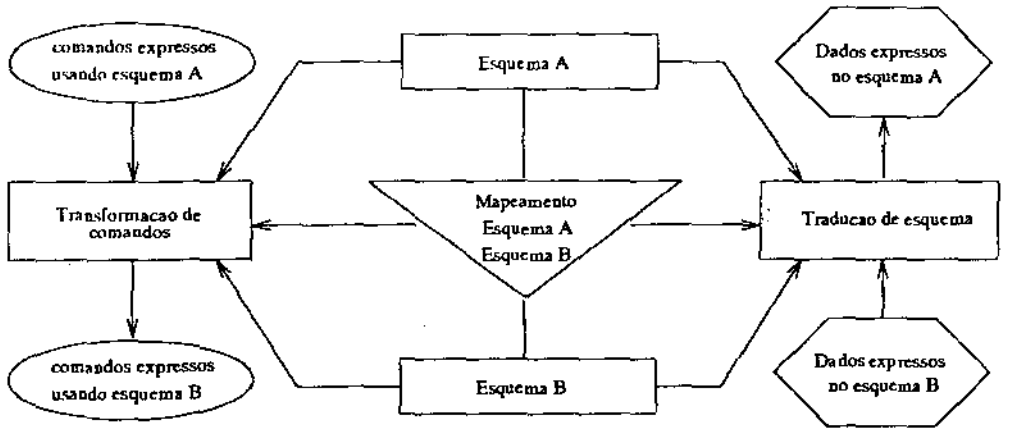

Figura 1.5: Relação entre conversão de esquemas e transformação de comandos

Para mapear os vários modelos de dados que podem existir em um SBDH duas abordagens são possíveis. A primeira abordagem é fazer um mapeamento direto entre cada par de modelos. Alguns trabalhos tratando da tradução direta entre modelos de dados já foram publicados [Zan79, VL80, Lie81, Dem83, Lar83, TYF86]. A segunda abordagem é utilizar um modelo de dados único e uma linguagem associada a esse modelo para servir como interface comum para todos os modelos. No caso de SBDHs, a segunda alternativa é claramente vantajosa em relação à primeira porque permite a uniformização da representação dos dados necessária para a integração de esquemas e também porque diminui o número de mapeamentos entre os modelos. Se existissem N modelos de dados diferentes no SBDH, o mapeamento direto exigiria $N * (N - 1)$ mapeamentos enquanto que o uso de um modelo de dados comum reduziria esse número para $2 * N$.

1.6 O modelo de dados comum

O modelo de dados comum (MDC) é um peça fundamental em um SBDH, e através desse modelo é possível obter a transparência de modelos de dados e a integração dos SBDs componentes. Para que um modelo de dados possa se adequar perfeitamente ao papel de MDC, ele precisa apresentar as seguintes propriedades:

1. Facilidades para representar informações contidas em outros modelos, de forma simples e clara;

2. Capacidade de suportar diferentes níveis de abstração usados nos projetos dos diversos SBDs componentes;

3. Possibilidade de acrescentar informações relevantes não contidas nos esquemas dos SBDs componentes como, por exemplo, restrições de integridade implícitas ou mantidas pelas aplicações;

4. Disponibilidade de uma linguagem de definição e manipulação de dados bem definida e abrangente, que possa ser utilizada como linguagem intermediária no processo de transformação de operações sobre os modelos de dados existentes no SBDH.

Vários projetos de pesquisa em SBDHs [Chu90, BDK+88, LA86, LR82] utilizam como modelo de dados comum o modelo relacional. Todavia, o modelo relacional não possui todas as propriedades citadas anteriormente. A limitação na representação de relacionamentos, de restrições de integridade e de abstrações como generalização e especialização, torna o modelo relacional inadequado para o processo de tradução e integração de esquemas.

Os chamados modelos semânticos [PM88, HK87] são, por outro lado, candidatos mais naturais para servirem como modelo de dados comum, devido à sua natureza tipicamente conceitual e ao seu poder de representação semântica. Apesar disso, poucos projetos em SBDHs [Car87, LR82] utilizam modelos semânticos como MDC.

Ultimamente alguns trabalhos publicados [ADD+91, KBGW91, HR90, KDN90, Man88] têm proposto a utilização de modelos de dados orientados a objetos para servirem como MDC. Uma das vantagens alegadas para a utilização de modelos orientados a objetos é a possibilidade de integrar não somente SBDs convencionais como também SBDs não convencionais, textos e outros tipos de componentes.

1.7 Objetivos e escopo da dissertação

Esta dissertação discute e propõe soluções visando suportar a transparência de modelos de dados em um SBDH formado por SBDs componentes nos modelos relacional e rede. Essas soluções incluem metodologias para tradução de esquemas e arquiteturas para transformação de operações. Ao contrário da maioria dos trabalhos existentes na literatura, não será adotada uma abordagem de acesso uniforme através da linguagem associada ao modelo de dados comum. Ao invés disso, será assumido que os usuários do SBDH

construirão programas de aplicação sobre esquemas externos, utilizando o mesmo modelo de dados do SBD componente em que eles trabalhavam antes da criação do ambiente integrado.

Para não estender em demasia as discussões relativas ao processo de transformação de operações, será assumido que todos os SBDs componentes no modelo relacional utilizam SQL padrão [vdL89] como LMD. Pelo mesmo motivo será usada uma única sintaxe de referência para a LMD rede (aquela descrita em [EN89]), embora reconhecidamente existam pequenas diferenças entre as LMDs usadas pelos SGBDs rede comercialmente disponíveis.

Duas simplificações serão efetuadas sobre a arquitetura de referência descrita na seção 1.4. Primeiro, serão excluídos os esquemas exportáveis, de modo que a integração de esquemas seja feita diretamente sobre os esquemas componentes; segundo, a tradução de esquemas federados para esquemas externos não poderá conter certas modificações estruturais, como, por exemplo, reorganizações ou mudança nos tipos dos dados. Esta última restrição visa simplificar as discussões envolvendo: a) a transformação de operações construídas sobre os esquemas externos; e b) a tradução de esquemas federados para esquemas externos. Dessa maneira, o esquema externo deixa de apresentar as características de uma "visão" do esquema federado, e passa a ser um subconjunto do esquema federado, representado segundo um outro modelo de dados. Com isso, evita-se o tratamento de problemas relacionados com a atualização de visões [SLW88].

O modelo de dados comum e a linguagem intermediária que servirão como referencial para este trabalho são aqueles propostos em [EWH85]. Trata-se do modelo Entidade-Categoria-Relacionamento (modelo ECR) e da linguagem GORDAS (Graph Oriented Data Selection Language) associada a esse modelo. O modelo ECR é uma extensão do modelo Entidade-Relacionamento (MER) proposto por Chen [Che76] e introduz o conceito de categoria: um conjunto de entidades que representam determinado papel em um relacionamento. Por exemplo, entidades que pertençam ao tipo de entidade AUTOMÓVEL podem ser reunidas com entidades que pertençam ao tipo de entidade CAMINHÃO para formar a categoria VEÍCULO que se relaciona com o tipo de entidade PESSOA através do relacionamento PROPRIEDADE. A introdução desse conceito permite a modelagem de conceitos importantes como, por exemplo, generalização e relacionamentos ISA (subclasses). O modelo ECR apresenta também uma definição mais geral de atributos (definição funcional) e uma especificação mais completa da cardinalidade e dos tipos dos relacionamentos, tornando as restrições de integridade estruturais mais claras em comparação com o modelo relacional e o MER. Portanto, a escolha do modelo ECR é natural, uma vez que o MDC é um modelo e como tal deve ser claro e semanticamente rico. Um outro motivo para a escolha desse modelo foi o fato de já existir uma linguagem bem definida associada a ele, que pode ser usada como linguagem intermediária no processo de transformação de

operações entre as linguagens distintas existentes no SBDH.

1.8 Estudo de caso

Para tornar mais claras as soluções propostas nesta dissertação, serão dados exemplos de sua utilização em um SBDH hipotético construído a partir da integração de duas aplicações desenvolvidas isoladamente. A primeira aplicação é um sistema de administração de pessoal, desenvolvido em um SBD rede, e a segunda é um sistema de controle de projetos, desenvolvido em um SBD relacional.

O sistema de pessoal apóia as atividades ligadas à administração de recursos humanos em uma empresa, como contratação de empregados, manutenção de informações cadastrais e pagamento. O diagrama que representa a modelagem dos dados dessa aplicação é mostrado na figura 1.6 e a definição do esquema rede é mostrado nas figuras 1.7, 1.8, 1.9 e 1.10. Além de informações cadastrais dos empregados, são mantidas informações sobre a participação dos mesmos nos projetos desenvolvidos pela empresa. Estas informações são usadas para pagar gratificações aos empregados que participam de projetos. Um empregado pode participar de vários projetos, mas pode gerenciar no máximo um projeto. Em cada projeto podem participar um ou mais empregados, sendo que um deles pode ser o gerente do projeto. O gerente recebe como gratificação o valor equivalente a um percentual sobre o valor total do projeto, e cada empregado que participa de um projeto recebe uma gratificação proporcional ao número de horas trabalhadas, sendo que o valor da hora de trabalho varia de acordo com o projeto.

O sistema de controle de projetos é utilizado para gerenciar a execução e manter informações cadastrais sobre os projetos. Cada projeto é realizado para um cliente, que pode, por sua vez, solicitar vários projetos. Cada projeto é dividido em etapas, e para cada etapa é feito um cronograma contendo o prazo previsto e a data efetiva da sua realização. Os custos de mão-de-obra de um projeto são divididos entre todos os departamentos da empresa que possuem empregados participando desse projeto. A definição do esquema relacional, que representa os dados do sistema de controle de projetos, é apresentada nas figuras 1.11 e 1.12.

1.9 Estrutura da dissertação

Além deste capítulo introdutório, a dissertação terá mais cinco capítulos. No capítulo 2 serão descritos o modelo ECR e a linguagem GORDAS que serão utilizados, respecti-

vamente, como MDC e linguagem intermediária para a conversão de esquemas e transformação de comandos. No capítulo 3 serão apresentadas metodologias para mapeamento de esquemas locais nos modelos relacional e rede em esquemas componentes no modelo ECR. Serão apresentadas ainda metodologias para mapeamento de esquemas federados no modelo ECR em esquemas externos nos modelos rede e relacional. Nos capítulos 4 e 5 serão propostas arquiteturas e algoritmos que permitam o acesso aos dados de um SBDH através de programas de aplicação construídos sobre esquemas externos rede e sobre esquemas externos relacionais. Finalmente, no capítulo 6 serão apresentadas as conclusões deste trabalho, incluindo a sua contribuição real e as sugestões para extensões e trabalhos futuros.

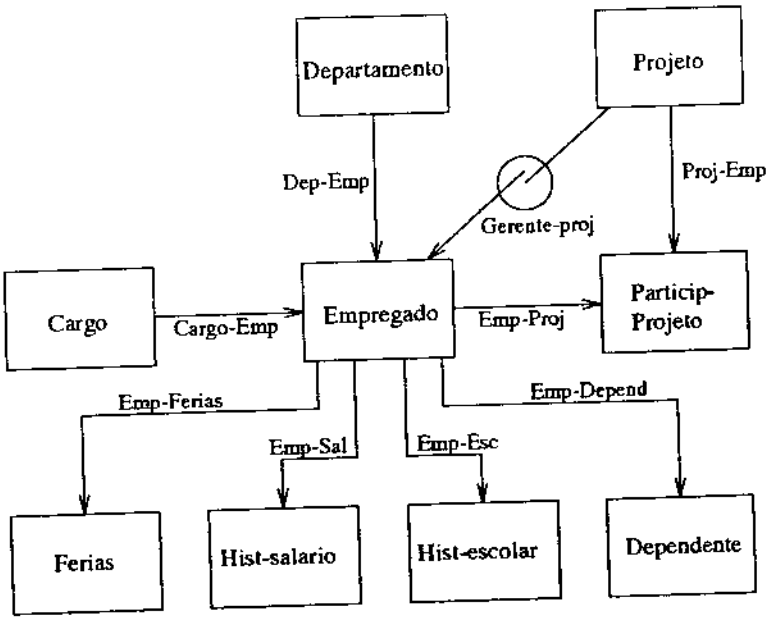

Figura 1.6: Diagrama rede do esquema do sistema de pessoal

```
SCHEMA NAME IS Pessoal.
AREA   NAME IS Area-Pes.

RECORD Empregado
LOCATION MODE CALC USING matricula
DUPLICATES NOT ALLOWED WITHIN Area-Pes.

02 matricula          type is decimal 6.
02 nome-empreg        type is char    40.
02 endereco.
   03 rua             type is char    30.
   03 numero          type is decimal 4.
   03 bairro          type is char    40.
02 telefone           type is decimal 7.
02 sexo               type is char    1.
02 data-nasc          type is decimal 8.
02 est-civil          type is char    1.
02 documentos-pessoais.
   03 cpf             type is decimal 14.
   03 identidade      type is char    15.
   03 tit-eleitor     type is char    15.
   03 doc-reservista  type is char    15.
   03 cart-motorista  type is char    15.
   03 cart-profiss    type is char    15.
02 nacionalidade      type is decimal 2.
02 data-admissao      type is decimal 8.

RECORD Departamento
LOCATION MODE CALC USING cod-depto
DUPLICATES NOT ALLOWED WITHIN Area-Pes.

02 cod-depto          type is char    5.
02 nome-depto         type is char    40.
02 matric-gerente     type is decimal 6.
02 gratif-gerente     type is decimal 13,2.

RECORD Cargo
LOCATION MODE CALC USING cod-cargo
DUPLICATES NOT ALLOWED WITHIN Area-Pes.

02 cod-cargo          type is char    5.
02 nome-cargo         type is char    40.
```

Figura 1.7: Definição do esquema local rede para o sistema de pessoal

```
RECORD Projeto
LOCATION MODE CALC USING cod-projeto
DUPLICATES NOT ALLOWED WITHIN Area-Pes.

02 cod-projeto       type is char    5.
02 nome-projeto      type is char    40.
02 valor-hora        type is decimal 13,2.
02 valor-total       type is decimal 13,2.

RECORD Particip-Projeto
LOCATION MODE VIA Emp-Proj WITHIN Area-Pes.

02 horas-trab        type is decimal  5.

RECORD Dependente
LOCATION MODE VIA Emp-Depend WITHIN Area-Pes.

02 data-nasc-depend  type is decimal 8.
02 cod-gemeos        type is decimal  1.
02 cod-parentesco    type is decimal 1.
02 nome-depend       type is char    40.

RECORD Ferias
LOCATION MODE VIA Emp-Ferias WITHIN Area-Pes.

02 periodo-aquisitivo.
   03 data-inic-per  type is decimal 8.
   03 data-fim-per   type is decimal 8.
02 data-inic-ferias  type is decimal 8.
02 data-fim-ferias   type is decimal 8.

RECORD Hist-salario
LOCATION MODE VIA Emp-Sal WITHIN Area-Pes.

02 data-vigencia     type is decimal 8.
02 valor-salario     type is decimal 13,2.

RECORD Hist-escolar
LOCATION MODE VIA Emp-Esc WITHIN Area-Pes.

02 cod-instrucao     type is decimal 3.
02 descricao-instr   type is char    40.
02 instituicao       type is char    40.
02 data-conclusao    type is decimal 8.
```

Figura 1.8: Definição do esquema local rede para o sistema de pessoal (cont.)

```
SET Gerente-Projeto
OWNER IS Projeto
ORDER IS BY SYSTEM DEFAULT
MEMBER IS Empregado
RETENTION IS OPTIONAL
INSERTION IS MANUAL
SET SELECTION IS BY VALUE OF cod-projeto IN Projeto

SET Dep-Emp
OWNER IS Departamento
ORDER IS LAST
MEMBER IS Empregado
RETENTION IS MANDATORY
INSERTION IS AUTOMATIC
SET SELECTION IS BY VALUE OF cod-depto IN Departamento

SET Cargo-Emp
OWNER IS Cargo
ORDER IS BY SYSTEM DEFAULT
MEMBER IS Empregado
RETENTION IS MANDATORY
INSERTION IS AUTOMATIC
SET SELECTION IS BY VALUE OF cod-cargo IN Cargo

SET Emp-Proj
OWNER IS Empregado
ORDER IS NEXT
MEMBER IS Particip-Projeto
RETENTION IS MANDATORY
INSERTION IS AUTOMATIC
SET SELECTION IS BY APPLICATION

SET Proj-Emp
OWNER IS Projeto
ORDER IS NEXT
MEMBER IS Particip-Projeto
RETENTION IS MANDATORY
INSERTION IS AUTOMATIC
SET SELECTION IS BY APPLICATION
```

Figura 1.9: Definição do esquema local rede para o sistema de pessoal (cont.)

```
SET Emp-Sal
OWNER IS Empregado
ORDER IS SORTED BY DEFINED KEYS DUPLICATES NOT ALLOWED
MEMBER IS Historico-salario
RETENTION IS FIXED
INSERTION IS AUTOMATIC
KEY IS DESCENDING data-vigencia, valor-salario
SET SELECTION IS BY VALUE OF matricula IN Empregado

SET Emp-Depend
OWNER IS Empregado
ORDER IS SORTED BY DEFINED KEYS DUPLICATES NOT ALLOWED
MEMBER IS Dependente
RETENTION IS FIXED
INSERTION IS AUTOMATIC
KEY IS DESCENDING data-nasc-depend, cod-gemeos
SET SELECTION IS BY VALUE OF matricula IN Empregado

SET Emp-Esc
OWNER IS Empregado
ORDER IS SORTED BY DEFINED KEYS DUPLICATES NOT ALLOWED
MEMBER IS Historico-escolar
RETENTION IS FIXED
INSERTION IS AUTOMATIC
KEY IS ASCENDING cod-instrucao
SET SELECTION IS BY VALUE OF matricula IN Empregado

SET Emp-Ferias
OWNER IS Empregado
ORDER IS SORTED BY DEFINED KEYS DUPLICATES NOT ALLOWED
MEMBER IS Ferias
RETENTION IS FIXED
INSERTION IS AUTOMATIC
KEY IS DESCENDING periodo-aquisitivo
SET SELECTION IS BY VALUE OF matricula IN Empregado
```

Figura 1.10: Definição do esquema local rede para o sistema de pessoal (cont.)

```
CREATE TABLE Empregado
      (matricula     DEC  (5)     NOT NULL UNIQUE,
       nome          CHAR (40)    NOT NULL,
       salario       DEC  (13,2) NOT NULL,
       cargo         CHAR (5)     NOT NULL,
       depto         CHAR (5)     NOT NULL)

CREATE TABLE Cargo
      (cod-cargo     CHAR (5)     NOT NULL UNIQUE,
       nome-cargo    CHAR (40)    NOT NULL,
       salario-medio DEC  (13,2) NOT NULL)

CREATE TABLE Depto
      (cod-depto     CHAR (5)     NOT NULL UNIQUE,
       nome          CHAR (40)    NOT NULL)

CREATE TABLE Cliente
      (cod-cliente   DEC  (5)     NOT NULL UNIQUE,
       nome-cliente  CHAR (40)    NOT NULL,
       endereco      CHAR (60)    NOT NULL,
       telefone      DEC  (7))

CREATE TABLE Projeto
      (cod-projeto    CHAR (5)     NOT NULL UNIQUE,
       nome-projeto   CHAR (40)    NOT NULL,
       num-contrato   DEC  (6)     NOT NULL,
       valor-contrato DEC  (13,2) NOT NULL,
       multa-rescisao DEC  (13,2),
       multa-atraso   DEC  (13,2))

CREATE TABLE Contab-Proj
      (cod-depto      CHAR (5)     NOT NULL,
       cod-projeto    CHAR (5)     NOT NULL,
       conta-contab   DEC  (7),
       custo-mao-obra DEC  (13,2) NOT NULL,
       UNIQUE (cod-depto, cod-projeto))
```

Figura 1.11: Definição do esquema local relacional para o sistema de controle de projetos

```
CREATE TABLE Empreg-Proj
      (matric         DEC  (5)     NOT NULL,
       cod-projeto    CHAR (5)     NOT NULL,
       num-etapa      DEC  (2)     NOT NULL,
       horas-trab     DEC  (5)      NOT NULL,
       UNIQUE (matric, cod-projeto, num-etapa))

CREATE TABLE Etapa-Projeto
      (cod-projeto    CHAR (5)     NOT NULL,
       num-etapa      DEC  (2)     NOT NULL,
       descr-etapa    CHAR (40)    NOT NULL,
       data-prev-inic DEC  (8)     NOT NULL,
       data-prev-fim  DEC  (8)     NOT NULL,
       data-inic      DEC  (8),
       data-fim       DEC  (8),
       UNIQUE (cod-projeto, num-etapa))

CREATE TABLE Cliente-Proj
      (cod-cliente    DEC  (5)     NOT NULL,
       cod-projeto    CHAR (5)     NOT NULL,
       UNIQUE (cod-cliente, cod-projeto))

CREATE TABLE Gerente-Proj
      (matric         DEC  (5)     NOT NULL,
       cod-projeto    CHAR (5)     NOT NULL,
       UNIQUE (matric, cod-projeto))
```

Figura 1.12: Definição do esquema local relacional para o sistema de controle de projetos (cont.)

Capítulo 2

O modelo ECR e a linguagem GORDAS

2.1 Modelos semânticos

Os modelos semânticos de dados [HK87, PM88] foram propostos para superar certas deficiências que os modelos tradicionais orientados a registros (modelo hierárquico, modelo rede e modelo relacional) apresentam na modelagem conceitual dos dados, particularmente na modelagem de relacionamentos entre entidades e na modelagem de abstrações importantes como generalização, agregação e classificação.

O modelo semântico que se tornou mais popular foi, sem dúvida, o modelo entidade-relacionamento (MER) proposto por Peter Chen [Che76]. As razões para essa popularidade são a simplicidade e clareza do modelo e a existência de uma representação gráfica bastante atraente. O MER apresenta uma visão lógica do mundo real baseada nos conceitos de **entidade** e **relacionamento**. Uma entidade é algum objeto existente no mundo real que pode ser distinguido de todos os outros objetos por um conjunto de propriedades ou atributos particulares. Uma pessoa ou uma empresa específica são exemplos de entidades. Um relacionamento é uma associação entre entidades particulares. Um exemplo de relacionamento é o vínculo empregatício entre uma empresa X e uma pessoa Y. As entidades que possuem características semelhantes são agrupadas em um **conjunto-entidade (entity-set)**. De maneira similar, os relacionamentos que possuem as mesmas características e são formados a partir dos mesmos conjuntos de entidades constituem um **conjunto-relacionamento (relationship-set)**. As informações sobre as entidades e os relacionamentos são coletadas como atributos, cujos valores individuais são obtidos de conjuntos de valores pré-definidos chamados **valueset**. Em termos formais, um atributo

pode ser definido como uma função que mapeia um conjunto de entidades ou um conjunto de relacionamentos para um conjunto de valores ou para o produto cartesiano de alguns conjuntos de valores. Assim, o atributo *idade* de um conjunto de entidades *Pessoa* mapeia esse conjunto de entidades para um conjunto de valores definidos no conjunto (*value set*) *número-de-anos*. No MER cada entidade é identificada univocamente por um grupo de atributos denominado **chave da entidade (entity-key)** que representa um mapeamento 1:1 (um para um) entre essa entidade e um conjunto de valores.

O MER classifica as entidades em **entidades regulares** e **entidades fracas**. Entidades regulares podem ser identificadas univocamente por um grupo de atributos pertencentes ao conjunto total de seus atributos. Entidades fracas só podem ser identificadas por uma concatenação de atributos pertencentes à própria entidade com um ou mais atributos de outra(s) entidade(s) que se relacionam com ela. Um exemplo de entidade regular seria a entidade *Empregado* identificada pelo atributo *matrícula*. Um exemplo de entidade fraca seria a entidade *Dependente* identificada pelo nome do dependente e pela matrícula do empregado responsável pelo dependente. O relacionamento formado apenas por entidades regulares é chamado **relacionamento regular**. Caso contrário, tem-se um **relacionamento fraco**.

A representação da cardinalidade de relacionamentos e de dependência de existência entre entidades são possíveis no MER. A cardinalidade de um relacionamento entre dois tipos de entidade A e B pode ser 1:1, 1:N ou N:M. A cardinalidade 1:1 indica que cada entidade do tipo A pode estar relacionada com um única entidade do tipo B e vice-versa. A cardinalidade 1:N indica que cada entidade do tipo A pode estar associada com N (N = 0,1,2...) entidades do tipo B e cada entidade do tipo B só pode estar associada a uma única entidade do tipo A. A cardinalidade N:M indica que cada entidade do tipo A ou B pode estar associada com um número arbitrário de entidades do outro tipo. A dependência de existência entre entidades reflete o fato de que uma entidade fraca só pode existir no banco de dados se estiver associada a uma entidade regular.

Apesar de oferecer uma maior capacidade de representação conceitual do que os modelos orientados a registros, o MER possui algumas restrições importantes como, por exemplo, o fato de não suportar diretamente modelagem de abstrações como subclasses (hierarquias ISA) e superclasses (generalização). Esses conceitos, particularmente a generalização, são muito importantes em SBDHs porque permitem conciliar os diferentes níveis de abstração usados nos projetos dos diversos bancos de dados componentes [BL81].

2.2 O modelo Entidade-Categoria-Relacionamento (ECR)

O Modelo ECR [EWH85] é um dos vários modelos propostos para estender a capacidade de modelagem do MER, mantendo as suas vantagens originais. Além de utilizar os conceitos de entidade e relacionamento do MER, o Modelo ECR introduz o conceito de categoria, que permite modelar grupos lógicos de entidades pertencentes a tipos de entidades distintas mas que desempenham um mesmo papel em um relacionamento. Os conceitos e características principais do Modelo ECR serão descritos a seguir.

2.2.1 Entidade, Tipo de Entidade e Categoria

Os conceitos de **entidade** e **tipo de entidade** no Modelo ECR são similares aos conceitos de entidade e conjunto de entidades no MER. Os tipos de entidades são conjuntos disjuntos na medida que uma entidade só pode pertencer a um único tipo de entidade.

Uma **categoria** é um grupo de entidades pertencentes a um ou mais tipos de entidade que desempenham um mesmo papel (função lógica) em um relacionamento. Esse relacionamento pode estar explícita ou implicitamente representado no esquema ECR. Um exemplo de relacionamento implicitamente representado em uma categoria é o relacionamento de vínculo empregatício com uma empresa que existe nas entidades pertencentes à categoria *Empregado*. Em termos formais, uma categoria é uma relação matemática assim definida:

$$C_i = T_1[S_1] \cup T_2[S_2] \cup T_3[S_3] \ldots \cup T_n[S_n] \text{ onde:}$$

C_i = categoria : $1 \leq i \leq n$; T_j = tipo de entidade : $1 \leq j \leq n$;
S_j = predicado de seleção que especifica as entidades de T_j que farão parte da categoria;

As categorias, ao contrário dos tipos de entidades, não são necessariamente disjuntas visto que uma entidade pode participar em várias categorias diferentes. Uma categoria pode, inclusive, ser definida em termos de outras categorias.

A introdução do conceito de categoria no Modelo ECR permite a modelagem explícita de duas abstrações importantes: generalizações e hierarquias ISA. Na figura 2.1 são mostrados um exemplo de categoria de generalização (*Conveniado*) e dois exemplos de categorias ISA (*Emp-Técnico* e *Emp-Administ*).

Tanto o conceito de tipo de entidade como o conceito de categoria são usados para representar conjuntos de entidades do mundo real. Observando-se a definição formal de

categoria pode-se perceber que uma categoria pode se confundir com um tipo de entidade (quando j = 1 e não existe predicado de seleção restritivo). Então é possível usar o conceito de categoria para se definir de forma mais abrangente o conceito de relacionamento.

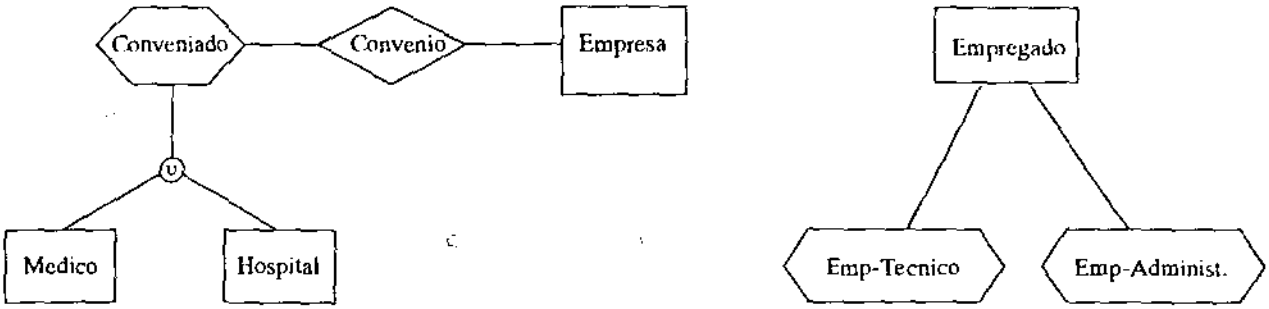

Figura 2.1: Exemplos de categorias: generalização e ISA

2.2.2 Relacionamentos

Um relacionamento é uma associação de entidades. Formalmente um relacionamento R é um subconjunto do produto cartesiano de n categorias, não necessariamente distintas:

$$R \subseteq X(C_1, C_2, ..., C_n) = \{(c_1, c_2, ..., c_n): c_j \in C_j \text{ para } 1 \leq j \leq n\} \text{ onde:}$$

c_j = entidade; C_j = categoria; X = produto cartesiano.

Um elemento de R (uma tupla) é denominado de **instância do relacionamento.**

As propriedades estruturais de um relacionamento são as regras que governam as diversas maneiras que entidades de uma categoria podem participar em um relacionamento. O Modelo ECR especifica a participação de uma categoria C_j em um relacionamento R através de um par de números inteiros: (i_1, i_2), $0 \leq i_1 \leq i_2$ e $i_2 > 0$. Esses números indicam que cada entidade pertencente à categoria C_j pode participar em, no mínimo, i_1 e em, no máximo, i_2 instâncias do relacionamento R. Utilizando-se esse par de números podem ser diferenciados os seguintes relacionamentos: total, parcial, específico e funcional.

Um relacionamento é **total** em relação à participação de C_j se $i_1 \geq 1$ e **parcial** se $i_1 = 0$. A participação total significa que uma entidade de C_j deve estar participando obrigatoriamente de uma instância do relacionamento em qualquer estado válido

do banco de dados. A participação parcial de C_j implica na possibilidade de existir uma entidade pertencente a essa categoria que não esteja participando de nenhuma instância do relacionamento R. Exemplos de relacionamento total e parcial podem ser vistos na figura 2.2. Uma entidade pertencente à categoria *Empregado* só pode existir no banco de dados se estiver ligada a uma entidade da categoria *Departamento* por uma instância do relacionamento *Dep-Emp*. Em outras palavras, todo empregado deve estar lotado em um departamento. Por outro lado, o relacionamento é parcial em relação às entidades da categoria *Departamento*, já que podem existir departamentos no banco de dados sem nenhum empregado.

Figura 2.2: Exemplo de participação total (*Empregado*) e parcial (*Departamento*) em um relacionamento

Um relacionamento é **específico** em relação à participação de C_j se ele é total, e, se uma entidade de C_j, uma vez ligada a uma instância de R, não puder ser removida dessa instância a menos que a própria entidade seja excluída do banco de dados. Um exemplo de relacionamento específico pode ser visto na figura 2.3. Uma entidade pertencente à categoria *Dependente*, uma vez ligada a uma entidade da categoria *Empregado* por uma instância do relacionamento *Emp-Depend*, não poderá ser transferida para outra instância do relacionamento. Dessa maneira, a exclusão de uma entidade da categoria *Empregado*, implicará na exclusão obrigatória das entidades associadas pertencentes à categoria *Dependente*.

Figura 2.3: Exemplo de participação específica (*Dependente*) em relacionamento

Um relacionamento é **funcional** em relação à participação de C_j se cada entidade dessa categoria aparece em no máximo uma instância do relacionamento R para cada estado do banco de dados. Um exemplo de relacionamento funcional pode ser visto na figura 2.1. Cada entidade pertencente à categoria *Empregado* só pode aparecer em uma única instância do relacionamento *Particip-Proj*, que liga entidades dessa categoria com entidades da categoria *Projeto*. Em outras palavras, cada empregado participa em no máximo um projeto.

As especificações das propriedades estruturais dos relacionamentos no Modelo ECR são equivalentes às especificações de restrições de cardinalidade e dependência encontradas em outros modelos como o MER, o modelo relacional e o modelo rede. A participação total define uma dependência de existência (referencial) entre entidades. A participação específica define uma restrição de exclusão sobre as instâncias de um relacionamento. A participação funcional restringe a cardinalidade do relacionamento para 1:N ou 1:1. Os valores assumidos para os números i_1 e i_2, na ausência da declaração explícita desses valores, são $i_1 = 0$ e $i_2 = \infty$ significando que não existem restrições sobre a participação da categoria no relacionamento em questão.

Figura 2.4: Exemplo de participação funcional (*Projeto*) em relacionamento

O Modelo ECR também é capaz de representar auto-relacionamentos em que uma única categoria desempenha múltiplos papéis em um único relacionamento. Um exemplo de auto-relacionamento é apresentado na figura 2.5 onde entidades da categoria *Empregado* estão associadas a outras entidades da mesma categoria através do relacionamento *Hierarquia*. O leitor pode perceber que a categoria desempenha dois papéis, *gerente* e *gerenciado*, indicando que um determinado empregado que participa do relacionamento em um papel está ligado a outros empregados que desempenham o outro papel.

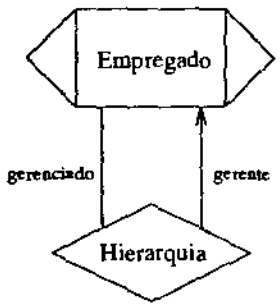

Figura 2.5: Exemplo de auto-relacionamento

2.2.3 Atributos

Entidades e relacionamentos possuem atributos que descrevem suas características e propriedades. Os atributos de uma entidade podem ser **básicos** ou **adquiridos**. Atributos

básicos são aqueles inerentes à entidade, enquanto que atributos adquiridos são aqueles que representam propriedades/características associadas à entidade como resultado de um processo de abstração, em que um relacionamento envolvendo aquela entidade foi omitido da descrição do banco de dados. Um exemplo de atributo básico é o atributo *sexo* de uma entidade *Pessoa*. Um exemplo de atributo adquirido é o atributo *matrícula* de uma entidade *Empregado*, que pode ser considerado como um atributo do relacionamento *vínculo-empregatício* que existe implicitamente no banco de dados. A diferenciação entre atributos básicos e adquiridos é importante no caso da categorização. Categorias formadas a partir de um subconjunto das entidades de um tipo de entidade básico podem possuir certos atributos que não se aplicam a todas as entidades pertencentes ao tipo de entidade básico.

Um atributo a é definido formalmente como uma função com domínio definido em um tipo de entidade T, categoria C ou relacionamento R e com imagem definida em um conjunto de valores (*valueset*) V ou no produto cartesiano de n conjuntos de valores:

$$a : T \rightarrow P(V) \text{ ou } a : C \rightarrow P(V) \text{ ou } a : R \rightarrow P(V) \text{ onde:}$$

P(V) é o conjunto potencial de valores;

Atributos definidos sobre um único conjunto de valores são chamados **atributos simples** e atributos definidos sobre o produto cartesiano de conjuntos de valores são chamados **atributos compostos**.

Restrições de cardinalidade e unicidade podem ser impostas sobre os atributos para representar certas situações do mundo real. A cardinalidade de um atributo determina o número de valores, obtidos de um conjunto de valores V, que podem ser associados a esse atributo. No Modelo ECR, a restrição de cardinalidade é especificada por um par de inteiros (i_1, i_2), onde $0 \leq i_1 \leq i_2$ e $i_2 > 0$. O número i_1 indica o número mínimo de valores e i_2 o número máximo. Usando-se esse par de inteiros podem ser distinguidos os seguintes tipos de atributos: **monovalorado** ($i_2 = 1$), **multivalorado** ($i_2 > 1$), **total** ($i_1 \geq 1$) e **parcial** ($i_1 = 0$). O valor default para i_1 e i_2 é 1, definindo um atributo monovalorado e total.

A unicidade de atributos é usada para especificar que determinados atributos identificam univocamente uma entidade. Um atributo a é **único (ou chave)** se cada valor v de $P(V)$ aparece como um valor de a em no máximo uma entidade pertencente ao domínio, para cada estado válido do banco de dados. Ao contrário do MER, o Modelo ECR não exige que um tipo de entidade possua um conjunto de atributos chave já que entidades com atributos contendo o mesmo valor podem ser diferenciadas pelos seus relacionamentos com outras entidades.

O conjunto de atributos de uma categoria é a união dos atributos básicos de todos os tipos de entidade participantes com os atributos que foram especialmente definidos para a categoria (atributos adquiridos). A definição dos atributos de uma categoria deve levar em consideração o fato de que um atributo a_i, pertencente a um determinado tipo de entidade, pode não estar definido para os outros tipos de entidade que formam a categoria. Por isso, o Modelo ECR define os atributos da categoria (a_c) da seguinte forma:

$$a_c = \begin{cases} a_i & \text{para aquelas entidades em que } a_i \text{ é definido} \\ nulo & \text{para outras entidades da categoria} \end{cases}$$

É importante ressaltar que não existe redundância de dados no Modelo ECR. Os atributos dos tipos de entidade sobre os quais é definido uma categoria não são duplicados na categoria e sim herdados dos tipos de entidade básicos. No exemplo da figura 2.1 tanto a categoria *Emp-Técnico* como a categoria *Emp-Administ* herdam do tipo de entidade *Empregado* atributos básicos como matrícula, nome e endereço.

2.2.4 Diagramas ECR

Assim como o MER, o Modelo ECR possui uma representação diagramática dos construtores utilizados na modelagem dos dados.

Tipos de entidade e relacionamentos são representados, respectivamente, por um retângulo e por um losângulo. Uma categoria é representada por um hexágono. Quando uma categoria é também um tipo de entidade, o hexágono é sobreposto ao retângulo que representa o tipo de entidade. As figuras 2.1 a 2.5 ilustram o uso desses símbolos no Modelo ECR.

Categorias são representadas como na figura 2.1. Casos particulares de categoria, como categorias ISA disjuntas, podem também ser representadas nos diagramas ECR, como mostra a figura 2.6.

Atributos são representados pelos seus nomes ligados por um arco[1] ao tipo de entidade, categoria ou relacionamento a que pertencem.

Os diferentes tipos de participação das categorias nos relacionamentos podem ser distinguidos nos diagramas ECR[2], como mostram as figuras 2.2, 2.3, 2.4 e 2.5 . Relacionamentos sem restrição com relação à participação da categoria ($i_1 = 0$ e $i_2 = \infty$) são

[1] Em [EWH85] os nomes dos atributos são envolvidos por formas ovais. Entretanto para tornar os diagramas menos "carregados" preferimos mudar ligeiramente essa representação.

[2] A representação dos relacionamentos apresentadas neste texto difere um pouco da representação usada em [EWH85] por questões de clareza e concisão dos diagramas.

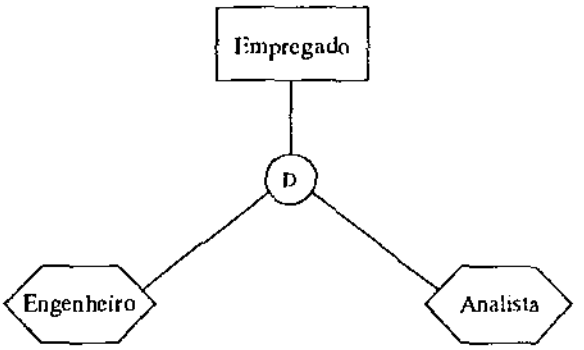

Figura 2.6: Diagrama ECR representando uma categoria ISA disjunta

representados com uma linha simples entre a categoria e o relacionamento em que a categoria participa. Um relacionamento total ($i_1 = 1$) é representado por uma linha com um ponto na extremidade que toca o losângulo que representa o relacionamento. Um relacionamento específico é representado por uma linha com um ponto em ambas as extremidades. Um relacionamento funcional é representado por uma linha com uma seta na extremidade em que está localizada a categoria que é funcionalmente determinada. Relacionamentos parcial e funcional, total e funcional e específico e funcional são representados usando combinações das convenções já citadas.

Valores particulares de i_1 e i_2, quando utilizados, são mostrados próximos da linha que indica a participação da entidade. A figura 2.7, por exemplo, apresenta uma restrição de cardinalidade que indica que um departamento pode ter no máximo vinte empregados.

Figura 2.7: Diagrama de relacionamento com cardinalidade máxima representada

2.3 A linguagem GORDAS

A linguagem GORDAS[3] é utilizada como linguagem de definição e manipulação de dados no Modelo ECR.

A definição formal dessa linguagem é baseada na representação do modelo através de grafos orientados e rotulados [EW83]. Um esquema ECR é representado por um

[3]acrônimo para Graph Oriented Data Selection

grafo em que cada vértice representa um tipo de entidade (vértice-TE), uma categoria (vértice-CA) ou um relacionamento (vértice-RE) e cada aresta A conecta um vértice-TE/CA a um vértice-RE, indicando a participação do tipo de entidade ou categoria naquele relacionamento. O rótulo de cada vértice contém o nome do tipo de entidade ou categoria ou relacionamento representado, o tipo do vértice (RE, CA ou TE) e os atributos com seus respectivos *valuesets*. As arestas são rotuladas com dois nomes que servem para referenciar o relacionamento a partir do tipo de entidade/categoria (*nome-1*) e para referenciar o tipo de entidade/categoria a partir do relacionamento (*nome-2*). Além desses nomes, os rótulos das arestas contém dois números inteiros i_1 e i_2 que definem as restrições de cardinalidade e dependência de existência já descritas na seção 2.2.2.

Da mesma forma como um esquema ECR é representado pelo **grafo de esquema** (GE), um banco de dados ECR é definido como um grafo orientado e rotulado denominado **grafo de banco de dados** (GBD). O GBD é uma representação abstrata do banco de dados real formado por entidades individuais e instâncias dos relacionamentos. Portanto, o GBD é uma instância de um GE particular e existe uma correspondência entre os vértices e arestas dos dois grafos.

Uma operação GORDAS nada mais é que a definição de um caminho no GE que gera, a seu turno, caminhos no GBD. Para definição desses caminhos são fundamentais os nomes existentes nos rótulos das arestas porque eles estabelecem ligações tanto no sentido entidade-relacionamento como no sentido relacionamento-entidade. Os nomes dos rótulos das arestas são conhecidos no Modelo ECR como **nomes de conexão**. São os nomes de conexão que possibilitam a referência direta entre entidades pertencentes às diferentes categorias envolvidas nos relacionamentos.

A figura 2.8 mostra um diagrama ECR contendo os nomes de conexão associados aos relacionamentos. Os nomes de conexão podem ser vistos como nomes "invertidos" dos papéis que uma categoria desempenha em um relacionamento. Desse modo, o nome de conexão *empregados* associado ao relacionamento *Dep-Emp* permite uma referência do tipo *empregados of Departamento*. De maneira similar, o nome de conexão *departamento* permite referenciar entidades pertencentes ao tipo de entidade *Departamento* a partir de entidades do tipo de entidade *Empregado*, através da expressão *departamento of Empregado*.

Nas seções seguintes serão descritas as principais funcionalidades suportadas pela linguagem GORDAS.

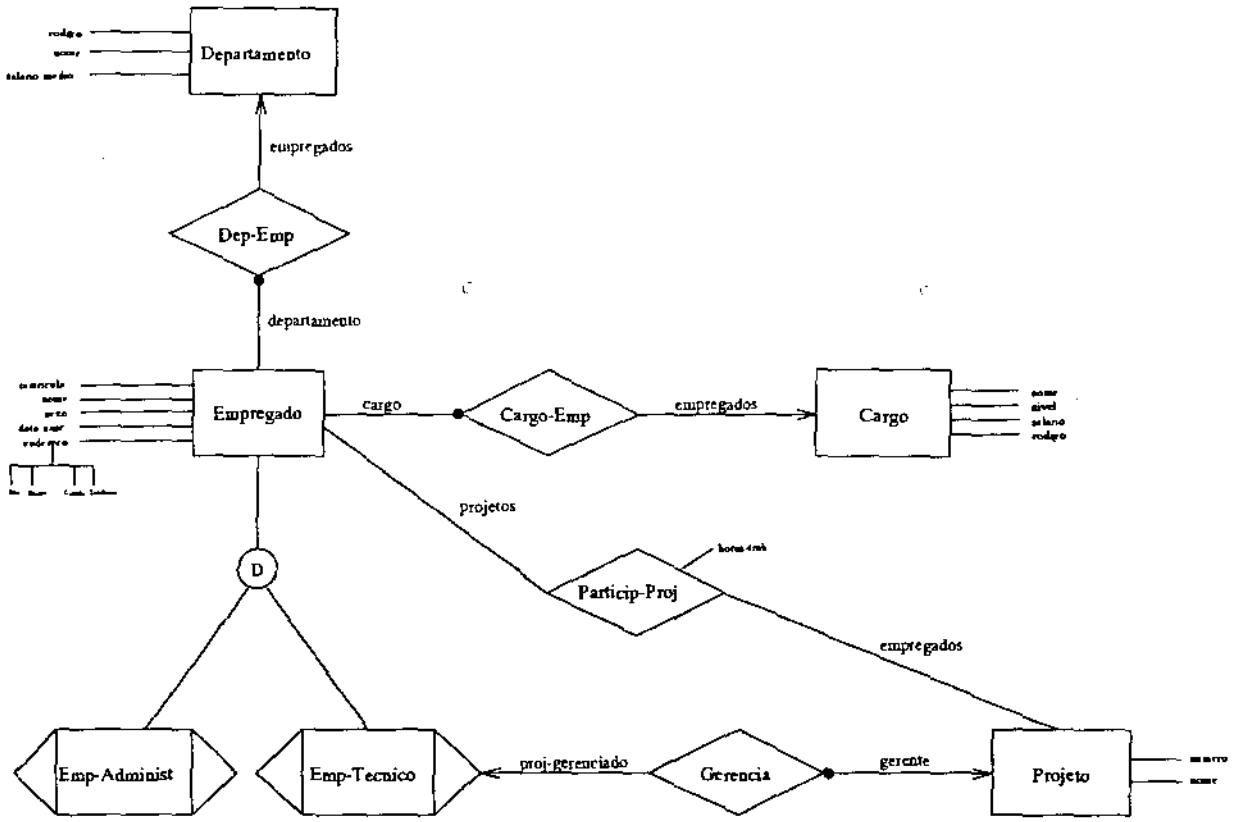

Figura 2.8: Exemplos de diagrama ECR com nomes de conexão

```
a.  DEFINE VALUESET data-nasc  AS DATE 1932:1978.
b.  DEFINE VALUESET nome       AS STRING 40.
c.  DEFINE VALUESET codigo     AS INTEGER RANGE 1:9999.
d.  DEFINE VALUESET sexo       AS ''masc'', ''fem''.
```

Figura 2.9: Exemplos de declaração de valuesets em GORDAS

2.3.1 Definição de Dados

A linguagem GORDAS permite a declaração de todos os construtores utilizados pelo Modelo ECR: conjunto de valores (*valueset*), tipo de entidade, categoria e relacionamento.

Na declaração de um conjunto de valores o usuário pode escolher entre um conjunto de valores pré-determinados ou um conjunto padrão (*Integer, String, Real, Date*). Também é possível utilizar um conjunto de valores já definido para outros atributos. No conjunto de valores padrão, o usuário pode incluir restrições sobre os valores permitidos como, por exemplo, limites superior e inferior para o caso de inteiros, reais e datas. A figura 2.9 mostra a definição de alguns conjuntos de valores obtidos a partir do exemplo da figura 2.8.

Na definição de um tipo de entidade, além do seu nome também são declarados os seus atributos. Os atributos podem ser declarados como *únicos* (*chaves*) ou como atributos *compostos* por outros atributos. Além disso, podem ser declarados para os atributos uma cardinalidade mínima e uma cardinalidade máxima que informam se o atributo é monovalorado ou multivalorado (cardinalidade máxima igual ou maior do que 1), e se o atributo é total ou parcial (cardinalidade mínima igual ou maior do que 0). O valor *default* tanto para a cardinalidade mínima como para a cardinalidade máxima é um. A figura 2.10 mostra a definição do tipo de entidade *Empregado* da figura 2.8. O atributo *endereço* é composto de outros atributos: *rua, bairro, cidade* e *telefones*. Os valores mínimo e máximo do atributo *telefones* indicam que o empregado pode ter de 0 a 3 telefones de contato.

```
DEFINE  ENTITY TYPE  Empregado
        ATTRIBUTES   matricula  VALUESET  codigo      KEY,
                     nome       VALUESET  nome,
                     data-nasc  VALUESET  data-nasc,
                     sexo       VALUESET  sexo,
                     endereco   COMPOUND  (rua         VALUESET nome,
                                           bairro      VALUESET bairro,
                                           cidade      VALUESET nome,
                                           telefones   VALUESET fone
                                                       MINIMUM VALUES 0,
                                                       MAXIMUM VALUES 3).
```

Figura 2.10: Exemplo de declaração de tipo de entidade em GORDAS

Na declaração de uma categoria são incluídos, além do seu nome, o(s) nome(s) do(s)

tipo(s) de entidade sobre o(s) qual(is) a categoria é formada, o predicado para seleção das entidades que formarão a categoria e os atributos específicos da categoria. A figura 2.11 ilustra a definição das categorias ISA obtidas a partir do tipo de entidade *Empregado*. O leitor deve observar o uso dos nomes de conexão nos predicados de seleção. A categoria *Emp-Técnico* contém os empregados cujo código do cargo é igual a 1111, 2222 ou 3333 e a categoria *Emp-Administ* contém os empregados que ocupam outros cargos.

```
a.  DEFINE  CATEGORY Emp-Tecnico FROM Empregado
            (codigo OF cargo OF Empregado INCLUDES {1111,2222,3333}).
b.  DEFINE  CATEGORY Emp-Adminis FROM Empregado
            (codigo OF cargo OF Empregado NOT INCLUDES {1111,2222,3333}).
```

Figura 2.11: Exemplos de declaração de categoria em GORDAS

Na declaração de um relacionamento são definidos o nome do relacionamento, os tipos de entidade e/ou categorias que participam desse relacionamento, as restrições impostas sobre a participação das entidades e os nomes de conexão. A figura 2.12 apresenta a definição dos relacionamentos *Dep-Emp* e *Gerência* da figura 2.8. A participação de entidades do tipo *Empregado* no relacionamento *Dep-Emp* é total e também funcional. Como a participação de entidades do tipo *Departamento* não é funcional, o relacionamento *Dep-Emp* é 1:N, isto é, um departamento para n empregados. A participação de entidades da categoria *Emp-Técnico* no relacionamento *Gerência* é parcial e funcional e a participação do tipo de entidade *Projeto* nesse relacionamento é total e funcional. Portanto, *Gerência* é um relacionamento 1:1.

```
a. DEFINE RELATIONSHIP Dep-Emp FROM
          Empregado   (departamento, empregados) MINIMUM PARTICIPATION 1
                                                 MAXIMUM PARTICIPATION 1,
          Departamento (empregados, departamento).

b. DEFINE RELATIONSHIP Gerencia FROM
          Emp-Tecnico (proj-gerenciado, gerente) MINIMUM PARTICIPATION 0
                                                 MAXIMUM PARTICIPATION 1,
          Projeto     (gerente, proj-gerenciado) MINIMUM PARTICIPATION 1
                                                 MAXIMUM PARTICIPATION 1.
```

Figura 2.12: Exemplos de declaração de relacionamento em GORDAS

2.3.2 Definição de Restrições de Integridade Explícitas

A linguagem GORDAS suporta a definição de restrições de integridade adicionais além daquelas intrínsecas ao Modelo ECR. O usuário pode definir restrições sobre determinados tipos de entidade, categorias ou relacionamentos através de uma expressão lógica que representa uma condição a ser satisfeita. Também é possível definir gatilhos que disparam certas ações quando determinada condição não é satisfeita.

A figura 2.13 ilustra a definição de uma restrição de integridade explícita sobre o relacionamento *Gerência* da figura 2.8. A restrição informa que um empregado só pode ser gerente de um projeto se o seu cargo for de nível superior. O leitor deve perceber novamente o uso de nomes de conexão e os mecanismos de herança que permitem referenciar-se o relacionamento de entidades do tipo *Empregado* com entidades do tipo *Cargo* através de entidades da categoria *Emp-Técnico* que é uma subclasse de *Empregado*.

```
DEFINE CONSTRAINT ON Gerencia:
                    nivel OF cargo OF gerente OF Projeto = ''SUP''.
```

Figura 2.13: Exemplo de declaração de restrição de integridade explícita em GORDAS

2.3.3 Definição de Atributos Derivados

Através da linguagem GORDAS é possível definir atributos a partir de outros atributos já existentes. O valor de um atributo derivado pode ser o resultado da aplicação de uma função como, por exemplo, desvio padrão ou até mesmo ser o resultado da execução de uma operação de seleção de dados.

A figura 2.14 exemplifica a derivação do atributo *salário-médio* de um departamento a partir dos salários individuais de cada empregado lotado no departamento.

```
DEFINE DERIVED ATTRIBUTE salario-medio OF Departamento TO BE
                AVERAGE OF salario OF cargo OF empregados OF Departamento.
```

Figura 2.14: Exemplo de declaração de atributo derivado em GORDAS

2.3.4 Operações de Seleção de Dados

Uma operação de seleção de dados em GORDAS é formada por duas cláusulas: a cláusula GET e a cláusula WHERE. A cláusula GET especifica a(s) classe(s)[4] sobre a(s) qual(is) deseja se obter dados e os dados, propriamente ditos, que deverão ser recuperados dos objetos[5] selecionados. A cláusula WHERE define uma expressão de seleção dos objetos da classe pesquisada.

Existe uma semelhança entre a sintaxe das operações de seleção GORDAS e a sintaxe das operações SQL[6] [vdl89] utilizada no modelo relacional. Entretanto existe uma diferença fundamental entre essas linguagens, além daquela, é claro, de atuarem sobre modelos de dados diferentes. Em GORDAS os relacionamentos são referenciados explicitamente através dos nomes de conexão, enquanto que em SQL o usuário precisa definir uma operação *join* entre atributos de relações que representam entidades interligadas.

Para ilustrar as potencialidades das operações de seleção de dados em GORDAS, serão apresentados vários exemplos de consultas sobre o diagrama ECR da figura 2.8. Nos exemplos, os nomes reservados da linguagem são apresentados em letras maiúsculas; os nomes de classes com letra inicial maiúscula; os nomes de atributos e os nomes de conexão com letras minúsculas.

(E1) Obtenha a matrícula e o endereço do empregado "José da Silva".

```
GET <matricula,endereco> OF Empregado
WHERE nome = ''Jose da Silva''.
```

Esse exemplo simples ilustra o mecanismo de um operação GORDAS. A classe de interesse é o tipo de entidade *Empregado* e os dados desejados são os valores dos atributos *matrícula* e *endereço*. Entretanto, deseja-se obter apenas a matrícula e o endereço do empregado cujo nome é "José da Silva" e essa condição é expressa na cláusula WHERE.

O mecanismo da operação é o seguinte: cada objeto na classe especificada no final da cláusula GET, no caso cada entidade do tipo *Empregado*, é testada pela expressão de seleção contida na cláusula WHERE; se a expressão de seleção é avaliada como verdadeira, o objeto é selecionado e as informações desejadas são retornadas ao usuário solicitante. Logo, se mais de um empregado chamar-se "José da Silva", uma lista de tuplas <matricula,endereco> será apresentada ao usuário.

---

[4] O termo classe é utilizado neste contexto como um termo genérico para referir-se a um conjunto de entidades (tipo de entidade ou categoria) ou a um conjunto de instancias de um relacionamento

[5] O termo objeto se refere a uma entidade particular ou a uma instancia de um relacionamento

[6] Acronimo para Structured Query Language

(E2) Obtenha os nomes e matrículas de todos os empregados do departamento "Material".

```
GET    <nome,matricula> OF Empregado
WHERE  nome OF departamento [OF Empregado] = ''Material''.
```

Esse exemplo ilustra o uso dos nomes de conexão em GORDAS. O nome de conexão *departamento* é usado para referenciar o departamento associado ao empregado através do relacionamento *Dep-Emp*. A partir da informação sobre a cardinalidade do relacionamento existente no esquema é possível saber que um empregado está lotado em apenas um departamento.

Para permitir a utilização de nomes de conexão associados a atributos e a nomes de classe é necessário impor certas restrições aos nomes usados em um esquema ECR. Essas restrições são:

1. Todos os nomes de tipos de entidade, categorias e relacionamentos devem ser únicos no esquema;

2. Todos os nomes dos atributos devem ser únicos dentro da classe a que se referem;

3. Se um tipo de entidade/categoria participar em mais de um relacionamento, os nomes de conexão que referenciam outros tipos de entidade/categorias a partir daquele tipo de entidade/categoria devem ser únicos para cada relacionamento. Na figura 2.8, por exemplo, o tipo de entidade *Empregado* participa do relacionamento *Dep-Emp* com o tipo de entidade *Departamento* e do relacionamento *Cargo-Emp* com o tipo de entidade *Cargo*. Nesse caso, o nome de conexão que referencia entidades do tipo *Departamento* a partir de entidades do tipo *Empregado* não pode ser igual ao nome de conexão que referencia entidades do tipo *Cargo* a partir de entidades do tipo *Empregado*. Contudo, deve-se observar que os nomes de conexão que referenciam entidades do tipo *Empregado* a partir de entidades do tipo *Departamento* e os nomes de conexão que referenciam entidades do tipo *Empregado* a partir de entidades do tipo *Cargo* podem ser iguais;

1. Para um relacionamento binário, se os nomes de conexão de um tipo de entidade/categoria participante forem (A, B) então os nomes de conexão do outro tipo de entidade/categoria participante devem ser (B, A). O exemplo da figura 2.12 ilustra esse fato;

5. Para um relacionamento de grau $n > 2$, se os nomes de conexão de um tipo de entidade/categoria participante forem (A, B) e de outro tipo de entidade/categoria participante forem (C, D) então $B \neq D$.

Sempre que um único nome de classe aparecer na cláusula GET, esse nome de classe pode ser omitido da cláusula WHERE, já que nesse caso a classe tem que ser obrigatoriamente a mesma que foi informada na cláusula GET. Assim no exemplo em questão, o nome de classe *Empregado* poderia ter sido omitido da cláusula WHERE. Tal fato é indicado pelos colchetes que envolvem o nome de classe *Empregado*.

(E3) Obtenha o nome e o cargo de cada empregado lotado no departamento "CPD".

```
GET  (<nome> ,
          <nome OF cargo>) OF Empregado
WHERE nome OF departamento [OF Empregado] = ''CPD''.
```

Esse exemplo apresenta referências à entidades relacionadas com um empregado tanto na cláusula GET (*nome of cargo*) como na cláusula WHERE (*nome of departamento*).

(E4) Obtenha os nomes e matrículas de todos os empregados do departamento "CPD" que são gerentes de projeto.

```
GET  <nome,matricula> OF Empregado
WHERE  nome OF departamento [OF Empregado] = ''CPD''
       AND COUNT projeto-gerenciado [OF Empregado] > 0.
```

Esse exemplo mostra a utilização de um função fundamental em GORDAS: a função COUNT. Essa função determina o número de entidades/valores/instâncias de relacionamento existentes em um conjunto específico. Um empregado é um gerente se estiver associado a um projeto através do relacionamento *Gerência* e a função COUNT na cláusula WHERE determina exatamente o número de instâncias do relacionamento *Gerência* em que cada empregado está incluído. No caso da figura 2.8 o número de projetos gerenciados por um empregado será 0 ou 1, porque o relacionamento *Gerência* é 1:1.

(E5) Para cada departamento, obtenha o seu nome, o número de empregados lotados no departamento e os nomes desses empregados.

```
GET  (<nome>
     <COUNT empregados>
     <nome of empregados>) OF Departamento
[WHERE  TRUE.]
```

Esse exemplo apresenta uma consulta em que não existe um predicado de seleção que restrinja as entidades da classe desejada. Nesse caso, todos os departamentos são selecionados. A cláusula WHERE TRUE é opcional.

(E6) Para cada departamento, obtenha o seu nome e o número de empregados lotados no departamento que têm o cargo de programador.

```
GET <nome>
    <COUNT empregados: nome of cargo = ''Programador''> OF Departamento
[WHERE TRUE.]
```

O exemplo em questão ilustra uma característica da linguagem GORDAS denominada expressão lógica restritiva. Na consulta do exemplo não se deseja contar todos os empregados de cada departamento, mas apenas os empregados que exercem o cargo de programador. A expressão *COUNT empregados* usa o nome de conexão empregados para referenciar todos os empregados lotados no departamento. Porém, a expressão que se segue ao nome de conexão limita a aplicação da função COUNT apenas aos empregados do departamento que exercem o cargo de programador. O nome de conexão *cargo* incluído na expressão lógica restritiva, permite referenciar o cargo de cada empregado selecionado anteriormente.

(E7) Obtenha os nomes de todos os departamentos que possuam pelo menos um empregado com nome "José" ou "João".

```
GET  <nome> OF Departamento
WHERE nome OF empregados [OF Departamento] INCLUDES {''Jose'',''Joao''}.
```

Esse exemplo mostra como podem ser usados operadores de comparação de conjuntos em GORDAS. A comparação de conjuntos é natural na linguagem GORDAS porque é sempre possível saber previamente se o resultado esperado para uma consulta é um valor/objeto isolado ou um conjunto de valores/objetos, através da cardinalidade dos relacionamentos.

(E8) Obtenha os nomes e matrículas de todos os empregados que trabalham em algum projeto em que o empregado "João da Silva" trabalha.

```
GET  <nome,matricula> OF Empregado
WHERE projetos [OF Empregado] INCLUDES
```

```
(GET projetos [OF Empregado]
 WHERE nome = ''Joao da Silva'').
```

Aqui está ilustrada a capacidade de se usar operações de seleção embutidas em outras operações de seleção. Nesse exemplo, o conjunto de projetos usados na cláusula WHERE é o resultado de uma outra operação de seleção no banco de dados. Todas as referências dentro da operação de seleção externa dizem respeito ao conjunto de objetos de interesse da operação de seleção interna.

Pode-se também referenciar objetos da operação externa dentro da operação interna. Esse fato não ocasionará nenhuma ambigüidade se os conjuntos de objetos na operação externa e interna forem diferentes. Porém, se for preciso referenciar-se um conjunto de objetos iguais será necessário diferenciar as ocorrências dos dois conjuntos através de sufixos compostos por números inteiros não repetidos.

(E9) Obtenha os nomes e matrículas de todos os empregados que trabalham em pelo menos dois projetos em que o empregado "João da Silva" trabalha.

```
GET  <nome,matricula> OF Empregado
WHERE COUNT (projetos [OF Empregado] INTERSECT
      (GET projetos [OF Empregado]
       WHERE nome = ''Joao da Silva'')) > 1.
```

O exemplo mostra a possibilidade que a linguagem GORDAS possui para definir novos conjuntos a partir de conjuntos já conhecidos. Para tal, são usados os operadores de construção de conjuntos UNION, INTERSECT e DIFFERENCE (-). Na consulta exemplificada, é formada a intersecção de dois conjuntos para se contar o número de elementos em comum. Cada empregado que trabalha em pelo menos dois projetos com o empregado "João da Silva" é então selecionado.

(E10) Obtenha os nomes de todos os departamentos em que o menor salário dos seus empregados é superior a 500000.

```
GET  <nome> OF Departamento
WHERE MIN salario of cargo OF empregados [OF Departamento] > 500000.
```

O exemplo ilustra o uso de funções padrão em GORDAS. Essas funções incluem: AVERAGE (média), SD (desvio padrão), SUM (somatório), MAX (valor máximo) e MIN

(valor mínimo). A linguagem também possui capacidade para expressar alguns tipos de operações aritméticas.

(E11) Obtenha os nomes dos empregados administrativos que estão lotados no departamento "CPD".

```
GET  <nome> OF Emp-Administ
WHERE nome OF departamento [OF Emp-Administ] = ''CPD''.
```

Essa consulta exemplifica os mecanismos de herança suportados pela linguagem GORDAS. Esses mecanismos permitem que atributos de um tipo de entidade, sobre o qual uma categoria é definida, sejam referenciados como se pertencessem à categoria. Esse é o caso do atributo *nome* na cláusula GET que está associado ao tipo de entidade *Empregado* sobre o qual foi definida a categoria *Emp-Administ*.

Da mesma forma é possível referenciar um relacionamento em que o tipo de entidade básica participa, como se fora um relacionamento definido para a categoria. Esse é o caso do relacionamento *Dep-Emp* que associa entidades do tipo *Empregado* com entidades do tipo *Departamento*. Esse relacionamento foi referenciado pelo nome de conexão *departamento* como se fora um relacionamento em que a categoria *Emp-Administ* participasse diretamente.

2.3.5 Operações de Atualização de Dados

São previstas as seguintes operações de atualização de dados em GORDAS:
inserir-entidade, excluir-entidade, incluir-relacionamento, excluir-relacionamento, incluir-em-categoria, excluir-de-categoria e modificar-atributo.

A operação *inserir-entidade* permite que novas entidades de um determinado tipo de entidade possam ser incluídas no banco de dados. O usuário deve informar o tipo de entidade e os valores que serão aplicados aos atributos básicos da entidade. A figura 2.15 mostra a codificação em GORDAS de uma operação de inserção de uma entidade do tipo *Empregado*.

A operação *excluir-entidade* serve para remover do banco de dados entidades pertencentes a um determinado tipo de entidade. Os parâmetros necessários para essa operação são o tipo de entidade e as condições que as entidades devem satisfazer para que possam ser excluídas. O exemplo da figura 2.16 mostra a exclusão das entidades do tipo *Empregado* que têm matrícula igual a 6531. Como o atributo *matrícula* do tipo de entidade é único (figura 2.10) somente uma entidade será excluída.

```
INSERT INTO Empregado
    ATTRIBUTES  matricula = 5, Nome = ''Pedro Campos'',
                data-nasc = 1960, Sexo  =  ''masc'',
                rua =  ''Ruas das Acacias 20'',
                bairro =  ''Bairro Feliz'',
                cidade = ''Itambica'',
                telefones = ''393983''.
```

Figura 2.15: Exemplo de operação de inclusão de entidade em GORDAS

```
DELETE FROM Empregado WHERE matricula = 6534.
```

Figura 2.16: Exemplo de operação de exclusão de entidade em GORDAS

A operação *incluir-relacionamento* permite criar uma instância de um relacionamento específico. Os parâmetros exigidos são o nome do relacionamento, a especificação de uma ou mais entidades para cada categoria que participa do relacionamento e os valores dos atributos da(s) instância(s) do relacionamento que será(ão) incluída(s), caso existam tais atributos. As entidades que serão associadas pelo relacionamento são determinadas através de um predicado de seleção. Caso seja selecionado um conjunto de entidades de cada categoria participante, então todas as associações de entidades que pertencerem ao produto cartesiano definido sobre esses conjuntos serão incluídas como instâncias do relacionamento. A figura 2.17 ilustra a inclusão de uma instância do relacionamento *Particip-Proj* que associa o empregado de matrícula 5753 ao projeto X.

```
ADD TO RELATIONSHIP Particip-Proj WHERE
                    Empregado :  matricula = 5753,
                    Projeto   :  nome = X,
        ATTRIBUTES horas-trab = 0.
```

Figura 2.17: Exemplo de operação GORDAS para inclusão de instância de relacionamento

A operação *excluir-relacionamento* exclui instâncias de um relacionamento específico. O usuário deve informar o nome do relacionamento e pode acrescentar predicados de seleção sobre as categorias participantes para restringir o número de instâncias excluídas. O exemplo da figura 2.18 apresenta a remoção das instâncias do relacionamento *Particip-*

Proj que envolvem a entidade empregado cuja matrícula é 3515. O predicado de seleção nesse exemplo inclui apenas um dos tipos de entidade que participam do relacionamento. Assim, a participação do empregado de matrícula 3515 em qualquer projeto será excluída do banco de dados.

```
REMOVE FROM RELATIONSHIP Particip-Proj WHERE
        Empregado: matricula = 3515.
```

Figura 2.18: Exemplo de operação GORDAS para exclusão de uma instância de relacionamento

A operação *incluir-em-categoria* permite a inclusão de uma entidade, pertencente a um tipo de entidade, em uma categoria. Os parâmetros para essa operação são o nome da categoria, o nome do tipo de entidade e os valores dos atributos específicos adquiridos pela entidade devido à sua inclusão na categoria (atributos específicos da categoria). Também pode ser incluído um predicado de seleção sobre o tipo de entidade para restringir as entidades selecionadas. Caso esse predicado seja omitido, todas as entidades pertencentes ao tipo de entidade básico serão selecionadas. A figura 2.19 mostra a inclusão do empregado de matrícula 1000 na categoria *Emp-Técnico*.

```
ADD TO CATEGORY Emp-Tecnico FROM
        Empregado WHERE
                matricula OF Empregado = 1000.
```

Figura 2.19: Exemplo de operação GORDAS para inclusão de entidade em uma categoria

A operação *excluir-de-categoria* é utilizada para remover entidades de uma categoria. Os parâmetros usados nessa operação são o nome da categoria e o predicado de seleção que especifica quais as entidades que serão excluídas da categoria. As entidades excluídas da categoria permanecem no banco de dados como entidades dos tipos de entidade originais. A figura 2.20 mostra a exclusão dos empregados que exercem o cargo de técnico-judiciário da categoria *Emp-Administ*.

A operação *modificar-atributo* é utilizada para substituir valores de atributos de entidades e de instâncias de relacionamento existentes no banco de dados. O usuário deve informar a classe que ele deseja tratar, o(s) atributo(s) que será(ão) atualizado(s), o predicado de seleção que especifica os objetos (entidades/instâncias de relacionamento) que serão atualizados e os novos valores dos atributos. No caso de atributos multivalorados

```
REMOVE FROM CATEGORY Emp-Administ WHERE
       nome OF cargo OF Empregado = ''Tecnico Judiciario''.
```

Figura 2.20: Exemplo de operação GORDAS para exclusão de entidades de uma categoria

é possível acrescentar novos valores ao conjunto de valores originais ou remover um ou mais valores desse conjunto. Quando os atributos se referem a um relacionamento, os predicados de seleção são definidos para cada categoria participante, como no caso da operação excluir-relacionamento. A figura 2.21 exemplifica algumas operações de modificação de atributos incluindo substituição de valores (a), acréscimo de valores em um atributo multivalorado (b) e exclusão de valores em um atributo multivalorado (c).

```
a. MODIFY <nome, data-nasc> OF Empregado TO <''Maria da Silva'', 1960>
          WHERE matricula = 3221.
b. MODIFY  telefones OF Empregado ADD VALUE-LIST 2252632
          WHERE matricula = 1857.
c. MODIFY  Telefones OF Empregado REMOVE VALUE-LIST 2251557
          WHERE matricula = 2613.
```

Figura 2.21: Exemplos de operações GORDAS para modificação de valores de atributos

2.3.6 Transações

A linguagem GORDAS suporta a definição de transações de atualização, que são unidades lógicas formadas por uma ou mais operações básicas de atualização descritas na seção 2.3.5. O grupamento de operações em transações permite:

1. Formar unidades de atualização de maior granularidade que estão mais próximas das operações lógicas executadas pelo usuário no seu dia-a-dia;
2. Diminuir o risco de violação da integridade semântica do banco de dados por operações isoladas;
3. Fornecer ao usuário não técnico transações pré-definidas e compiladas.

Uma transação GORDAS é formada por uma parte de declaração de parâmetros e por uma parte de operações, conhecida como *corpo da transação*. O corpo de um transação

é uma seqüência de operações de atualização básicas que usam os parâmetros declarados pela transação.

Operações para verificação de restrições de integridade podem ser incluídas em uma transação para assegurar que as restrições de integridade especificadas no esquema não sejam violadas.

A figura 2.22 apresenta a codificação de uma transação para substituição de um gerente de projeto no esquema ECR representado na figura 2.8. Essa transação envolve duas operações que não podem ser separadas. A primeira operação desassocia um determinado projeto do empregado técnico que era gerente desse projeto. A segunda operação associa o mesmo projeto a um novo gerente. A execução separada da primeira operação violaria a restrição de integridade que define o relacionamento *Gerência* como total em relação à participação de entidades do tipo *Projeto*. Portanto, a definição de um transação é essencial para que as operações possam formar um todo indivisível e consistente.

```
DEFINE TRANSACTION Substitui-gerente
                   (nome-proj, matric-novo-gerente)
BEGIN  TRANSACTION
       REMOVE FROM RELATIONSHIP Gerencia WHERE
                   Projeto: nome = nome-proj
       ADD    TO   RELATIONSHIP Gerencia WHERE
                   Projeto: nome = nome-proj,
                   Emp-Tecnico: matricula = matric-novo-gerente;
END    TRANSACTION
```

Figura 2.22: Exemplo de transação GORDAS

Capítulo 3

Tradução de esquemas

3.1 Introdução

Como foi visto no capítulo 1 desta dissertação, a tradução de esquemas locais, representados em diferentes modelos de dados, para esquemas componentes no MDC, objetiva, em primeiro lugar, uniformizar a representação dos dados permitindo a sua integração e, em segundo lugar, obter os mapeamentos que serão usados para a transformação de operações sobre o MDC em operações sobre o modelos de dados adotado pelo SGBD local.

Por outro lado, a tradução de esquemas federados no modelo ECR para esquemas externos nos modelos de dados originais dos SBDs componentes visa permitir ao usuário do SBDH a utilização do modelo de dados e da LMD que lhe são familiares, garantindo assim uma maior transparência do sistema como um todo.

Neste capítulo serão apresentadas metodologias para mapear esquemas locais nos modelos relacional e rede para esquemas componentes no modelo ECR e para mapear esquemas federados no modelo ECR para esquemas externos nos dois primeiros modelos.

3.2 Tradução Rede-ECR

3.2.1 Considerações gerais

Uma metodologia para tradução Rede-ECR que possa ser utilizada no contexto de SBDHs deve levar em consideração cinco pontos importantes.

Primeiro, a separação que existe entre entidades e relacionamentos no modelo rede, que possui construtores distintos para modelar esses dois conceitos semânticos, embora em algumas situações específicas, como no caso de relacionamentos N:M, essa separação não aconteça plenamente.

Segundo, a forma como são representados os relacionamentos no esquema rede. O único tipo de relacionamento representado diretamente no modelo rede é o relacionamento 1:N. Outros tipos de relacionamento ou são garantidos nos programas de aplicação (relacionamento 1:1) ou são simulados por técnicas de projeto (relacionamentos N:M, relacionamentos não binários).

Terceiro, as restrições impostas à participação dos registros membros em um *set-type*. Essas restrições devem ser consideradas na definição do tipo de participação das entidades nos relacionamentos representados no esquema ECR.

Quarto, a presença no esquema rede de informações referentes à organização física dos dados. Essas informações não podem ser transferidas para o esquema ECR mas precisam ser mantidas localmente para que possam ser utilizadas no processo de transformação de operações GORDAS em operações sobre o esquema rede.

Quinto e último, a maior riqueza semântica do modelo ECR que implica na possibilidade de inclusão no esquema ECR de informações que não podem ser obtidas diretamente do esquema rede e devem ser supridas pelo usuário, no caso o DBA local.

Cada um dos pontos citados será discutido com mais detalhes na seção 3.2.3 que analisa a correspondência entre os construtores dos dois modelos de dados.

3.2.2 Trabalhos correlatos

São poucos os trabalhos na literatura que apresentam metodologias para tradução do modelo rede para modelos baseados no modelo entidade-relacionamento (MER).

Dumpala e Arora [DA83] propuseram um algoritmo bastante limitado para converter diagramas rede em diagramas ER. O algoritmo baseia-se quase que exclusivamente na correspondência entre *record-type* e tipo de entidade e entre *set-type* e relacionamento. Entretanto, essa correspondência não é sempre verdadeira devido à presença dos registros conectores que simulam relacionamentos M:N, relacionamentos de grau maior do que dois e auto-relacionamentos.

Fong [Fon92] apresenta uma metodologia para tradução de esquemas para ser usada no caso de uma migração de dados de um SBD rede ou de um SBD hierárquico para um SBD relacional. A metodologia usa um modelo entidade-relacionamento estendido para captu-

rar a semântica do esquema rede original, particularmente com relação às dependências funcionais e às restrições de integridade referenciais existentes entre os *record-type*. As dependências funcionais são obtidas pelo exame dos itens de dados dos *record-type*, e as restrições de integridade referenciais através do exame do tipo de retenção e inserção dos *set-type* em que os *record-type* estão envolvidos. Durante todo o processo de tradução o usuário pode intervir para validar ou modificar os resultados obtidos. A representação no modelo entidade-relacionamento estendido é traduzida para um esquema relacional em que são mantidas as restrições estruturais do modelo original. Nesse esquema resultante todas as relações bases estão pelo menos na terceira forma normal.

3.2.3 Equivalência de Construtores

A tradução de esquemas locais dos SBDs componentes representados no modelo rede para esquemas componentes no modelo de dados comum (modelo ECR) pode ser feita baseada na correspondência entre os construtores dos dois modelos. A seguir serão analisadas essas correspondências.

Record-type e Set-type

De maneira geral, os *record-type* do modelo rede são equivalentes aos tipos de entidade no modelo ECR e os *set-type* são equivalentes a relacionamentos binários funcionais (1:N) entre o tipo de entidade que representa o *record-type* mestre e o tipo de entidade que representa o *record-type* membro de um *set-type*. Esse fato é ilustrado na figura 3.1.

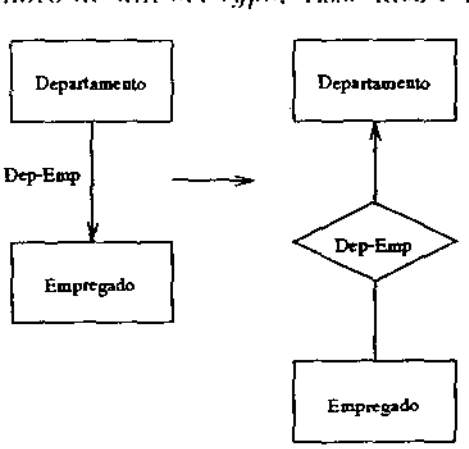

Figura 3.1: Correspondência entre *record-type* e tipo de entidade e entre *set-type* e relacionamento

Contudo, certos tipos de relacionamentos são modelados usando registros conectores:

relacionamentos N:M, auto-relacionamentos e relacionamentos que envolvem mais de dois tipos de entidades. Os conectores, apesar de serem representados por *record-type* no modelo rede, devem ser traduzidos para relacionamentos no modelo ECR, como ilustra as figuras 3.2 e 3.3.

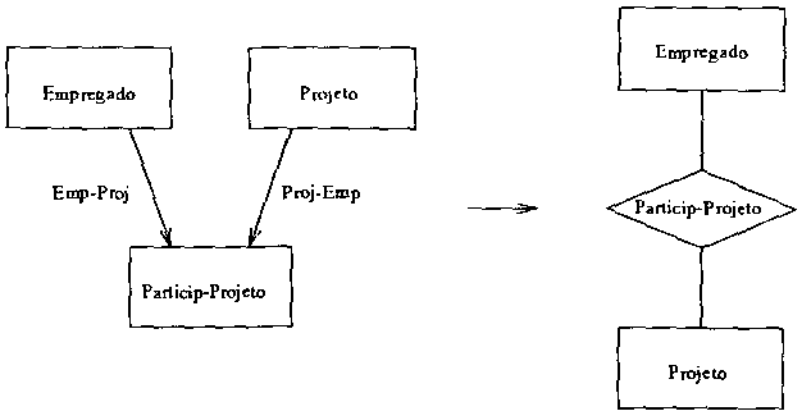

Figura 3.2: Mapeamento de relacionamento N:M no modelo rede para o modelo ECR

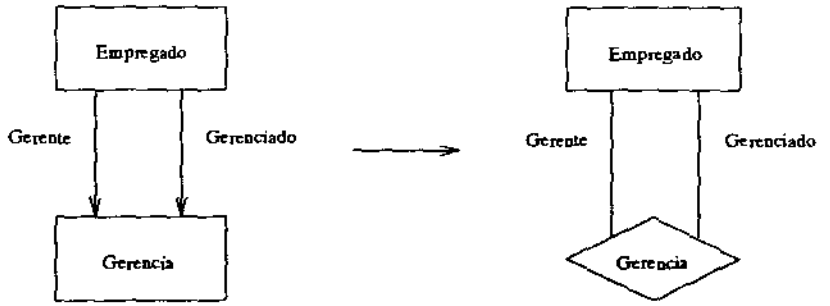

Figura 3.3: Mapeamento de auto-relacionamento no modelo rede para o modelo ECR

Uma metodologia para tradução de esquemas rede para esquemas ECR deverá incluir um algoritmo para identificar esses possíveis conectores. Sabe-se que os *record-type* conectores são membros em mais de um *set-type* e ao mesmo tempo não podem ser o *record-type* mestre em nenhum *set-type*, já que nesse caso estariam fortemente caracterizados como um tipo de entidade propriamente dito e não como uma estrutura auxiliar. Contudo, pode ser que um *record-type* satisfaça essas condições e ainda assim não represente um conector. Logo, os candidatos a conectores devem ser apresentados ao usuário (DBA local) para que ele os confirmem ou não como tal.

Set Membership (Set Insertion e Set Retention)

Na descrição de um *set-type* em um esquema rede são definidas algumas restrições em relação à participação de seus registros membros. São declaradas restrições quanto à inserção (AUTOMATIC ou MANUAL) e quanto à retenção desses registros (*FIXED*, *MANDATORY*, *OPTIONAL*). Um tipo de retenção *FIXED* ou *MANDATORY* é geralmente acompanhada de um tipo de inserção AUTOMATIC e um tipo de retenção *OPTIONAL* é normalmente acompanhada de um tipo de inserção MANUAL.

A retenção *FIXED* caracteriza um tipo de relacionamento *específico* entre as entidades que representam registros mestres e membros, e a retenção *MANDATORY* caracteriza um tipo de relacionamento *total* entre essas entidades. Ambos as retenções indicam que, para todo estado válido do banco de dados, a entidade que representa o registro membro deve estar obrigatoriamente ligada por uma instância desse relacionamento a uma entidade que representa o registro mestre. Logo, a cardinalidade mínima do relacionamento em relação à participação das entidades que representam os registros membros deverá ser em ambos os casos igual a 1. No caso do relacionamento específico, além dessa restrição, é imposta uma outra que proíbe a transferência de uma entidade que representa um registro membro para outra instância do relacionamento.

Por outro lado, o tipo de retenção *OPTIONAL* caracteriza um tipo de relacionamento *parcial* das entidades que representam os registros membros com as entidades que representam os registros mestres. Assim, uma entidade que representa um registro membro poderá existir em um estado válido do banco de dados sem estar incluída em uma instância do relacionamento. Nesse caso, a cardinalidade mínima do relacionamento em relação à participação das entidades que representam os registros membros deverá ser igual a 0.

Para todos os tipos de retenção a cardinalidade máxima do relacionamento em relação à participação do tipo de entidade que representa o *record-type* membro será igual a N, a menos que, durante o processo de tradução, o DBA local informe que a cardinalidade máxima será outra. Esse tratamento de exceção se justifica porque no modelo rede pressupõe-se que os relacionamentos são todos 1:N e outros tipos de relacionamentos, como por exemplo 1:1, só podem ser estabelecidos através de restrições impostas sobre o valor de N nos programas de aplicação.

Se o relacionamento modelado for funcional (1:N), a cardinalidade mínima e máxima do relacionamento em relação à participação do tipo de entidade que representa o *record-type* membro são iguais a 0 e 1, respectivamente. Se o relacionamento for não funcional (N:M), a cardinalidade máxima é igual a N.

Itens de dados

Os itens de dados descritos na definição dos *record-type* no modelo rede são equivalentes aos atributos dos tipos de entidade e relacionamentos no modelo ECR. Os itens de dados dos *record-type* que foram convertidos em tipos de entidade ou categorias no modelo ECR, são neles incluídos. Os itens de dados pertencentes aos conectores no esquema rede são incluídos nos relacionamentos que esses conectores geraram no esquema ECR.

Cada atributo no modelo ECR tem um conjunto de valores associados (*valueset*) que podem ser obtidos a partir da descrição do tipo e do tamanho dos itens de dados no esquema rede. Os itens de dados compostos ou multivalorados podem ser representados diretamente no modelo ECR. As cardinalidades mínima e máxima dos atributos monovalorados são iguais a 1 (um). As cardinalidades mínima e máxima dos atributos multivalorados são iguais a 1 e a N, respectivamente, onde N é o número de ocorrência definido na declaração dos itens de dados no esquema rede.

No modelo ECR, os tipos de entidade não precisam apresentar atributos identificadores (chaves) uma vez que as entidades com mesmos conteúdos de atributos podem ser distinguidas pelos relacionamentos que possuem com outras entidades. Entretanto, alguns atributos identificadores podem ser inferidos a partir do esquema rede. Por exemplo, os registros que têm a cláusula *LOCATION MODE CALC DUPLICATES NOT ALLOWED* podem ser identificados univocamente pelos atributos que representam os itens de dados contidos nessa cláusula. Da mesma forma, os registros membros de um *set-type* que são ordenados pela cláusula *SORTED BY DEFINED KEY DUPLICATES NOT ALLOWED* podem ser identificados univocamente pela concatenação dos atributos definidos nessa cláusula com o(s) atributo(s) que identifica(m) o seu mestre. O valor da *DATABASE-KEY* que em um banco de dados rede identifica univocamente cada registro, não pode ser usada como identificador no modelo ECR porque esse valor expressa unicamente a identidade física do registro e, portanto, não tem utilidade no nível lógico (conceitual) em que o modelo de dados comum é utilizado.

Cláusulas referentes à organização física dos dados

Os esquemas no modelo rede trazem algumas informações relativas às particularidades da organização física dos dados, como localização física (*LOCATION MODE*), métodos para identificação de uma ocorrência de um *set-type* (*SET SELECTION*), partições lógicas de registros (*AREA*) e ordenamento dos registros membros dentro de um *set-type* (*SET ORDER*).

Essas informações são relevantes para a definição de estratégias de recuperação e arma-

zenamento dos dados e por conseguinte são importantes no processamento das operações nesse modelo. Porém, o modelo ECR, por ser um modelo conceitual, não possui construtores para modelar essa características físicas. Por esse motivo, essas informações devem ser armazenadas em um repositório de dados local que pode ser um dicionário de dados (DD) ou um banco de dados auxiliar. As informações nele contidas serão utilizadas pelo processador responsável pela transformação de operações expressas no modelo ECR em operações equivalentes no modelo rede. Além disso, essas informações precisam ser armazenadas a nível global para que seja possível criar esquemas externos no modelo rede a partir de esquemas federados no modelo ECR.

Construtores específicos do modelo ECR

Alguns construtores importantes utilizados no modelo ECR não possuem equivalentes no modelo rede e são usados para modelar explicitamente conceitos semânticos importantes. Entre esses construtores destacam-se as categorias (de generalização e de especialização) e os nomes de conexão, discutidos no capítulo 2 dessa dissertação.

As abstrações de generalização e especialização são modeladas de forma intuitiva no modelo rede. Por exemplo, a figura 3.4 apresenta um *set-type* associando *Pessoa* e *Empregado*. Pode ser que a intenção do projetista seja a de representar o fato de que todo empregado é uma pessoa, isto é, que *Empregado* é uma subclasse de *Pessoa*. Da mesma forma, a figura 3.5 pode sugerir que *Empregado* seja uma generalização de *Empregado Técnico* e *Empregado Administrativo*. O problema é que a intenção do projetista não pode ser percebida através de um exame objetivo do esquema rede. Fica evidente a necessidade de intervenção humana para a inclusão de categorias no esquema componente ECR.

Contudo, a inclusão de categorias no esquema componente, embora desejável do ponto de vista da clareza de representação, não é uma boa alternativa no contexto de SBDHs pois modifica sensivelmente a organização original dos dados no esquema local, aumentando o número de mapeamentos necessários entre os construtores nos vários níveis da arquitetura do SBDH. Conseqüentemente os processos de tradução de esquemas e, principalmente, de transformação de operações se torna muito mais complexo.

Os nomes de conexão são usados no modelo ECR para permitir que se façam referências às entidades de uma categoria a partir de uma outra entidade que está relacionada com as primeiras através de um determinado relacionamento. Esses nomes de conexão estabelecem, de certa maneira, caminhos de navegação entre os registros no modelo rede.

Os nomes de conexão não podem ser obtidos das estruturas existentes no esquema rede e, portanto, devem ser supridos pelo DBA local que deverá ter o cuidado de escolher

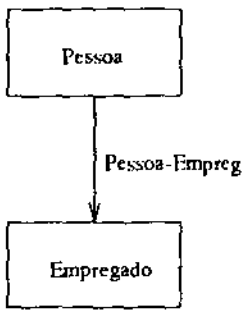

Figura 3.4: Set-type representando um relacionamento ISA

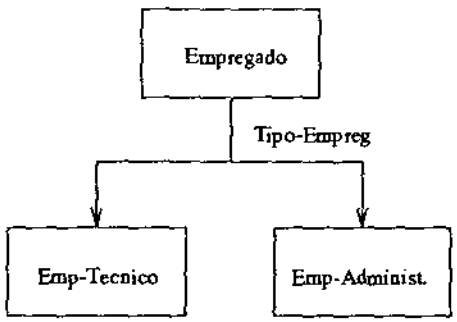

Figura 3.5: Set-type representando uma generalização

nomes significativos para tornar mais claras as referências entre as entidades. As correspondências entre os nomes dos *set-type* e os nomes de conexão deverão ser guardadas no DD local para serem usadas no processo de transformação de operações GORDAS em operações na LMD rede.

Restrições de integridade explícitas

O modelo rede provê, pelo menos em tese, um mecanismo que pode ser utilizado para a verificação de restrições de integridade semânticas. Esse mecanismo é baseado em procedimentos especiais, conhecidos como *database procedures* [TF76], que são invocados sempre que determinadas condições sobre os dados não são satisfeitas.

Na verdade, a maioria dos SGBDs rede comercialmente disponíveis não suporta esse tipo de facilidade, deixando a cargo dos programas de aplicação a manutenção das restrições de integridade que não são intrínsecas ao modelo rede.

De qualquer forma, as restrições semânticas podem ser incorporadas ao esquema componente utilizando-se os recursos providos pela linguagem GORDAS.

3.2.4 Metodologia proposta

Nas seção anterior foram feitas várias considerações sobre a tradução de esquemas rede para esquemas ECR. Foram discutidas, em particular, as correspondências entre os construtores utilizados nos dois modelos e a interação do DBA local no processo de tradução para tornar possível a inclusão no esquema alvo de informações que não podem ser inferidas diretamente do esquema original.

Baseado nessas discussões, pode-se então sugerir uma metodologia para a tradução Rede-ECR composta do seguintes passos:

1. Exame dos *record-type* identificando tipos de entidades provisórios;
2. Exame dos *set-type* identificando relacionamentos provisórios;
3. Identificação dos conectores que representam relacionamentos N:M e relacionamentos recursivos (auto-relacionamentos);
1. Definição dos nomes de conexão;
5. Refinamento do esquema ECR.

Cada um desses passos será discutido a seguir.

Exame dos record-type

Cada *record-type* deverá dar origem a um tipo de entidade provisória. Esse mapeamento é provisório porque alguns dos *record-type* podem representar conectores que são convertidos para relacionamentos no modelo ECR. Porém, só será possível identificar exatamente o conceito semântico representado por um *record-type* ao serem examinados os *set-type* em que ele participa.

Os itens de dados dos *record-type* serão transformados em atributos dos tipos de entidade gerados, juntamente com seus respectivos *valueset*, que serão definidos no esquema ECR com o mesmo nome dos atributos e com as mesmas características de tipo e tamanho descritas nos itens de dados correspondentes do esquema rede.

As informações relativas às cláusulas *LOCATION MODE* e *AREA* deverão ser armazenadas no DD local para que possam ser utilizadas na transformação de operações GORDAS para operações rede interpretáveis pelo SGBD local.

As figuras 3.6 e 3.7 mostram, respectivamente, o trecho do esquema ECR gerado (expresso em CORDAS) a partir da definição do *record-type Empregado* da figura 1.7 e as entradas do DD local incluídas nessa etapa da metodologia.

```
DEFINE VALUESET matricula     AS INTEGER RANGE 0:999999.
DEFINE VALUESET nome          AS STRING  40.
                .
                .
                .
DEFINE VALUESET data-admissao AS DATE.

DEFINE ENTITY-TYPE Empregado
       ATTRIBUTES
           matricula   VALUESET matricula KEY,
           nome-empreg VALUESET nome,
           endereco    COMPOUND (rua    VALUESET rua,
                                 numero VALUESET numero,
                                 bairro VALUESET bairro),
           telefone    VALUESET telefone,
                       .
                       .
                       .
           data-admissao VALUESET data-admissao.
```

Figura 3.6: Trecho do esquema ECR obtido no passo 1 da metodologia Rede-ECR

Entrada	Tipo	Construtor Rede	Location Mode	Area
Empregado	entidade	record-type	calc matricula	A1

Figura 3.7: Mapeamento gerado no passo 1 da metodologia Rede-ECR

Exame dos set-type

Para se identificar quais dentre os *record-type* existentes no esquema rede representam conectores é preciso examinar todos os *set-type* do esquema, porque um *record-type* só é

considerado conector se ele participa como *record-type* membro em mais de um *set-type* e não participa como *record-type* mestre em nenhum outro *set-type*.

Entretanto, deve-se gerar para cada *set-type* examinado um relacionamento binário 1:N provisório. Dessa forma, evita-se um novo exame dos *set-type* para gerar esses relacionamentos após a identificação dos conectores.

O tipo de relacionamento (específico, total ou parcial) é definido pelo número de participações mínima e máxima das entidades nesse relacionamento, que podem ser obtidos através do exame das cláusulas de inserção e retenção dos *record-type* membros, como foi mostrado na seção 3.2.3. A participação do tipo de entidade que representa o *record-type* mestre é assumida como não funcional (caracterizando um relacionamento 1:N). Contudo essa cardinalidade poderá ser alterada no passo de refinamento do esquema ECR (passo 5).

Para os *set-type* com mais de um *record-type* membro são criados relacionamentos binários 1:N entre o *record-type* mestre e cada um dos *record-type* membros. O nome de cada relacionamento é formado pelo nome do *set-type* original concatenado com um sufixo diferenciador que pode ser, por exemplo, um número inteiro seqüencial e crescente. Todos os *set-type* criados são mapeados para o *set-type* original para que as operações sobre o esquema componente possam ser transformadas em operações usando os construtores do esquema local.

As informações contidas nas cláusulas *SET SELECTION* e *SET ORDER* de cada *set-type* examinado devem ser armazenadas no DD local para serem usadas na transformação de operações. Além disso, para cada *record-type* devem ser guardados os *set-type* em que ele participa como membro e os *set-type* em que ele participa como mestre. Essas informações serão usadas no próximo passo da metodologia.

As figuras 3.8 mostra o trecho do esquema ECR obtido pela aplicação do passo 2 sobre a definição dos *set-type* das figuras 1.9 e 1.10. A figura 3.9 mostra o mapeamento gerado no DD local.

Identificar conectores

A identificação dos possíveis conectores é feita examinando-se as entradas do DD local que associam cada *record-type* com os *set-type* em que ele participa.

Os possíveis candidatos a conectores são apresentados ao DBA local que deve determinar se o *record-type* em questão é um conector ou não. Se o *record-type* não for considerado um conector nenhuma ação é executada, porque os relacionamentos definidos pelos *set-type* em que ele participa já foram identificados e declarados no passo anterior

```
DEFINE RELATIONSHIP Dep-Emp FROM
                    Departamento,
                    Empregado          MINIMUM PARTICIPATION 1,
                                       MAXIMUM PARTICIPATION 1.
DEFINE RELATIONSHIP Emp-Proj FROM
                    Empregado,
                    Particip-projeto   MINIMUM PARTICIPATION 1,
                                       MAXIMUM PARTICIPATION 1.
DEFINE RFLATIONSHIP Proj-Emp FROM
                    Projeto,
                    Particip-projeto   MINIMUM PARTICIPATION 1,
                                       MAXIMUM PARTICIPATION 1.
DEFINE RELATIONSHIP Cargo-Emp FROM
                    Cargo,
                    Empregado          MINIMUM PARTICIPATION 1,
                                       MAXIMUM PARTICIPATION 1.
DEFINE RELATIONSHIP Emp-Sal FROM
                    Empregado,
                    Historico-salario SPECIFIC
                                       MINIMUM PARTICIPATION 1,
                                       MAXIMUM PARTICIPATION 1.
DEFINE RELATIONSHIP Emp-Depend FROM
                    Empregado,
                    Dependente         SPECIFIC
                                       MINIMUM PARTICIPATION 1,
                                       MAXIMUM PARTICIPATION 1.
DEFINE RELATIONSHIP Emp-Esc    FROM
                    Empregado,
                    Historico-escolar SPECIFIC
                                       MINIMUM PARTICIPATION 1,
                                       MAXIMUM PARTICIPATION 1.
DEFINE RELATIONSHIP Emp-Ferias FROM
                    Empregado,
                    Ferias             SPECIFIC
                                       MINIMUM PARTICIPATION 1,
                                       MAXIMUM PARTICIPATION 1.
DEFINE RELATIONSHIP Gerente-Proj FROM
                    Projeto,
                    Empregado          MINIMUM PARTICIPATION 0,
                                       MAXIMUM PARTICIPATION 1.
```

Figura 3.8: Trecho do esquema obtido no passo 2 da metodologia Rede-ECR

Entrada	Tipo	Const. Rede	Location Mode	Area	Mestre nos sets	Membro nos sets
Empregado	entidade (TE)	Rec-type (RT)	Calc (matricula)	Area-Pes	Emp-Proj, Emp-Sal, Emp-Depend, Emp-Esc, Emp-Ferias	Dep-Emp, Gerente-Proj, Cargo Emp
Departamento	TE	RT	Calc (cod-depto)	Area-Pes	Dep-Emp	
Cargo	TE	RT	Calc (cod-cargo)	Area-Pes	Cargo-Emp	
Projeto	TE	RT	Calc (cod-projeto)	Area-Pes	Gerente-Proj, Proj-Emp	
Particip-projeto	TE	RT	Via (Emp-Proj)	Area-Pes		Emp-Proj, Proj-Emp
Historico-salario	TE	RT	Via (Emp-Sal)	Area-Pes		Emp-Sal
Historico-escolar	TE	RT	Via (Emp-Esc)	Area-Pes		Emp-Esc
Dependente	TE	RT	Via (Emp-Depend)	Area-Pes		Emp-Depend
Ferias	TE	RT	Via (Emp-Ferias)	Area-Pes		Emp-Ferias

Entrada	Tipo	Construtor Rede	Set Selection	Set Order
Dep-Emp	Relac.bin (RI)	set-type	value of (cod-depto)	Last
Cargo-Emp	RI	set-type	value of (cod-cargo)	System default
Emp-Sal	RI	set-type	value of (matricula)	Descending (data-vigencia, valor-salario)
Emp-Depend	RI	set-type	value of (matricula)	Ascending (data-nasc, cod-gemeos)
Emp-Esc	RI	set-type	value of (matricula)	Ascending (cod-instrucao)
Emp-Ferias	RI	set-type	value of (matricula)	Descending (periodo-aquisitivo)
Emp-Proj	RI	set-type	Application	Next
Proj-Emp	RI	set-type	Application	Next
Gerente-Proj	RI	set-type	Application	System default

Figura 3.9: Mapeamentos gerados no passo 2 da metodologia Rede-ECR

da metodologia.

Se um *record-type* é identificado como conector, então o próximo passo é determinar se esse conector está sendo usado para representar um relacionamento N:M ou um auto-relacionamento. Se os *record-type* mestres dos *set-type* em que um conector participa como membro são distintos, fica caracterizado um relacionamento N:M entre esses *record-type* mestres. No caso dos *record-type* mestres serem idênticos fica caracterizado um auto-relacionamento.

As declarações dos tipos de entidade que representavam os registros conectores no esquema ECR provisório devem ser substituídas pelas declarações dos relacionamentos que eles realmente representam. Os atributos do tipo de entidade passam a ser atributos do relacionamento. A correspondência entre os *set-type* originais e os relacionamentos gerados deve ser guardada no DD local. A figura 3.10 apresenta as modificações introduzidas no esquema ECR ocasionadas pela identificação do conector *Particip-projeto* e a figura 3.11 apresenta as modificações introduzidas nos mapeamentos armazenados no DD local.

```
DEFINE  ENTITY-TYPE   Particip-Projeto...          ⟹ Excluído

DEFINE  RELATIONSHIP  Emp-Proj            FROM...  ⟹ Excluído

DEFINE  RELATIONSHIP  Proj-Emp             FROM...  ⟹ Excluído

DEFINE  RELATIONSHIP  Particip-Projeto     FROM
                      Empregado,                    ⟹ Incluído
                      Projeto.
```

Figura 3.10: Modificações no esquema feitas no passo 3 da metodologia Rede-ECR

Definir nomes de conexão

A declaração dos relacionamentos feita nos passos 2 e 3 da metodologia ainda não está completa porque falta incluir os nomes de conexão para cada tipo de entidade participante. Como foi visto na seção 2.3, cada participação de uma classe[1] em um relacionamento está associada a dois nomes de conexão, *nome-conexão-1* e *nome-conexão-2*, que determinam um caminho da classe para o relacionamento e um caminho do relacionamento para a

[1]Uma classe é definida aqui como um conjunto de entidades, que pode ser tanto um tipo de entidade como uma categoria

Entrada	Tipo	Const. Rede	Location Mode	Area	Mestre nos sets	Membro nos sets
Particip-Projeto	relac.N:M (R2)	RT	Via (Emp-proj)	Area-Pes		Emp-Proj, Proj-Emp

Entrada	Tipo	Construtor Rede	Set Selection	Set Order
Emp-Proj		set-type	Application	
Proj-Emp		set-type	Application	

Figura 3.11: Mapeamentos modificados no passo 3 da metodologia Rede-ECR

classe, respectivamente. Os nomes de conexão possibilitam que as classes participantes em um relacionamento se referenciem umas às outras. No caso de relacionamentos binários, o *nome-conexão-1* permite referenciar não só o relacionamento e seus eventuais atributos mas também a outra classe participante. Para um relacionamento que associa mais de duas classes, a referência à uma classe B a partir de uma classe A é feita através do caminho definido pela concatenação do *nome-conexão-2* da classe B com o *nome-conexão-1* de A. Nos auto-relacionamentos, a referência às entidades de uma classe, que participam do relacionamento desempenhando um papel Y, a partir de uma entidade da mesma classe que desempenha um papel X, é feita utilizando o *nome-conexão-1* associado ao tipo de participação X.

A responsabilidade da definição dos nomes de conexão é do DBA local juntamente com o DBA do sistema global. Para facilitar essa tarefa seria interessante prover uma visão gráfica do esquema ECR construído até o passo 3 da metodologia. Os nomes de conexão podem ser definidos livremente desde que obedeçam às restrições citadas na seção 2.3.1.

As correspondências entre os caminhos de busca de dados definidos pelos nomes de conexão no esquema ECR e os caminhos de navegação no esquema rede deverão ser guardados no DD local para que sejam utilizados durante o processo de transformação de operações GORDAS em operações sobre o esquema rede. Esses mapeamentos podem ser obtidos examinando-se os relacionamentos gerados no esquema ECR.

Seja $R = \{R_1, R_2, \cdots, R_n\}$ o conjunto dos relacionamentos gerados no esquema ECR.

Seja $R_i = \{(E_1, na_1, nb_1), (E_2, na_2, nb_2), \cdots, (E_n, na_n, nb_n)\}, 1 \leq i \leq n$, um relacionamento qualquer de R que associa as classes $E_1, E_2, \cdots, E_n$ através dos nomes de conexão na_i e nb_i.

Sejam na_j o *nome-conexão-1* e nb_j o *nome-conexão-2* definidos para a classe E_j, $1 \leq j \leq n$, que participa do relacionamento R_i. Para cada E_j devem ser gerados os seguintes mapeamentos:

1. Um mapeamento do caminho definido pela expressão $<of\ E_j>$ para o *record-type* correspondente a E_j;

2. Um mapeamento do caminho definido pela expressão $<na_j\ of\ E_j>$ para o caminho definido pelo *set-type* S correspondente a R_i, se R_i é um relacionamento 1:N. O caminho $E_j \rightarrow S$ pode ter duas formas: (*mestre-S* $\rightarrow$ *membro-S*) ou (*membro-S* $\rightarrow$ *mestre-S*) conforme E_j seja mestre ou membro de S;

3. Um mapeamento do caminho definido pela expressão $<na_j\ of\ E_j>$ para o caminho definido pelo *set-type* T que tem como membro o *record-type* correspondente a R_i e como mestre o *record-type* correspondente a E_j, se R_i é um relacionamento N:M ou um relacionamento de grau maior do que dois. O caminho definido pelo *set-type* T é da forma: (*mestre-T* $\rightarrow$ *membro-T*). Se o relacionamento R_i é binário deve ser gerado um mapeamento adicional da expressão $<na_j\ of\ E_j>$ para o *record-type* correspondente à outra classe $E_k, k \neq j$, do relacionamento. Esse mapeamento é da forma (*mestre-T* $\rightarrow$ *membro-T/membro-U* $\rightarrow$ *mestre-U*) onde U é o *set-type* que tem o *record-type* correspondente a E_k como mestre e o *record-type* correspondente a R_i como membro.

4. Um mapeamento do caminho definido pela expressão $<na_j\ of\ E_j>$ para o caminho que vai do *record-type* que corresponde a E_j até o *record-type* que corresponde a R_i através do *set-type* V que liga R_i às entidades de E_j que desempenham um determinado papel P_1, se R_i é um auto-relacionamento. O caminho correspondente no esquema rede tem a forma (*mestre-V* $\rightarrow$ *membro-V*). Deve ser gerado também um mapeamento adicional do caminho definido pela expressão $<na_j\ of\ E_j>$ para o caminho que vai do *record-type* correspondente a E_j até ele mesmo, passando pelo *set-type* V e pelo *set-type* W que liga R_i às entidades de E_j que representam o papel P_2 no relacionamento. Esse caminho tem a forma (*mestre-V* $\rightarrow$ *membro-V/membro-W* $\rightarrow$ *mestre-W*).

5. Um mapeamento do caminho definido por cada expressão $<nb_k\ of\ na_j\ of\ E_j>$, $1 \leq k \leq n$, $k \neq j$, para o *record-type* correspondente a E_k, se R_i é um relacionamento que envolve mais de duas classes. O caminho no esquema rede é da forma (*mestre-X* $\rightarrow$ *membro-X/membro-Y* $\rightarrow$ *mestre-Y*), onde X é o *set-type* que tem como mestre o *record-type* correspondente a E_j e como membro o *record-type* correspondente a R_i e Y é o *set-type* que que tem como mestre o *record-type* correspondente a E_k e como membro o *record-type* correspondente a R_i.

A figura 3.12 mostra a definição de alguns dos relacionamentos das figura 3.8 e 3.10 já com os nomes de conexão incluídos. A figura 3.13 mostra os mapeamentos que serão guardados no DD local. O leitor deve perceber que os mapeamentos de caminhos como *<projetos of empregados of Departamento>*, embora não apareçam explicitamente no DD local, podem ser facilmente obtidos com a substituição da subexpressão *<empregados of Departamento>* pelo *record-type* equivalente, no caso *Empregado*.

```
DEFINE RELATIONSHIP Dep-Emp FROM
                    Departamento (empregados, departamento),
                    Empregado    (departamento, empregados)
                                  MINIMUM PARTICIPATION 1,
                                  MAXIMUM PARTICIPATION 1.
DEFINE RELATIONSHIP Cargo-Emp FROM
                    Cargo        (empregados, cargo),
                    Empregado    (cargo, empregados)
                                  MINIMUM PARTICIPATION 1,
                                  MAXIMUM PARTICIPATION 1.
DEFINE RELATIONSHIP Emp-Sal FROM
                    Empregado    (hist-salario, empregado),
                    Historico-salario
                                 (empregado, hist-salario)
                                 SPECIFIC
                                 MINIMUM PARTICIPATION 1,
                                 MAXIMUM PARTICIPATION 1.
DEFINE RELATIONSHIP Particip-projeto FROM
                    Projeto      (empregados, projetos),
                    Empregado    (projetos, empregados).
```

Figura 3.12: Nomes de conexão incluídos no passo 4 da metodologia Rede-ECR

Refinar esquema

Como resultado da execução dos quatro passos anteriores obtém-se um esquema ECR estruturalmente completo e equivalente ao esquema rede original. Contudo, o modelo ECR permite que certas informações importantes não contidas no esquema rede possam ser incorporadas ao esquema ECR gerado.

Restrições de integridade que estavam sendo verificadas nos programas de aplicação locais podem ser definidas diretamente no esquema componente ECR. Podem ser in-

Caminho ECR	Caminho rede 1	Caminho rede 2	Record-type alcançado
OF Departamento	Departamento		Departamento
OF Empregado	Empregado		Empregado
OF Cargo	Cargo		Cargo
OF Projeto	Projeto		Projeto
OF Hist-salario	Hist-salario		Hist-salario
OF Hist-escolar	Hist-Escolar		Hist-escolar
OF Dependente	Dependente		Dependente
OF Ferias	Ferias		Ferias
empregados OF Departamento	Dep-Emp (mestre — membro)		Empregado
departamento OF Empregado	Dep-Emp (membro — mestre)		Departamento
empregados OF Cargo	Cargo-Emp (mestre — membro)		Empregado
cargo OF Empregado	Cargo-Emp (membro — mestre)		Cargo
empregados OF Projeto	Proj-Emp — Emp-Proj (mestre-S1 — membro-S1 — mestre-S2)	Proj-Emp (mestre — membro)	Particip-Projeto, Empregado
projetos OF Empregado	Emp-Proj — Proj-Emp (mestre-S1 — membro-S1 — mestre-S2)	Emp-Proj (mestre — membro)	Particip-Projeto, Projeto
⋮	⋮	⋮	⋮
gerente OF Projeto	Gerente-Proj (mestre — membro)		Empregado
projeto-gerenciado OF Empregado	Gerente-Proj (membro — mestre)		Projeto

Figura 3.13: Mapeamentos gerados no passo 4 da metodologia

corporadas ao esquema componente, por exemplo, restrições sobre a cardinalidade dos relacionamentos representados pelos *set-type* e restrições sobre os valores permitidos para um atributo.

A inclusão de restrições de integridade semânticas no esquema componente é muito importante, porque permite que o próprio sistema se responsabilize pela sua verificação e manutenção, durante o processo de transformação de operações submetidas pelos usuários federados em operações sobre os SBDs componentes. Logo, o esquema componente funciona como uma espécie de interface semântica [Oli92] para os SBDs que formam o SBDH. Essa interface semântica permite uma maior independência dos dados em relação as aplicações dos usuários. Em contrapartida, exige-se uma maior complexidade dos processadores que constituem o sistema gerenciador do SBDH.

No último passo da metodologia proposta, o DBA local e o DBA global devem avaliar a conveniência de se incluir no esquema componente ECR as restrições de integridade que não são controladas pelo SGBD rede local. Essa decisão representa uma espécie de compromisso entre a simplicidade e o grau de segurança desejados para o ambiente. Se por acaso, a opção escolhida for a de não incluir as restrições de integridade no esquema ECR, então essas restrições devem ser divulgadas entre a comunidade de usuários federados para que as operações submetidas por eles não violem a integridade dos dados.

A figura 3.14 mostra alguns exemplos de modificações feitas durante o passo de refinamento para incluir no esquema componente restrições de integridade não modeladas no esquema local rede. Os *valuesets* dos atributos *sexo* e *data-nasc* são alterados para permitir apenas determinados valores. Assim, o atributo *sexo* só admite os valores *"m"* para sexo masculino e *"f"* para sexo feminino. Da mesma forma, o atributo *data-nasc* só admite datas válidas entre os anos de 1943 e 1977 indicando que a empresa só admite empregados em determinada faixa etária. No relacionamento *Gerente-Proj* são alteradas a participação mínima e máxima do tipo de entidade *Projeto* para indicar que o relacionamento é 1:1 e não 1:N como foi modelado durante o passo 2 da metodologia.

A figura 3.15 mostra o diagrama ECR que representa o esquema componente obtido pela aplicação da metodologia no esquema local rede descrito nas figuras 1.7 a 1.10.

3.3 Tradução Relacional-ECR

3.3.1 Considerações gerais

O modelo relacional possui um único construtor (*relação*) para modelar entidades e relacionamentos entre entidades. Como conseqüência dessa sobrecarga semântica existe a

```
DEFINE VALUESET sexo AS {''m'',''f''}.

DEFINE VALUESET data-nasc AS DATE 01011943:31121977.

DEFINE RELATIONSHIP Gerente-Proj FROM
                    Projeto     (projeto-gerenciado, gerente)
                                MINIMUM PARTICIPATION 1,
                                MAXIMUM PARTICIPATION 1
                    Empregado   (gerente, projeto-gerenciado)
                                MINIMUM PARTICIPATION 0,
                                MAXIMUM PARTICIPATION 1.
```

Figura 3.11: Exemplos de alterações feitas durante refinamento do esquema na metodologia Rede-ECR

necessidade de se definir relacionamentos através de domínios comuns entre os atributos das relações. Além disso, o modelo relacional apresenta pouquíssimas restrições de integridade estruturais intrínsecas o que permite alternativas diferentes para se modelar certos conceitos semânticos. Por exemplo, um relacionamento 1:N pode ser modelado utilizando-se uma chave estrangeira na relação com cardinalidade N ou através de uma relação separada cuja chave primária seja a concatenação das chaves das relações que representam as entidades participantes.

Dentre as alternativas possíveis para o projeto de uma aplicação em um banco de dados relacional, algumas podem facilitar a percepção das entidades e relacionamentos e outras podem dificultar essa percepção, omitindo certas relações que representam tipos de entidade e representando apenas o seu relacionamento com outros tipos de entidade. Em geral, a normalização das relações ajuda na obtenção de "bons" projetos, tornando mais claro o papel desempenhado por cada relação [NA88] e impedindo uma série de anomalias durante a atualização dos dados [Dat86]. Contudo, nem sempre a normalização é suficiente para contornar todos os problemas de representação do modelo relacional [Ken79].

As limitações semânticas do modelo relacional dificultam a tradução Relacional-ECR, exigindo uma grande interação com o DBA local para que ele complemente informações omitidas no esquema relacional e resolva certas ambigüidades de projeto.

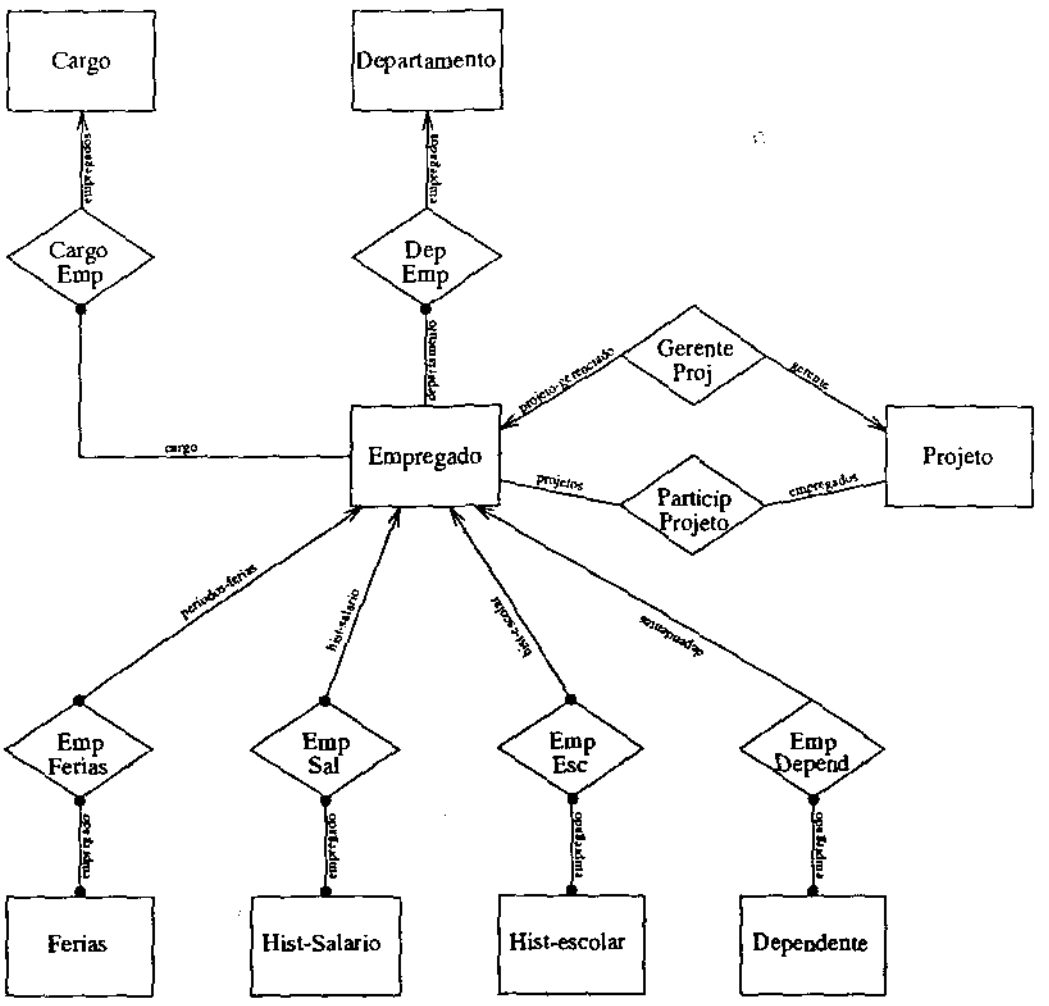

Figura 3.15: Diagrama do esquema componente obtido pela metodologia Rede-ECR

3.3.2 Trabalhos correlatos

Muitas metodologias para tradução do modelo relacional para modelos baseados no MER já foram propostas na literatura [CS83, DA83, BDH+88, DA88, NA88, JK90]. Todas essas metodologias visam prover um melhor entendimento e uma melhor documentação das informações contidas em SBDs relacionais através da modelagem conceitual das mesmas. No entanto, elas diferem entre si em muitos pontos: embasamento teórico, tipo de modelo tratado (ER puro ou ER estendido), limitações impostas sobre as relações, grau de interação com o usuário e resultados obtidos.

Casanova e Sá

O objetivo da metodologia proposta em [CS83] é obter uma representação conceitual das informações contidas em sistemas de arquivos tradicionais, facilitando uma possível conversão para a abordagem de bancos de dados. Todavia, essa metodologia pode ser aplicada na tradução do modelo relacional para o MER desde que algumas restrições sejam feitas sobre as relações, caracterizando o que os autores denominaram *forma normal entidade-relacionamento* (FNER).

A metodologia baseia-se nos conceitos de *chave*, *dependência de inclusão (DI)* e *referência*. Uma dependência de inclusão (*DI*) entre duas relações R e S, denotada por $R(x) \ll S(y)$, indica que existe uma dependência entre os valores do atributo (ou conjunto de atributos) x da relação R e os valores do atributo (ou conjunto de atributos) y da relação S, isto é, os valores de x formam um subconjunto dos valores de y em todas as tuplas das relações R e S. Uma referência é uma *DI* em que y é chave de S. Em uma referência, x é denominada *chave-sai* de R e y é chamada *chave-ent* de S. Um referência entre R e S é uma *referência bem formada* se ela é a única que existe entre as duas relações.

Um esquema relacional está em FNER se cada relação está na forma normal de Boyce-Codd (FNBC), cada *DI* é uma referência, toda referência é bem formada e cada relação define um tipo de entidade regular ou um tipo de entidade fraca ou um relacionamento.

A metodologia estabelece três critérios para a tradução. Segundo o primeiro critério, uma relação R define um tipo de entidade se ela não referencia outras relações. De acordo com o segundo critério, uma relação R define um relacionamento se ela referencia mais de uma relação e o conjunto de suas *chave-sai* forma a sua chave primária. O terceiro critério afirma que uma relação R define um tipo de entidade fraca quando ela referencia apenas uma relação e sua *chave-sai* é um subconjunto de sua chave primária.

Antes de verificar se as relações satisfazem os critérios citados anteriormente, a metodo

logia prevê uma operação de aglutinação em que relações com mesma chave (se existirem) são reunidas em uma única relação. A justificativa para essa operação é a suposição de que essas relações representam o mesmo tipo de entidade.

As relações que após a operação de aglutinação não satisfazem nenhum dos três critérios são desmembradas em outras relações que obedeçam aos mesmos. Durante o processo de desmembramento as referências à relação original são substituídas por referências às novas relações. Esse processo se repete até que todas as relações obedeçam a um dos critérios já mencionados.

Dumpala e Arora

A metodologia proposta em [DA83] supõe relações em 3FN e está baseada principalmente na correspondência entre as chaves primárias das relações.

O primeiro passo da metodologia é classificar as relações e atributos em grupos específicos. As relações cujas chaves primárias não contêm chaves primárias de outras relações são agrupadas como *relações primárias (RP)*. As relações cujas chaves primárias são compostas total ou parcialmente por chaves primárias de outras relações são agrupadas como *relações secundárias (RS)*. As RS são subdivididas em $RS1$, $RS2$ e $RS3$. As $RS1$ são as RS cujas chaves são formadas pela concatenação total de chaves de outras relações primárias. As $RS2$ são as relações cujas chaves são formadas pela concatenação de chaves de relações primárias e secundárias. As $RS3$ são as demais relações secundárias. Os atributos que compõem a chave de uma RS e são chaves de RP são agrupados como *atributos-chave-referenciais (ACR)*. Os atributos que compõem a chave de uma RS e não são chaves de RP são denominados *atributos-chave-não-referenciais (ACNR)*. Os atributos que não fazem parte da chave da relação são os *atributos-não-chave (ANCs)*. Os ANC se subdividem em $ANC1$, $ANC2$ e $ANC3$. Os $ANC1$ são aqueles que só aparecem em uma relação. Os $ANC2$ são aqueles que aparecem em outras RP. Os demais ANC são $ANC3$.

É importante ressaltar que antes do processo de classificação das relações e atributos, o usuário responsável pelo esquema relacional deve verificar os nomes dos atributos-chave para compatibilizá-los, evitando ambigüidades ou incoerências entre os mesmos.

Após a classificação das relações e atributos são aplicadas as seguintes regras de tradução:

- Cada RP é convertida em um tipo de entidade com mesmos atributos e mesma chave;

- Cada $RS1$ é convertida em um relacionamento identificado pela chave primária e envolvendo os tipos de entidade definidas pelos ACR. Os $ANC1$ são convertidos em atributos do relacionamento;

- Cada $RS2$ gera um tipo de entidade fraca e um relacionamento entre o tipo de entidade fraca e o tipo de entidade regular de quem ele depende. A chave e os $ANC1$ passam a compor o tipo de entidade criado;

- Para cada $ACNR$ de uma $RS3$ cria-se um tipo de entidade correspondente. É gerado também um relacionamento entre os tipos de entidade criados e os tipos de entidade referenciados pelas ACR dessa relação. Os atributos $ANC1$ da relação tornam-se atributos do relacionamento criado;

- Para cada $ANC2$ de uma RP cria-se um relacionamento entre o tipo de entidade representado pela RP e o tipo de entidade referenciado pela relação cuja chave é o $ANC2$ (chave-estrangeira);

- Para cada relacionamento binário obtido que envolve os tipos de entidade $E1$ e $E2$ verifica-se a existência das dependências funcionais (DFs) $E1 \rightarrow E2$ e $E2 \rightarrow E1$. Se existir apenas a primeira, o relacionamento é 1:N; se existir apenas a segunda o relacionamento é N:1; se existirem ambas, o relacionamento é 1:1; se não existir nenhuma delas, o relacionamento é N:M;

- Atributos $ANC3$ são associados aos tipos de entidade correspondentes, após a verificação e eventual substituição de nomes.

Briand et al

A metodologia proposta em [BDH+88] é diferente das demais porque está baseada exclusivamente nos conceitos formais de dependência funcional (DF) e dependência de inclusão (DI). O modelo conceitual adotado é um modelo entidade-relacionamento estendido com os conceitos de generalização e agregação [SS77].

A metodologia em questão supõe que sejam fornecidas a cobertura mínima das dependências funcionais, o grafo de dependências funcionais associado a cobertura mínima e as dependências de inclusão envolvidas no esquema relacional. A *cobertura mínima de dependências funcionais (CM)* pode ser definida informalmente como o menor conjunto de DFs que pode ser usado para caracterizar um esquema, de modo que as outras DFs não contidas nesse conjunto possam ser inferidas a partir dele. O *grafo de dependências funcionais (GDF)* <U,F> é um grafo orientado, onde os vértices correspondem aos atributos e as arestas às DFs que existem entre os atributos. Assim, uma aresta que parte

do vértice que corresponde ao atributo x e chega no vértice que corresponde ao atributo y representa a DF $x \rightarrow y$.

O primeiro passo da metodologia visa obter o *grafo de dependências funcionais reduzido (GDFR)* através da eliminação dos nós terminais do GDF.

No segundo passo é definido o conjunto de tipos de entidade que será representado no esquema conceitual. Cada atributo x que aparece em um vértice do grafo é o identificador de um tipo de entidade, a menos que exista uma DI sobre x ($x \ll y$).

No terceiro passo é definido um conjunto de relacionamentos N:M. Cada vértice do $GDFR$ composto (rotulado) por mais de um atributo gera um relacionamento entre os tipos de entidade identificados por cada um desses atributos, a menos que exista uma DI entre eles. O quarto passo consiste na definição de um conjunto de agregações. Se um vértice v_1 que gerou um relacionamento no passo anterior estiver ligado a um outro vértice v_2 por uma DF, então o relacionamento gerado é transformado em uma agregação.

No passo seguinte, é definido um conjunto de relacionamentos 1:N. Cada aresta entre dois vértices do $GDFR$ determina um relacionamento entre os dois tipos de entidade identificados pelos atributos de cada vértice.

No penúltimo passo, é definido um conjunto de auto-relacionamentos. A existência de um auto-relacionamento é caracterizada por duas situações. A primeira situação é a presença de um vértice composto por mais de um atributo, todos com mesmo domínio. A segunda situação é a presença de dois atributos com mesmo domínio e com uma DF entre si.

O último passo da metodologia inclui nos tipos de entidades e relacionamentos gerados até o passo anterior os atributos simples que correspondem aos vértices terminais do GDF.

Davis e Arora

A preocupação básica da metodologia apresentada por Davis e Arora [DA88] é a de traduzir para um modelo conceitual não só a estrutura, mas também o comportamento dos dados existentes em um banco de dados relacional. Com esse objetivo, os autores distinguem modelos de dados e modelos conceituais. Um modelo de dados é composto por objetos de dados (estruturas de dados + operadores que atuam sobre elas) e pelas restrições estruturais que definem o comportamento intrínseco desses objetos. Um modelo conceitual é o modelo de dados mais o comportamento explícito associado a ele, representado por assertivas introduzidas no esquema.

A idéia da metodologia é traduzir um modelo de dados relacional (MDR) para um

modelo conceitual entidade-relacionamento (*MC'ER*). Para tal, certas restrições de integridade intrínsecas ao modelo relacional são transformadas em restrições de integridade explícitas no modelo alvo. A vantagem dessa abordagem é que ela obtém um projeto mais flexível do banco de dados facultando ao DBA a possibilidade de alterar o comportamento dos dados sem modificar (ou modificando menos) a estrutura do esquema.

A metodologia supõe que as relações obedeçam a 3FN e associa a cada um dos seus passos duas ações, uma ligada à tradução de construtores e outra à tradução do comportamento dos dados.

No passo 1 são tratadas as relações cujas chaves primárias são compostas por um único atributo. Cada relação é traduzida para um tipo de entidade como mesmos atributos. Nenhum comportamento explícito é acrescentado.

No passo 2 são tratadas as relações cujas chaves primárias são compostas por mais de um atributo. Se os atributos que compõem a chave são sempre usados como um conjunto indivisível nas chaves primárias de outras relações, ou se os atributos não são usados em nenhuma outra relação, então a relação é traduzida para um tipo de entidade com mesma chave e atributos. Também nesse passo nenhum comportamento explícito é acrescentado ao esquema conceitual.

No passo 3 são examinadas as chamadas *chaves livres* (*Dangling keys*) que aparecem quando a chave de uma relação tem pelo menos um atributo em comum com a chave de algum dos tipos de entidade gerados até o passo 2, e pelo menos um atributo que não aparece nesses tipos de entidade. As chaves livres são formadas justamente pelos atributos não comuns e a sua presença caracteriza um tipo de entidade fraca. As restrições (dependências de inclusão) usadas para definir o relacionamento fraco entre os dois tipos de entidade são modeladas como comportamentos explícitos no modelo alvo.

No passo 4 são tratadas as chaves primárias compostas que não derivaram tipos de entidade nas etapas anteriores. Todas as relações cujas chaves são formadas pela concatenação de atributos que são chaves de tipos de entidade já criados são convertidas em relacionamentos N:M que associam esses tipos de entidade. Nesse passo são adicionados ao *MC'ER* comportamentos explícitos sobre as operações de inclusão e exclusão de entidades para garantir a manutenção das dependências referenciais entre elas.

No último passo da metodologia são examinados os atributos das relações que não estão contidos nas suas chaves primárias para detectar a presença de chaves estrangeiras. As chaves estrangeiras são convertidas em relacionamentos 1:N . As restrições de integridade referenciais entre os tipos de entidade relacionados são modelados como comportamentos explícitos sobre as operações de inclusão e exclusão que envolvem esses tipos de entidade.

Navathe e Awong

A metodologia proposta em [NA88] estende a proposta de Dumpala e Arora [DA83] para o modelo ECR. A adoção de um modelo semanticamente mais rico do que o MER exigiu uma metodologia mais sofisticada e com uma maior interação com o usuário.

Os autores supõem que as relações obedecem a 3FN e que os nomes dos atributos-chave usados em mais de uma relação sejam únicos. Caso essas condições não sejam satisfeitas é necessário um pré-processamento para modificar as relações e atributos de modo que eles passem a satisfazer as condições citadas. Esse pré-processamento envolve necessariamente a participação do usuário (projetista ou DBA).

A metodologia classifica as relações e atributos de acordo com a correspondência entre as chaves e atributos das relações. Entretanto, antes da classificação é feita uma análise das chaves alternativas das relações para que possam ser evitadas certas dificuldades na identificação de relacionamentos entre as entidades. Essas dificuldades devem-se basicamente às possíveis correspondências entre as chaves primárias de algumas relações com chaves alternativas de outras relações. Caso essas correspondências existam, a metodologia prevê um passo em que são substituídas as chaves primárias pelas chaves alternativas (ou vice-versa). Após essa substituição, as referências entre as relações passam a ser feitas exclusivamente pelas chaves primárias.

Uma relação cuja chave primária não contém chaves primárias de outras relações é denominada *relação primária do tipo 1 (RP1)*. Uma relação cuja chave primária k contém a chave primária k' de outra relação e que só pode ser identificada por k dentro do contexto de uma valor específico de k' é denominada *relação primária do tipo 2 (RP2)*. As relações cujas chaves primárias são compostas total ou parcialmente por chaves primárias de outras relações são denominadas *relações secundárias (RSs)*. As RS são subdivididas em $RS1$ e $RS2$. As $RS1$ são aquelas relações cujas chaves são formadas pela concatenação total de chaves primárias de outras relações. As $RS2$ são as relações cujas chaves são formadas pela concatenação parcial de chaves primárias de outras relações. Os atributos que compõem a chave de uma RS e são chaves de RP são denominados *atributos-chave-primária (ACP)*. Os atributos que compõem a chave de uma RS e não são chaves de RP são denominados *atributos-chave-geral (ACG)*. Os atributos que não compõem a chave de uma RP mas que são chaves de outras RP são denominados *atributos-chave-estrangeira (ACE)*. Os demais atributos são denominados *atributos-não-chave (ANC)*.

Após a classificação das relações e atributos, são examinadas as RP e RS cujas chaves possuem atributos com mesmo nome, para eliminar eventuais ambigüidades de referência entre as relações, através da inclusão do nome da relação referenciada como prefixo dos atributos.

A seguir são examinados os grupos de relações e atributos formados. Cada $RP1$ é convertida em um tipo de entidade com mesma chave. Os ANC' da relação são convertidos em atributos do tipo de entidade. Cada $RP2$ origina um tipo de entidade fraca e um relacionamento entre o tipo de entidade gerado e o tipo de entidade regular do qual ele depende. Os ANC' da relação são convertidos em atributos do tipo de entidade fraca. Cada $RS1$ gera um relacionamento entre os tipos de entidade que formam a sua chave primária. Os ANC' da relação tornam-se atributos do relacionamento. Cada ACE de uma RP define um relacionamento entre o tipo de entidade representado pela relação e o tipo de entidade que tem o ACE como chave primária. A cardinalidade dos relacionamentos é determinada do mesmo modo que em [DA83].

Nos últimos dois passos da metodologia são identificadas as categorias de generalização e ISA.

Os tipos de entidade que possuem chaves iguais são apresentados ao usuário que deve determinar quais dentre eles representam categorias ISA e quais representam tipos de entidades básicos. Se existir algum relacionamento entre uma categoria e um tipo de entidade básico da categoria, esse relacionamento é excluído. Da mesma forma, cada tipo de entidade que está envolvido em mais de um relacionamento é apresentado ao usuário para que ele determine se esse tipo de entidade desempenha ou não o mesmo papel em todos os relacionamentos. Se o papel desempenhado for o mesmo, os tipos de entidade que se relacionam com o tipo de entidade examinado são reunidos em uma categoria de generalização. A categoria formada recebe um identificador próprio e todos os relacionamentos anteriores são convertidos em um único relacionamento que associa o tipo de entidade examinado com a categoria criada.

Johannesson e Kalman

O modelo conceitual usado em [JK90] é um modelo entidade-relacionamento estendido que incorpora o conceito de subtipo (subclasse). Assim como nos trabalhos citados anteriormente também é assumido relações na 3FN.

A metodologia classifica as relações de acordo com o tipo de participação das suas chaves primárias nas dependências de inclusão (DIs) definidas para o esquema. Se nenhum subconjunto da chave de uma relação ocorre do lado esquerdo de uma DI, essa relação é *primária* (RP). Se a chave de uma relação é formada pela concatenação de pelo menos dois conjuntos de atributos que aparecem do lado esquerdo de DIs, essa relação é *secundária* (RS). Relações que não são RP nem RS são relações *terciárias* (RT).

Cada RP origina um tipo de entidade. Uma RS é convertida em um tipo de entidade se sua chave primária completa ou um atributo não contido na sua chave primária apa-

rece em uma *DI*. Caso contrário, a *RS* é convertida em um relacionamento. Uma *RT* corresponde à um tipo de entidade se um atributo não contido na sua chave aparece em uma *DI*. As outras relações terciárias podem ser modeladas como tipo de entidade ou como relacionamento. Cabe ao usuário (DBA ou projetista) definir, nesse caso, a opção mais correta já que essa definição depende do conhecimento do conteúdo semântico das relações.

O próximo passo consiste na análise das *DIs* para tentar estabelecer as ligações entre os construtores obtidos no passo anterior. Cada *DI* pode influenciar o esquema conceitual resultante de várias maneiras, dependendo dos tipos de atributos e construtores envolvidos. Os autores descrevem as possibilidades para cada tipo de *DI*. Cada uma dessas possibilidades é apresentada ao usuário para que ele confirme ou não uma situação provável. A presença de chaves alternativas nas *DIs* aumenta o número de possibilidades de modelagem porque essa presença pode indicar a existência de uma relação implícita no esquema relacional que deve ser explicitada no modelo conceitual.

O último passo da metodologia trata os atributos das relações que não aparecem em *DIs*. Em primeiro lugar são examinados os atributos contidos nas chaves das *RT* que foram interpretadas pelo usuário como relacionamentos. Para cada um desses atributos é criado um tipo de entidade. Os tipos de entidade criados são então associados ao relacionamento correspondente à *RT* examinada. Todos os outros atributos que não foram examinados são simplesmente convertidos em atributos do tipo de entidade ou relacionamento correspondente à relação a que pertencem.

3.3.3 Metodologia Proposta

As metodologias citadas na seção 3.3.2 apresentam algumas deficiências que impedem a sua utilização direta no contexto de SBDHs.

[CS83] restringe excessivamente o tipo de relações que podem ser tratadas o que se contrapõe à autonomia de projeto que deve ser garantida aos SBDs componentes. Além disso, a metodologia não analisa a cardinalidade dos relacionamentos gerados no esquema conceitual.

[DA83] não trata alguns tipos de relacionamentos como, por exemplo, relacionamentos existentes entre duas relações primárias com chaves interdependentes. A metodologia também apresenta alguns critérios ambíguos para classificação das relações e atributos.

[BDH+88] apresenta uma metodologia de pouca praticidade porque exige que o usuário forneça a cobertura mínima das dependências funcionais.

[DAS88] apresenta um critério para definição de tipos de entidade e relacionamentos baseado na correspondência entre as chaves das relações o que pode ocasionar problemas se os nomes dos atributos que compõem a chave não forem compatíveis, refletindo a semântica das relações.

[NAS88] apresenta uma metodologia que prevê alguns passos para a modificação de nomes e a substituição de chaves primárias por chaves alternativas (ou vice-versa), uniformizando as referências entre as relações. Essa uniformidade é necessária porque o critério de classificação das relações é baseado na comparação do nome das suas chaves primárias. Contudo, a substituição de nomes de atributos não é desejável no contexto de SBDHs porque exige que sejam feitos muitos mapeamentos adicionais entre os nomes originais e os nomes modificados, tornando o sistema muito mais complexo. Um outro problema da metodologia é não tratar conceitos importantes para o modelo ECR como as restrições em relação à participação de entidades nos relacionamentos.

As metodologias propostas em [CS83, JK90] apresentam um grande vantagem em relação às demais. Ela utiliza dependências de inclusão ao invés de dependências funcionais ou comparação de chaves. As dependências de inclusão, além de caracterizar as ligações entre as relações, são independentes de nomes de atributos e aparecem em menor número do que as dependências funcionais. Todavia, a metodologia exige uma interação excessiva com o usuário pela presença de chaves alternativas nas *DIs* e pelo fato de que muitas ligações entre os tipos de entidades e os relacionamentos terem sido feitas em uma etapa posterior à identificação dos relacionamentos.

A seguir será apresentada uma metodologia para tradução de esquemas relacionais em esquemas ECR no contexto de SBDHs. A metodologia proposta leva em consideração a autonomia de projeto dos SBDs componentes e os mapeamentos que devem ser guardados durante a tradução para permitir a transformação de operações sobre o esquema ECR em operações equivalentes sobre o esquema relacional. Algumas idéias contidas nas metodologias descritas anteriormente, como o uso de dependências de inclusão para a classificação de relações, serão utilizadas na metodologia proposta.

O processo de tradução é dividido em quatro passos:

1. Classificação de relações e atributos;
2. Definição de tipos de entidades e relacionamentos;
3. Definição dos nomes de conexão;
4. Refinamento e validação do esquema ECR resultante;

Classificação de relações e atributos

As relações são classificadas em relações primárias, secundárias, terciárias e quaternárias.

Se nenhum subconjunto da chave primária de uma relação aparece do lado esquerdo de uma dependência de inclusão (*DI*), a relação é uma **relação primária (RP)**. Isto significa que a relação não depende da existência de outras relações. As *DIs* em que uma chave primária de uma *RP* aparece do lado direito serão denominadas **referências primárias**.

Se a chave primária de uma relação é formada pela concatenação de dois ou mais atributos[2] que aparecem do lado esquerdo de referências primárias, a relação é uma **relação secundária (RS)**. As *RS* servem como elo de ligação entre outras relações. As DIs que apresentam a chave de uma *RS* no seu lado direito são chamadas **referências secundárias**.

Se a chave de uma relação é composta por alguns atributos que aparecem do lado esquerdo de referências primárias ou secundárias e por outros atributos que não aparecem do lado esquerdo de nenhuma referência primária ou secundária, então a relação é uma **relação terciária (RT)**. As *DIs* que possuem do seu lado direito a chave primária de uma *RT* são denominadas **referências terciárias**.

Uma relação que não puder ser classificada como *RP*, *RS* ou *RT* é uma **relação quaternária (RQ)**. As *RQ* possuem na composição de sua chave primária pelo menos um atributo que aparece do lado esquerdo de uma referência terciária.

Como se pode perceber a classificação das relações é baseada na forma como as chaves primárias das relações estão contidas nas *DIs*. No entanto, alguns SGBDs relacionais como o SISTEMA R [Dat86] ou o ORACLE [ORA85] não exigem a presença de chaves primárias nas suas tabelas. A incompatibilidade aparente entre esses sistemas e a metodologia pode ser resolvida se o DBA local acrescentar ao esquema relacional as chaves primárias e alternativas que existiriam se as relações não permitissem tuplas duplicadas. Essas chaves seriam marcadas para que pudessem ser usadas na classificação e descartadas nos passos seguintes.

Segundo as *DIs* definidas na figura 3.16 para o esquema local mostrado nas figuras 1.11 e 1.12, as relações *Cliente*, *Empregado*, *Projeto* e *Cargo* são classificadas como *RP*, as relações *Contab-Proj*, *Cliente-Proj* e *Gerente-Proj* são classificadas como *RS*, a relação *Etapa-Projeto* como *RT* e a relação *Empreg-Proj* como *RQ*.

Os atributos são classificados, conforme a função que desempenham na relação, em

[2] O termo atributo está se referindo a atributos simples ou compostos

```
Empregado.cargo << Cargo.cod-cargo
Empregado.depto << Depto.cod-depto
Empreg-Proj.matric << Empregado.matricula
Empreg-Proj.num-etapa << Etapa-Projeto.num-etapa
Empreg-Proj.cod-projeto << Etapa-Projeto.cod-projeto
Etapa-Projeto.cod-projeto << Projeto.cod-projeto
Cliente-Proj.cod-cliente << Cliente.cod-cliente
Cliente-Proj.cod-projeto << Projeto.cod-projeto
Gerente-Proj.matric << Empregado.matricula
Gerente-Proj.cod-projeto << Projeto.cod-projeto
Contab-Proj.cod-depto << Depto.cod-depto
Contab-Proj.cod-projeto << Projeto.cod-projeto
```

Figura 3.16: Dependências de inclusão no esquema relacional das figuras 1.11 e 1.12

atributos chave-primária (ACP), atributos chave-alternativa (ACA) e atributos não-chave (ANC). Os ANC são subdivididos em atributos que entram na composição da chave primária da relação ($ANC1$) e os que não entram na composição da chave primária da relação ($ANC2$).

Definição de tipos de entidade e relacionamentos

Para se identitificar os tipos de entidade e os relacionamentos representados pelas relações são examinados os grupos obtidos no passo anterior na seguinte ordem: 1) RP; 2) RS; 3) RT; 4) RQ.

Relações Primárias

Cada RP origina um tipo de entidade no esquema ECR. Os atributos da relação tornam-se atributos do tipo de entidade criado. Para cada atributo do tipo de entidade será definido um *valueset* correspondente com mesmo nome, tipo e tamanho do atributo correspondente no esquema relacional. Os atributos do tipo de entidade que correspondem à chave primária ou a uma chave alternativa da relação serão identificados como tal no esquema ECR, através das cláusulas KEY e UNIQUE, respectivamente. Chaves primárias ou alternativas compostas por mais de um atributo geram atributos compostos no tipo de entidade.

Normalmente, os SGBDs relacionais trabalham com relações em 1FN, embora existam propostas para modelos relacionais que não apresentem essa limitação [Set86, SS86]. Para que as relações obedeçam a 1FN é necessário que os atributos multivalorados de um tipo de entidade sejam colocados em uma relação a parte juntamente com o atributo identificador da relação que representa esse tipo de entidade. A chave da relação usada para modelar um atributo multivalorado é formada pela concatenação do próprio atributo multivalorado com o atributo correspondente à chave da relação que representa o tipo de entidade.

Alguns SGBDs relacionais aceitam valores nulos para os atributos. No modelo ECR essa possibilidade é expressa atribuindo-se à cardinalidade mínima do atributo o valor zero. Se a cardinalidade mínima for omitida no esquema ECR, então o atributo não pode ter valores nulos.

Se existir uma *DI* entre os atributos identificadores de duas relações primárias, então será criado um relacionamento 1:1 entre os tipos de entidade que representam essas relações. A participação do tipo de entidade que representa a relação cuja chave primária aparece do lado esquerdo da *DI* é total e funcional. A participação do outro tipo de entidade é parcial e funcional.

As correspondências entre os tipos de entidade gerados e as relações originais devem ser armazenadas no DD local para que possam ser utilizadas na transformação de operações CORDAS em operações em uma LMD relacional. A figura 3.17 mostra parte do esquema ECR gerado a partir do esquema relacional das figuras 1.11 e 1.12. A figura 3.18 mostra os mapeamentos armazenados no DD local.

```
DEFINE VALUESET matricula AS INTEGER RANGE 0:99999.
                    .
                    .
                    .
DEFINE ENTITY-TYPE Empregado
       ATTRIBUTES
           matricula VALUESET matricula KEY,
           nome      VALUESET nome,
           salario   VALUESET salario,
           cargo     VALUESET cargo,
           depto     VALUESET depto.
```

Figura 3.17: Trecho de esquema ECR obtido pelo exame de relações primárias

Relações Secundárias

Entrada	Tipo	Relação
Empregado	TE	Empregado

Figura 3.18: Mapeamento gerado durante o exame de relações primárias

Cada *RS* pode originar um tipo de entidade ou um relacionamento. Se a chave primária ou um atributo não contido na chave primária aparecer em uma *DI*, então a *RS* é convertida em um tipo de entidade fraca. Caso contrário, a *RS* origina um relacionamento que associa os tipos de entidade referenciados pelos atributos que compõem a sua chave primária. Os atributos da relação tornam-se atributos do tipo de entidade ou do relacionamento gerado.

Se a *RS* for convertida em um tipo de entidade fraca, cria-se também um relacionamento associando o tipo de entidade fraca e os tipos de entidade referenciados pelos atributos que compõem a chave primária da *RS*. A participação do tipo de entidade fraca no relacionamento é definida como específica e funcional, caracterizando a sua dependência em relação aos demais tipos de entidade que participam do relacionamento. A participação dos outros tipos de entidade é definida como parcial e não funcional.

No caso da *RS* corresponder a um relacionamento, as dependências funcionais entre os atributos componentes da chave dessa relação podem indicar se o tipo de participação dos tipos de entidade envolvidos no relacionamento é funcional ou não. Por exemplo, se a chave da *RS* examinada é composta pelos atributos x, y e z que referenciam os tipos de entidade $E1$, $E2$ e $E3$, respectivamente, e existe a dependência funcional $x \rightarrow yz$, então a participação do tipo de entidade $E1$ deve ser funcional. Todavia, para evitar o tratamento de dependências funcionais, pode-se assumir que a participação dos tipos de entidade seja não funcional e deixar para o passo de refinamento e validação do esquema ECR as correções necessárias. O tipo de participação total ou parcial dos tipos de entidade pode ser avaliado diretamente das *DIs*. Se existir uma *DI* da relação correspondente ao tipo de entidade para a *RS* que deu origem ao relacionamento, a participação do tipo de entidade é total. Caso contrário é parcial.

Devem ser guardados no DD local as correspondências entre os tipos de entidade e relacionamentos gerados e as relações originais. A figura 3.19 apresenta parte do esquema ECR obtido através do exame das *RSs* do esquema relacional definido nas figuras 1.11 e 1.12. A figura 3.20 mostra os mapeamentos incluídos no DD local.

```
DEFINE RELATIONSHIP Cliente-Proj FROM
        Cliente, Projeto
        ATTRIBUTES cod-cliente VALUESET cod-cliente,
                   cod-projeto VALUESET cod-projeto.
```

Figura 3.19: Trecho de esquema ECR obtido pelo exame de relações secundárias

Entrada	Tipo	Relação	Dependências de Inclusão
Cliente-Proj	Rel. Bin.	Cliente-Proj	Cliente-Proj.cod-cliente << Cliente.cod-cliente, Cliente-Proj.cod-projeto << Projeto.cod-projeto

Figura 3.20: Mapeamento gerado durante o exame de relações secundárias

Relações Terciárias

Uma *RT* pode representar um tipo de entidade fraca, um atributo multivalorado de um tipo de entidade ou um relacionamento. Na primeira opção, os atributos que compõem a chave primária e não aparecem do lado esquerdo de nenhuma *DI* são considerados identificadores próprios do tipo de entidade fraca. Na segunda opção, esses mesmos atributos são considerados atributos multivalorados do tipo de entidade referenciado pelos outros atributos que compõem a chave. Na terceira opção, tais atributos são considerados como chaves primárias de tipos de entidades implicitamente referenciadas no projeto e que podem ser explicitadas no esquema ECR. O problema é que não existem critérios objetivos genéricos para se decidir qual o conceito representado. A decisão depende do conhecimento do contexto da aplicação e, portanto, deve ser tomada pelo DBA local.

Se o DBA caracterizar a *RT* como um tipo de entidade fraca, então todos os atributos da relação tornam-se atributos do tipo de entidade criado. É gerado também um relacionamento que associa o tipo de entidade fraca com os tipos de entidade referenciados pelos atributos que compõem a chave da *RT*. A participação do tipo de entidade fraca no relacionamento é específica e funcional enquanto que a participação dos demais é parcial e não funcional.

Se, por outro lado, o DBA considerar que a relação está sendo usada para modelar um atributo multivalorado de um tipo de entidade, então esse atributo deve ser incluído no tipo de entidade e o DBA deve informar qual a cardinalidade máxima permitida para ele.

Finalmente, se o DBA decidir que a relação representa um relacionamento, as seguintes ações são executadas. Em primeiro lugar, é criado um tipo de entidade no esquema ECR para cada atributo componente da chave que não aparece do lado esquerdo de nenhuma referência primária ou secundária. O único atributo de um tipo de entidade criado é o próprio atributo que deu origem a ele. Em segundo lugar, é criado um relacionamento de grau k, $k \geq 2$, associando os k tipos de entidade referenciados na chave primária da *RT* examinada. A participação de cada tipo de entidade no relacionamento é parcial e não funcional. Os atributos da *RT* tornam-se atributos do relacionamento criado.

As *DIs* que referenciam no seu lado direito os atributos que motivaram a criação dos tipos de entidade devem ser substituídas por *DIs* que passam a referenciar as relações implicitamente identificadas por esses atributos, que foram explicitadas no esquema ECR. Assim, se o atributo a da *RT* $R1(a,b,c,d)$ deu origem ao tipo de entidade E, então a *DI* $S(x) \ll R1(a)$ deve ser substituída pela *DI* $S(x) \ll E(a)$. Essa substituição é importante porque permite que essas dependências de inclusão sejam tratadas como referências primárias nos próximos passos da metodologia.

Nas figuras 3.21 e 3.22 são apresentadas a tradução da *RT Etapa-projeto* da figura 1.12, interpretada como um tipo de entidade fraca, e os mapeamentos gerados no DD local a partir dessa tradução.

```
DEFINE VALUESET descr-etapa AS STRING 40.
                    .
                    .
                    .
DEFINE ENTITY-TYPE Etapa-Projeto
       ATTRIBUTES
             etapa-projeto-id COMPOUND
                (cod-projeto VALUESET cod-projeto,
                 num-etapa   VALUESET num-etapa) KEY,
             descr-etapa     VALUESET descr-etapa,
             data-prev-inic  VALUESET data-prev-inic,
             data-prev-fim   VALUESET data-prev-fim,
             data-inic       VALUESET data-inic,
             data-fim        VALUESET data-fim.

DEFINE RELATIONSHIP Cronog-Projeto FROM
       Projeto,
       Etapa-Projeto SPECIFIC MINIMUM PARTICIPATION 1
                              MAXIMUM PARTICIPATION 1.
```

Figura 3.21: Trecho de esquema ECR obtido pelo exame de relações terciárias

Entrada	Tipo	Relação	Dependências de Inclusão
Etapa-projeto	entid.fraca (TE2)	Etapa-projeto	Etapa-projeto.cod-projeto << Projeto.cod-projeto

Entrada	Tipo	Entidades Participantes
Cronog-Projeto	Relac.fraco	Projeto, Etapa-Projeto

Figura 3.22: Mapeamentos gerados durante o exame de relações terciárias

Relações Quaternárias

As *RQ* são tratadas da mesma forma que as relações terciárias que representam relacionamentos. É importante destacar, no entanto, que a separação entre *RT* e *RQ* é essencial para a metodologia proposta. As *RQ* precisam ser examinadas após as *RT*, uma vez que as primeiras referenciam as últimas e, portanto, o resultado das modificações nas dependências de inclusão que envolvem as RTs são importantes no exame das *RQ*.

As figuras 3.23 e 3.24 mostram, respectivamente, a tradução da *RQ Empreg-Proj* da figura 1.8 e os mapeamentos armazenados no DD local.

```
DEFINE VALUESET horas-trab AS INTEGER.

DEFINE RELATIONSHIP Empreg-Proj FROM
            Empregado,
            Etapa-projeto
                ATTRIBUTES
                   empreg-proj-id COMPOUND
                         (matric VALUESET matricula,
                          cod-projeto VALUESET cod-projeto,
                          num-etapa   VALUESET num-etapa) KEY,
                   horas-trab VALUESET horas-trab.
```

Figura 3.23: Trecho de esquema ECR obtido pelo exame de relações quaternárias

Entrada	Tipo	Relação	Dependencias de Inclusão
Empreg-Proj	Relac.	Empreg-Proj	Empreg-Proj.matricula << Empregado.matricula, Empreg-Proj.cod-projeto << Etapa-proj.cod-projeto, Empreg-Proj.num-etapa << Etapa-proj.num-etapa,

Figura 3.24: Mapeamento gerado durante o exame de relações quaternárias

Chaves estrangeiras

Dependências de inclusão envolvendo atributos não-chave (*ANC*1 e *ANC*2) de relações que originaram tipos de entidade podem indicar relacionamentos entre esses tipos de entidade. As *DIs* em que aparecem *ANC*1 do lado esquerdo foram consideradas durante

o exame das *RS*, *RT* e *RQ*. Nesse momento, são tratadas as *DIs* em que atributos *ANC1* ou *ANC2* ou *ACA* aparecem do lado direito e as *DIs* em que atributos *ANC2* ou *ACA* aparecem do lado esquerdo.

Cada uma das *DIs* examinada dá origem a um relacionamento binário associando os tipos de entidade referenciadas do seu lado esquerdo e do seu lado direito. Se o atributo do tipo de entidade referenciado do lado esquerdo da *DI* pode conter valores nulos, a participação do tipo de entidade em questão no relacionamento é funcional e parcial. Caso contrário a participação é funcional e total. Para o tipo de entidade referenciado no lado direito de uma *DI* é atribuída uma participação não funcional e parcial.

A figura 3.25 mostra o relacionamento entre as relações *Empregado* e *Cargo* da figura 1.11. A figura 3.26 mostra os mapeamentos armazenados no DD local.

```
DEFINE RELATIONSHIP Cargo-Empregado FROM
            Empregado MINIMUM PARTICIPATION 1
                      MAXIMUM PARTICIPATION 1,
            Cargo.
```

Figura 3.25: Trecho de esquema ECR obtido pelo exame de atributos não chave

Entrada	Tipo	Relação	Relação Referenciada	Chave Estrang.	Chave Referenciada
Cargo-Empregado	chave estrangeira	Empregado	Cargo	cargo	cod-cargo

Figura 3.26: Mapeamento gerado durante o exame de atributos não chave

Definição dos nomes de conexão

Neste passo da metodologia são definidos pelo DBA local e pelo DBA do SBDH, os nomes de conexão que serão associados à participação de cada classe nos relacionamentos.

Após a inclusão dos nomes de conexão no esquema ECR devem ser definidos os mapeamentos entre eles e as relações no esquema relacional. Esses mapeamentos serão utilizados para transformar os caminhos de busca expressos na linguagem GORDAS em referências às relações na linguagem utilizada pelo SGBD relacional. Os mapeamentos em questão são obtidos examinando-se a definição dos relacionamentos no esquema ECR.

Seja $R = \{R_1, R_2, \cdots, R_n\}$ o conjunto dos relacionamentos gerados no esquema ECR.

Seja $R_i = \{(E_1, na_1, nb_1), (E_2, na_2, nb_2), \cdots, (E_n, na_n, nb_n)\}$, $1 \leq i \leq n$, um relacionamento qualquer de R que associa as classes $E_1, E_2, \cdots, E_n$ através dos nomes de conexão na_i e nb_i.

Sejam na_j o *nome-conexão-1* e nb_j o *nome-conexão-2* definidos para a classe E_j, $1 \leq j \leq n$, que participa do relacionamento R_i. Para cada E_j devem ser gerados os seguintes mapeamentos:

1. Um mapeamento do caminho definido pela expressão $<of\ E_j>$ para a relação correspondente a E_j;

2. Um mapeamento do caminho definido pela expressão $<na_j\ of\ E_j>$ para a relação correspondente a R_i no esquema relacional, se tal relação existir. Se o relacionamento R_i é binário deve ser gerado um mapeamento adicional da expressão $<na_j\ of\ E_j>$ para a relação correspondente à outra classe E_k que participa do relacionamento. A classe E_k pode ser inclusive igual a E_j, no caso de R_i ser um auto-relacionamento.

3. Um mapeamento do caminho definido pela expressão $<na_j\ of\ E_j>$ para a relação referenciada pela chave estrangeira existente na relação correspondente a E_j, caso o relacionamento R_i tenha sido definido pela presença de uma chave estrangeira. A relação referenciada pela chave estrangeira pode ser obtida na entrada do DD local que guardou a correspondência entre o relacionamento R_i e os construtores do esquema relacional (ver figura 3.26).

4. Mapeamentos do caminho definido por cada expressão $<nb_k\ of\ na_j\ of\ E_j>$, $1 \leq k \leq n$, $k \neq j$, para cada relação correspondente a E_k, se R_i é um relacionamento que envolve mais de duas classes.

As figuras 3.27 e 3.28 apresentam, respectivamente, parte de um esquema ECR contendo os nomes de conexão e os mapeamentos que serão guardados no DD local. O leitor deve observar que os mapeamentos de caminhos como $<departamento\ of\ empregados\ of\ Cargo>$ embora não declarados explicitamente no DD local, podem ser facilmente obtidos com a substituição da subexpressão $<empregados\ of\ Cargo>$ pela relação equivalente, no caso a relação *Empregado*.

Refinamento e validação

No último passo da metodologia, o DBA local e o DBA do SBDH devem examinar o esquema obtido até o passo anterior para avaliar a necessidade de inclusão ou modificação

```
DEFINE RELATIONSHIP Cargo-Empregado FROM
                Cargo     (empregados, cargo),
                Empregado (cargo, empregados)
                          MIMIMUM PARTICIPATION 1,
                          MAXIMUM PARTICIPATION 1.

DEFINE RELATIONSHIP Departamento-Empregado FROM
                Departamento (empregados, departamento),
                Empregado    (departamento, empregados)
                          MIMIMUM PARTICIPATION 1
                          MAXIMUM PARTICIPATION 1.
```

Figura 3.27: Nomes de conexão incluídos no passo 3 da metodologia Relac.-ECR

Caminho ECR	Relação
OF Empregado	Empregado
OF Departamento	Departamento
OF Cargo	Cargo
OF Projeto	Projeto
⋮	⋮
departamento OF Empregado	Departamento
cargo OF Empregado	Cargo
empregados OF Departamento	Empregado
empregados OF Cargo	Empregado

Figura 3.28: Mapeamentos gerados no passo 3 da metodologia Relac.-ECR

de certas informações nesse esquema. Assim como na tradução Rede-ECR, também na tradução Relacional-ECR é importante garantir que todas as informações relevantes não contidas explicitamente no esquema local sejam inseridas no esquema componente para permitir uma maior independência dos dados em relação às aplicações dos usuários federados.

A figura 3.29 mostra o diagrama ECR do esquema componente obtido pela aplicação da metodologia ao esquema local relacional definido nas figuras 1.11 e 1.12.

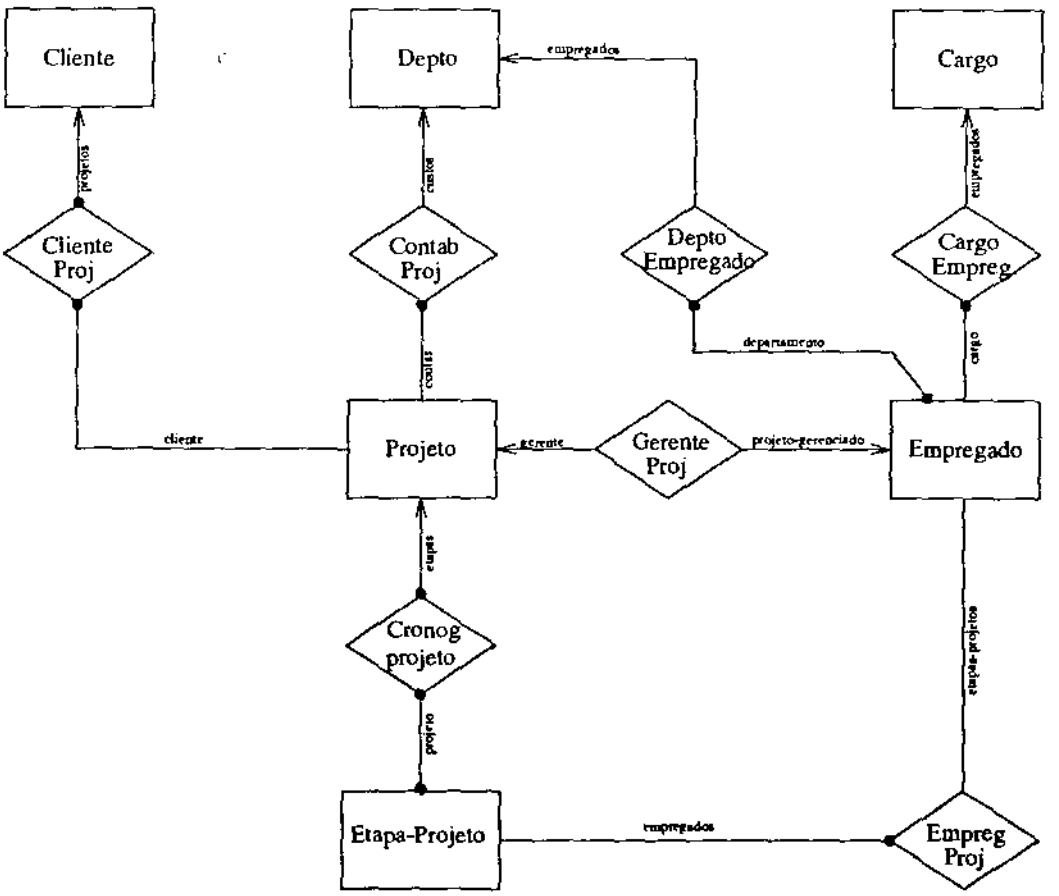

Figura 3.29: Diagrama do esquema componente obtido pela metodologia Relac.-ECR

3.4 Integração de esquemas

Após a tradução dos esquemas locais para o MDC é feita a integração desses esquemas obtendo se um ou mais esquemas federados. A integração de esquemas é um processo complexo que pode envolver a resolução de conflitos estruturais e semânticos existentes entre

os esquemas componentes [dO92]. Várias metodologias têm sido propostas para suportar o processo de integração [DH84, BOT86, BL84, NEL86, Sou86, DAT87, SLCN90, LNE89, SPY92]. Algumas dessas metodologias [NEL86, SLCN90, LNE89] se aplicam inclusive ao Modelo ECR, adotado como modelo comum neste trabalho. O detalhamento dessas metodologias está fora do escopo desta dissertação e, por conseguinte, será assumido que o(s) esquema(s) federado(s) e os mapeamentos entre ele(s) e os esquemas componentes já estejam disponíveis. Contudo, as seguintes premissas devem ter sido respeitadas pela metodologia empregada:

1. A aglutinação de tipos de entidade (ou relacionamentos), contidos em esquemas componentes diferentes, em um único tipo de entidade (ou relacionamento) no esquema federado foi feita se, e somente se, os tipos de entidade (relacionamentos) dos esquemas componentes representavam o mesmo conjunto de entidades (instâncias de relacionamento) em um *estado válido* do SBDH. Entenda-se como estado válido do SBDH o conjunto de todos os esquemas e dados locais dos SBDs componentes em um determinado instante do tempo. Assim, a aglutinação de um tipo de entidade E_1, presente no esquema componente de um SBD S_1, com o tipo de entidade E_2, presente no esquema componente de um SBD S_2, acontece se, e somente se, em qualquer instante do tempo todas as entidades de E_1 representadas em S_1 estiverem representadas como entidades de E_2 no SBD $S2$ e vice-versa. Isto pode acontecer mesmo que os atributos dos dois tipos de entidade não sejam exatamente os mesmos. O mesmo raciocínio se aplica aos relacionamentos;

2. Tipos de entidade dos esquemas componentes que representavam diferentes subconjuntos de um mesmo conjunto de entidades do mundo real foram modelados no esquema federado através de categorias de generalização ou especialização;

3. A integração de conceitos semânticos idênticos mas representados por construtores diferentes (conflitos estruturais) em esquemas componentes distintos foi feita através do construtor mais abrangente. Assim, um mesmo conceito representado em um esquema componente por um tipo de entidade e em outro esquema componente por um atributo foi integrado como um tipo de entidade no esquema federado;

1. As redundâncias de atributos-chave encontradas nos esquemas componentes foram eliminadas no esquema federado para que se pudesse obter esquemas externos mais claros e menos sujeitos a problemas com relação as operações de atualização de dados. Basicamente essas redundâncias são causadas pela presença de chaves estrangeiras nos esquemas componentes dos SBDs relacionais que compõem o SBDH.

Para simplificar a discussão do problema da identificação de dados replicados nos SBDs componentes, será também assumido que os atributos identificadores dos tipos de

entidade do esquema federado estão replicados em todos os esquemas componentes em que as entidades daquele tipo estão definidas.

A figura 3.30 mostra o diagrama ECR do esquema federado obtido pela integração dos esquemas componentes das figuras 3.15 e 3.29. As figuras 3.31 e 3.32 apresentam os mapeamentos obtidos durante a integração.

3.5 Tradução ECR-Rede

3.5.1 Considerações gerais

A tradução de esquemas federados no modelo ECR para esquemas externos no modelo rede pode ser vista, em termos estruturais, como o inverso da tradução de esquemas locais rede para esquemas componentes ECR. Entretanto, a tradução ECR-Rede apresenta algumas características particulares.

Em primeiro lugar, o maior poder de representação semântica do modelo ECR em relação ao modelo rede faz com que muitas restrições impostas sobre o esquema federado como, por exemplo, restrições semânticas e certas restrições sobre o tipo de participação das entidades nos relacionamentos não possam ser incluídas no esquema externo. Para garantir que estas restrições sejam obedecidas existem duas alternativas básicas: a) deixar que os usuários que atuam sobre o esquema externo incluam nos seus programas de aplicação rotinas para validação dessas restrições; ou b) verificar durante o processo de transformação das operações do nível externo para o nível federado e do nível federado para o nível local, se as restrições não são violadas. A primeira alternativa é bastante cômoda mas prejudica a independência dos dados em relação às aplicações, porque depende do conhecimento que os usuários têm da semântica dos dados e da precisão com que são construídos os programas de aplicação. A segunda alternativa exige um processamento complexo mas garante um tratamento mais seguro.

Em segundo lugar, o esquema federado pode conter objetos oriundos de SBDs que não utilizam o modelo rede. Logo, para esses objetos não existem informações sobre algumas características físicas que devem ser informadas no esquema externo rede. Para incluir estas informações no esquema externo serão feitas algumas inferências que serão posteriormente verificadas pelo DBA do SBDH na fase de refinamento do esquema obtido.

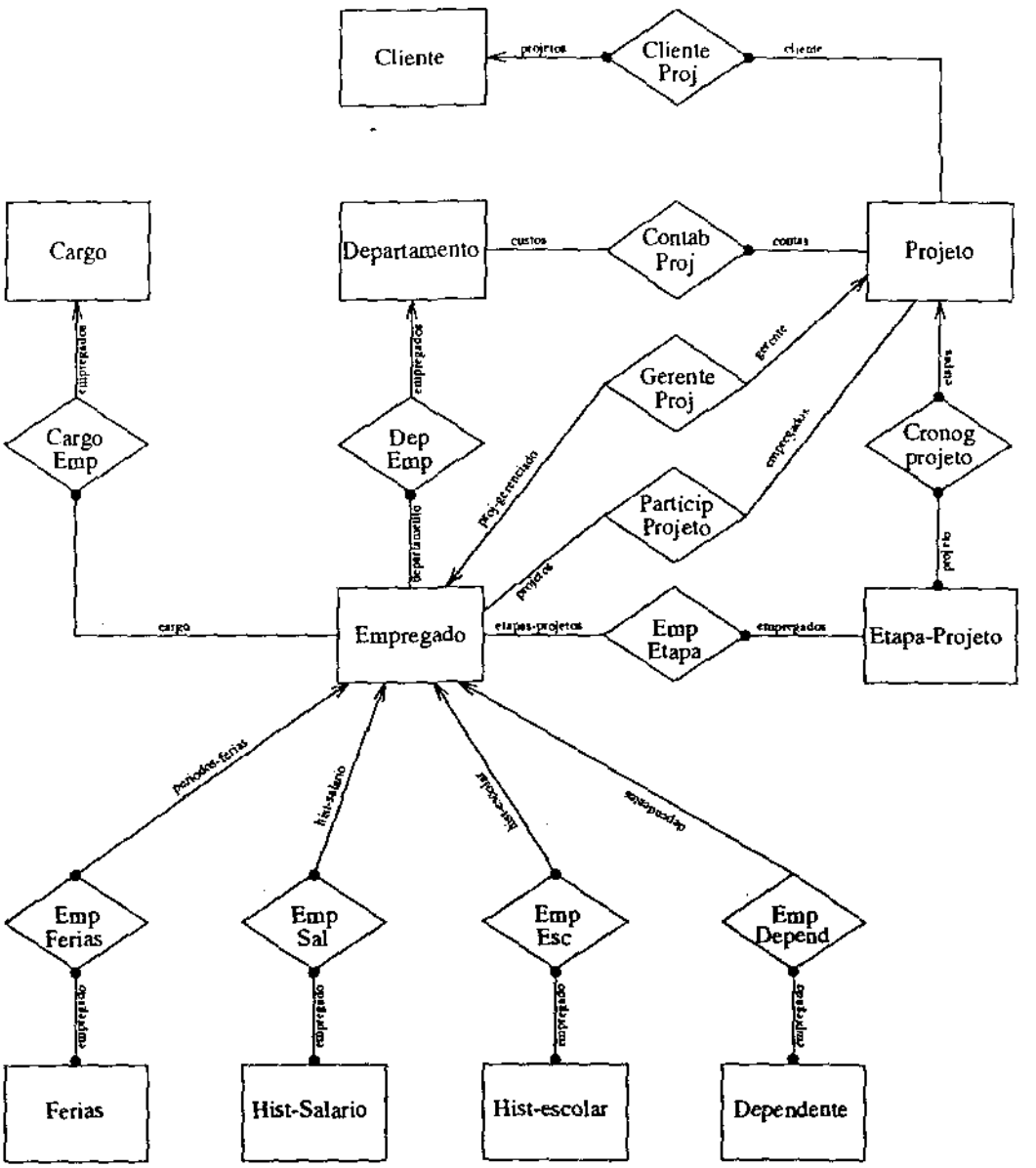

Figura 3.30: Diagrama ECR do esquema federado obtido na integração de esquemas

Construtor Esq.Federado	Construtor Esq.Componente 1 (SBD rede)	Construtor Esq.Componente 2 (SBD relacional)
Departamento	Departamento	Depto
Departamento.cod-depto	Departamento.cod-depto	Depto.cod-depto
Departamento.nome-depto	Departamento.nome-depto	Depto.nome
Departamento.gerente	Departamento.matric-gerente	
Departamento.gratif-gerente	Departamento.gratif-gerente	
Empregado	Empregado	Empregado
Empregado.matricula	Empregado.matricula	Empregado.matricula
Empregado.nome-empreg	Empregado.nome-empreg	Empregado.nome
Empregado.endereco	Empregado.endereco	
⋮	⋮	⋮
Empregado.salario-atual		Empregado.salario
Projeto	Projeto	Projeto
Projeto.cod-projeto	Projeto.cod-projeto	Projeto.cod-projeto
Projeto.nome-projeto	Projeto.nome-projeto	Projeto.nome-projeto
Projeto.valor-total	Projeto.valor-total	
⋮	⋮	⋮
Projeto.multa-atraso		Projeto.multa-atraso
Cargo	Cargo	Cargo
Cargo.cod-cargo	Cargo.cod-cargo	Cargo.cod-cargo
Cargo.nome-cargo	Cargo.nome-cargo	Cargo.nome-cargo
Cargo.salario-medio		Cargo.salario-medio
Cliente		Cliente
Etapa-projeto		Etapa-projeto
Ferias	Ferias	
Hist-salario	Hist-salario	
Hist-escolar	Hist-escolar	
Dependente	Dependente	
Dep-Emp	Dep-Emp	Depto-Empregado
Cargo-Emp	Cargo-Emp	Cargo-Empregado
Gerente-Proj	Gerente-Proj	Gerente-Proj
Cliente-Proj		Cliente-Proj

Figura 3.31: Mapeamento entre esquema federado e esquemas componentes

Construtor Esq.Federado	Construtor Esq.Componente 1 (SBD rede)	Construtor Esq.Componente 2 (SBD relacional)
Cronog-projeto		Cronog-projeto
Emp-Etapa		Empreg-Proj
Particip-Projeto	Particip-Projeto	
Contab-Proj		Contab-Proj
Emp-Ferias	Emp-Ferias	
Emp-Sal	Emp-Sal	
Emp-Esc	Emp-Esc	
Emp-Depend	Emp-Depend	

Figura 3.32: Mapeamento entre esquema federado e esquemas componentes (cont.)

3.5.2 Trabalhos correlatos

A representação dos dados através do MER ou de modelos similares é um recurso muito utilizado no projeto conceitual de aplicações em bancos de dados [EN89]. No entanto, como praticamente não existem SGBDs que implementam diretamente o modelo entidade-relacionamento é necessária uma tradução da representação conceitual para a representação no modelo de dados rede em que a aplicação será construída. Várias metodologias para essa tradução são propostas na literatura [DA83, DC83, EWH85, EN89, Lin89].

Dumpala e Arora

A Metodologia apresentada em [DA83] trata o modelo entidade-relacionamento sem extensões e baseia-se na correspondência entre tipo de entidade e *record-type* e entre relacionamento e *set-type*. Trata-se, na verdade, de uma metodologia muito simplificada que se propõe única e exclusivamente a obter uma representação diagramática alternativa para o MER.

Dogac e Chen

Dogac e Chen [DC83] discutem alguns problemas envolvidos na tradução de esquemas de um modelo ER estendido para esquemas rede. Os autores argumentam que certos tipos de informações previstas no esquema rede, como *LOCATION MODE* e *SET SE-*

LECTION, não podem ser obtidas do esquema ER usando-se uma regra geral, porque essas informações dependem da aplicação modelada. Além disso, eles mostram que a introdução de conectores para permitir a modelagem de relacionamentos N:M faz com que algumas restrições estruturais existentes no modelo ER, como a participação total de um tipo de entidade no relacionamento, não sejam modeladas no esquema rede.

Dogac e Chen também apresentam sugestões para a tradução das generalizações e restrições de integridade presentes no esquema do modelo ER estendido.

Para as generalizações, é sugerido que seja criado um *set-type* cujo *record-type* mestre represente o conjunto dos objetos individualizados e o *record-type* membro o conjunto dos objetos generalizados. Essa abordagem difere de outra adotada em [EWH85] que sugere que o *record-type* mestre represente os objetos generalizados e o *record-type* membro os objetos individualizados. Segundo os autores, a vantagem da abordagem adotada por eles é a manutenção da cardinalidade 1:N nos *set-type* evitando-se a necessidade de restringir a cardinalidade 1:1 nos programas de aplicação. Entretanto, a abordagem adotada apresenta como desvantagem o fato de não modelar as restrições de dependência das subclasses, que representam os objetos individualizados, em relação às superclasses, que representam os objetos generalizados.

Para garantir as restrições de integridade referenciais e restrições de integridade semânticas existentes no esquema ECR é sugerido o uso de *Database Procedures* [TF76] que são procedimentos especiais construídos para verificar se determinadas condições são satisfeitas antes de uma atualização no banco de dados.

Elmasri, Weeldreyer e Hevner

Elmasri, Weeldreyer e Hevner [EWH85] ao apresentarem o modelo ECR também sugerem regras para o seu mapeamento para o modelo rede. Cada tipo de entidade ou categoria é convertido em um *record-type* cujos itens de dados correspondem aos atributos originais. Atributos multivalorados são modelados como itens de dados repetitivos. Uma categoria de generalização gera também um *set-type* que tem como *record-type* mestre o *record-type* correspondente à categoria e como *record-type* membros, os *record-type* correspondentes aos tipos de entidade básicos da categoria. Uma categoria ISA gera, da mesma forma, um *set-type* que tem como mestre o *record-type* correspondente ao tipo de entidade básico que formou a categoria e como membro o *record-type* que corresponde a própria categoria. Para ambos os tipos de categoria, o *set-type* representa um relacionamento 1:1 e essa restrição deve ser reforçada nos programas de aplicação. O tipo de inclusão-retenção para os *record-type* membros é definido como *AUTOMATIC MANDATORY* para que o próprio SGBD rede garanta a restrição de integridade referencial entre as entidades das

subclasses e as entidades da superclasse.

Um relacionamento binário 1:N entre as classes A e B é modelado diretamente como um *set-type* que tem como *record-type* mestre o *record-type* correspondente a A e como *record-type* membro o *record-type* correspondente a B. Um relacionamento 1:1 é representado da mesma forma. A única diferença é que a restrição em relação ao número de registros membros deve ser garantida nos programas de aplicação. Um relacionamento N:M é representado por um *record-type* conector juntamente com os *set-type* que associam cada classe participante no relacionamento (mestre) com o *record-type* conector (membro).

Elmasri e Navathe

O modelo ER estendido usado em [EN89] apresenta uma maior variedade de relacionamentos *subclasse-superclasse*. Esses relacionamentos são classificados conforme os critérios de formação das superclasses ou subclasses (definidos por predicado ou pelo usuário), conforme a participação ou não de uma mesma entidade em mais de uma subclasse de uma mesma superclasse (disjunto ou sobreposto) e conforme a obrigatoriedade ou não de uma entidade pertencente a superclasse pertencer a pelo menos uma subclasse (total ou parcial).

Os autores apresentam alternativas para o mapeamento desses tipos de relacionamento para o modelo rede. Algumas dessas alternativas convertem cada superclasse e subclasse em um *record-type* diferente associando-os através de um *set-type*. Outras aglutinam as subclasses e superclasses em um único *record-type* que contém, além dos atributos das subclasses e da superclasse, atributo(s) auxiliar(es) indicando qual(is) a(s) subclasse(s) que é(são) referenciada(s) por um registro específico do *record-type*. Outras ainda, utilizam um *set-type* com mais de um *record-type* membro, da mesma forma que [EW1185].

Ling

Em seu artigo, Ling [Lin89] propõe algumas regras para introduzir no esquema rede as informações referentes ao tipo de retenção (*SET RETENTION*), ao tipo de inserção (*SET INSERTION*) e ao tipo de seleção (*SET SELECTION*) dos registros membros nas *set-occurrence*.

Para um *set-type* obtido a partir da um relacionamento fraco, o tipo de retenção é *MANDATORY*, o tipo de inserção é *MANUAL* e o tipo de seleção é *BY APPLICATION*.

Para um *set-type* obtido a partir de um relacionamento binário 1:N ou 1:1, o tipo de retenção é *OPTIONAL*, o tipo de inserção é *MANUAL* e o tipo de seleção é *BY*

APPLICATION.

Para os set type *AC* e *BC* que representam um relacionamento N:M entre os tipos de entidade *A* e *B*, não necessariamente distintos, o tipo de retenção, inserção e seleção associados ao conector *C* são, respectivamente, *MANDATORY*, *AUTOMATIC* e *BY STRUCTURAL*. A escolha do tipo de seleção *BY STRUCTURAL* é baseada na premissa de que as chaves de *A* e *B* foram duplicadas em *C*.

Para os *set-type* A_iC, $1 \leq i \leq n$ e $n > 2$, que representam um relacionamento de grau maior do que dois, o tratamento é quase o mesmo adotado para os *set-type* que representam relacionamentos binários N:M. Apenas são mudados o tipo de inserção e seleção para os *set-type* que ligam o conector *C* com os *record-type* correspondentes a tipos de entidade com cardinalidade 1. Nesses *set-type*, a chave do *record-type* mestre não é duplicada no *record-type* membro, o tipo de retenção é *MANUAL* e o tipo de seleção é *BY APPLICATION*.

A metodologia proposta em [Lin89] também apresenta regras de conversão diferenciadas para relacionamentos ISA e atributos multivalorados. Dois tipos de entidade associados através de um relacionamento ISA são convertidos em um único *record-type* que recebe os atributos de ambos os tipos de entidade. Cada atributo multivalorado de um tipo de entidade *A* é colocado em um *record-type B* separado que recebe também a chave de *A* como atributo. O *record-type B* é ligado ao *record-type A* por um *set-type AB* que tem *A* como mestre e *B* como membro. O tipo de retenção, tipo de inserção e tipo de seleção associados ao *set-type AB* são *MANDATORY*, *MANUAL* e *BY STRUCTURAL*, respectivamente.

3.5.3 Metodologia proposta

A metodologia para tradução ECR-Rede que será proposta nesta dissertação, envolve não só a tradução de construtores de um modelo para outro, mas também os mapeamentos entre os *set-type* e os nomes de conexão que serão utilizados na transformação de operações sobre o esquema externo em operações GORDAS sobre o esquema federado. Os passos da metodologia são:

1. Exame dos tipos de entidade;
2. Exame das categorias;
3. Exame dos relacionamentos;
4. Refinamento e validação do esquema externo;

Exame dos tipos de entidade

Para cada tipo de entidade existente no esquema ECR é definido um *record-type* correspondente. Os atributos do tipo de entidade são convertidos em itens de dados com mesmas características de tipo e tamanho. Atributos compostos e atributos com cardinalidade máxima maior do que 1 (multivalorados) são traduzidos para itens de dados compostos e repetitivos (definidos pela cláusula *OCCURS*), respectivamente. Restrições de integridade que não puderem ser representadas no esquema rede devem ser verificadas durante o processo de transformação de operações rede em operações GORDAS.

Exame das categorias

As categorias de generalização e categorias ISA que aparecem no esquema federado foram geradas durante o processo de integração de esquemas para modelar os diferentes níveis de abstração usados pelos projetistas das aplicações contidas nos SBDs componentes.

Categorias ISA representam o fato de que em um esquema componente o conjunto de entidades representadas por um tipo de entidade é um subconjunto das entidades representadas por um tipo de entidade em outro esquema componente.

Categorias de generalização representam o fato de que um conjunto de entidades no mundo real está dividido em subconjuntos diferentes nos SBDs componentes. As categorias de generalização podem ser disjuntas (subconjuntos sem elementos em comum) ou sobrepostas (com alguns elementos em comum).

As categorias representam relacionamentos inter-esquema e, portanto, não faz sentido representá-los como *set-type* no esquema externo rede. Ao invés disso, cada tipo de entidade presente em uma categoria é convertida em um *record-type* com mesmo nome dentro do esquema alvo. Assim, os tipos de entidade que formam uma categoria de generalização no esquema federado são convertidos em *record-type* de mesmo nome no esquema externo. Os atributos de cada *record-type* são os atributos do tipo de entidade que o originou mais os atributos específicos da categoria de generalização. No caso de categorias ISA, tanto a categoria como o tipo de entidade que serviu como base para a formação da categoria são convertidos em *record-type* no esquema externo. Os atributos de cada *record-type* são os mesmos da categoria ou do tipo de entidade que o originou.

O relacionamento entre os registros de cada *record-type* gerado no esquema externo devem ser informados aos usuários para que eles possam associá-los no próprio programa de aplicação.

Exame dos relacionamentos

Nesta etapa da metodologia são examinados os relacionamentos existentes no esquema federado.

Cada relacionamento binário 1:1 que não possui atributos é convertido em um *set-type* com mesmo nome. Se uma das classes que participam do relacionamento tem participação total, então o *record-type* correspondente a essa classe será o *record-type* membro do *set-type* criado. Essa escolha permite que a participação total da classe seja reforçada no esquema rede através do tipo de retenção *MANDATORY* e do tipo de inserção *AUTOMATIC*. Se ambas as classes tem participação parcial, então qualquer uma delas pode ser escolhida como *record-type* membro do *set-type*. Nesse caso, o tipo de retenção é definido como *OPTIONAL* e o tipo de inserção como *MANUAL*. Um relacionamento 1:1 com atributos é convertido em um *record-type* conector se ambas as classes relacionadas têm participação parcial nesse relacionamento. Nesse caso, são usados dois *set-type*, cada um ligando um *record-type* que representa uma das classes participantes ao *record-type* que representa o relacionamento. O nome de cada *set-type* é igual ao nome de conexão associado à participação da classe correspondente ao *record-type* mestre no relacionamento examinado. Um relacionamento 1:1 com atributos em que pelo menos uma das classes têm participação obrigatória (total) recebe o mesmo tratamento dos relacionamentos 1:1 sem atributos. Nesse caso, os atributos do relacionamento são incluídos no *record-type* membro do *set-type*. A restrição que estabelece a presença de um único registro membro em cada *set-occurrence* deve ser garantida no processo de transformação de operações.

Cada relacionamento binário 1:N que não possui atributos é convertido em um *set-type* com mesmo nome. O *record-type* correspondente à classe com participação funcional será o *record-type* membro do *set-type* criado e a outra classe será o *record-type* mestre. O tipo de participação do *record-type* membro será *AUTOMATIC MANDATORY* se a classe correspondente tiver participação total no relacionamento e, *MANUAL OPTIONAL* se a classe correspondente tiver participação parcial. Cada relacionamento 1:N com atributos é convertido em um *record-type* conector se a classe com participação funcional no relacionamento não tiver participação total no mesmo. Nesse caso, são criados dois *set-type* da mesma forma como nos relacionamentos 1:1 com atributos em que ambas as classes participantes têm participação parcial. Os atributos do relacionamento são colocados no *record-type* conector. Relacionamentos 1:N com atributos são convertidos em *set-type* com mesmo nome se a classe com participação funcional também tem participação total. Nesse caso, os atributos do relacionamento são incluídos no *record-type* membro do *set-type*.

Cada relacionamento binário N:M ou relacionamento de grau maior do que 2, é convertido em um *record-type* conector com mesmo nome. São criados também tantos *set-type* quantos forem as classes participantes no relacionamento. Cada um desses *set-type* liga

um *record-type* correspondente a uma das classes participantes (mestre) com o *record-type* conector (membro). O nome de cada *set-type* é igual ao *nome-conexão-1* associado à participação da classe, correspondente ao *record-type* mestre, no relacionamento examinado. O tipo de participação do *record-type* conector em cada *set-type* é *AUTOMATIC MANDATORY*. Nos relacionamentos de grau maior do que dois algumas classes podem ter participação funcional. Nesse caso, a restrição quanto ao número de registros membros no *set-type*, cujo *record-type* mestre corresponde àquela classe, deve ser mantida durante o processo de transformação de operações.

Cada auto-relacionamento é convertido em um *record-type* conector com mesmo nome. São criados dois *set-type*, um para cada papel desempenhado pelas entidades da classe no auto-relacionamento. O nome de cada *set-type* é o *nome-conexão-1* definido para o papel associado ao *set-type*. O tipo de participação do *record-type* conector em ambos os *set-type* é *AUTOMATIC MANDATORY*.

O tipo de seleção BY APLICATION é definido para identificar as *set-ocurrence* dos *set-type* criados nesse passo da metodologia. Desse modo, o programa construído sobre o esquema externo deve localizar explicitamente os registros mestres de cada *set-type* através de um comando FIND da linguagem de manipulação de dados rede.

O ordenamento dos registros membros no *set-type* (cláusula SET ORDER) é definido de acordo com os seguintes critérios:

1. Se o relacionamento no esquema federado que deu origem ao *set-type* estiver definido no esquema do SBD rede local então a cláusula SET ORDER é igual àquela definida no esquema do SBD local;

2. Se o relacionamento não estiver estiver definido no esquema componente do SBD rede local mas estiver definido em outros esquemas componentes de SBDs remotos rede, então a cláusula SET ORDER é igual à cláusula SET ORDER contida em algum dos esquemas desses SBDs remotos. O SBD remoto escolhido será o primeiro a ser pesquisado durante a navegação no *set-type* definido no esquema externo rede, garantindo a semântica da aplicação construída pelo usuário. A identificação do SBD escolhido deve ser guardada no mapeamento entre o esquema externo e o esquema federado.

3. Se o relacionamento no esquema federado estiver definido apenas em SBDs componentes relacionais, então a cláusula SET ORDER é definida como SYSTEM DEFAULT indicando a ausência de um ordenação lógica na inserção e recuperação dos registros no *set-type*.

Os mapeamentos entre os construtores criados no esquema externo rede e os relaciona-

mentos do esquema federado e entre os caminhos de navegação definidos nos *set-type* do esquema externo e os nomes de conexão definidos no esquema federado devem ser mantidos no DD local para que possam ser utilizadas durante a transformação de operações construídas sobre o esquema externo em operações GORDAS sobre o esquema federado. Os nomes de conexão são mapeados da seguinte forma. O caminho que sai de um registro mestre e percorre os membros é mapeado para o *nome-conexão-1* associado à participação da classe que corresponde ao *record-type* mestre enquanto que o caminho que sai do registro membro e chega ao registro mestre é mapeado para o *nome-conexão-2* da classe que corresponde ao *record-type* mestre.

As regras para conversão de relacionamentos apresentadas até aqui são suficientes para modelar no esquema externo rede as restrições que se referem ao tipo de participação de uma classe no relacionamento, desde que essa classe corresponda ao *record-type* membro do *set-type* que representa o relacionamento. Porém, a participação total de uma classe em um relacionamento não pode ser modelada no esquema rede se essa classe foi convertida no *record-type* mestre do *set-type* que representa o relacionamento. Tal situação acontece, por exemplo, em relacionamentos binários em que ambas as classes têm participação total. Nos SBDs rede, a participação total de um *record-type* mestre é mantida pelo programador da aplicação, que ao incluir um registro pertencente ao *record-type* mestre inclui também um registro do *record-type* membro declarado com tipo de retenção MANDATORY e tipo de inserção AUTOMATIC. Assim, ao se incluir o registro membro o relacionamento obrigatório é garantido pelo próprio SGBD rede. Portanto, é exigida a inclusão de dois registros para que a restrição seja mantida. No caso de SBDIIs em que os usuários atuam sobre um esquema externo rede usando uma linguagem de manipulação de dados procedimental, a mesma limitação se aplica, apesar de existir no esquema federado restrições de integridade estruturais que modelam a participação total das duas classes participantes. A manutenção automática desse tipo de restrição pelos processadores de transformação de operações é difícil porque as operações na LMD rede são executadas uma a uma e, no momento da inclusão do registro mestre, não se sabe quais são os dados do registro membro que devem ser associados ao registro incluído.

Refinamento e validação do esquema

No último passo da metodologia, o esquema externo rede obtido é apresentado ao DBA para que ele possa modificar certas informações como, por exemplo, nomes de *set-type* ou cláusulas relacionadas com aspectos de implementação, como as cláusulas LOCATION MODE e AREA. As modificações feitas podem implicar em mudanças nos mapeamentos gerados nos passos anteriores. É importante ressaltar que essas modificações não afetam a correspondência conceitual entre os esquemas externo e federado, mas simplesmente ajustam, quando necessário, informações ligadas às características físicas do modelo rede

que foram inferidas nos passos anteriores.

A figura 3.33 apresenta o diagrama do esquema rede obtido pela aplicação da metodologia ao esquema federado da figura 3.30.

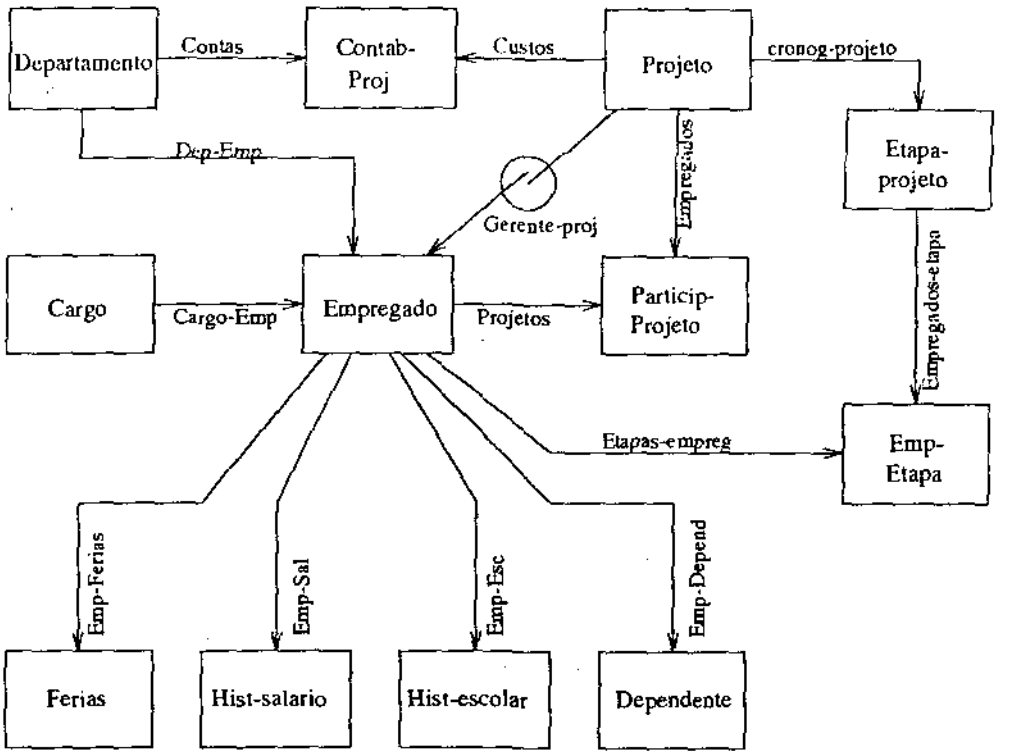

Figura 3.33: Diagrama do esquema externo rede obtido pela metodologia ECR-Rede

3.6 Tradução ECR-Relacional

3.6.1 Considerações gerais

A tradução ECR-Relacional é um pouco mais simples do que a tradução ECR-Rede porque um esquema externo no modelo relacional não menciona características relacionadas com a implementação física dos dados como um esquema externo no modelo rede faz. Contudo, o modelo relacional também não consegue representar todas as restrições estruturais e semânticas contidas no esquema ECR e, por conseguinte, essas restrições precisam ser verificadas durante a transformação das operações construídas sobre o esquema externo relacional para operações GORDAS equivalentes e/ou durante a transformação de operações GORDAS em operações sobre os dados dos SBDs componentes.

Uma outra observação importante sobre a tradução ECR-Relacional diz respeito à conversão de relacionamentos e categorias em relações. Como o modelo relacional não possui construtores específicos para modelar relacionamentos, essa conversão exige a duplicação de atributos de algumas relações em outras para que possam ser executadas operações de *junção* (*join*) entre elas. Os mapeamentos entre as operações de *junção* e os nomes de conexão e entre as operações de *junção* e as categorias são essenciais para a transformação de operações construídas sobre os esquema externo relacional em operações sobre o esquema federado.

3.6.2 Trabalhos correlatos

Vários trabalhos discutem a tradução do modelo entidade-relacionamento e seus assemelhados para o modelo relacional no contexto de projeto lógico de bancos de dados [CNC83, DA83, DC83, EWH85, AP87, EN89].

Dumpala e Arora

Em [DA83] é apresentado um algoritmo bastante simplificado para conversão de construtores básicos do MER. Cada tipo de entidade é traduzido para uma relação correspondente. Cada relacionamento de grau k ($k \geq 2$) ou relacionamento binário N:M gera uma relação correspondente cuja chave é composta pela concatenação das chaves dos tipos de entidade que participam do relacionamento. Cada relacionamento binário 1:N é modelado introduzindo-se na relação, que representa o tipo de entidade com cardinalidade N, a chave do tipo de entidade com cardinalidade 1. Essa regra também se aplica aos relacionamentos binários 1:1, apenas com a observação de que qualquer relação pode receber a chave da outra. Um tipo de entidade fraca é convertido em uma relação cuja chave é formada pela concatenação da sua própria chave com a chave do tipo de entidade do qual ele é dependente.

Chung, Nakamura e Chen

Chung, Nakamura e Chen [CNC83] propõem um algoritmo para normalizar um diagrama ER antes da tradução para o modelo relacional. Essa normalização visa garantir que as dependências entre os dados sejam preservadas na tradução. A forma normal proposta pelos autores é denominada *Forma Normal Entidade-Relacionamento* (FNER). A relação construída sobre os atributos de uma entidade (*relação entidade*) ou de um relacionamento (*relação relacionamento*) está em FNER se e somente se todos os seus atributos são

monovalorados e todas as dependências funcionais não triviais nela definidas incluem pelo menos um atributo identificador como atributo que determina a funcionalidade. É demonstrado que as relações em FNER estão também em forma normal Boyce-Codd (FNBC), mas que o inverso não é verdadeiro.

O algoritmo de normalização parte dos atributos, das dependências funcionais entre os atributos e dos relacionamentos entre os *valuesets* dos atributos multivalorados e obtém os tipos de entidade e relacionamentos em FNER que serão convertidos diretamente em relações em FNBC. Um aspecto interessante do algoritmo é que os relacionamentos são sempre convertidos como relações separadas no esquema relacional gerado.

Dogac e Chen

Dogac e Chen [DC83] discutem a inserção no modelo relacional de restrições de integridade referenciais existentes entre as entidades que participam nos relacionamentos e nas abstrações de generalização e agregação de um modelo ER estendido. Os autores sugerem que sejam criados *gatilhos* (*triggers*) no esquema relacional para definir ações que devem ser executadas sempre que um determinado evento de atualização de dados ocorrer.

Azar e Pichat

Azar e Pichat [AP87] consideram a tradução de um modelo ER estendido para o modelo relacional. As extensões tratadas pelos autores são generalização, agregação de relacionamentos, restrições quanto à cardinalidade mínima e máxima de atributos e ao tipo de participação de entidades nos relacionamentos, dependências funcionais entre atributos e tipos de entidade e dependências entre valores de atributos multivalorados.

O algoritmo proposto converte um diagrama nesse modelo ER estendido para uma *relação universal (RU)* com definição explícita das suas dependências funcionais (*DFs*), de inclusão (*DIs*) e de junção (*DJs*). Algumas *DFs* são inferidas a partir da cardinalidade dos relacionamentos e outras a partir das *DFs* entre os atributos e entre os tipos de entidade que são informadas pelo projetista do diagrama ER. As *DJs* são obtidas à partir das dependências entre os *valuesets* dos atributos multivalorados. As *DIs* são definidas a partir dos relacionamentos e generalizações. A *RU* obtida como resultado da aplicação do algoritmo é posteriormente normalizada de acordo com as formas normais conhecidas [Dat86].

Elmasri,Weeldreyer e Hevner

Em [EWH85] são apresentadas regras para se obter uma representação relacional de um esquema ECR. Cada relação obtida na tradução da representação ECR para a representação relacional recebe um atributo identificador definido pelo sistema e invisível aos usuários finais (*surrogate*). O *surrogate* é necessário para identificar univocamente cada tupla da relação, já que o modelo ECR não exige a declaração de atributos identificadores para os tipos de entidade e categorias.

Cada tipo de entidade é convertido em uma relação com um único *surrogate*. Todos os atributos monovalorados do tipo de entidade são incluídos na relação criada. Cada atributo multivalorado é colocado em uma relação à parte cuja chave é formada pela concatenação do próprio atributo multivalorado com o *surrogate* da relação criada para o tipo de entidade.

Cada categoria origina uma relação que recebe os atributos monovalorados da categoria mais o *surrogate*. A restrição de integridade referencial entre os *surrogate* da relação que representa a categoria e os *surrogates* das relações que representam os tipos de entidade básicos deve ser declarada no esquema relacional. Desse modo, para uma categoria ISA C_1 formada a partir de um tipo de entidade E_1 deve ser declarada a restrição $surrogate(C_1) \subseteq surrogate(E_1)$.

Elmasri e Navathe

Elmasri e Navathe [EN89] apresentam várias alternativas para conversão de um modelo ER estendido para o modelo relacional. Relacionamentos entre superclasses e subclasses que modelam abstrações de generalização/especialização podem ser convertidos de quatro maneiras diferentes. A primeira alternativa é criar uma relação para a superclasse com os mesmos atributos e uma relação para cada subclasse com os atributos específicos da subclasse mais a chave da relação que representa a superclasse. A segunda alternativa é criar uma relação para cada subclasse e_i contendo todos os atributos da subclasse mais todos os atributos da superclasse. Essa opção causa uma grande redundância de dados e torna as relações não normalizadas. A terceira alternativa é criar uma relação contendo todos os atributos da superclasse e das subclasses e mais um atributo t que indica a qual subclasse uma tupla da relação se refere. Essa opção pode ser usada para melhorar o desempenho das aplicações no caso das subclasses serem disjuntas e possuírem poucos atributos específicos. A última alternativa é indicada para subclasses sobrepostas com poucos atributos específicos e consiste na criação de uma única relação com todos os atributos das subclasses e da superclasse mais um atributo para cada subclasse que estiver representada na relação. Esse atributo é uma variável booleana que indica se a tupla

contém ou não valores para os atributos da subclasse em questão.

São apresentados também mapeamentos para subclasses definidas sobre mais de uma superclasse (subclasse compartilhadas) e para categorias. Para as subclasses compartilhadas são usadas as mesmas opções citadas para os relacionamentos *superclasse-subclasse*.

3.6.3 Metodologia proposta

A estrutura da metodologia para tradução ECR-Relacional que será proposta nesta dissertação é basicamente a mesma que foi proposta para a tradução ECR-Rede. Cada tipo de construtor usado no modelo ECR é examinado separadamente começando pelos tipos de entidade, passando pelas categorias e terminando nos relacionamentos. Esse ordenamento é natural porque categorias são definidas sobre os tipos de entidade e relacionamentos são definidos sobre tipos de entidade e categorias.

Assim como em [EWH85], a metodologia utiliza *surrogates* nas relações em que não são definidos atributos chave. A função dos *surrogates* é estabelecer ligações entre uma relação que foi gerada a partir de um tipo de entidade que não possui atributos identificadores e outras relações que representam tipos de entidade que se relacionam com o primeiro tipo de entidade no esquema federado. Ao contrário de [EWH85], porém, os *surrogates* são visíveis aos usuários e, embora não sejam passíveis de atualização, podem ser usados para a construção de operações de *junção* das relações.

Exame dos tipos de entidade

Cada tipo de entidade no esquema ECR é convertido em uma relação com mesmo nome. A chave primária da relação é a chave do tipo de entidade que a originou ou, no caso do tipo de entidade não ter atributos identificadores, um *surrogate* formado pela concatenação do nome da relação com o sufixo *ID*. Atributos monovalorados (cardinalidade máxima = 1) do tipo de entidade tornam-se atributos da relação. Atributos compostos são decompostos nos atributos simples que o compõem e incluídos na relação. Restrições sobre os valores de atributos, como por exemplo, a permissão de valores nulos (cardinalidade mínima = 0), devem ser verificadas durante a transformação de operações sobre o esquema externo em operações sobre o esquema federado.

As correspondências entre as relações geradas e os tipos de entidade no esquema federado devem ser armazenadas no DD local para serem usadas no processo de transformação de operações construídas sobre o esquema externo em operações sobre o esquema federado.

Como o modelo relacional supõe relações em 1FN, cada atributo multivalorado (car-

dinalidade máxima > 1) do tipo de entidade é colocado em uma relação à parte cujo nome é formado pela concatenação do nome do tipo de entidade com o nome do atributo multivalorado. A chave de tal relação é formada pela concatenação do atributo multivalorado com o atributo identificador (ou *surrogate*) da relação que representa o tipo de entidade que possui o atributo em questão. A dependência de inclusão definida entre as duas relações pode ser explicitada no esquema externo para facilitar a compreensão do usuário sobre a ligação entre as relações. O mapeamento que associa a relação que representa o atributo multivalorado com o tipo de entidade no esquema federado deve ser mantido junto com o esquema externo para ser utilizado durante a transformação de operações.

Exame das categorias

Cada categoria ISA é convertida em uma relação cuja chave é a chave da categoria ou o *surrogate* criado para ela. Os atributos específicos das categorias devem ser incluídos na relação gerada, levando-se em consideração as mesmas observações feitas na discussão da conversão dos atributos dos tipos de entidade.

Nas categorias de generalização cada tipo de entidade que compõe a categoria é convertido em uma relação cuja chave é a chave associada à categoria ou o *surrogate* gerado para ela. Os atributos da relação são todos os atributos da categoria mais os atributos específicos do tipo de entidade em questão.

As dependências de inclusão entre as chaves (ou *surrogates*) das relações que representam as categorias e as chaves (ou *surrogates*) das relações que representam os tipos de entidade que compõem essas categorias podem ser declaradas no esquema externo para que o usuário perceba mais claramente a interdependência entre as relações.

Exame dos relacionamentos

Relacionamentos binários M:N ou relacionamentos de grau maior do que dois são convertidos em uma relação cuja chave é formada pela concatenação dos atributos identificadores (ou *surrogates*) das relações que representam as classes que participam desse relacionamento. Os atributos do relacionamento são incluídos na relação gerada obedecendo-se os mesmos critérios definidos para os atributos dos tipos de entidade.

Relacionamentos 1:N ou 1:1 são convertidos em uma relação separada se eles possuem atributos próprios. Nesse caso, valem as mesmas considerações feitas para os relacionamentos M:N. Se os relacionamentos não têm atributos próprios, então deve ser incluído na relação que representa a classe com participação funcional o atributo identificador (ou

surrogate) da relação que representa a outra classe participante. O atributo identificador (ou *surrogate*) incluído nessa relação funciona como chave estrangeira para a outra relação. Se a participação do tipo de entidade que corresponde à relação que contém a chave estrangeira for parcial, então será admitido o valor nulo para essa chave estrangeira.

Devem ser armazenados no DD local as correspondências entre as relações geradas no esquema externo e os relacionamentos do esquema federado. Devem ser guardadas também as correspondências entre as chaves estrangeiras contidas nas relações geradas e o caminho de navegação definido pelo nome de conexão no esquema federado.

As dependências de inclusão entre cada atributo que compõe a chave (ou *surrogate*) da relação que representa o relacionamento e as chaves (ou *surrogates*) das relações que representam as classes participantes podem ser declaradas no esquema externo para facilitar a compreensão do usuário final. As restrições referentes ao tipo de participação das classes no relacionamento (total, parcial, específica, funcional e não funcional) devem ser averiguadas durante o processo de transformação de operações.

Para possibilitar a transformação de operações sobre o esquema externo em operações GORDAS equivalentes sobre o esquema federado é necessário identificar as correspondências entre as operações de *junção* das relações e os nomes de conexão definidos no esquema federado. Uma *junção* que envolve uma relação correspondente a uma classe e uma relação que representa um relacionamento em que a classe participa é mapeada para o caminho definido pelo *nome-conexão-1* associado à participação da classe no relacionamento. O mesmo mapeamento é feito para uma *junção* que envolve duas relações que representam classes participantes em um relacionamento binário. A combinação de duas *junções* como essas, envolvendo as classes A e B e o relacionamento R em que ambas participam, é mapeada para o caminho definido pelo *nome-conexão-1* associado à participação de uma das classes, se o relacionamento R é binário, e para o caminho definido pela concatenação do *nome-conexão-2* de uma das classes com o *nome-conexão-1* da outra classe (*nome-conexão-2 of nome-conexão-1 of* $A|B$), se o relacionamento tem grau maior do que dois.

Outros mapeamentos que podem ser feitos para facilitar a transformação de operações são os que associam cada classe com os relacionamentos em que elas participam e os mapeamentos que associam cada relacionamento com as classes participantes, separadas em classes com participação total e classes com participação parcial. Essas listas invertidas evitam um exame extensivo do esquema ECR para se verificar a necessidade de propagação de atualizações submetidas pelos usuários do esquema externo.

Refinamento do esquema

Nesta etapa da metodologia podem ser modificadas pelo DBA do SBDH algumas informações contidas no esquema externo relacional como os nomes dos atributos ou os nomes das relações. Podem ser também acrescentadas ao esquema externo restrições de integridade sobre os valores dos atributos e restrições de integridade referenciais, mesmo que o SGBD do SBD relacional local não suporte essas restrições. A inclusão dessas restrições facilita o entendimento da semântica das aplicações pelos usuários. Outro refinamento que pode ser feito no esquema obtido é a eliminação de *surrogates* que não sejam essenciais para as operações de *junção* entre as relações.

A figura 3.31 apresenta as relações do esquema externo relacional obtido pela aplicação da metodologia proposta ao esquema federado representado na figura 3.30. Os atributos sublinhados representam as chaves primárias da relação.

```
Departamento(cod-depto,nome-depto,matric-gerente,gratif-gerente)
Empregado(matricula,nome,...,salario-atual,cod-depto,cod-cargo)
Projeto(cod-projeto,nome-projeto,...,multa-atraso,cod-cliente)
Cliente(cod-cliente,nome-cliente,endereco,telefone)
Cargo(cod-cargo,nome-cargo,salario-medio)
Ferias(dt-ini-per,dt-fim-per,dt-ini-ferias,dt-fim-ferias,matricula)
Hist-salario(data-vigencia,valor-salario,matricula)
Hist-escolar(cod-instrucao,instituicao,data-conclusao,matricula)
Dependente(data-nasc,cod-gemeos,cod-parentesco,nome-depend,matricula)
Etapa-projeto(cod-projeto,num-etapa,...,data-inic,data-fim)
Contab-projeto(cod-depto,cod-projeto,conta-contab,custos-mao-obra)
Emp-Etapa(matric,cod-projeto,num-etapa,horas-trab)
Particip-Projeto(matric,cod-projeto,horas-trab)
Gerente-Proj(matric,cod-projeto)
```

Figura 3.31: Relações do esquema externo obtidas na metodologia Relac.-ECR

Capítulo 4

Transformação de operações sobre esquema externo rede

4.1 Introdução

Para que um programa de aplicação construído sobre um esquema externo rede obtenha os dados disponíveis no SBDH é preciso que os processadores que formam o sistema gerenciador global (SGBDH) identifiquem os SBDs componentes referenciados em cada comando e decomponham cada operação sobre o esquema virtual externo em operações sobre os esquemas dos SBDs componentes. Além disso, o SGBDH deve transformar as operações construídas através da LMD rede em operações equivalentes nas LMDs utilizadas pelos SBDs componentes.

A transformação de operações é dificultada pela presença de LMDs com características bem diferentes. Enquanto que a LMD rede é procedimental, acessando registro a registro do banco de dados através de caminhos de navegação definidos explicitamente pelos usuários, outras LMDs, como o SQL, obtém de uma só vez conjuntos de registros que satisfazem determinadas condições de seleção preestabelecidas.

As características da LMD rede exigem a presença de *indicadores de estado corrente* (*currency indicators*) nos programas de aplicação, que funcionam como ponteiros para indicar a posição corrente do programa em relação aos caminhos de navegação possíveis, de modo que a próxima operação seja iniciada a partir de uma das posições marcadas.

Alguns trabalhos existentes na literatura [TFS+79, LDJS80, DS82, Dem83] estudam mecanismos para conversão de programas que utilizam LMDs procedimentais em programas equivalentes que utilizam LMDs não procedimentais. A maioria desses trabalhos

sugere maneiras para se analisar os programas e determinar o fluxo de dados nele contido, e a partir desse fluxo de dados obter a sua estruturação semântica, identificando blocos lógicos independentes que possam ser convertidos em um comando da LMD não procedimental. O fluxo de dados do programa é determinado através dos comandos da LMD procedimental que estabelecem os indicadores de estado corrente e dos comandos da linguagem hospedeira que alteram o fluxo de dados (comandos de desvio e teste de condição).

Entretanto, a análise de programas, conforme sugerida por esses trabalhos, apresenta alguns problemas, porque depende da maneira como o programa é construído e da linguagem hospedeira utilizada. Assim, a análise completa dos programas para o caso geral se torna extremamente complexa, senão inviável.

No contexto de SBDHs essa abordagem é ainda menos indicada, por não se tratar de uma migração dos dados e aplicações de um SBD para outro, e sim do compartilhamento de dados envolvendo SBDs independentes. Uma solução mais adequada é construir, a partir do programa de aplicação sobre o esquema externo rede, um programa de aplicação sobre o esquema do SBD componente rede presente no mesmo local em que o esquema externo está definido. Neste programa cada comando que referencia dados contidos em SBDs componentes remotos deve ser substituído por uma chamada (*call*) a um processador de comunicação, que encaminhará uma mensagem a cada SBD remoto solicitando a execução de uma operação para obtenção dos dados desejados. Dessa maneira mantém-se o controle local e aproveita-se toda a estrutura de execução do programa construído pelo usuário, modificando-se apenas a parte que se refere à LMD rede contida na linguagem hospedeira.

Para manter a característica de acesso registro a registro em SBDs remotos que usam o SQL como LMD deve-se utilizar o conceito de cursor [vdL89] que permite a associação de um ponteiro ao resultado de uma expressão de seleção (*Select-From-Where*), de modo que seja possível obter uma tupla por vez. O cursor funciona então como uma espécie de indicador da última tupla recuperada.

No presente capítulo será proposta uma arquitetura para permitir o acesso transparente e integrado aos dados de um SBDH a partir de um programa construído sobre um esquema externo rede. Essa arquitetura prevê módulos para:

1. pré-compilação do programa original gerando um programa sobre o esquema local rede;

2. tradução de operações rede que referenciam SBDs remotos relacionais para operações na linguagem intermediária (GORDAS) sobre o esquema federado;

3. definição do cursor que será associado a cada chamada no programa remoto, identificando a rotina a ser executada por esse programa;

4. construção de programas de aplicação nos nós remotos da rede que possuem SBDs referenciados no programa original;

5. troca de mensagens entre os processos que executam o programa local rede e os programas remotos.

A descrição da arquitetura será feita na seção 4.2. A seção 4.3 apresenta os algoritmos usados para analisar os principais comandos da LMD-Rede durante a pré-compilação. Na seção 4.4 são mostrados os algoritmos usados durante a tradução de operação na LMD rede para operações equivalentes na linguagem GORDAS. Finalmente na seção 4.5 são discutidos algoritmos usados pelos construtores de programas remotos.

4.2 Arquitetura proposta

A figura 4.1 mostra a arquitetura proposta para permitir o acesso aos dados de um SBDH a partir de um programa de aplicação construído sobre um esquema externo rede. Os principais módulos que compõem essa arquitetura são o *Pré-Compilador* (1), o *Tradutor* (2), o *Cursor-Def* (3), o *Construtor-de-Programas* (4) e o *Monitor-de-Comunicação* (5). Cada um desses módulos será descrito a seguir.

4.2.1 Pré-Compilador

Esse módulo converte um programa de aplicação escrito sobre um esquema externo rede em um programa de aplicação sobre o esquema local rede com chamadas ao monitor de comunicação local, responsável pela troca de mensagens com os monitores de comunicação remotos. Para cumprir sua função o Pré-Compilador deve executar os seguintes passos:

1. Verificar a correção sintática dos construtores do esquema externo rede que foram usados no programa de aplicação do usuário;

2. Substituir na área de trabalho (*user-work-area ou uwa*) do programa gerado a descrição do esquema externo pela descrição do esquema local. Todos os itens de dados contidos no esquema externo e não contidos no esquema local devem ser transformados em variáveis auxiliares na *uwa* do programa gerado, mantendo-se o mesmo nome para facilitar o processo de pré-compilação.

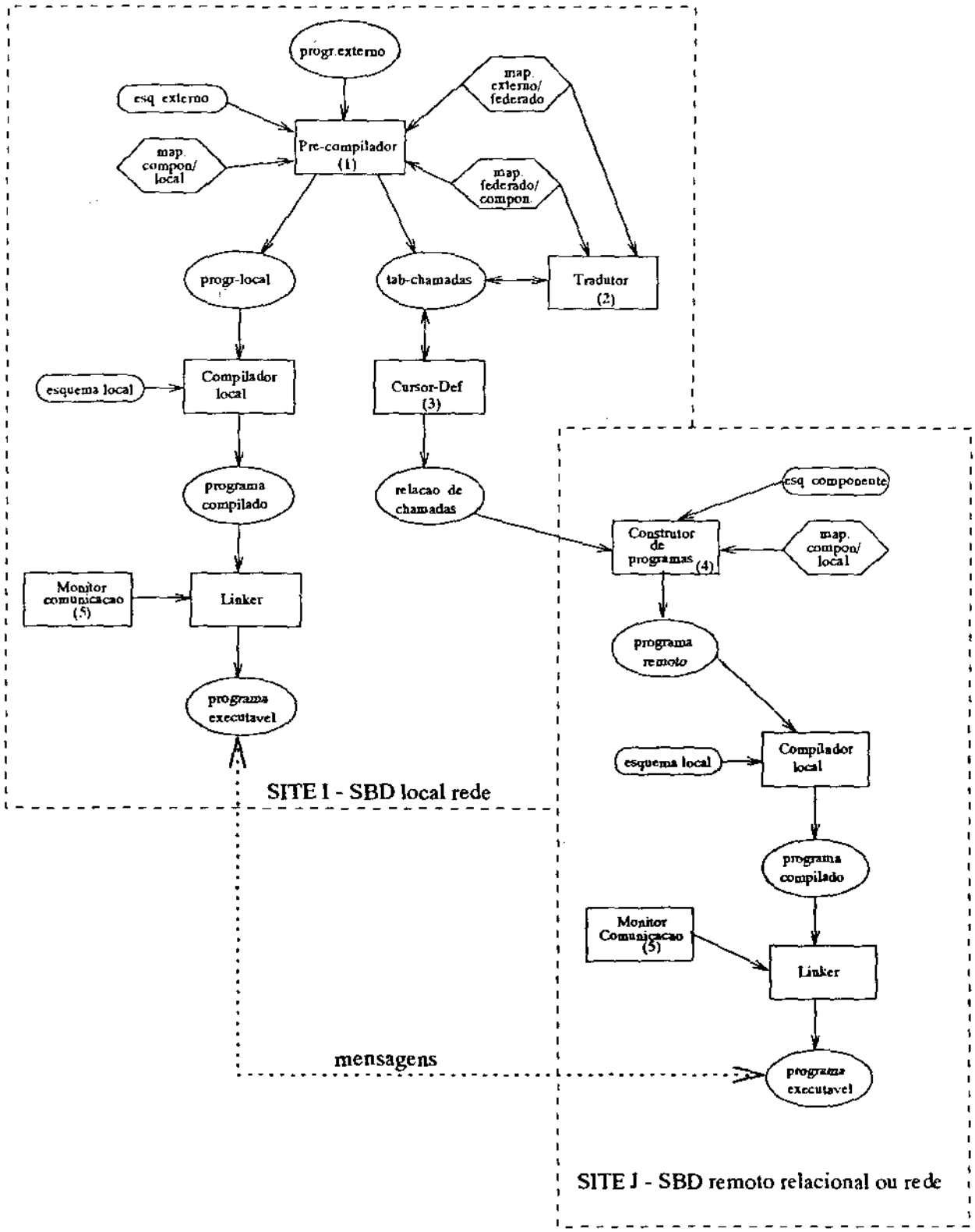

Figura 1.1: Arquitetura proposta

3. Incluir na *uwa* do programa gerado a variável de controle *call-status* que será usada no programa gerado para verificar se uma operação sobre um SBD remoto foi bem sucedida ou não.

4. Analisar cada comando da LMD rede embutida na linguagem hospedeira determinando a distribuição dos dados referenciados nesse comando. Para tal, devem ser examinados os mapeamentos que guardam as correspondências entre os construtores do esquema externo e do esquema federado e entre os construtores do esquema federado e dos esquemas componentes. Se todos os dados referenciados em um comando de recuperação de dados estiverem disponíveis no SBD local, então esse comando deverá ser substituído por um comando equivalente usando a sintaxe do esquema local. Se alguns ou todos os dados referenciados no comando não existirem no SBD local, então deve ser incluída no programa gerado uma chamada ao monitor de comunicação que enviará uma mensagem para cada SBD remoto que possui alguns dos dados requisitados. Cada chamada individual a um SBD remoto será identificada pelo nome do programa e pelo número da chamada, gerado de forma seqüencial e crescente pelo Pré-Compilador. Cada chamada conterá também como parâmetros a variável *call-status*, as variáveis do programa local que receberão os dados obtidos nos SBDs remotos e os valores das variáveis do programa local que serão utilizados pelos SGBDs remotos na seleção dos seus dados.

5. Incluir em uma tabela de chamadas informações sobre as chamadas inseridas no programa rede local. A tabela de chamadas é uma estrutura de dados existente em cada nó da rede que possui um SBD componente rede. Os dados dessa tabela são:

   - Programa: nome do programa em que a chamada foi incluída;
   - Número: número de ordem da chamada dentro do programa gerado;
   - SBD: identificador do SBD remoto que é o destinatário da chamada. Essa informação será utilizada pelo monitor de comunicação local para endereçamento da mensagem;
   - Tipo: tipo da operação que está sendo requisitada pela chamada. Essa informação será usada pelo Tradutor e pelo Construtor-de-Programas. Os tipos de chamada possíveis são:
     - FA (FIND ANY);
     - FD (FIND DUPLICATE);
     - FFS (FIND FIRST SET);
     - FNS (FIND NEXT SET);
     - FLS (FIND LAST SET);
     - FPS (FIND PRIOR SET);

- FOS (FIND OWNER SET);
- SR (STORE RECORD);
- ER (ERASE RECORD);
- MR (MODIFY RECORD);
- CRS (CONNECT RECORD SET);
- DRS (DISCONNECT RECORD SET);
- RRS (RECONNECT RECORD SET).

- Cursor: nome do cursor que será definido para a chamada pelo módulo Cursor-Def. A concatenação do tipo da chamada com o cursor será utilizada para identificar no programa remoto o procedimento que deverá ser executado;
- Record: nome do *record-type* do esquema externo citado no comando do programa original do usuário. Essa informação será usada pelo Tradutor;
- Set: nome do *set-type* do esquema externo citado no comando do programa original do usuário. Para os tipos de chamada que não referenciam nenhum set-type, como por exemplo FA, esta coluna não é preenchida. Essa informação será utilizada pelo Tradutor;
- Set-currency: nome dos *set-type* cujos indicadores de estado corrente devem ser mantidos inalterados no programa que manipula os dados do SBD remoto rede referenciado na chamada;
- Atributos: nome dos itens de dados que se deseja obter nos SBDs remotos e que estão contidos no *record-type* citado no comando do programa original;
- Parâmetros: nome das variáveis contidas na *uwa* do programa gerado cujos valores serão utilizados como parâmetros para as operações sobre o SBD remoto referenciado na chamada;
- Transação GORDAS: transação na linguagem intermediária que será enviada ao SBD componente remoto referenciado na chamada.

6. Incluir no programa gerado rotinas para verificação de possíveis restrições de integridade, que foram incorporadas ao esquema componente obtido na fase de tradução do esquema local rede para o modelo de dados comum.

4.2.2 Tradutor

O Tradutor recebe do Pré-Compilador a tabela de chamadas e gera para cada tipo de chamada um conjunto de operações equivalente na linguagem intermediária, usando os construtores do esquema componente do SBD remoto referenciado na chamada. Para

tal, o Tradutor examina os mapeamentos entre os construtores do esquema externo e do esquema federado e os mapeamentos entre os contrutores do esquema federado e dos esquemas componentes.

Durante a tradução podem ser verificadas algumas restrições de integridade estáticas presentes no esquema federado como, por exemplo, os valores permitidos para um determinado atributo.

Restrições de integridade mantidas automaticamente pelo SGBD local rede, como a participação obrigatória de registros em uma ocorrência de um set-type, devem ser garantidas para os SBDs remotos durante a tradução Rede-Cordas. Suponha, por exemplo, que um record-type RT no esquema externo rede esteja mapeado para um tipo de entidade ET no esquema federado e que esse tipo de entidade esteja definido no SBD local rede e em outro SBD componente X. Suponha também que esse record-type seja o record-type membro de um set-type ST no esquema externo e que o tipo de retenção e inserção desse record-type seja AUTOMATIC MANDATORY. Suponha ainda que esse set-type ST esteja mapeado para um relacionamento R no esquema federado e que esse relacionamento esteja definido no SBD local e no SBD componente X. Nesse caso, uma operação de inclusão de um registro do record-type R, presente na tabela de chamadas, deve ser traduzida para uma transação CORDAS que contenha uma operação para inclusão de uma entidade em ET e uma operação para inclusão de uma instância do relacionamento R envolvendo a entidade incluída anteriormente e a entidade que corresponde ao registro mestre do set-type ST.

Essas e outras restrições de integridade, que precisam ser validadas com base nos valores dos dados armazenados nos SBDs remotos, devem ser incluídas nos programas gerados pelo construtor de programas remoto durante a transformação de operações sobre o esquema componente em operações sobre os esquemas locais dos SBDs componentes.

4.2.3 Cursor-Def

O Cursor-Def define o nome do cursor associado a cada chamada e ao nome da rotina do programa remoto que executa a operação solicitada pela chamada.

Se o SBD remoto referenciado na chamada segue o modelo rede, então é definido um nome de cursor diferente para cada entrada na tabela de chamadas, já que as operações no modelo rede tratam um registro por vez. Para entradas na tabela que tenham o mesmo tipo de chamada e que referenciem os mesmos dados (record-type, set-type, set-currency, lista de atributos e lista de parâmetros) é definido o mesmo cursor.

Se o SBD remoto referenciado na chamada é relacional, então é definido um mesmo

cursor para as chamadas que se referem ao mesmo conjunto de dados em operações de seleção de dados. Por exemplo, a chamada cujo tipo é FFS (FIND FIRST SET) e que se refere ao *record-type X* e ao *set-type Y* recebe o mesmo cursor que a chamada cujo tipo é FNS (FIND NEXT SET) e que se refere aos mesmos *record-type X* e *set-type Y*. Embora as duas chamadas sejam semanticamente diferentes, elas se referem ao mesmo conjunto de dados, isto é, ao conjunto dos registros pertencentes ao *record-type X* que são membros do *set-type Y* e, por isso, devem ser associadas ao mesmo cursor no programa remoto.

Deve-se ressaltar que para alguns tipos de chamada o cursor só pode ser identificado em tempo de execução. Esse é o caso dos tipos de chamada ER (ERASE RECORD) e MR (MODIFY RECORD) que se aplicam ao último registro de um determinado *record-type* recuperado no banco de dados. Para contornar esse problema é necessário manter no programa remoto relacional o último cursor utilizado. Dessa forma, esse programa remoto ao receber uma chamada do tipo ER ou MR executa o procedimento identificado pelo tipo da chamada e pelo nome do último cursor tratado.

O nome do cursor definido para cada tipo de chamada deve ser armazenado na tabela de chamadas para que possa ser incorporado à mensagem gerada pelo monitor de comunicação local.

Uma relação das chamadas que referenciam um SBD remoto relacional é enviada ao módulo Construtor-de-Programas, localizado no mesmo nó da rede de comunicação onde se encontra o SBD em questão. A relação de chamadas contém o identificador do SBD local, o nome do programa local rede em que as chamadas foram submetidas, o tipo da chamada, o nome do cursor e a transação GORDAS associados à cada chamada.

A relação de chamadas enviada para cada construtor de programas remotos rede é similar à relação enviada aos construtores de programas remotos relacionais. A diferença é que na primeira podem ser incluídas outras informações como, por exemplo, a lista dos *set-type* cujos indicadores de estado corrente serão mantidos inalterados durante a execução da operação solicitada.

4.2.4 Construtor-de-Programas

Este processador recebe a relação de chamadas gerada pelo Cursor-Def em outro nó da rede de comunicação e gera o programa de aplicação que executa as operações correspondentes sobre o SBD local, usando a linguagem hospedeira e a LMD locais.

O programa de aplicação é construído de forma modular e compõe-se de um módulo de controle e n módulos de operações. O tipo desses módulos pode variar conforme a linguagem hospedeira utilizada.

Os módulos de operação são construídos a partir da transformação das operações GORDAS sobre o esquema componente em operações equivalentes sobre o esquema local utilizando a LMD do SBD componente. O nome de cada módulo de operação é formado pela concatenação do tipo da chamada com o nome do cursor da chamada, já que mais de um cursor pode ser associado ao mesmo tipo de chamada.

O procedimento de controle recebe do monitor de comunicação local o tipo da chamada, o cursor ao qual se refere a chamada e os parâmetros necessários para execução da operação requisitada pela chamada. Através do tipo de chamada e do nome do cursor este módulo determina o módulo de operação que deve ser ativado. As únicas exceções para este tratamento acontecem quando as chamadas são do tipo exclusão de registro (ER) ou alteração de registro (MR) e o SBD referenciado nas chamadas segue o modelo relacional. Nesse caso, o nome do cursor a que a chamada se refere é obtido diretamente da variável auxiliar que mantém o nome do último cursor tratado.

4.2.5 Monitor-de-Comunicação

Esse processador é na verdade um programa de propósito especial responsável pelo intercâmbio de mensagens entre os nós da rede de comunicação que contém os SBDs componentes.

No nó de origem, isto é, no nó em que foi construído o programa externo rede, esse programa é chamado pelo programa local rede obtido no processo de pré-compilação. O programa de aplicação passa para o monitor de comunicação local o nome do programa, o número da chamada e os valores das suas variáveis que servirão como parâmetros para a operação que será executada no SBD remoto. O monitor de comunicação cria então uma mensagem que será enviada ao monitor de comunicação em outro nó da rede. Os dados contidos nessa mensagem são:

- o endereço do nó origem;
- o endereço do nó destino;
- o nome do programa local que solicitou uma operação sobre um SBD remoto;
- o tipo da chamada e o nome do cursor que identificam a rotina que será executada no programa remoto;
- o indicador do status da operação que será usado pelo programa local para verificar se a operação remota foi executada com sucesso ou não;

- os valores dos parâmetros passados pelo programa local rede.

O endereço origem e destino são obtidos no mapa da rede de comunicação através dos identificadores do SBDs local e remoto. O tipo de chamada e o cursor são obtidos na entrada da tabela de chamadas identificada pelo nome do programa e número da chamada.

No nó destino, o monitor de comunicação recebe a mensagem do nó origem e chama o programa de aplicação previamente criado pelo módulo Construtor-de-Programas. Ao receber o resultado da operação requisitada o monitor envia uma mensagem de resposta ao monitor de comunicação do nó origem e passa a aguardar uma nova mensagem.

Supõe-se que o sistema de gerenciamento da rede de comunicação forneça as facilidades necessárias para a comunicação ponto a ponto entre os computadores onde estão localizados os diversos SBDs componentes. O detalhamento dos mecanismos de comunicação não serão tratados nesta dissertação.

4.3 Algoritmos para análise da LMD-REDE

Os algoritmos do Pré-Compilador que analisam os comandos da LMD rede utilizam as seguintes estruturas de dados:

1) Mapeamento esquema-externo-esquema-federado (Mef)
Mef(E,F) onde E = nome do construtor no esquema externo
F = nome do construtor no esquema federado que corresponde a E
2) Mapeamento esquema-federado-esquema-componente (Mfc)
Mfc(F,C_1,C_2,...,C_n)
onde F = nome do construtor no esquema federado
C_i = nome do construtor no esquema componente i, $1 \geq i \geq n$, que corresponde a F
3) Mapeamento esquema-componente-esquema-local-rede (Mcl)
Mcl(C_L,L) onde
C_L= nome do construtor no esquema componente definido para o SBD local rede
L = nome do construtor no esquema local rede que corresponde a C_L
1) Tabela de chamadas (TC)
TC(Prog,NC,Tipo,Cursor,Rec,Set,Set-cur-list,{SBD,A-list,P-list,Trans})
onde Prog = nome do programa que gerou a chamada

NC = número da chamada gerado pelo pré-compilador
Tipo = tipo da operação da chamada
Cursor =cursor associado a chamada
Rec = *record-type* referenciado
Set = *set-type* referenciado
Set-cur-list = nome dos set-type que terão os indicadores de estado preservados
SBD = identificador do SBD remoto
A-list = lista com o nome dos atributos que se deseja obter no SBD remoto
P-list = lista com o nome dos parâmetros que serão utilizados pelo SBD remoto para executar as operações solicitadas
Trans = transação Gordas que será encaminhada ao SBD remoto

5) NCALL = variável auxiliar usada para numerar as chamadas geradas

4.3.1 Comandos para localização e recuperação de registros de um record-type

I) FIND ANY <record-type> USING <field-list>

A: Verificar nos mapeamentos Mef e Mfe a distribuição dos dados existentes em <record-type> e <field-list>

Se todos os itens de dados contidos em <field-list> estão definidos no SBD local rede

Então

B: Incluir no programa gerado um comando equivalente ao comando examinado usando a sintaxe do esquema local

Se (<record-type> está definido somente no SBD local rede) ou
(todos os dados contidos em <record-type> estão presentes no SBD local rede e não existem no programa comandos para exclusão ou alteração de <record-type>)

Então

Passar a examinar outro comando do programa externo rede

Senão

C: Incluir no programa gerado o teste do DB-STATUS da operação e o comando GET para tornar disponível na *uwa* os dados do registro localizado no passo **B**

D: Incluir no programa gerado uma chamada ao monitor de comunicação para os SBDs remotos que possuem réplicas de <record-type> contendo dados não definidos no SBD local rede. A chamada deve incluir uma lista contendo os SBDs remotos, o número de chamada gerado para cada um deles, os dados que devem ser buscados nos SBDs remotos e os parâmetros para cada uma dessas chamadas. Os parâmetros são os atributos identificadores de <record-type> que servirão para localizar o registro replicado no SBD remoto.

E: Incluir na tabela de chamadas os dados de cada chamada e incrementar o número da chamada (NCALL). O tipo da chamada é FA já que a operação equivalente no SBD remoto é localizar um registro, dado o valor de seus atributos identificadores.

Senão

Se pelo menos alguns dos dados contidos em <field-list> estão definidos no SBD local

Então

F: Idem **B**

G: Incluir no programa gerado um laço (loop) para recuperar os registros replicados de maneira similar ao passo **D**. A diferença é que os parâmetros de cada chamada são formados não só pelos atributos identificadores do registro obtido em **F**, mas também pelos atributos do SBD remoto contidos em <field-list>. No final do laço deve ser incluído um comando FIND DUPLICATE que recupera o próximo registro local que satisfaz a condição de busca, caso as operações remotas não tenham sido executadas com sucesso (*call-status* diferente de zeros). As condições para saída do laço são a execução com sucesso da chamada ou o fim dos registros locais (passo **F**) que satisfaçam as condições de seleção.

H: Idem **E**

Senão

I: Incluir no programa gerado a chamada a um dos SBDs remotos que contenha dados de <field-list>. Incluir também o teste do status da chamada.

J: Incluir no programa gerado para o caso de uma operação bem sucedida no passo **I** (*call-status* = 0) um laço similar ao definido em **G**, se os campos em <field-list> estão contidos em mais de um SBD remoto. A diferença principal entre os dois laços é que no passo **J** o registro base da pesquisa é um registro contido em um SBD remoto e, portanto, ao invés de haver um comando FIND DUPLICATE para repetição do laço haverá uma nova chamada ao mesmo SBD remoto que iniciou o laço, mudando-se o tipo da chamada de FA (FIND ANY) para FD (FIND DUPLICATE). Uma outra diferença é que no laço definido em **J** pode ser incluído um comando FIND ANY para localizar um registro no SBD rede local, usando como chave de pesquisa o(s) atributo(s) identificador(es) do registro obtido no SBD remoto da chamada inicial.

K: Idem **E**

A seguir são mostrados exemplos da aplicação do algoritmo. Os exemplos se referem ao esquema externo rede da figura 3.33. A linguagem hospedeira usada é a linguagem COBOL.

Exemplo 1.1: FIND ANY *Empregado* USING *nome-empreg,est-civil*

Passo A - Verifica-se nos mapeamentos que as entidades do tipo *Empregado* correspondente ao *record-type* do comando estão replicadas nos SBDs componentes SBD1 (SBD local rede) e SBD2 (SBD remoto relacional) com fragmentação vertical de atributos. Verifica-se também que os atributos *nome-empreg* e *est-civil* estão contidos em SBD1.

Passo B - Inclui-se no programa gerado:

```
FIND ANY Empregado USING nome-empreg,est-civil
```

Passo C - Inclui-se no programa gerado:

```
IF DB-STATUS = 0
   GET Empregado
```

Passo D - Inclui-se no programa gerado:

```
CALL "Monitor" USING  programa, "ncall", call-status,
                      salario-atual, matricula
IF   call-status NOT EQUAL 0
     GO TO cancela-programa.
```

O parâmetro *programa* é uma constante definida no programa gerado que contém o nome desse programa. O parâmetro *ncall* colocado entre aspas é um literal definido pelo Pré-Compilador e corresponde ao valor da variável NCALL. A rotina *cancela-programa* deve ser incluída no programa gerado pelo Pré-Compilador e deve conter um comando para ROLL-BACK nas transações executadas localmente e uma chamada ao monitor de comunicação pedindo uma operação ROLL-BACK em cada um dos SBDs componentes remotos referenciados nas chamadas do programa.

Passo E - Inclui-se os dados da chamada na tabela de chamadas:

TC.prog	←	programa
TC.NC	←	"ncall"
TC.SBD	←	"SBD2"
TC.Tipo	←	"FA"
TC.Rec	←	"Empregado"
TC.A-list	←	"salario-atual"
TC.P-List	←	"matricula"
NCALL	←	NCALL + 1

Exemplo 1.2: FIND ANY *Empregado* USING *nome-empreg,salario-atual*

Passo A - Verifica-se nos mapeamentos que as entidades do tipo Empregado correspondente ao *record-type* do comando estão replicadas nos SBDs componentes SBD1 (SBD local rede) e SBD2 (SBD remoto relacional). Verifica-se também que os atributos *nome-empreg* e *salario-atual* estão contidos em SBD1 e SBD2, respectivamente.

Passo F - Inclui-se no programa gerado:

```
FIND ANY Empregado USING nome-empreg
```

Passo G - Inclui-se no programa gerado:

```
loop.
```

```
IF DB-STATUS = 0
   GET Empregado
   CALL "Monitor" USING programa, "ncall", call-status,
                        salario-atual, matricula
   IF call-status NOT = 0
      FIND DUPLICATE Empregado USING nome-empreg
      GO TO loop.
```

Passo H - Inclui-se os dados da chamada na tabela de chamadas:

TC.prog	←	programa
TC.NC	←	"ncall"
TC.SBD	←	"SBD2"
TC.Tipo	←	"FA"
TC.Rec	←	"Empregado"
TC.A-list	←	"salario-atual"
TC.P-List	←	"salario-atual, matricula"
NCALL	←	NCALL + 1

Exemplo 13: FIND ANY *Empregado* USING *salario-atual*

Passo A - Verifica-se nos mapeamentos que as entidades do tipo Empregado correspondente ao *record-type* do comando estão replicadas nos SBDs componentes SBD1 (SBD local rede) e SBD2 (SBD remoto relacional). Verifica-se também que o atributo *salario-atual* está contido em SBD2.

Passo I - Inclui-se no programa gerado:

```
CALL "Monitor" USING programa, "ncall", call-status,
                     matricula, salario-atual
IF call-status = 0
   FIND ANY Empregado USING matricula
   GET Empregado
ELSE
   MOVE call-status TO DB-STATUS.
```

Passo K - Inclui-se os dados da chamada na tabela de chamadas:

TC.prog	←	programa
TC.NC	←	"ncall"
TC.SBD	←	"SBD2"
TC.Tipo	←	"FA"
TC.Rec	←	"Empregado"
TC.A-list	←	"salario-atual,matricula"
TC.P-List	←	"salario-atual"
NCALL	←	NCALL + 1

II) FIND DUPLICATE <record-type> USING <field-list>

Algoritmo similar ao algoritmo usado para o comando FIND ANY. Muda-se apenas a sintaxe do comando nos passos **B** e **F** e o tipo de chamada no passo **J**, de FA para FD, indicando a natureza da operação no SBD remoto.

III) GET <record-type>

A: Verificar nos mapeamentos Mef e Mfc a distribuição dos dados existentes em <record-type>

Se todos os itens de dados contidos em <record-type> estão definidos no SBD local rede

Então

B: Incluir no programa gerado um comando equivalente ao comando examinado usando a sintaxe do esquema local

Senão

C: Incluir no programa gerado um comando para atribuir zero a variável DB-STATUS

4.3.2 Comandos para seleção dentro de um set-type

I) FIND FIRST <record-type> WITHIN <set-type>

A: Verificar nos mapeamentos Mef e Mfc se <set-type> está definido no SBD local rede ou não

Se <set-type> está definido no SBD local rede
Então

B: Verificar nos mapeamentos Mef e Mfc a distribuição dos dados existentes em <record-type>

C: Incluir no programa gerado um comando equivalente ao comando examinado usando a sintaxe do esquema local

Se (<record-type> está definido somente no SBD local rede) ou
(todos os dados contidos em <record-type> estão presentes no SBD local rede e não existem no programa comandos para exclusão ou alteração de <record-type>)

Então
Passar a examinar outro comando do programa externo rede
Senão
D: Idem passo **C** do comando FIND ANY
E: Idem passo **D** do comando FIND ANY
F: Idem passo **E** do comando FIND ANY

Senão

G: Incluir no programa gerado uma chamada a um dos SBDs remotos em que o *set-type* referenciado está definido. Se existir mais de um SBD remoto candidato, verificar no mapeamento entre o esquema externo e o esquema federado qual o SBD remoto que deve ser pesquisado primeiramente (ver seção 3.5.3). O tipo da chamada será "FFS" e o parâmetro será o atributo identificador do *record-type* mestre.

H: Incluir no programa gerado uma chamada para cada SBD componente que possui uma réplica de <record-type> contendo dados não definidos no registro localizado no passo **G**. Se alguns dos dados contidos em <record-type> estão definidos no SBD local rede, então deve ser incluído no programa gerado comandos para localizar e recuperar o registro (FIND e GET).

I: Incluir na tabela de chamadas os dados das chamadas dos passos **G** e **H**.

Exemplo L1: FIND FIRST *Empregado* WITHIN *Dep-Emp*

Passo A - Verifica-se que o *set-type Dep-Emp* está definido no SBD1 (SBD local rede).

Passo B - Verifica-se que o atributo *salario-atual* do *record-type Empregado* está definido em SBD2 e os demais atributos estão definidos em SBD1.

Passo C - Inclui-se no programa gerado:

```
FIND FIRST Empregado WITHIN Dep-Emp
```

Passo D - Inclui-se no programa gerado:

```
IF DB-STATUS = 0
   GET Empregado
```

Passo E - Inclui-se no programa gerado:

```
CALL "Monitor" USING  programa, "ncall", call-status,
                      salario-atual, matricula
IF    call-status NOT EQUAL 0
      GO TO cancela-programa.
```

Passo F - Inclui-se os dados da chamada na tabela de chamadas:

TC.prog	←	programa
TC.NC	←	"ncall"
TC.SBD	←	"SBD2"
TC.Tipo	←	"FA"
TC.Rec	←	"Empregado"
TC.A-list	←	"salario-atual"
TC.P-List	←	"matricula"
NCALL	←	NCALL + 1

Exemplo 4.2: FIND FIRST *Projeto* WITHIN *Cliente-Proj*

Passo A - Verifica-se nos mapeamentos que o *set-type Cliente-Proj* está definido no SBD2 (SBD remoto relacional).

Passo G - Inclui-se no programa gerado:

```
CALL "Monitor" USING programa, "ncall", call-status,
                     cod-projeto, nome-projeto,
                     valor-hora-projeto, num-contrato,
```

```
                              valor-contrato, multa-rescisao,
                              multa-atraso, cod-cliente
        IF call-status NOT = 0
           MOVE call-status to DB-STATUS
        ELSE
```

Passo H - Inclui-se no programa gerado:

```
FIND ANY Projeto USING cod-projeto
GET Projeto
```

Passo I - Inclui-se os dados da chamada na tabela de chamadas:

TC.prog	←	programa
TC.NC	←	"ncall"
TC.SBD	←	"SBD2"
TC.Tipo	←	"FFS"
TC.Rec	←	"Projeto"
TC.A-list	←	"cod-projeto, nome-projeto, ...,multa-atraso"
TC.P-List	←	"cod-cliente"
NCALL	←	NCALL + 1

II) FIND FIRST <record-type> WITHIN <set-type> USING <field-list>

O algoritmo usado para este comando é similar ao algoritmo usado para o comando anterior. A diferença é que no comando em questão são usados dados do record-type membro para restringir a seleção. Se os dados em <field-list> estiverem distribuídos em mais de um SBD componente, então pode ser necessária a inclusão de um laço no programa gerado para realizar uma pesquisa exaustiva nos registros de uma *set-occurrence* até que a combinação de atributos desejada seja encontrada ou todos os registros tenham sido examinados. Portanto deve-se alterar os passos **E**, **G** e **H** do algoritmo anterior da seguinte forma:

Passo E: Incluir no programa gerado uma chamada ao monitor de comunicação para os SBDs remotos que possuem réplicas de <record-type> contendo dados existentes em <field-list> não disponíveis localmente. Os parâmetros de cada chamada são os atributos identificadores do registro obtido no passo **B** mais os dados contidos em <field-list> que estão definidos no SBD remoto da chamada. Se todas as chamadas forem executadas com

sucesso então o registro encontrado satisfaz os critérios de busca, senão deve ser obtido o próximo registro do *set-occurrence* para se repetir o processo.

Passo G: Incluir nos parâmetros os dados contidos em <field-list> que estão definidos no SBD remoto da chamada.

Passo H: Incluir no programa gerado uma chamada para cada SBD componente remoto que possui uma réplica de <record-type> contendo dados em <field-list> que não estão disponíveis no SBD remoto referenciado no passo **G**. Se alguns dos dados existentes em <record-type> estão definidos no SBD local rede, então devem ser incluídos no programa gerado comandos para localizar o registro e torná-lo disponível na *uwa* desse programa. Se alguma das chamadas ou algum dos comandos incluídos para seleção no SBD local não for executado com sucesso, então deve ser procurado o próximo registro do *set-type* definido no SBD remoto do passo *G* para repetição do processo.

Exemplo II.1: FIND FIRST *Empregado* WITHIN *Dep-Emp* USING salario-atual, nome-empreg

Passo A - Verifica-se que o *set-type Dep-Emp* está definido no SBD1 (SBD local rede).

Passo B - Verifica-se que o atributo *salario-atual* do *record-type Empregado* está definido em SBD2 e os demais atributos estão definidos em SBD1.

Passo C - Inclui-se no programa gerado:

```
FIND FIRST Empregado WITHIN Dep-Emp USING nome-empreg
```

Passo D - Inclui-se no programa gerado:

```
IF DB-STATUS = 0
   GET Empregado.
```

Passo E - Inclui-se no programa gerado:

```
loop.
     CALL "Monitor" USING  programa, "ncall", call-status,
                           salario-atual, matricula
     IF   call-status NOT EQUAL 0
          FIND NEXT Empregado WITHIN Dep-Emp USING nome-empreg
```

```
IF   DB-STATUS = 0
     GET Empregado
     GO TO loop.
```

Passo F - Inclui-se os dados da chamada na tabela de chamadas:

TC.prog	←	programa
TC.NC	←	"ncall"
TC.SBD	←	"SBD2"
TC.Tipo	←	"FA"
TC.Rec	←	"Empregado"
TC.A-list	←	"salario-atual"
TC.P-List	←	"salario-atual, matricula"
NCALL	←	NCALL + 1

III) FIND FIRST <record-type> WITHIN <set-type> [USING <field-list>] RETAINING <set-list> CURRENCY

Este comando é praticamente idêntico aos comandos dos itens I e II. A única diferença é a presença de uma lista dos *set-type* que não terão seus indicadores de estado corrente alterados nos programas que tratam os SBDs componentes rede referenciados no comando. As modificações nos algoritmos I e II são feitas na sintaxe dos comandos sobre o esquema local rede e na tabela de chamadas para os SBDs remotos rede, que passam a incluir a lista dos *set-type* contidos em <set-list>.

IV) FIND (NEXT|LAST|PRIOR) <record-type> WITHIN <set-type> [USING <field-list>] [RETAINING <set-list> CURRENCY]

Os algoritmos usados para o comando FIND NEXT|LAST|PRIOR e suas variações são basicamente os mesmos usados para o comando FIND FIRST e suas variações (itens I, II e III). As modificações serão feitas na sintaxe dos comandos FIND e no tipo das chamadas (FNS|FLS|FPS ao invés de FFS).

V) FIND OWNER WITHIN <set-type> [RETAINING <set-list> CURRENCY]

Os algoritmos usados para tratar o comando FIND OWNER e suas variações são similares àquele apresentado no item I. Contudo, além da sintaxe dos comandos FIND e do tipo das chamadas (FOS ao invés de FFS) são alterados também os parâmetros das

chamadas no passo G que passam a ser os atributos identificadores do registro membro da *set-occurrence* pesquisada.

Exemplo V.1: FIND OWNER WITHIN *Cliente-proj*

Passo A - Verifica-se nos mapeamentos que o *set-type Cliente-Proj* está definido somente no SBD2 (SBD remoto relacional).

Passo G - Inclui-se no programa gerado:

```
CALL "Monitor" USING programa, "ncall", call-status,
                     cod-cliente, nome-cliente, endereco,
                     telefone, cod-projeto
IF call-status NOT = 0
   MOVE call-status to DB-STATUS.
```

Passo I - Inclui-se os dados da chamada na tabela de chamadas:

TC.prog	←	programa
TC.NC	←	"ncall"
TC.SBD	←	"SBD2"
TC.Tipo	←	"FOS"
TC.Rec	←	"Cliente"
TC.A-list	←	"cod-cliente, nome-cliente, endereco, telefone"
TC.P-List	←	"cod-projeto"
NCALL	←	NCALL + 1

4.3.3 Comandos para atualização de registros

I) STORE <record-type>

A: Verificar nos mapeamentos Mef e Mfc a distribuição dos dados existentes em <record-type>

Se <record-type> está definido no SBD local

Então

B: Incluir no programa gerado um comando equivalente ao comando examinado usando a sintaxe do esquema local

Se <record-type> não está definido em outro SBD componente

Então

Passar a examinar outro comando do programa externo rede

Senão

C: Incluir no programa gerado o teste do status da operação do passo **B**

D: Incluir no programa gerado uma chamada ao monitor de comunicação para os SBDs remotos em que <record-type> está definido. Os parâmetros de cada chamada são os valores dos atributos identificadores dos registros mestres de todos os *set-type* definidos no esquema externo em que <record-type> é membro com participação obrigatória, mais os valores dos atributos pertencentes ao <record-type>.

E: Similar ao passo **E** do comando FIND ANY mudando-se o tipo da chamada e a lista de parâmetros

Senão

F: Idem passo **D**
G: Idem passo **E**

Exemplo 4.1: STORE *Projeto*

Passo A - Verifica-se que o *record-type Projeto* está replicado em SBD1 e SBD2 com fragmentação vertical de atributos.

Passo B e C - Inclui-se no programa gerado:

```
STORE Projeto
IF DB-STATUS = 0
```

Passo D - Inclui-se no programa gerado:

```
CALL "Monitor" USING  programa, "ncall", call-status,
```

```
                                cod-cliente
                                cod-projeto, nome-projeto,
                                valor-hora, ... , multa-atraso,

        IF    call-status NOT EQUAL 0
              GO TO cancela-programa.
```

Passo E - Inclui-se os dados da chamada na tabela de chamadas:

TC.prog	←	programa
TC.NC	←	"ncall"
TC.SBD	←	"SBD2"
TC.Tipo	←	"SR"
TC.Rec	←	"Projeto"
TC.P-List	←	"cod-cliente, cod-projeto, nome-projeto, ..., multa-atraso"
NCALL	←	NCALL + 1

Exemplo 1.2: STORE *Emp-Etapa*

Passo A - Verifica-se que o *record-type Emp-Etapa* está definido somente no SBD2 (SBD remoto).

Passo F - Inclui-se no programa gerado:

```
CALL "Monitor" USING  programa, "ncall", call-status,
                      cod-projeto, num-etapa,
                      matricula, horas-trab

IF    call-status NOT EQUAL 0
      GO TO cancela-programa.
```

Passo G - Inclui-se os dados da chamada na tabela de chamadas:

TC.prog	←	programa
TC.NC	←	"ncall"
TC.SBD	←	"SBD2"
TC.Tipo	←	"SR"
TC.Rec	←	"Emp-Etapa"
TC.P-List	←	"cod-projeto, num-etapa, matricula, horas-trab"
NCALL	←	NCALL + 1

II) ERASE <record-type>

Algoritmo similar ao algoritmo do comando STORE. As diferenças estão nos passos **B**, **D** e **E**. No passo **B** muda-se a sintaxe do comando. No passo **D** não são utilizados parâmetros para as chamadas e no passo **E** muda-se o tipo da chamada de SR para ER.

III) ERASE ALL <record-type>

O comando ERASE ALL tem um tratamento bastante especial, porque ele não só exclui um determinado registro de um record-type, como propaga a exclusão para todos os registros que estão subordinados direta ou indiretamente ao registro excluído dentro da hierarquia representada pelos *set-type* do esquema rede. Assim, se o registro excluído é o registro mestre de uma *set-occurrence* então todos os registros daquela set-ocurrence são excluídos e assim sucessivamente até chegar ao último nível da hierarquia.

Nos SBDs rede isolados, o SGBD é responsável pela propagação das exclusões. No contexto de SBDHs, em que os dados estão distribuídos em SBDs distintos, é necessário informar aos SGBDs dos SBDs remotos quais são os registros que terão de ser excluídos e para isso devem ser recuperados um a um no SBD local os atributos que identificam esses registros. Por isso, cada comando ERASE ALL contido no programa externo rede é convertido primeiramente numa rotina que percorre todos os *set-type* em que o registro objeto do comando é mestre, direta ou indiretamente, e exclui cada registro individualmente através de um comando ERASE simples. As exclusões devem ser feitas do nível mais baixo da hierarquia até chegar ao nível do registro objeto do ERASE ALL.

Como um exemplo desse processo considere o diagrama do esquema rede mostrado na figura 4.2 e o comando *ERASE ALL Reg-b*. Este comando é substituído pelo trecho de programa descrito na figura 4.3 que é então analisado pelo Pré-Compilador usando os algoritmos já descritos.

IV) MODIFY <record-type>

Algoritmo similar ao algoritmo usado para o comando STORE. Muda-se apenas o tipo do comando rede no passo **B**, os parâmetros utilizados no passo **D** e o tipo da chamada no passo **E**.

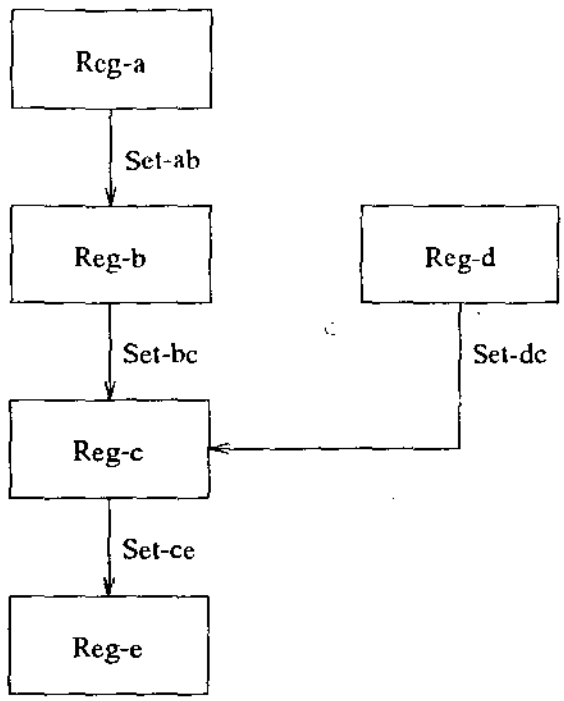

Figura 4.2: Diagrama de um esquema genérico rede

```
A. FIND NEXT Reg-c WITHIN Set-bc
   IF DB-STATUS = END-OF-SET
      ERASE Reg-b
      GO TO C.
B. FIND NEXT Reg-e WITHIN Set-ce
   IF DB-STATUS = END-OF-SET
      ERASE Reg-c
      GO TO A.
   ERASE Reg-e
   GO TO B.
C. EXIT.
```

Figura 4.3: Exemplo de trecho de programa que substitui o comando ERASE ALL

4.3.4 Comandos para atualizar set-occurrences

I) CONNECT <record-type> TO <set-type>

A: Verificar nos mapeamentos Mef e Mfe em quais SBDs componentes o <set-type> está definido

Se <set-type> está definido no SBD local

Então

B: Incluir no programa gerado um comando equivalente ao comando examinado usando a sintaxe do esquema local

Se <set-type> não está definido em outros SBDs componentes

Então

Passar a examinar outro comando do programa externo rede

Senão

C: Incluir no programa gerado o teste do status da operação do passo **B**

D: Incluir no programa gerado uma chamada ao monitor de comunicação para os SBDs remotos em que <set-type> está definido. Os parâmetros de cada chamada são os valores dos atributos identificadores de <record-type> mais os atributos identificadores do registro mestre de <set-type>.

E: Similar ao passo **E** do comando FIND ANY mudando-se apenas o tipo de chamada de FA para CRS

Senão

F: Idem passo **D**
G: Idem passo **E**

Exemplo I.1: CONNECT *Empregado* TO *Gerente-Proj*

Passo A - Verifica-se que o *set-type Gerente-Proj* está definido nos dois SBDs componentes.

Passo B - Inclui-se no programa gerado:

```
CONNECT Empregado TO Gerente-Proj
```

Passo C - Inclui-se no programa gerado:

```
IF DB-STATUS = 0
```

Passo D - Inclui-se no programa gerado:

```
CALL "Monitor" USING  programa, "ncall", call-status,
                      matricula, cod-projeto

IF   call-status NOT EQUAL 0
     GO TO cancela-programa.
```

Passo E - Inclui-se os dados da chamada na tabela de chamadas:

TC.prog	←	programa
TC.NC	←	"ncall"
TC.SBD	←	"SBD2"
TC.Tipo	←	"CRS"
TC.Rec	←	"Empregado"
TC.Set	←	"Gerente-Proj"
TC.P-List	←	"matricula, cod-projeto"
NCALL	←	NCALL + 1

II) DISCONNECT <record-type> FROM <set-type>

Mesmo algoritmo usado para o comando CONNECT trocando-se apenas o tipo da chamada de CRS para DRS.

III) RECONNECT <record-type> TO <set-type>

Mesmo algoritmo usado para o comando CONNECT trocando-se apenas o tipo da chamada de CRS para RRS.

4.4 Algoritmos para tradução REDE-GORDAS

O módulo Tradutor complementa as informações da tabela de chamadas incluindo nela as transações GORDAS sobre o esquema componente referenciado na chamada.

As transformações feitas para cada tipo de chamada são mostradas a seguir.

4.4.1 Operações de localização e recuperação de dados

1) Tipo da chamada = FA ou FD

Para esses tipos de chamada é gerada a seguinte transação GORDAS:

GET $<a_1,a_2,\ldots,a_n>$ OF <ECR-name>
WHERE $<p_1>$ = param-1
AND $<p_2>$ = param-2
⋮
AND $<p_n>$ = param-n

onde:

- <ECR-name> é o nome do construtor (tipo de entidade ou relacionamento) no esquema componente que corresponde ao *record-type* do esquema externo (TC.Rec)
- $<a_1,a_2,\ldots,a_n>$ é a lista dos nomes dos atributos de <ECR-name> que correspondem aos nomes dos campos em TC.A-list.
- $<p_1,p_2,\ldots,p_n>$ é a lista dos nomes dos atributos de <ECR-name> que correspondem aos nomes dos campos em TC.P-list.

Exemplo: Para a chamada incluída no exemplo 1.2 da seção 4.3.1 (TC.Tipo = "FA", TC.Rec = "Empregado", TC.A-list = "salario-atual", TC.P-list = "salario-atual, matricula") a transação GORDAS gerada é:

```
GET   salario OF Empregado
WHERE salario = param-1
AND   matricula = param-2
```

II) Tipo da chamada = FFS ou FNS ou FLS ou FPS

Para esses tipos de chamada é gerada a seguinte transação GORDAS:

GET <a_1,a_2,...,a_n> OF <nome-conexão> OF <ECR-name>
WHERE <p_1> [OF <nome-conexão>] = param-1
AND <p_2> [OF <nome-conexão>] = param-2
⋮
AND <p_n> [OF <nome-conexão>] = param-n

onde:

- <ECR-name> é o nome do tipo de entidade no esquema componente que corresponde ao *record-type* mestre do *set-type* contido em TCSet.
- <nome-conexão> é o nome de conexão 1 associado à participação de <ECR-name> no relacionamento do esquema componente que corresponde a TCSet.
- <a_1,a_2,...,a_n> é a lista dos nomes dos atributos do construtor do esquema componente (tipo de entidade ou relacionamento) que correspondem aos nomes dos campos em TCA-list.
- <p_1,p_2,...,p_n> é a lista dos nomes dos atributos dos construtores do esquema componente que correspondem aos nomes dos campos em TCP-list.

Exemplo: Para a chamada incluída no exemplo 1.2 da seção 4.3.2 (TCTipo = "FPS", TCRec = "Projeto", TCSet = "Cliente-proj", TCA-list = "cod-projeto, nome-projeto, ..., multa-atraso", TCP-list = "cod-cliente") a transação GORDAS gerada é:

```
GET <cod-projeto,nome-projeto,...,multa-atraso> OF projetos OF Cliente
WHERE cod-cliente = param-1
```

III) Tipo da chamada = FOS

Para esse tipo de chamada será gerada a seguinte transação GORDAS:

GET <a_1,a_2,...,a_n> OF <ECR-name>

WHERE <p_1> OF <nome-conexão> OF <ECR-name> = param-1
AND <p_2> OF <nome-conexão> OF <ECR-name> = param-2
⋮
AND <p_n> OF <nome-conexão> OF <ECR-name> = param-n

onde:

- <ECR-name> é o nome do tipo de entidade no esquema componente que corresponde ao *record-type* mestre do *set-type* contido em TC.Set (TC.Rec).
- <nome-conexão> é o nome de conexão 1 associado à participação de <ECR-name> no relacionamento do esquema componente que corresponde a TC.Set.
- <a_1,a_2,...,a_n> é a lista dos nomes dos atributos de <ECR-name> que correspondem aos nomes dos campos em TC.A-list.
- <p_1,p_2,...,p_n> é a lista dos nomes dos atributos do construtor do esquema componente (tipo de entidade ou relacionamento) que correspondem aos nomes dos campos em TC.P-list.

Exemplo: Para a chamada incluída no exemplo VIII.1 da seção 1.3.1 (TC.Tipo = "FOS", TC.Rec = "Cliente", TC.Set = "Cliente-proj", TC.A-list = "cod-cliente, nome-cliente, endereco, telefone", TC.P-list = "cod-projeto") a transação GORDAS gerada é:

```
GET   <cod-cliente,nome-cliente,endereco,telefone> OF Cliente
WHERE cod-projeto OF projetos OF Cliente = param-1
```

4.4.2 Operações de atualização de dados

I) Tipo da chamada = SR

Se TC.Rec corresponde a um tipo de entidade no esquema componente

Então

A: Incluir uma transação GORDAS contendo um comando para inclusão de uma entidade no tipo de entidade do esquema componente que corresponde a TC.Rec e um comando para inclusão de uma instância em cada relacionamento do esquema componente que corresponda a um *set-type* do esquema externo em que TC.Rec é membro com participação obrigatória (SET RETENTION FIXED/MANDATORY)

Senão

B: Incluir uma transação GORDAS contendo um comando para inclusão de uma instância no relacionamento do esquema componente que corresponde a TC.Rec.

O formato geral das transações para o caso **A** é:

INSERT <entity-name>
ATTRIBUTES <a_i> = param-i,...,<a_j> = param-j
ADD TO RELATIONSHIP <relac-name$_k$>
WHERE <ent-part$_k$> : <ent-part$_k$.key> = param-m
<entity-name> : <entity-name.key> = param-p
ATTRIBUTES <r_1> = param-s,...,<r_n> = param-t

onde:

. <entity-name> é o nome do tipo de entidade no esquema componente que corresponde ao *record-type* contido em TC.Rec.

. <relac-name$_k$>, $1 \leq k \leq n$, é um relacionamento no esquema componente que corresponde a um *set-type* no esquema externo em que TC.Rec participa obrigatoriamente como membro.

. <ent-part$_k$> é o tipo de entidade do esquema componente que corresponde ao *record-type* mestre do *set-type* representado por <relac-name$_k$>.

. <a_i,...,a_j>, $1 \leq i, j \leq n$, são atributos de <entity-name> que estão contidos em TC.P-list.

. <ent-part$_k$.key> é o atributo identificador de <ent-part$_k$>.

. <entity-name.key> é o atributo identificador de <entity-name>.

. <r_1,...,r_n> são os atributos de <relac-name$_k$>.

O formato geral das transações para o caso **B** é:

ADD TO RELATIONSHIP <relac-name>
WHERE <ent-part$_1$> : <ent-part$_1$.key> = param-j
ATTRIBUTES <r_1> = param-k
$\vdots$
<r_n> = param-p

onde:

- <relac-name> é o relacionamento no esquema componente que corresponde ao *record-type* contido em TC.Rec.
- <ent-part$_i$>, $1 \leq i \leq n$, é o tipo de entidade do esquema componente que corresponde ao *record-type* mestre de um dos *set-type* do esquema externo que tem TC.Rec como membro.
- <ent-part$_i$.key> é o atributo identificador de <ent-part$_i$>.
- <r_1,...,r_n> são os atributos de <relac-name>.

Exemplo-1: Para a chamada incluída no exemplo I.1 da seção 1.3.3 (TC.Tipo = "SR", TC.Rec = "Projeto", TC.P-list = "cod-projeto,nome-projeto, valor-hora-projeto, num-contrato, valor-contrato, multa-rescisao, multa-atraso, cod-cliente") a transação GORDAS gerada é:

```
INSERT INTO Projeto
       ATTRIBUTES cod-projeto = param-1,
                  nome-projeto = param-2, valor-hora = param-3,
                  num-contrato = param-4, valor-contrato = param-5,
                  multa-rescisao = param-6, multa-atraso = param-7
ADD TO RELATIONSHIP Cliente-Proj
       WHERE Cliente : cod-cliente = param-8
             Projeto : cod-projeto = param-1
```

Exemplo-2: Para a chamada incluída no exemplo I.2 da seção 1.3.3 (TC.Tipo = "SR", TC.Rec = "Emp-Etapa", TC.P-list = "cod-projeto, num-etapa, matricula, horas-trab) a transação GORDAS gerada é:

```
ADD TO RELATIONSHIP Empreg-Proj
```

```
WHERE Empregado : matricula = param-3
      Etapa-projeto : cod-projeto = param-1
                      num-etapa = param-2
ATTRIBUTES horas-trab = param-4
```

II) Tipo da chamada = ER

Se TC.Rec corresponde a um tipo de entidade no esquema componente

Então

A: Incluir uma transação GORDAS contendo um comando para exclusão de uma entidade no tipo de entidade do esquema componente que corresponde a TC.Rec

Senão

B: Incluir uma transação GORDAS contendo um comando para exclusão de uma instância no relacionamento do esquema componente que corresponde a TC.Rec.

O formato geral das transações para o caso **A** é:

DELETE CURRENT FROM <entity-name>

onde:

. <entity-name> é o nome do tipo de entidade no esquema componente que corresponde ao *record-type* contido em TC.Rec.

O formato geral das transações para o caso **B** é:

REMOVE CURRENT FROM RELATIONSHIP <relac-name>

onde:

. <relac-name> é o relacionamento no esquema componente que corresponde ao *record-type* contido em TC.Rec.

III) Tipo da chamada = MR

Para esse tipo de chamada é gerada a seguinte transação GORDAS:

MODIFY CURRENT <p_1,p_2,...,p_n> OF <ECR-name> TO
param-1, param-2, ..., param n

onde:

. <ECR-name> é o nome do construtor do esquema componente que corresponde ao *record-type* do esquema externo (TC.Rec).

. <p_1,p_2,...,p_n> é a lista dos nomes dos atributos de <ECR-name> que correspondem aos nomes dos campos em TC.P-list.

IV) Tipo da chamada = CRS

Para esse tipo de chamada é gerada a seguinte transação GORDAS:

ADD TO RELATIONSHIP <relac-name>
WHERE <ent-part> : <ent-part.key> = param-k
<entity-name> : <entity-name.key> = param-m

onde:

. <entity-name> é o nome do tipo de entidade no esquema componente que corresponde ao *record-type* contido em TC.Rec.

. <relac-name> é o relacionamento no esquema componente que corresponde ao *set-type* contido em TC.Set.

. <ent-part> é o tipo de entidade do esquema componente que corresponde ao *record-type* mestre do *set-type* contido em TC.Set.

. <ent-part.key> é o atributo identificador de <ent-part>.

. <entity-name.key> é o atributo identificador de <entity-name>

V) Tipo da chamada = DRS

Para esse tipo de chamada é gerada a seguinte transação GORDAS:

REMOVE FROM RELATIONSHIP <relac-name>
WHERE <ent-part> : <ent-part.key> = param-k
<entity-name> : <entity-name.key> = param-m

onde:

. <entity-name> é o nome do tipo de entidade no esquema componente que corresponde ao *record-type* contido em TC.Rec.

. <relac-name> é o relacionamento no esquema componente que corresponde ao *set-type* contido em TC.Set.

. <ent-part> é o tipo de entidade do esquema componente que corresponde ao *record-type* mestre do *set-type* contido em TC.Set.

. <ent-part.key> é o atributo identificador de <ent-part>

. <entity-name.key> é o atributo identificador de <entity-name>

VI) Tipo da chamada = RRS

Para esse tipo de chamada é gerada a seguinte transação GORDAS:

REMOVE FROM RELATIONSHIP <relac-name>
WHERE <entity-name> : <entity-name.key> = param-m

ADD TO RELATIONSHIP <relac-name>
WHERE <ent-part> : <ent-part.key> = param-k
<entity-name> : <entity-name.key> = param-m

onde:

. <entity-name> é o nome do tipo de entidade no esquema componente que corresponde ao *record-type* contido em TC.Rec.

. <relac-name> é o relacionamento no esquema componente que corresponde ao *set-type* contido em TC.Set.

. <ent-part> é o tipo de entidade do esquema componente que corresponde ao *record-type* mestre do *set-type* contido em TC.Set.

. <ent-part.key> é o atributo identificador de <ent-part>

. <entity-name.key> é o atributo identificador de <entity-name>

4.5 Algoritmos para tradução GORDAS-SQL

No processo de construção de programas remotos que atuam sobre um SBD relacional, cada transação GORDAS sobre o esquema componente, contida na relação de chamadas, é convertida em uma transação SQL equivalente sobre o esquema local do SBD em questão.

Os algoritmos utilizam os mapeamentos entre o esquema componente e o esquema local para identificarem o nome das relações e atributos que devem ser utilizados nos comando SQL. Como já foi comentado nesta dissertação, é possível incluir rotinas para validação de restrições de integridade introduzidas no esquema componente durante o processo de tradução de esquemas locais para o modelo de dados comum. Todavia, para não estender em demasia as discussões, só serão consideradas as restrições de integridade estruturais previstas no esquema externo rede, que foram garantidas na tradução da LMD rede para a linguagem intermediária GORDAS (seção 4.4.2).

A seguir serão apresentados os algoritmos para os diversos tipos de transações GORDAS apresentadas na seção 4.4. As transações SQL obtidas se referem ao esquema do SBD relacional das figuras 1.11 e 1.12.

4.5.1 Operações de localização e recuperação de dados

I) tipo da chamada = FA ou FD

Transação GORDAS:

GET <a_1,a_2,...,a_n> OF <ECR-name>
WHERE <p_1> = param-1
AND <p_2> = param-2
⋮
AND <p_n> = param-n

Transação SQL:

SELECT <a_{L1},a_{L2},...,a_{Ln}>
FROM <R1>
WHERE <p_{L1}> = param-1
AND <p_{L2}> = param-2
⋮

AND $<p_{Ln}>$ = param-n

onde:

- <R1> é o nome da relação do esquema local que corresponde ao construtor <ECR-name> do esquema componente
- $<a_{L1},a_{L2},\ldots,a_{Ln}>$ é a lista dos nomes dos atributos de <R1> que correspondem aos atributos $<a_1,a_2,\ldots,a_n>$ de <ECR-name>.
- $<p_{L1},p_{L2},\ldots,p_{Ln}>$ é a lista dos nomes dos atributos de <R1> que correspondem aos atributos $<p_1,p_2,\ldots,p_n>$ de <ECR-name> .

Exemplo: Para a transação GORDAS gerada no item I da seção 1.1.1 é gerada a seguinte transação SQL:

```
SELECT salario
FROM   Empregado
WHERE  salario = param-1
AND    matricula = param-2
```

II) tipo da chamada = FFS, FNS, FLS ou FPS

Transação GORDAS:

GET $<a_1,a_2,\ldots,a_n>$ OF <nome-conexão> OF <ECR-name>
WHERE $<p_1>$ [OF <nome-conexão>] = param-1
AND $<p_2>$ [OF <nome-conexão>] = param-2
⋮
AND $<p_n>$ [OF <nome-conexão>] = param-n

Transação SQL:

Caso A: Se o relacionamento do esquema componente alcançado através do caminho *<nome-conexão> OF <ECR-name>* for representado por uma relação à parte no esquema local, então a transação SQL gerada é:

SELECT $<(R2|R3).a_{L1}, (R2|R3).a_{L2}, \ldots, (R2|R3).a_{Ln}>$
FROM <R2, R3>
WHERE $<(R2|R3).p_{L1}>$ = param-1

```
AND     <(R2|R3).p_L2> = param-2
        ⋮
AND     <(R2|R3).p_Ln> = param-n
AND     <R3.fk_R2> = <R2.k>
```

onde:

- <R2> é o nome da relação do esquema local que corresponde ao tipo de entidade do esquema componente alcançado através do caminho *<nome-conexão> OF <ECR-name>*.
- <R3> é o nome da relação do esquema local que corresponde ao relacionamento do esquema componente alcançado através do caminho *<nome-conexão> OF <ECR-name>*.
- $<a_{L1}, a_{L2}, \ldots, a_{Ln}>$ é a lista dos nomes dos atributos de <R2> ou de <R3> que correspondem aos atributos $<a_1, a_2, \ldots, a_n>$.
- $<p_{L1}, p_{L2}, \ldots, p_{Ln}>$ é a lista dos nomes dos atributos de <R2> ou de <R3> que correspondem aos atributos $<p_1, p_2, \ldots, p_n>$.
- $<R3.fk_{R2}>$ é o atributo da relação <R3> que referencia a relação <R2>.
- <R2.k> é a chave primária da relação <R2> referenciada pelo atributo da relação <R3>.

Caso B: Se o relacionamento do esquema componente alcançado através do caminho *<nome-conexão> OF <ECR-name>* for representado no esquema local pela presença de uma chave estrangeira em uma das relações que representam as entidades participantes do relacionamento, então a transação SQL gerada é:

```
SELECT  <R2.a_L1, R2.a_L2, ..., R2.a_Ln>
FROM    <R1, R2>
WHERE   <(R1|R2).p_L1> = param-1
AND     <(R1|R2).p_L2> = param-2
        ⋮
AND     <(R1|R2).p_Ln> = param-n
AND     <R2.fk_R1> = < R1.k >
```

onde:

. <R2> é o nome da relação do esquema local que corresponde ao tipo de entidade do esquema componente alcançado através do caminho *<nome-conexão> OF <ECR-name>*.

. <R1> é o nome da relação do esquema local que corresponde ao construtor <ECR-name> do esquema componente.

. $<a_{L1},a_{L2},\ldots,a_{Lm}>$ é a lista dos nomes dos atributos de <R2> que correspondem aos atributos $<a_1,a_2,\ldots,a_n>$.

. $<p_{L1},p_{L2},\ldots,p_{Lm}>$ é a lista dos nomes dos atributos de <R1> ou de <R2> que correspondem aos atributos $<p_1,p_2,\ldots,p_n>$.

. $<R2.fk_{R1}>$ é a chave estrangeira da relação <R2> que referencia a relação <R1>.

. <R1.k> é a chave primária da relação <R1> referenciada pela chave estrangeira de <R2>.

Exemplo: Para a transação GORDAS gerada no item II da seção 4.1.1 é gerada a seguinte transação SQL:

```
SELECT Projeto.cod-projeto, ..., Projeto.multa-atraso
FROM   Projeto, Cliente-Proj
WHERE  Cliente-Proj.cod-cliente = param-1
AND    Cliente-Proj.cod-projeto = Projeto.cod-projeto
```

III) tipo da chamada = FOS

Transação GORDAS:

```
GET    <a1,a2,...,an> OF <ECR-name>
WHERE  <p1> OF <nome-conexão> = param-1
AND    <p2> OF <nome-conexão> = param-2
                  :
AND    <pn> OF <nome-conexão> = param-n
```

Transação SQL:

Caso A: Se o relacionamento do esquema componente alcançado através do caminho *<nome-conexão> OF <ECR-name>* for representado por uma relação à parte no esquema local, então a transação SQL gerada é:

```
SELECT   <R1.a_L1,R1.a_L2, ..., R1.a_Ln>
FROM     <R1, R3>
WHERE    <R3.p_L1> = param-1
AND      <R3.p_L2> = param-2
         :
AND      <R3.p_Ln> = param-n
AND      <R3.fk_R1> = < R1.k >
```

onde:

- <R1> é o nome da relação do esquema local que corresponde ao construtor <ECR-name> do esquema componente.
- <R3> é o nome da relação do esquema local que corresponde ao relacionamento do esquema componente alcançado através do caminho *<nome-conexão> OF <ECR-name>*.
- $<a_{L1},a_{L2},\ldots,a_{Ln}>$ é a lista dos nomes dos atributos de <R1> que correspondem aos atributos $<a_1,a_2,\ldots,a_n>$.
- $<p_{L1},p_{L2},\ldots,p_{Ln}>$ é a lista dos nomes dos atributos de <R3> que correspondem aos atributos $<p_1,p_2,\ldots,p_n>$.
- $<R3.fk_{R1}>$ é o atributo da relação <R3> que referencia a relação <R1>.
- <R1.k> é a chave primária da relação <R1> referenciada pelo atributo da relação <R3>.

Caso B: Se o relacionamento do esquema componente alcançado através do caminho *<nome-conexão> OF <ECR-name>* for representado no esquema local pela presença de uma chave estrangeira em uma das relações que representam as entidades participantes do relacionamento, então a transação SQL gerada é:

```
SELECT   <R1.a_L1, R1.a_L2, ..., R1.a_Ln>
FROM     <R1, R2>
```

WHERE <R2.p_{L1}> = param-1
AND <R2.p_{L2}> = param-2
⋮
AND <R2.p_{Ln}> = param-n
AND <R2.fk_{R1}> = < *R1.k* >

onde:

- <R1> é o nome da relação do esquema local que corresponde ao construtor <ECR-name> do esquema componente.
- <R2> é o nome da relação do esquema local que corresponde ao tipo de entidade do esquema componente alcançado através do caminho <*nome-conexão*> *OF* <*ECR-name*>.
- <a_{L1},a_{L2},...,a_{Ln}> é a lista dos nomes dos atributos de <R1> que correspondem aos atributos <a_1,a_2,...,a_n>.
- <p_{L1},p_{L2},...,p_{Lm}> é a lista dos nomes dos atributos de <R2> que correspondem aos atributos <p_1,p_2,...,p_m>.
- <R2.fk_{R1}> é a chave estrangeira da relação <R2> que referencia a relação <R1>.
- <R1.k> é a chave primária da relação <R1> referenciada pela chave estrangeira de <R2>.

Exemplo: Para a transação GORDAS gerada no item III da seção 4.4.1 é gerada a seguinte transação SQL:

```
SELECT Cliente.cod-Cliente, ..., Cliente.telefone
FROM   Cliente, Cliente-Proj
WHERE  Cliente-Proj.cod-projeto = param-1
AND    Cliente-Proj.cod-cliente = Cliente.cod-cliente
```

4.5.2 Operações de atualização de dados

I) Tipo da chamada = SR

Transação GORDAS - Caso A:

INSERT <entity-name>
ATTRIBUTES <a_i> = param-i, ..., <a_j> = param-j

ADD TO RELATIONSHIP <relac-name$_k$>
WHERE <ent-part$_k$> : <ent-part$_k$.key> = param-m
<entity-name> : <entity-name.key> = param-p
ATTRIBUTES <r_1> = param-s, ..., <r_n> = param-t

Algoritmo:

A: Obter no mapeamento componente-local (Mcl) o nome da relação (<R0>) do esquema local que corresponde a <entity-name> e os nomes dos atributos de <R0> (<a_{Li}>, ..., <a_{Lj}>) que correspondem aos nomes dos atributos da lista <a_i>, ..., <a_j>.

B: Incluir na transação SQL o seguinte comando

INSERT INTO <R0> (<a_{Li}, ..., a_{Lj}>)
VALUES (param-i, ..., param-j)

C: Para cada relacionamento <relac-name$_k$>, $1 \leq k \leq n$, contido na transação GORDAS buscar em Mcl o construtor correspondente no esquema local.

Se <relac-name$_k$> é representado através de uma chave estrangeira em <R0>

Então

D: Obter em Mcl o nome da relação <R_k> que corresponde a <ent-part$_k$> no esquema componente

E: Incluir na lista de atributos do comando inserido no passo **B** a chave estrangeira (fk_{R_k}) de <R0> que referencia a relação <R_k> correspondente a <ent-part$_k$>. O comando alterado fica assim:

INSERT INTO <R0> (<a_{Li}>, ..., <a_{Lj}>.<fk_{R_k}>)
VALUES (param-i, ..., param-j.param-m)

Senão

G: Obter em Mcl o nome dos atributos (R_k.K1 e R_k.K2) de $<R_k>$ que referenciam as relações <R1> e <R2> do esquema local as quais correspondem aos tipos de entidade $<entity\text{-}name_k>$ e $<ent\text{-}part_k>$ do esquema componente. Obter também os nomes dos atributos ($<r_{L1}>, \ldots, <r_{Lm}>$) de $<R_k>$ que correspondem aos atributos da lista $<r_1>, \ldots, <r_n>$.

H: Incluir na transação SQL o seguinte comando:

INSERT INTO $<R_k>$ ($<R_k.K1>$,$<R_k.K2>$,$<r_{L1}>, \ldots, <r_{Lm}>$)
VALUES (param-m,param-p,param-s, ..., param-t)

Transação GORDAS - Caso B :

ADD TO RELATIONSHIP <relac-name>
WHERE $<ent\text{-}part_1>$: $<ent\text{-}part_1.key>$ = param-i
⋮
$<ent\text{-}part_n>$: $<ent\text{-}part_n.key>$ = param-j
ATTRIBUTES $<r_1>$ = param-k, ..., $<r_n>$ = param-m

Algoritmo:

A: Obter no mapeamento componente-local (Mcl) o nome da relação (<R0>) do esquema local que corresponde ao relacionamento <relac-name> e os nomes dos atributos de <R0> ($<r_{Li}>, \ldots, <r_{Lj}>$) que correspondem aos nomes dos atributos da lista $<r_1>, \ldots, <r_n>$.

B: Obter em Mcl o nome dos atributos de <R0> ($<fk_{R_i}>$, $1 \leq i \leq n$) que referenciam as chaves primárias das relações $<R_1>, \ldots, <R_n>$ que correspondem aos tipos de entidade $<ent\text{-}part_1>, \ldots, <ent\text{-}part_n>$.

C: Incluir na transação SQL o seguinte comando:

INSERT INTO <R0> ($<fk_{R_1}>, \ldots, <fk_{R_n}>$,$<r_{L1}>, \ldots, <r_{Lm}>$)
VALUES (param-i, ..., param-j, param-k, ..., param-m)

Exemplos: As transações GORDAS geradas nos exemplos da seção 4.1.2 são traduzidas para as seguintes transações SQL:

```
Ex1:   INSERT INTO Projeto(cod-projeto, ..., multa-atraso)
```

```
                    VALUES (param-1, ..., param-7)

          INSERT INTO Cliente-Proj(cod-cliente, cod-projeto)
                    VALUES (param-8, param-1)

Ex2:      INSERT INTO Empreg-Proj(matricula, cod-projeto,
                                  num-etapa, horas-trab)
                    VALUES (param-3, param-1, param-2, param-4)
```

II) Tipo da chamada = ER

Transação GORDAS - Caso A:

DELETE CURRENT FROM <entity-name>

Algoritmo

A: Obter no mapeamento federado-componente (Mfc) e componente-local (Mcl) o nome da relação (<R0>) do esquema local que corresponde a <entity-name>.

B: Para cada cursor (< cursor-name$_i$>, $1 \leq i \leq n$), que foi declarado no programa criado pelo módulo *Construtor-de-Programas* e que manipula dados da relação <R0> gerar o seguinte comando SQL:

DELETE FROM <R0>
WHERE CURRENT OF <cursor-name$_i$>

C: Incluir cada transação gerada no passo B em um módulo separado do programa. O módulo é rotulado pelo tipo da chamada (ER) concatenado com o nome do cursor (<cursor-name$_i$>).

Transação GORDAS - Caso B:

REMOVE CURRENT FROM RELATIONSHIP <relac-name>

Algoritmo:

A: Obter no mapeamento componente-local (Mcl) o tipo de construção do esquema local que representa o relacionamento <relac-name> no esquema componente.

Se <relac-name> corresponde a relação <R0> no esquema local
Então

B: Gerar as transações SQL da mesma forma que foram geradas as transações GORDAS no caso A.

Senão

C: Obter em Mcl o nome das relações <R1> e <R2> que representam os tipos de entidade do esquema componente que participam de <relac-name>. Obter também o atributo da relação <R1> que funciona como chave estrangeira de <R2> (<R1.fk_{R2}>).

D: Para cada cursor (<cursor-name$_i$>, $1 \leq i \leq n$), que foi declarado no programa criado e que manipula dados da relação <R1> gerar o seguinte comando SQL:

UPDATE <R1>
SET <R1.fk_{R2}> = NULL
WHERE CURRENT OF <cursor-name$_i$>

E: Incluir cada transação gerada no passo D em um módulo separado do programa. O módulo é rotulado pelo tipo da chamada (ER) concatenado com o nome do cursor (<cursor-name$_i$>).

III) Tipo da chamada = MR

Transação GORDAS

MODIFY CURRENT <a_1,a_2,...,a_n> OF <ECR-name> TO
param-1, param-2, ..., param-n

Algoritmo

A: Obter no mapeamento componente-local (Mcl) o nome da relação (<R0>) que corresponde a < ECR-name>. Obter também o nome dos atributos de (<R0>) que correspondem aos atributos <a_1,a_2,...,a_n> de <ECR-name>.

B: Para cada cursor (< cursor-name$_i$>, $1 \leq i \leq n$), que foi declarado no programa criado e que manipula dados da relação <R0> gerar o seguinte comando SQL:

UPDATE <R0>

SET <a_{L1}> = param-1,
<a_{L2}> = param-2,
⋮
<a_{Ln}> = param-n
WHERE CURRENT OF <cursor-name$_i$>

C: Incluir cada transação gerada no passo B em um módulo separado do programa. O módulo é rotulado pelo tipo da chamada (MR) concatenado com o nome do cursor (<cursor-name$_i$>).

IV) Tipo da chamada = CRS

Transação GORDAS

ADD TO RELATIONSHIP <relac-name>
WHERE <entity-name-1> : <entity-name-1.key> = param-i
<entity-name-2> : <entity-name-2.key> = param-j

Algoritmo:

A: Obter no mapeamento componente-local (Mcl) o tipo de construção do esquema local que representa o relacionamento <relac-name> no esquema componente.

Se <relac-name> corresponde a relação <R0> no esquema local
Então

B: Obter em Mcl o nome das relações <R1> e <R2> que representam os tipos de entidade do esquema componente que participam de <relac-name>. Obter também os atributos da relação <R0> que referenciam as relações <R1> e <R2> (<R0.fk$_{R1}$> e <R0.fk$_{R2}$>).

C: Gerar seguinte transação SQL:

INSERT INTO <R0>
(<R0.fk$_{R1}$>, <R0.fk$_{R2}$>) VALUES (param-i, param-j)

Senão

D: Obter em Mcl o nome das relações <R1> e <R2> que representam os tipos de entidade <entity-name-1> e <entity-name-2> do esquema componente.

E: Obter em Mcl o nome do atributo de <R2> que funciona como chave estrangeira de <R1> (<R2.fk$_{R1}$>)

F: Gerar a seguinte transação SQL:

UPDATE <R2>
SET <R2.fk$_{R1}$> = param-i
WHERE <R2.key> = param-j

V) Tipo da chamada = DRS

Transação CORDAS

REMOVE FROM RELATIONSHIP <relac-name>
WHERE <entity-name-1> : <entity-name-1.key> = param-i
<entity-name-2> : <entity-name-2.key> = param-j

Algoritmo:

A: Obter no mapeamento componente-local (Mcl) o tipo de construção do esquema local que representa o relacionamento <relac-name> no esquema componente.

Se <relac-name> corresponde a relação <R0> no esquema local
Então

B: Obter em Mcl o nome das relações <R1> e <R2> que representam os tipos de entidade do esquema componente que participam de <relac-name>. Obter também os atributos da relação <R0> que referenciam as relações <R1> e <R2> (<R0.fk$_{R1}$> e <R0.fk$_{R1}$>).

C: Gerar a seguinte transação SQL:

DELETE FROM <R0>
WHERE <R0.fk$_{R1}$> = param-i
AND <R0.fk$_{R2}$> = param-j

Senão

D: Obter em Mcl o nome das relações <R1> e <R2> que representam os tipos de entidade <entity-name-1> e <entity-name-2> do esquema componente.

E: Obter em Mcl o nome do atributo de <R2> que funciona como chave estrangeira de <R1> (<R2.fk$_{R1}$>)

F: Gerar a seguinte transação SQL:

UPDATE <R2>
SET <R2.fk$_{R1}$> = NULL
WHERE <R2.key> = param-j

VI) Tipo da chamada = RRS

Transação GORDAS:

REMOVE FROM RELATIONSHIP <relac-name>
WHERE <entity-name-1> : TRUE
<entity-name-2> : <entity-name-2.key> = param-m

ADD TO RELATIONSHIP <relac-name>
WHERE <entity-name-1> : <entity-name-1.key> = param-k
<entity-name-2> : <entity-name-2.key> = param-m

Algoritmo:

A: Obter no mapeamento componente-local (Mcl) o tipo de construção do esquema local que representa o relacionamento <relac-name> no esquema componente.

Se <relac-name> corresponde a relação <R0> no esquema local
Então

B: Obter em Mcl o nome das relações <R1> e <R2> que representam os tipos de entidade do esquema componente que participam de <relac-name>. Obter também os atributos da relação <R0> que referenciam as relações <R1> e <R2> (<R0.fk$_{R1}$> e <R0.fk$_{R2}$>).

C: Gerar a seguinte transação SQL:

UPDATE <R0>
SET <R0.fk$_{R1}$> = param-k
WHERE <R0.fk$_{R2}$> = param-m

Senão

D: Obter em Mcl o nome das relações <R1> e <R2> que representam os tipos de entidade <entity-name-1> e <entity-name-2> do esquema componente.

E: Obter em MeI o nome do atributo de <R2> que funciona como chave estrangeira de <R1> (<$R2.fk_{R1}$>)

F: Gerar a seguinte transação SQL:

```
UPDATE <R2>
        SET <R2.fk_R1> = param-k
        WHERE <R2.key> = param-m
```

Capítulo 5

Transformação de operações sobre esquema externo relacional

5.1 Introdução

Neste capítulo será proposta uma arquitetura para suportar o acesso transparente aos dados de um SBDH através de programas de aplicação construídos sobre esquemas externos no modelo relacional.

Os programas de aplicação dos usuários de um SBDH que atuam sobre esquemas externos relacionais são construídos usando comandos de uma LMD não procedimental embutidos nos comandos de uma linguagem hospedeira de propósito geral.

O uso de uma LMD não procedimental permite a simplificação, em alguns aspectos, da arquitetura necessária à obtenção de acesso transparente e integrado aos dados do SBDH. Como os dados são recuperados como conjuntos, não é mais exigida a manutenção da correspondência e sincronização entre os indicadores de estado corrente do programa local e os cursores ou indicadores de estado corrente do programa que atua sobre um SBD remoto, tal como foi descrito no capítulo anterior.

Em contrapartida, os comandos de uma linguagem não procedimental são muito mais solisticados e diversificados do que os comandos de uma linguagem procedimental, exigindo um esforço maior no processo de transformação de operações. Além disso, existem várias maneiras diferentes para se dividir e processar um comando de uma linguagem não procedimental que envolve dados distribuídos e a escolha da melhor alternativa é um problema a mais que deve ser enfrentado.

A descrição da arquitetura será feita na seção 5.2. Na seção 5.3 são mostrados algoritmos usados pelo tradutor SQL-GORDAS. Finalmente na seção 5.4 são discutidos algoritmos usados pelos construtores de programas remotos na tradução GORDAS-REDE. Os exemplos da aplicação dos algoritmos apresentados neste capítulo se referem às relações contidas na figura 3.34 e utilizam os mapeamentos obtidos durante o processo de tradução de esquemas tratado no capítulo 3.

5.2 Arquitetura proposta

A figura 5.1 mostra a arquitetura proposta para permitir o acesso aos dados de um SBDH a partir de um programa de aplicação construído sobre um esquema externo relacional. Os principais módulos que compõem essa arquitetura são o *Pré-Compilador* (1), o *Tradutor-Decompositor* (2), o *Construtor-de-Programas* (3), o *Executor* (4) e o *Monitor-de-Comunicação* (5). Cada um desses módulos será descrito a seguir.

5.2.1 Pré-Compilador

Esse módulo modifica o programa de aplicação escrito pelo usuário, substituindo comandos SQL que referenciam dados não disponíveis no SBD local relacional por chamadas aos SBDs remotos que contêm esses dados.

O Pré-Compilador examina cada declaração de cursor no programa original do usuário e verifica nos mapeamentos entre o esquema externo e o esquema federado (Mef) e entre o esquema federado e os esquemas componentes (Mfc) a distribuição e a localização dos dados contidos na expressão *Select-From-Where* (SFW) associada ao cursor declarado. Três casos podem ocorrer: (a) todos os dados referenciados na expressão SFW estão definidos exclusivamente no SBD local; (b) pelo menos um dos dados não está definido no SBD local; (c) todos os dados estão definidos no SBD local e pelo menos um deles está replicado em outro SBD componente.

Para o caso (a) o Pré-Compilador simplesmente substitui cada comando SQL associado ao cursor declarado por um comando SQL equivalente, usando os construtores do esquema local. Para tal, são pesquisados os mapeamentos entre o esquema componente ECR e o esquema local relacional (Mel).

Para o caso (b) o Pré-Compilador executa as seguintes ações:

1. Retira do programa pré-compilado a declaração do cursor;

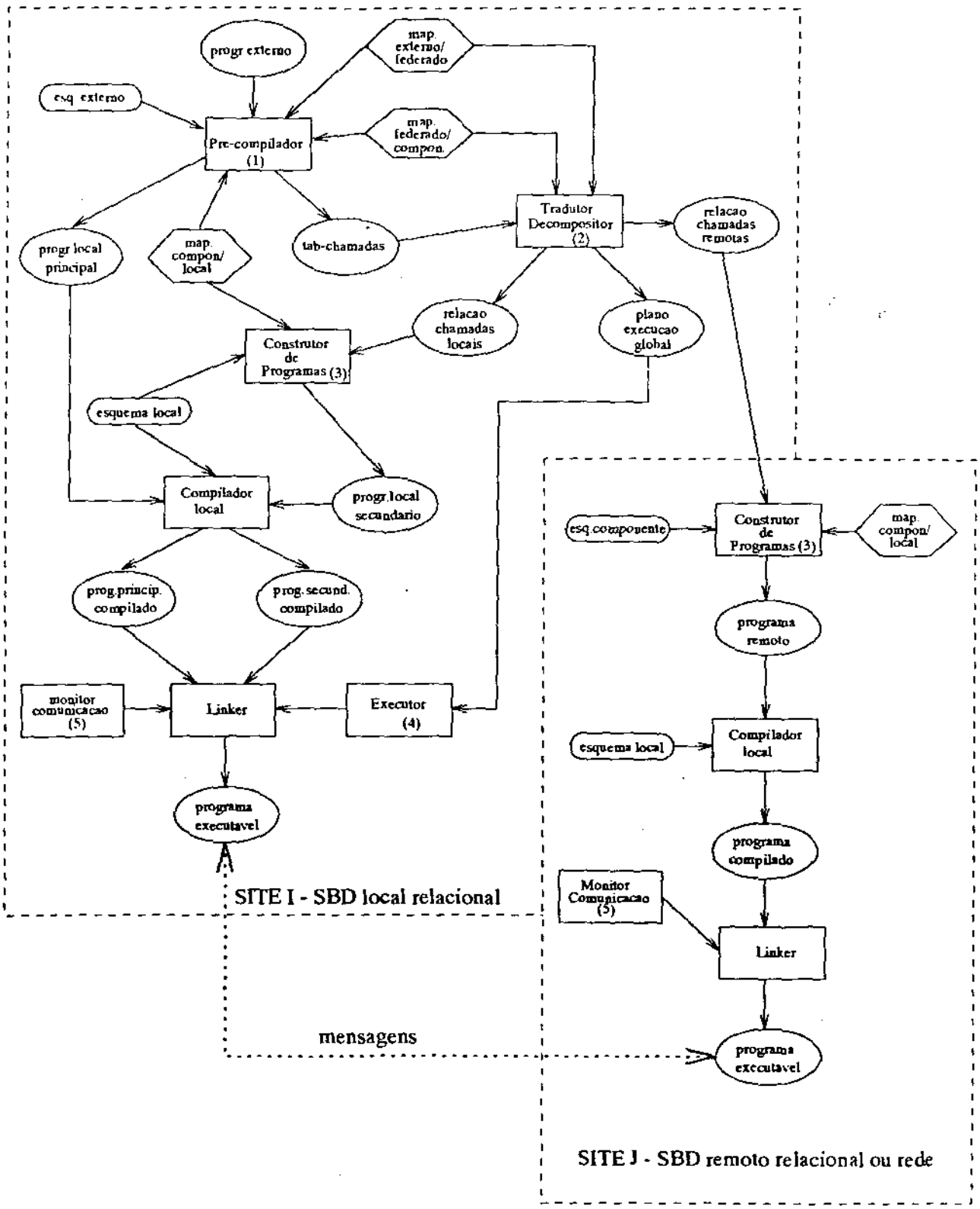

Figura 5.1: Arquitetura proposta

2. Substitui cada comando que abre o cursor no programa original (comando OPEN) por uma chamada ao módulo Executor solicitando o processamento do plano de execução global definido pelo módulo Tradutor-Decompositor para a expressão SFW original. A chamada contém o nome do programa, o número da chamada e a variável (SQLCODE) que retorna ao programa o resultado do processamento do plano de execução global;

3. Substitui cada comando SQL que recupera uma tupla da relação resultante do processamento da expressão SFW associada ao cursor (comando FETCH) por uma chamada ao módulo Executor solicitando os dados desejados. A chamada contém os mesmos dados constantes na chamada anterior mais os nomes das variáveis do programa pré-compilado que receberão os dados selecionados.

1. Substitui cada comando SQL que exclui a tupla associada à posição corrente do cursor (comando DELETE <*table-name*> WHERE CURRENT OF <*cursor-name*>) por uma chamada ao módulo Executor solicitando a operação desejada. Os dados contidos na chamada são os mesmos que foram incluídos na chamada do comando OPEN;

5. Substitui cada comando que modifica o conteúdo dos dados contidos na tupla associada à posição corrente do cursor (comando UPDATE ... WHERE CURRENT OF <*cursor-name*>) por uma chamada ao módulo Executor. Os dados contidos na chamada são os mesmos contidos na chamada gerada para o comando OPEN;

6. Substitui cada comando que fecha o cursor (comando CLOSE) no programa original por uma chamada ao módulo Executor. Os dados contidos na chamada são os mesmos usados na chamada gerada para o comando OPEN.

Para o caso (c) Pré-Compilador verifica se existe algum comando SQL que exclui ou modifica as tuplas correntes da relação resultante associada à expressão SFW contida na declaração do cursor. Se tais comandos existirem, o Pré-Compilador procede como no caso (b) e, em caso contrário, como no caso (a).

Depois de tratar os comandos SQL associados aos cursores, o Pré-Compilador examina os outros comandos SQL. Cada um desses comandos que referencia dados não existentes no SBD local é substituído por uma chamada ao módulo Executor. Os dados contidos nas chamadas são iguais aqueles usados para os comandos de mesma natureza associados aos cursores.

Para cada chamada introduzida no programa pré-compilado inclui-se em uma tabela de chamadas as seguintes informações: o nome do programa pré-compilado, o número da chamada, o nome do cursor associado à chamada, o tipo de operação solicitada pela chamada (tipo da chamada), o comando SQL definido na declaração do cursor e os nomes das

variáveis auxiliares do programa pré-compilado que recebem os valores dos dados obtidos nos SBDs componentes. Para os comandos SQL que não referenciam cursores são gerados nomes arbitrários de cursores. A concatenação do tipo de operação solicitada na chamada com o nome do cursor associado à mesma identificará a rotina que será executada nos programas construídos pelo módulo Construtor-de-Programas para manipular os dados dos SBDs componentes. Os tipos de operação que podem ser associados às chamadas são:

- OC (OPEN CURSOR);
- FC (FETCH CURSOR);
- DC (DELETE CURRENT OF CURSOR);
- MC (MODIFY CURRENT OF CURSOR);
- CC (CLOSE CURSOR);
- SEL (SELECT);
- DEL (DELETE);
- INS (INSERT);
- UPD (UPDATE).

A última tarefa do Pré-Compilador é incluir no programa gerado rotinas para verificação de possíveis restrições de integridade, que foram incorporadas ao esquema componente durante a fase de tradução do esquema local relacional para o modelo de dados comum.

5.2.2 Tradutor-Decompositor

O Tradutor-Decompositor recebe do Pré-Compilador a tabela de chamadas, examina o Mef e transforma cada comando SQL sobre o esquema externo em uma transação GORDAS equivalente usando os construtores do esquema federado. Em seguida, o Tradutor-Decompositor examina o Mfc para determinar os SBDs componentes que possuem os dados referenciados no comando SQL.

Uma vez identificados os SBDs componentes que possuem os dados, o módulo define qual a melhor forma para decompor a transação original em subtransações e qual a melhor estratégia para executar essa subtransações. O plano de execução global obtido define quais os SBDs componentes que serão utilizados, qual a seqüência de execução

das subtransações, como os resultados de uma subtransação são usados pela próximas subtransações e como são reunidos esses resultados em uma relação final que será usada pelo programa pré-compilado.

Existem trabalhos propondo soluções para a questão do processamento e otimização de transações em sistemas de bancos de dados distribuídos [CP84, YC84]. As soluções propostas utilizam como parâmetros informações relacionadas com o estado do sistema distribuído [HB91] como, por exemplo, a capacidade de processamento dos sistemas que formam o sistema distribuído e os custos de comunicação em cada trecho da rede de comunicação. Contudo, existem muitas outras variáveis que podem influir na escolha de estratégias de processamento distribuído o que torna o problema bastante complexo, particularmente no caso de SBDHs em que os SBDs componentes são autônomos e heterogêneos.

A discussão detalhada de soluções para otimização do processamento de transações em SBDHs está fora do escopo desta dissertação e, portanto, será assumido que algum algoritmo eficiente foi utilizado para a construção do plano de execução global. O leitor interessado no assunto pode tomar como referência soluções adotadas em alguns projetos em SBDHs [CBTY89, DL87, LR82].

Após processar todas as entradas da tabela de chamadas, o Tradutor-Decompositor envia a cada módulo Construtor-de-Programas, situado em um nó da rede que contém um SBD componente, uma relação das chamadas que referenciam esse SBD componente. A relação contém o identificador do SBD componente, o nome do programa pré-compilado em que as chamadas foram incluídas, o tipo da operação, o nome do cursor e a transação GORDAS associada a cada chamada.

5.2.3 Construtor-de-Programas

Este processador recebe a relação de chamadas gerada pelo Tradutor-Decompositor e gera o programa de aplicação que executa as operações correspondentes sobre o SBD local, usando a linguagem hospedeira e a LMD locais.

O programa de aplicação é construído de forma modular e compõe-se de um módulo de controle e n módulos de operações. O tipo desses módulos pode variar conforme a linguagem hospedeira utilizada.

Os módulos de operação são construídos a partir da transformação das operações GORDAS sobre o esquema componente em operações equivalentes utilizando o esquema e a LMD locais. O nome de cada módulo de operação é formado pela concatenação do tipo da chamada com o nome do cursor da chamada, já que mais de um cursor pode ser

associado ao mesmo tipo de chamada.

O módulo de controle recebe do Monitor-de-Comunicação ou do Executor local o tipo da chamada, o cursor associado à chamada e os parâmetros necessários para execução da operação requisitada pela chamada. Através do tipo de chamada e do nome do cursor este módulo determina o módulo de operação que deve ser ativado.

5.2.4 Monitor-de-Comunicação

Esse processador é na verdade um programa de propósito específico responsável pelo intercâmbio de mensagens entre os nós da rede de comunicação que contém os SBDs componentes.

No nó de origem, isto é, no nó em que foi construído o programa do usuário, esse programa é ativado pelo Executor que passa o nome do programa pré-compilado em que as chamadas foram incluídas, o número da chamada e os parâmetros que serão utilizados pelo SGBD componente referenciado na chamada para a execução da operação solicitada. O Monitor-de-Comunicação cria então uma mensagem que será enviada ao Monitor-de-Comunicação em outro nó da rede. Os dados contidos nessa mensagem são:

- o endereço do nó origem;
- o endereço do nó destino;
- o nome do programa local que solicitou uma operação sobre um SBD remoto;
- o tipo da chamada e o nome do cursor que identificam a rotina que será executada no programa remoto;
- o indicador do status da operação que será usado pelo programa local para verificar se a operação remota foi executada com sucesso ou não;
- os valores dos parâmetros passados pelo programa local relacional.

O endereço origem e destino são obtidos no mapa da rede de comunicação através dos identificadores do SBDs local e remoto. O tipo de chamada e o cursor são obtidos na entrada da tabela de chamadas identificada pelo nome do programa e número da chamada.

No nó destino, o Monitor-de-Comunicação recebe a mensagem do nó origem e chama o programa de aplicação previamente criado pelo módulo Construtor-de-Programas. Ao receber o resultado da operação requisitada o monitor envia uma mensagem de resposta ao monitor de comunicação do nó origem e passa a aguardar uma nova mensagem.

5.2.5 Executor

O Executor é o módulo que controla a execução das operações que envolvem dados contidos em SBDs componentes remotos. Esse módulo é chamado pelo Pré-Compilador e sua função é processar o plano de execução global definido pelo Tradutor-Decompositor.

Para processar uma subtransação que se refere a um determinado SBD componente remoto, o Executor ativa o Monitor-de-Comunicação local informando o número da chamada à qual a subtransação está vinculada, o nome do programa que originou a chamada, a identificação do SBD componente referenciado e os parâmetros usados para execução da operação solicitada e para retorno do resultados obtidos.[c]

Se a operação solicitada pela chamada é uma seleção de dados vinculada à abertura de um cursor (tipo da chamada = OC), então a relação resultante do processamento do plano de execução global é mantida pelo Executor para que outras operações associadas ao cursor possam manipular os dados nela contidos. Assim, uma chamada solicitando uma operação para recuperação de uma tupla da relação associada ao cursor (tipo de chamada = FC) faz com que o Executor simplesmente retorne ao programa local o conteúdo dos atributos da próxima tupla dessa relação. Da mesma forma, as chamadas solicitando operações de atualização de dados vinculados ao cursor (tipo de chamada = DC/MC) utilizam os valores dos atributos contidos na última tupla recuperada da relação como parâmetros para determinar os dados que serão objetos da operação.

A relação associada ao cursor é mantida pelo Executor até que uma chamada do programa local solicitando o fechamento do cursor (tipo da chamada = CC) seja recebida.

As chamadas que solicitam operações de seleção de dados não vinculadas a cursores (tipo da chamada = SEL) recebem tratamento semelhante aquele citado para a operação de abertura de cursor. A única diferença relevante é que os resultados da operação são retornados ao programa local imediatamente após o seu processamento pelo Executor. As chamadas que solicitam operações de atualização de dados não vinculadas a cursores (tipo da chamada = INS/DEL/UPD) também recebem tratamento semelhante.

5.3 Algoritmos para tradução SQL-GORDAS

A linguagem SQL permite uma variedade muito grande de construções sintáticas para manipulação de dados. Para não estender em demasia as discussões, serão tratados aqui apenas os comandos SQL mais freqüentemente usados pela comunidade de usuários. Contudo deve ficar registrado que a linguagem intermediária GORDAS é potencialmente capaz de suportar todos os comandos da LMD SQL porque possui um grande poder de expressão

e porque a sua sintaxe pode ser estendida facilmente sem prejuízo para o sistema.

A seguir são apresentados os algoritmos. Os exemplos da sua aplicação se referem às relações da figura 3.34.

5.3.1 Seleção de dados

Comando SQL:

SELECT <a-list> [INTO <v-list>]
FROM <R-list>
[WHERE <P_1> AND <P_2> ... AND <P_n>]

onde:

- <R-list> é a lista das relações contidas no esquema externo que estão sendo referenciadas no comando
- <a-list> é a lista dos atributos contidos nas relações de <R-list> que estão sendo selecionados
- <v-list> é a lista das variáveis do programa local que receberão os valores dos atributos de <a-list> no caso de seleção de uma única tupla
- <P_i>, $1 \leq i \leq n$, é um predicado de seleção usado para restringir as tuplas selecionadas. Cada predicado <P_i> pode assumir os seguintes formatos:
  1. $[R_i]$.a = $[R_j]$.b onde:
     .. $[R_i]$.a é um atributo da relação R_i, $1 \leq i \leq n$, contida em <R-list>
     .. $[R_j]$.b é um atributo da relação R_j, $1 \leq j \leq n$, contida em <R-list>
  2. $[R_k]$.c <op> <param> onde:
     .. $[R_k]$.c é um atributo da relação R_k, $1 \leq k \leq n$, contida em <R-list>
     .. <op> é um operador de comparação contido no conjunto {"=","<>","<",">","<=",">="}
     .. <param> é uma constante ou variável que será comparada com o valor de $[R_k]$.c

Algoritmo:

A: Identificar no Mef os construtores do esquema federado que correspondem aos construtores do esquema externo usados no comando SQL:

A1: Examinar os predicados de seleção do comando SQL e verificar no Mef se algum predicado <P_i> contém uma chave estrangeira que representa um relacionamento no esquema federado. Se a relação referenciada pela chave estrangeira não estiver contida na <R-list> do comando SQL, então obter no Mef o nome do construtor do esquema federado que corresponde à essa relação e incluí-lo na lista auxiliar <C-list>. Incluir na lista auxiliar de predicados (<P-list>) o(s) predicado(s), substituindo-se o atributo chave estrangeira pelo atributo identificador da relação referenciada.

A2: Obter no Mef o nome do construtor do esquema federado que corresponde a cada relação contida em <R-list>. Incluir os construtores obtidos em <C-list>.

B: Incluir em <P-list> todos os predicados contidos na cláusula WHERE do comando SQL, com exceção dos predicados de junção e dos predicados já incluídos no passo **A1**.

C: Determinar o construtor <Cs> contido em <C-list> que vai servir como referência na cláusula GET do comando GORDAS:

C1: Verificar quais os construtores contidos em <C-list> que aparecem no maior número de predicados de <P-list>.

C2: Definir um dos construtores obtidos no passo **C1** como <Cs>. Se não existir nenhum candidato, definir como <Cs> o construtor contido em <C-list> que tem o maior número de atributos em <a-list>.

D: Obter no esquema federado os caminhos de navegação (<N_i> *OF* <*Cs*>), definidos pelos nomes de conexão, que correspondem aos predicados de junção no comando SQL e que permitem alcançar a partir de <Cs> cada um dos outros construtores contidos em <C-list>.

E: Transformar o comando SQL no comando GORDAS

E1: Excluir cláusula FROM

E2: Obter no Mef o nome dos atributos dos construtores contidos em <C-list> que correspondem aos atributos contidos no comando SQL. Agrupar os nomes dos atributos contidos em <a-list> em sub-listas $<a_{C_j}\text{-list}>$, cada uma contendo os atributos vinculados a um único construtor $<C_j>$ contido em <C-list>.

E3: Substituir cada atributo <a> contido em <P-list> por um caminho $<a>$ *OF* $<N_i>$ *OF* $<C_s>$

E4: Substituir a cláusula SELECT pela seguinte cláusula:

GET ([$<a_{C_s}$-list>],
[$<a_{C_i}$-list> OF $<N_i>$]) OF $<C_s>$

E5: Substituir a cláusula WHERE pela cláusula:

WHERE <P-list>

Exemplo 1:

```
SELECT matricula,nome,est-civil,salario-atual
FROM Empregado
WHERE depto = "DEP1"
```

Passo A1 - Verifica-se no Mef que o atributo *depto* da relação *Empregado* é uma chave estrangeira usada para representar o relacionamento *Dep-Emp* do esquema federado, que relaciona o tipo de entidade *Departamento* com o tipo de entidade *Empregado*. Verifica-se também que o tipo de entidade *Departamento* não está contido em <R-list>. Inclui-se em <C-list> o tipo de entidade *Departamento* que corresponde à relação com mesmo nome referenciada pela chave estrangeira. Inclui-se o predicado *cod-depto = "DEP1"* em <P-list>.

Passo A2 - Obtém-se no Mef o tipo de entidade *Empregado* do esquema federado que corresponde à relação de mesmo nome no esquema externo. Inclui-se *Empregado* em <C-list>.

Passo C1 - Verifica-se que *Departamento* é o único construtor contido em <C-list> que aparece também em <P-list>.

Passo C2 - Define-se *Departamento* como $<C_S>$.

Passo D - Obtém-se no esquema federado o caminho *empregados OF Departamento* que permite alcançar o tipo de entidade *Empregado* a partir do tipo de entidade *Departamento*.

Passo E2 - Obtém-se a lista:
$<a_{Empregado}$-list> = (matricula,nome-empreg,est-civil,salario-atual).

Passo E3 - Substitui-se em <P-list> o atributo *cod-depto* pelo caminho *cod-depto OF Departamento*.

Passo E1 e E5 - Obtém-se o comando GORDAS:

```
GET <matricula,nome-empreg,est-civil,salario-atual> OF
                                        empregados OF Departamento
WHERE cod-depto OF Departamento = ''DEP1''
```

Exemplo 2:

```
SELECT nome,salario-atual,est-civil,nome-cargo,
       nome-projeto,horas-trab
FROM Empregado,Particip-Projeto,Cargo,Projeto,Cliente
WHERE Empregado.depto = "DEP1"
AND Empregado.matricula = Particip-projeto.matric
AND Empregado.cargo = Cargo.cod-cargo
AND Projeto.cod-projeto = Particip-projeto.cod-projeto
AND Projeto.cod-cliente = Cliente.cod-cliente
AND Cliente.nome-cliente = "Telebras"
```

Passo A1 - Verifica-se no Mef que existem na relação de predicados de seleção do comando SQL as chaves estrangeiras *Empregado.depto*, *Empregado.cargo* e *Projeto.cod-cliente*. Verifica-se também que a relação *Departamento* referenciada pela primeira chave estrangeira não está contida em <R-list>. Inclui-se em <C-list> o tipo de entidade *Departamento* do esquema federado que corresponde à relação *Departamento*. Inclui-se em <P-list> o predicado *Departamento.cod-depto = "DEP1"*.

Passo A2 - Obtém-se no Mef os tipos de entidade *Empregado*, *Cargo*, *Projeto*, *Cliente* e o relacionamento *Particip-Projeto* do esquema federado que correspondem às relações de mesmo nome no esquema externo. Inclui-se os construtores obtidos em <C-list>.

Passo B - Inclui-se em <P-list> o predicado *Cliente.nome-cliente = "Telebras"*.

Passo C1 - Verifica-se que *Departamento* e *Cliente* aparecem o mesmo número de vezes nos predicados de seleção contidos em <P-list>.

Passo C2 - Define-se *Departamento* como <Cs>.

Passo D - Obtém-se no esquema federado os seguintes caminhos:
Empregado: *empregados OF Departamento*
Cargo: *cargo OF empregados OF Departamento*
Projeto: *projetos OF empregados OF Departamento*
Particip-Projeto: *projetos OF empregados OF Departamento*
Cliente: *cliente of projetos OF empregados OF Departamento*

Passo E2 - Obtém-se as listas:.
<$a_{Empregado}$-list> = (nome-empreg,est-civil,salario-atual)
<a_{Cargo}-list> = (nome-cargo)
<$a_{Projeto}$-list> = (nome-projeto)
<$a_{Particip-Projeto}$-list> = (horas-trab)

Passo E3 - Substitui-se em <P-list> os atributos *Departamento.cod-depto* e *Cliente.nome-cliente* pelos caminhos *cod-depto OF Departamento* e *nome-cliente OF cliente OF projetos OF Empregados OF Departamento.*

Passo E4 e E5 - Obtém-se o comando GORDAS:

```
GET (<nome-empreg,est-civil,salario-atual> OF empregados
     <nome-cargo> OF cargo OF empregados,
     <nome-projeto,horas-trab> OF projetos OF empregados)
                                              OF Departamento
WHERE cod-depto OF Departamento = ''DEP1''
AND   nome-cliente OF cliente OF projetos OF empregados
                                OF Departamento = ''Telebras''
```

5.3.2 Exclusão de dados

Comando SQL:

DELETE FROM <R>
[WHERE <P_1> AND <P_2> ... AND <P_n>]

onde:

. <R> é a relação que está sendo objeto da operação de exclusão

. <P_i>, $1 \leq i \leq n$, é um predicado de seleção usado para restringir as tuplas que serão excluídas.

Algoritmo:

A: Identificar no Mef qual o construtor <C> do esquema federado que é representado pela relação <R> do esquema externo

Se <C> é um tipo de entidade
Então

B: Verificar se existe alguma chave estrangeira nos predicados de seleção representando um relacionamento binário em que <C> participa. Em caso afirmativo obter no Mef o tipo de entidade <E> que se relaciona com <C> através do relacionamento representado pela chave estrangeira. Obter também o nome de conexão <N> que permite alcançar <E> a partir de <C>. Substituir no predicado de seleção a chave estrangeira pelo caminho $<a_k>$ *OF* <N> *OF* <C>, onde $<a_k>$ é o atributo identificador de <E>. Incluir o predicado de seleção modificado em <P-list>.

C: Substituir cada atributo de <R> contido nos predicados de seleção pelo atributo correspondente que está contido em <C>. Incluir os predicados modificados em <P-list>

D: Gerar o seguinte comando GORDAS:
DELETE FROM <C>
WHERE <P-list>

Senão
Se <C> é um relacionamento
Então

E: Identificar no Mef as relações $<R_1>,\ldots,<R_n>$ que são associadas pela relação <R> e os tipos de entidade correspondentes no esquema federado ($<E_1>,\ldots,<E_n>$).

F: Substituir cada predicado $<P_i>$ que contém um atributo que referencia o atributo identificador de alguma relação $<R_j>$, $1 \leq j \leq n$, por um predicado da forma $<E_j> : <P_i>$. Incluir esses predicados em <P-list>. Incluir os demais predicados em <W-list>.

G: Substituir os atributos contidos em <P-list> e <W-list> pelos nomes dos atributos correspondentes no esquema federado.

H: Gerar o seguinte comando GORDAS:

REMOVE FROM RELATIONSHIP <C'>
[WHERE <P-list> AND <W-list>]

Senão

I: Obter no Mef o nome do construtor <E> do esquema federado que contém o atributo multivalorado <C'>

J: Substituir os atributos contidos nos predicados de seleção do comando SQL pelos atributos correspondentes de <E>. Incluir cada predicado modificado em <P-list>.

K: Gerar o seguinte comando GORDAS:

MODIFY <C'> OF <E> REMOVE VALUE-LIST
[WHERE <P-list>]

Exemplo 1:

DELETE FROM Empregado
WHERE depto = "DEP1"
AND salario-atual > 10000000

Passo A - Verifica-se no Mef que a relação *Empregado* corresponde ao tipo de entidade *Empregado* no esquema federado.

Passo B - Verifica-se que o atributo *depto* é uma chave estrangeira que representa o relacionamento *Dep-Emp* entre o tipo de entidade *Empregado* e o tipo de entidade *Departamento*. Substitui-se no predicado de seleção o atributo *depto* pelo caminho *cod-depto OF departamento OF Empregado*. Inclui-se o predicado modificado em <P-list>.

Passo C - Inclui-se o predicado *salario-atual* > *10000000* em <P-list>.

Passo D - Cria-se o comando GORDAS:

```
DELETE FROM Empregado
WHERE cod-depto OF Departamento = ''DEP1''
AND  salario-atual OF EMPREGADO > 10000000
```

Exemplo 2:

DELETE FROM Particip-Projeto
WHERE matric = 3515
AND horas-trab < 5

Passo A - Verifica-se no Mef que a relação *Particip-Projeto* corresponde ao relacionamento *Particip-Projeto* no esquema federado.

Passo E - Identificam-se os tipos de entidade *Empregado* e *Projeto* do esquema federado que correspondem às relações de mesmo nome no esquema externo, associadas pela relação *Particip-Projeto*.

Passo F - O predicado *matric = 3515* é substituído pelo predicado *Empregado: matric = 3515* que é incluído em <P-list>. O predicado *horas-trab < 5* é incluído em <W-list>.

Passo G - Substitui-se os nomes dos atributos contidos em <P-list> e <W-list> pelos nomes dos atributos correspondentes no esquema federado.

Passo D - Cria-se o comando GORDAS:

```
REMOVE FROM RELATIONSHIP Particip-Projeto
WHERE Empregado: matric = 3515
AND  horas-trab < 5
```

5.3.3 Inclusão de dados

Comando SQL:

INSERT INTO <R>
(<a-list>) VALUES (<val-list>)

onde:

. <R> é a relação que está sendo objeto da operação de inclusão

. <a-list> contém a lista de atributos de <R> que estão recebendo valores

. <val-list> contém os valores atribuídos aos elementos de <a-list>

Algoritmo:

A: Identificar no MeI qual o construtor <C> do esquema federado que é representado pela relação <R> do esquema externo

Se <C> é um tipo de entidade
Então

B: Associar os elementos correspondentes de <a-list> e <val list>, de acordo com a suas posições na lista, obtendo os predicados de atribuição <A_i>, $1 \leq i \leq n$, da forma <a_i> = <val_i>, onde <a_i> $\in$ <a-list> e <val_i> $\in$ <val-list>. Incluir cada predicado <A_i> na lista <A-list>.

C: Retirar de <A-list> os predicados de atribuição que se referem às chaves estrangeiras de <R> e incluí-los em <Rel-list>.

D: Substituir os atributos contidos em <A-list> pelos atributos correspondentes no esquema federado

E: Incluir o seguinte comando na transação GORDAS:

INSERT INTO <C>
ATTRIBUTES <A-list>

F: Para cada predicado de atribuição <Pa> contido em <Rel-list> verificar o tipo de valor atribuído à chave estrangeira

Se o valor atribuído é um valor não nulo ($\neq$ NULL)
Então

G: Obter no MeI o nome do relacionamento binário <Rb> que é representado pela chave estrangeira e o nome do tipo de entidade <E> que é associado à <C> através desse relacionamento.

H: Substituir a chave estrangeira em <Pa> pelo atributo identificador de <E>

I: Obter em <A-list> o predicado <Pk> que atribui um valor ao atributo chave de <C>.

J: Incluir o seguinte comando na transação GORDAS:

ADD TO RELATIONSHIP <Rb>
WHERE <C> : <Pk>
<E> : <Pa>

Senão
Continuar a examinar <Rel-list>

Senão
Se <C> é um relacionamento
Então

K: Idem B

L: Retirar de <A-list> os predicados de atribuição que se referem aos atributos que compõem a chave da relação <R> e incluí-los em <Rel-list>

M: Idem D

N: Identificar no Mef os tipos de entidade $<E_1>,\ldots,<E_n>$ que correspondem às relações associadas pela relação <R> do comando SQL

O: Substituir em <Rel-list> os atributos da relação <R> pelos atributos identificadores dos tipos de entidade obtidas no passo **N**

P: Gerar o seguinte comando GORDAS:

ADD TO RELATIONSHIP <C>
WHERE $<E_1>$: $<Pk_1>$
$<E_2>$: $<Pk_2>$
⋮
$<E_n>$: $<Pk_n>$
ATTRIBUTES <A-list>

onde:

- $<Pk_i>$, $1 \leq i \leq n$, é o predicado de atribuição contido em <Rel-list> que referencia o atributo identificador de E_i

Senão

Q: Obter no Mef o nome do construtor <E> do esquema federado que contém o atributo multivalorado <C>

R: Identificar em <a-list> o atributo de <R> que corresponde ao atributo identificador de <E> e associá-lo ao valor correspondente em <val-list> obtendo o predicado <Pk>. Retirar de <val-list> o valor incluído em <Pk>.

S: Incluir o seguinte comando GORDAS:

```
MODIFY <C> OF <E>
ADD VALUE-LIST <val-list>
[WHERE <Pk>]
```

Exemplo 1:

```
INSERT INTO Projeto
        (cod-projeto,nome-projeto,...,multa-atraso,cod-cliente)
        VALUES
        ("PROJY","Projeto OMEGA",...,1000000000,10)
```

Passo A - Verifica-se no Mef que a relação *Projeto* corresponde ao tipo de entidade *Projeto* no esquema federado.

Passo B - Incluem-se os seguintes predicados em <A-list>:

cod-projeto = "PROJY"
nome-projeto = "Projeto OMEGA",
⋮
multa-atraso = 1000000000,
cod-cliente = 10

Passo C - Retira-se de <A-list> o predicado *cod-cliente = 10.* Inclui-se o predicado em questão em <Rel-list>

Passo E - Inclui-se na transação GORDAS o seguinte comando:

```
INSERT INTO Projeto
ATTRIBUTES cod-projeto = ''PROJY'',
           nome-projeto = ''Projeto OMEGA'',
           multa-atraso = 1000000000
```

Passo G - Obtém-se o relacionamento *Cliente-Proj* do esquema federado representado pela chave estrangeira na relação *Projeto* e o tipo de entidade *Cliente* referenciado por essa chave estrangeira

Passo I - Obtém se o predicado *cod-projeto = "PROJY"*

Passo J - Inclui-se na transação GORDAS o seguinte comando:

```
ADD TO RELATIONSHIP Cliente-Proj
WHERE  Projeto : cod-projeto = ''PROJY''
       Cliente : cod-cliente = 10
```

Exemplo 2:

```
INSERT INTO Particip-Projeto
       (matric,cod-projeto,horas-trab)
       VALUES
       (3515,"PROJX",10)
```

Passo A - Verifica-se no Mef que a relação *Particip-Projeto* corresponde ao relacionamento *Particip-Projeto* no esquema federado.

Passo K - Incluem-se os seguintes predicados em <A-list>:
matric = 3515
cod-projeto = "PROJX",
horas-trab = 10

Passo L - Retiram-se de <A-list> os predicado *matric = 3515* e *cod-projeto = "PROJX"* e incluem-nos em <Rel-list>

Passo N - Identificam-se os tipos de entidade *Empregado* e *Projeto* do esquema federado que correspondem às relações de mesmo nome no esquema externo associadas pela relação *Particip-Projeto*.

Passo O - Substitui-se em <Rel-list> o predicado *matric = 3515* pelo predicado *matricula = 3515*

Passo P - Gera-se o comando GORDAS:

```
ADD TO RELATIONSHIP Particip-Projeto WHERE
       Empregado: matricula = 3515
       Projeto  : cod-projeto = ''PROJX''
ATTRIBUTES horas-trab = 10
```

5.3.4 Alteração de dados

Comando SQL:

UPDATE <R>
SET <S_1> [,<S_2>, ..., <S_n>]

[WHERE <P_1> AND <P_2> ... AND <P_n>]

onde: [6]

- <S_i> é um predicado de atribuição da forma <*a*> = <*param*>, onde <a> é um atributo de <R> e <*param*> é um valor que está sendo atribuído ao atributo.

Algoritmo:

A: Identificar no Mef qual o construtor <C> do esquema federado que é representado pela relação <R> do esquema externo

Se <C> é um tipo de entidade
Então

B: Verificar quais os atributos de <R> que estão sendo tratados nos predicados de atribuição do comando SQL

Se nenhuma chave estrangeira de <R> aparece nos predicados de atribuição

Então

C: Incluir em <A-list> os atributos de <R> que aparecem nos predicados de atribuição e substituí-los pelos atributos correspondentes de <C>.

D: Para cada predicado de seleção <P_i> que contém uma chave estrangeira obter no Mef o nome do tipo de entidade que corresponde à relação referenciada pela chave estrangeira. Substituir a chave estrangeira pelo atributo identificador do tipo de entidade obtida concatenado com o caminho definido pelo nome de conexão que permite referenciar o tipo de entidade a partir de <C>, usando o relacionamento representado pela chave estrangeira. Incluir os predicados de seleção examinados em <P-list>.

E: Para cada predicado de seleção não examinado no passo anterior substituir o atributo de <R> pelo atributo correspondente de <C>. Incluir os predicados examinados em <P-list>.

F: Incluir em <val-list> os valores que aparecem nos predicados de atribuição

G: Gerar o seguinte comando GORDAS:

MODIFY <A-list> OF <C> TO <val-list>
[WHERE <P-list>]

Senão

Se somente chaves estrangeiras de <R> aparecem nos predicados de atribuição e pelo menos um atributo de <R> que não seja uma chave estrangeira aparece nos predicados de seleção

Então

H: Idem **C**

I: Idem **D**

J: Para cada predicado de atribuição $<S_i>$, $1 \leq i \leq n$, que contém uma chave estrangeira faça:

J1: Obter no Mef o relacionamento $<R_i>$ representado pela chave estrangeira e o nome do tipo de entidade $<E_i>$ que corresponde à relação referenciada pela chave estrangeira

J2: Substituir no predicado examinado a chave estrangeira pelo atributo identificador do tipo de entidade $<E_i>$

J3: Se o valor atribuído em $<S_i>$ é o valor nulo, então incluir o seguinte comando na transação GORDAS:

REMOVE FROM RELATIONSHIP $<R_i>$
WHERE <C> : <P-list>

J4: Se o valor atribuído em <S_i> não é o valor nulo, então incluir os seguintes comandos na transação GORDAS:

```
REMOVE FROM RELATIONSHIP <R_i>
        WHERE <C> : <P-list>

ADD     TO RELATIONSHIP <R_i>
        WHERE <C> : <P-list>
        AND   <E_a> : <S_i>
```

Senão

K: Obter no Mef o atributo <c_k> que identifica o tipo de entidade <C>

L: Idem **H**

M: Idem **I**

N: Incluir em <A-list> os atributos de <R> que não são chaves estrangeiras e que aparecem nos predicados de atribuição. Substituir em <A-list> os atributos de <R> pelos atributos correspondentes de <C>.

O: Idem **F**

P: Incluir na transação GORDAS os seguintes comandos:

```
param := GET <c_k> OF <C>
         WHERE <P-list>
MODIFY <A-list> OF <C>
         TO <val-list>
WHERE <c_k> = param
```

Q: Idem **J**

Q1: Idem **J1**

Q2: Idem **J2**

Q3: Se o valor atribuído em <S_i> é o valor nulo, então incluir o seguinte comando na transação GORDAS:

REMOVE FROM RELATIONSHIP <R_i>
WHERE <C> : <ck> = param

Q4: Se o valor atribuído em <S_i> não é o valor nulo, então incluir os seguintes comandos na transação GORDAS:

REMOVE FROM RELATIONSHIP <R_i>
WHERE <C> : <ck> = param

ADD TO RELATIONSHIP <R_i>
WHERE <C> : <ck> = param
AND <E_n> : <S_i>

Senão

Se <C> é um relacionamento

Então

R: Incluir em <A-list> os nomes dos atributos de <R> contidos nos predicados de seleção que não compõem a chave primária. Substituir os atributos contidos em <A-list> pelos atributos correspondentes de <C>

S: Obter no Mef os tipos de entidade <E_1>,<E_2>,...,<E_n> que são associados pelo relacionamento <C>. Substituir nos predicados de seleção (<P_i>) e atribuição (<S_i>) os atributos que compõem a chave de <R> pelos atributos identificadores dos tipos de entidade referenciados por esses atributos

T: Agrupar os predicados de seleção de acordo com o construtor do esquema federado referenciado por eles. Obter as listas: <E_1.S-list>,<E_2.S-list>,...,<E_n.S-list>,<E_1.P-list>,<E_2.P-list>,..., <E_n.P-list>.

U: Se existe alguma <E_i.S-list> não vazia, incluir na transação GORDAS os seguintes comandos

REMOVE FROM RELATIONSHIP <C>
WHERE <E_1> : <E_1.P-list>
⋮
<E_n> : <E_n.P-list>

ADD TO RELATIONSHIP <C>
WHERE <E_1> : <E_1.S-list>
⋮
<E_n> : <E_n.S-list>

V: Se <A-list> é não vazia, incluir na transação GORDAS os seguintes comandos:

MODIFY <A-list> OF <C> TO <val-list>
WHERE <E_1> : <E_1.S-list>|<E_1.P-list>
⋮
<E_n> : <E_n.P-list>|<E_1.S-list>

Senão

X: Obter no Mef o nome do construtor <E> no esquema federado que contém o atributo multivalorado <C>

W: Substituir nos predicados de seleção, o atributo da relação <R> que referencia o construtor <E> pelo atributo identificador de <E>

Y: Se o valor atribuído a <C> é o valor nulo, então gerar o seguinte comando GORDAS:

MODIFY <C> OF <E>
REMOVE VALUE-LIST
[WHERE <P_1> AND <P_2> ... AND <P_n>]

Z: Se o valor atribuído a <C> não é o valor nulo, então gerar o seguinte comando GORDAS:

MODIFY <C> OF <E>
TO VALUE-LIST <valor>
[WHERE <P_1> AND <P_2> ... AND <P_n>]

onde:

. <valor> é o valor atribuído a <C> no comando SQL

Exemplo 1:

```
UPDATE Empregado
        SET nome = "Maria da Silva"
            est-civil = "c"
        WHERE matricula = 100
```

Passo A - Verifica-se no Mef que a relação *Empregado* corresponde ao tipo de entidade *Empregado* no esquema federado.

Passo B - Verifica-se que os atributos de <R> tratados nos predicados de atribuição não são chaves estrangeiras

Passo C - Obtém-se <A-list> = (nome-empreg,est-civil)

Passo E - Obtém-se <P-list> = (matricula = 100)

Passo F - Obtém-se <val-list> = ("Maria da Silva, "c")

Passo G - Cria-se o seguinte comando GORDAS:

```
MODIFY <nome-empreg,est-civil> OF Empregado TO
       (''Maria da Silva'', ''c'')
WHERE matricula = 100
```

Exemplo 2:

```
UPDATE Empregado
        SET cod-cargo = "C5"
            cod-depto = "DEP3".
        WHERE salario-atual < 10000000
        AND     cod-depto = "DEP1"
```

Passo A - Verifica-se no Mef que a relação *Empregado* corresponde ao tipo de entidade *Empregado* no esquema federado.

Passo B - Verifica-se que todo os atributos de <R> que aparecem nos predicados de atribuição são chaves estrangeiras e que o atributo *salario-atual* presente nos predicados de seleção não é uma chave estrangeira.

Passo I - Obtém-se <P-list> = (cod-depto OF departamento OF empregado = "DEP1", salario atual < 10000000)

Passo J - Iteração 1 - Examina-se o predicado *cod-cargo = "C5"*

Passo J1 - Obtém-se o relacionamento *Cargo-Emp* representado pela chave estrangeira e o tipo de entidade *Cargo* que corresponde à relação referenciada

Passo J1 - Incluem-se na transação GORDAS os seguintes comandos:

```
REMOVE FROM RELATIONSHIP Cargo-Emp WHERE
       Empregado : cod-depto OF departamento = ''DEP1'',
                   salario-atual < 10000000

ADD    TO   RELATIONSHIP Cargo-Emp WHERE
       Empregado : cod-depto OF departamento = ''DEP1'',
                   salario-atual < 10000000
       Cargo     : cod-cargo = ''C5''
```

Passo J - Iteração 2 - Examina-se o predicado *cod-depto = "DEP3"*

Passo J1 - Obtém-se o relacionamento *Dep-Emp* representado pela chave estrangeira e o tipo de entidade *Departamento* que corresponde à relação referenciada

Passo J1 - Incluem-se na transação GORDAS os seguintes comandos

```
REMOVE FROM RELATIONSHIP Dep-Emp WHERE
       Empregado : cod-depto OF departamento = ''DEP1'',
                   salario-atual < 10000000

ADD    TO   RELATIONSHIP Dep-Emp WHERE
       Empregado : cod-depto OF departamento = ''DEP1'',
                   salario-atual < 10000000
       Departamento : cod-depto = ''DEP3''
```

Exemplo 3:

```
UPDATE Empregado
       SET cod-cargo = "C5"
           cod-depto = "DEP3"
           salario-atual = 5000000
```

WHERE salario-atual < 4000000
AND cod-depto = "DEP1"

Passo A - Idem exemplo anterior

Passo B - Verifica-se que nos predicados de atribuição aparecem atributos de <R> que são chaves estrangeiras (*cod-depto* e *cod-cargo*) e o atributo *salario-atual* que não é uma chave estrangeira.

Passo K - Obtém-se o atributo *matricula* que identifica univocamente as entidades do tipo de entidade *Empregado*.

Passo L - Idem passo H do exemplo anterior

Passo M - Idem passo I do exemplo anterior

Passo N - Obtém-se <A-list> = (salario-atual)

Passo O - Obtém-se <val-list> = (5000000)

Passo P - Incluem-se na transação GORDAS os seguintes comandos:

```
param := GET matricula OF Empregado
         WHERE cod-depto OF departamento OF Empregado = ''DEP1''
         AND   salario-atual < 4000000

MODIFY salario-atual OF Empregado TO 5000000
       WHERE matricula = param
```

Passo Q - Iteração 1 - Examina-se o predicado *cod-cargo* = "C5"

Passo Q1 - Obtém-se o relacionamento *Cargo-Emp* representado pela chave estrangeira e o tipo de entidade *Cargo* que corresponde à relação referenciada

Passo Q1 - Incluem-se na transação GORDAS os seguintes comandos

```
REMOVE FROM RELATIONSHIP Cargo-Emp WHERE
       Empregado : matricula = param

ADD    TO   RELATIONSHIP Cargo-Emp WHERE
       Empregado : matricula = param
       Cargo     : cod-cargo = ''C5''
```

Passo Q - Iteração 2 - Examina-se o predicado *cod-depto = "DEP3"*

Passo Q1 - Obtém-se o relacionamento *Dep-Emp* representado pela chave estrangeira e o tipo de entidade *Departamento* que corresponde à relação referenciada

Passo Q1 - Incluem-se na transação GORDAS os seguintes comandos

```
REMOVE FROM RELATIONSHIP Dep-Emp WHERE
       Empregado : matricula = param

ADD    TO   RELATIONSHIP Dep-Emp WHERE
       Empregado : matricula = param
       Departamento : cod-depto = ''DEP3''
```

Exemplo 1:

```
UPDATE Particip-Projeto
          SET horas-trab = 10
              cod-projeto = "PROJ-K"
WHERE matricula = 3838
AND     cod-projeto = "PROJ-X"
```

Passo A - Verifica-se que a relação *Particip-Projeto* corresponde ao relacionamento de mesmo nome no esquema federado

Passo R - Obtém-se <A-list> = (horas-trab)

Passo S - Obtém-se os tipos de entidade *Empregado* e *Projeto* que são associados pelo relacionamento *Particip-Projeto*

Passo T - Obtém-se as listas
Projeto.S-list = (cod-projeto = "PROJ-K")
Empregado.P-list = (matricula = 3838)
Projeto.P-list = (cod-projeto = "PROJ-X")

Passo U - Inclui-se na transação GORDAS os seguintes comandos:

```
REMOVE FROM RELATIONSHIP Particip-Projeto WHERE
       Empregado : matricula = 3838
       Projeto   : cod-projeto = ''PROJ-X''
```

```
ADD    TO   RELATIONSHIP Particip-Projeto WHERE
       Empregado : matricula = 3838
       Projeto   : cod-projeto = ''PROJ-K''
```

Passo V - Inclui-se na transação GORDAS os seguintes comandos:

```
MODIFY horas-trab OF Particip-Projeto WHERE
       Empregado : matricula = 3838
       Projeto   : cod-projeto = ''PROJ-K''
```

5.3.5 Comandos SQL vinculados aos cursores

Existem dois comandos SQL de atualização de dados que se referem aos cursores declarados no programa de aplicação dos usuários. Esses comandos são:

DELETE FROM <R>
WHERE CURRENT OF <cursor>

e

UPDATE <R>
SET <S_1> [,<S_2>,...,<S_n>]
WHERE CURRENT OF <cursor>

Tais comandos dependem do resultado da execução de comandos de seleção de dados associados à declaração dos cursores e portanto devem ser substituídos pelos comandos:

DELETE FROM <R>
WHERE <R.key> = <param>

e

UPDATE <R>
SET <S_1> [,<S_2>,...,<S_n>]
WHERE <R.key> = <param>

onde:

. <R.key> é o atributo identificador da relação <R>

. <param> é o valor de <R.key> da última tupla da relação associada ao cursor, que foi recuperada por uma chamada do tipo FC (FETCH CURSOR)

Os comandos modificados são transformados em transações GORDAS utilizando os algoritmos apresentados anteriormente. É importante observar que se o comando de seleção de dados, associado à declaração do cursor, não incluir nos atributos a selecionar o atributo identificador da relação, então o módulo Tradutor-Decompositor deve incluí-lo no comando SQL antes de se aplicar o algoritmo da seção 5.3.1.

5.4 Algoritmos para tradução GORDAS-REDE

Nesta seção serão apresentados os algoritmos usados pelo Construtor-de-Programas na transformação das transações GORDAS, recebidas do módulo Tradutor-Decompositor, em rotinas que serão incluídas no programa construído para operar sobre um SBD componente que usa o modelo rede.

Os algoritmos para tradução GORDAS-REDE utilizam os mapeamentos entre o esquema componente, sobre o qual a transação GORDAS é construída, e o esquema local do SBD componente rede. Esses mapeamentos, obtidos durante a construção do SBDH, contêm, além das correspondências entre os construtores, informações referentes à organização física dos dados no SBD rede que são usadas para se determinar o tipo de comando da LMD rede que pode ser aplicado na execução de uma operação específica.

Em termos gerais, o primeiro passo executado pelos algoritmos é a identificação dos caminhos de navegação no esquema local (definidos pelos *record-type* e *set-type*) que correspondem aos caminhos estabelecidos pelos nomes de conexão existentes nas transações GORDAS. Para a representação e armazenamento desses caminhos pode-se utilizar grafos orientados em que os vértices e arestas representam, respectivamente, os *record-type* e os *set-type* do esquema local envolvidos na operação que se deseja efetuar. O próximo passo é percorrer esse grafo, definindo para cada vértice ou aresta visitada um conjunto de comandos da LMD rede que implemente a navegação efetuada.

A natureza procedimental da LMD rede exige que a varredura de registros contidos em uma *set-occurrence* seja feita com o auxílio de estruturas de repetição (loops) construídas através de comandos da própria linguagem hospedeira. Portanto, os algoritmos para construção de programas rede a partir de transações GORDAS dependem da linguagem hospedeira utilizada. Os algoritmos que serão apresentados a seguir se referem à linguagem COBOL, por se tratar da linguagem hospedeira mais utilizada pelos usuários dos SBDs rede.

5.4.1 Transação GORDAS para seleção de dados

Transação GORDAS:

GET <A_1-list> [,<A_2-list>, ..., <A_n-list>]
[WHERE <P_1> AND <P_2> ... AND <P_n>]

onde:

. <A_i-list>, $1 \leq i \leq n$, tem a forma:
<a_1,a_2,...,a_n> [OF <N>] OF <C>, onde:

.. <C> é o construtor do esquema federado que serve como ponto de partida para a navegação

.. <N> é o caminho definido pelos nomes de conexão do esquema federado que permite referenciar a partir de <C> o construtor do esquema federado que possui os atributos <a_1,a_2,...,a_n>

. <P_i-list>, $1 \leq i \leq n$, tem a forma:
<a> [OF <N>] OF <C> <op> <param>, onde:

.. <a> é o atributo do construtor alcançado pelo caminho [OF <N>] OF <C>

.. <op> é um operador de comparação contido no conjunto {"=","<>","<",">","<=",">="}

.. <param> é uma constante ou variável que será comparada com o valor de <a>

Algoritmo:

Entradas:

. Ent-1: tipo de operação da chamada à qual a transação se refere
. Ent-2: cursor ao qual a chamada se refere
. Ent-3: mapeamento entre o esquema componente e o esquema local (Mcl)
. Ent-4: Transação GORDAS

Estruturas Auxiliares

. T1: tabela que armazena os predicados de seleção contidos na cláusula WHERE da transação GORDAS. Cada entrada da tabela contém os seguintes dados:

.. RT1: *record-type* a que se refere o predicado de seleção
.. NT1: caminho de navegação no esquema componente representado pelos nomes de conexão
.. ST1: caminho de navegação no esquema local representado pelos *set-type*
.. PT1: lista dos predicados de seleção que se referem ao *record-type* contido em RT1. Cada predicado da lista é dividido em três campos: atributo (a-pred), operador de comparação (o-pred) e parâmetro de comparação (p-pred)

. T2: tabela que armazena a lista de atributos contidos na cláusula GET da transação GORDAS. Cada entrada da tabela contém os seguintes dados:

.. RT2: *record-type* que contém os atributos
.. NT2: caminho de navegação no esquema componente representado pelos nomes de conexão usados para alcançar o *record-type* contido em RT2
.. ST2: caminho de navegação no esquema local representado pelos *set-type*
.. AT2: lista dos atributos contidos em RT2.

. G : Grafo orientado (G=V,A) usado para representar os caminhos de navegação no esquema local contidos nas tabelas T1 e T2. Cada vértice $v \in V$ é rotulado com o nome do *record-type* (RT) e com os predicados de seleção (P-list) e atributos de seleção (A-list) que se referem ao *record-type*. Cada aresta $a \in A$ é rotulada com o *set-type* (ST) percorrido para ir de um *record-type* a outro e com o número de ordem de pesquisa (NO) que serve para indicar a ordem de visitação das arestas que partem de um mesmo vértice

Saídas:

. Sai-1: Rotina (seção) de um programa COBOL com comandos embutidos da LMD rede que executam as operações solicitadas na transação GORDAS. O nome da seção é formado pela concatenação do tipo de operação com o nome do cursor associado à chamada. O nome dos parágrafos contidos na seção são formados pelo nome da seção acrescido de um sufixo diferenciador formado por um número inteiro, seqüencial e crescente.

. Sai-2: Definição de um arquivo cujos registros são formados pelos atributos contidos na cláusula GET da transação GORDAS. Os nomes do arquivo e do registro contido no arquivo são formados do mesmo modo que o nome da seção apenas com a inclusão dos sufixos ARQ e REG, respectivamente. O arquivo criado é utilizado para retornar o conjunto de dados selecionados ao módulo Monitor-de-Comunicação que ativa o programa rede.

. Sai-3: Definição das variáveis auxiliares que são usadas na rotina construída.

Passos:

A: Incluir no programa a declaração do arquivo de saída e das variáveis de controle FLAG-ERRO e FLAG-FIM.

B: Para cada predicado de seleção $<P_i>$, $1 \leq i \leq n$, obter no Mcl o construtor do esquema componente referenciado pelo caminho definido pelos nomes de conexão e o *record-type* ($<R_i>$) correspondente no esquema local. Obter também o caminho de navegação no esquema local que corresponde ao caminho definido em $<P_i>$. Incluir os dados de cada $<P_i>$ na tabela T1.

C: Para cada $<A_i$-list>, $1 \leq i \leq n$, executar as ações do passo **B** menos a inclusão na tabela T1. Incluir os dados de cada $<A_i$-list> na tabela T2.

D: Reunir as entradas da tabela T1 que contiverem o mesmo caminho de navegação em uma única entrada, incluindo nesta entrada os predicados de seleção originais. Classificar as entradas da tabela de acordo com o tamanho do caminho de navegação, do menor para o maior. Dessa forma, a primeira entrada da tabela será aquela que contiver os predicados que se referem diretamente ao *record-type* do esquema local que corresponde ao construtor <C> da transação GORDAS.

E: Construir o grafo G assim:

E1: Incluir para cada entrada da tabela T1 um vértice contendo o nome do *record-type* alcançado e os predicados de seleção que contiverem os atributos desse *record-type*. Para as entradas que contiverem caminhos definidos pelos *set-type* incluir as arestas que representam os mesmos, numerando essas arestas em uma ordem seqüencial crescente.

E2: Examinar T2 e para cada entrada, cujo caminho de navegação não está contido em T1, incluir no grafo o vértice e as arestas da mesma forma que no passo **E1**. Para as entradas cujos caminhos já estavam na tabela T1 incluir no vértice do grafo que representa o *record-type* alcançado, a lista de atributos que se deseja obter.

E3: Eliminar do grafo os vértices que não possuem arestas partindo de si e cujas listas de atributos e predicados sejam vazias

F: Incluir no programa gerado a declaração da seção que será associada à transação GORDAS

G: Percorrer o grafo em profundidade ou amplitude a partir do vértice fonte (vértice que representa o *record-type* correspondente a <C>). Se existir para um mesmo vértice mais de uma aresta candidata tomar a que tiver o menor número de ordem de busca.

Para cada vértice visitado faça

H: Incluir um parágrafo no programa

I: Incluir o atributo de cada predicado de seleção que tem um operador de igualdade em uma lista de parâmetros (<param-list>) e inserir no programa rede o seguinte comando:

MOVE <v-pred> TO <a-pred>

J: Se o vértice visitado é o vértice fonte

Então

J1: Incluir o seguinte comando no programa:

FIND ANY <RT> USING <param-list>

IF DB-STATUS NOT EQUAL 0

```
        MOVE 1 TO FLAG-ERRO
    ELSE
```

Senão

Se o *record-type* (<RT>) representado pelo vértice visitado é membro no *set-type* (<ST>) representado pela aresta percorrida para alcançá-lo

Então

J2: Incluir o seguinte comando no programa:

```
FIND FIRST <RT> WITHIN <ST>
                    USING <param-list>

IF DB-STATUS NOT EQUAL 0
   MOVE 1 TO FLAG-ERRO
ELSE
```

Senão

J3: Incluir o seguinte comando no programa:

```
FIND OWNER WITHIN <ST>

IF DB-STATUS NOT EQUAL 0
   MOVE 1 TO FLAG-ERRO
ELSE
```

K: Se os atributos contidos em <param-list> são suficientes para identificar univocamente o *record-type* <RT> representado pelo vértice, ou se esse *record-type* é mestre do *set-type* <ST> representado pela aresta percorrida para alcançá-lo

Então

K1: Incluir no programa gerado os seguintes comandos:

```
GET <RT>
```

K2: Se a lista de atributos (<A-list>) contida no vértice é não vazia

Então

K2.1: Incluir no programa gerado o seguinte comando:

MOVE <A list> TO <sai list>

onde:

. <sai-list> é a lista de atributos contida no registro do arquivo de saída declarado no passo **A**

K3: Se não existem mais vértices a visitar no grafo
Então
K3.1: Incluir no programa gerado o comando para gravar o registro de saída
Senão
K3.2: Incluir no programa gerado o seguinte comando

PERFORM <prox-parag>

onde:

. <prox-parag> é o nome do próximo parágrafo que será associado ao próximo vértice visitado.
O nome desse parágrafo é obtido acrescentando uma unidade ao sufixo do último parágrafo incluído no programa.

Senão
K4: Incluir no programa os seguintes comandos:

```
MOVE 0 TO FLAG-FIM
PERFORM <prox-parag> UNTIL
                     FLAG-FIM = 1
MOVE 0 TO FLAG-FIM.
<prox-parag>.
GET <RT>
```

K5: Incluir na lista de predicados de teste (<IF-list>) os predicados de seleção do vértice visitado cujos operadores não sejam operadores de igualdade

Se <IF-list> é vazia
Então

K5.1: Idem K2

K5.2: Idem K3

Senão

K5.3: Incluir no programa o seguinte comando

IF <IF-list>

K5.4: Idem K2

K5.5: Idem K3

K6: Se o vértice visitado é o vértice fonte
Então

K6.1: Incluir no programa os seguintes comandos:

```
FIND DUPLICATE <RT> USING
                          <param-list>.
IF DB-STATUS = END-OF-SET
   MOVE 1 TO FLAG-FIM
ELSE
   IF DB-STATUS NOT EQUAL 0
      MOVE 1 TO FLAG-ERRO
                FLAG-FIM.
```

Senão

K6.2: Incluir no programa os seguintes comandos:

```
FIND NEXT <RT> USING <param-list>
IF DB-STATUS = END-OF-SET
   MOVE 1 TO FLAG-FIM
ELSE
   IF DB-STATUS NOT EQUAL 0
      MOVE 1 TO FLAG-ERRO
                FLAG-FIM.
```

L: Incluir no programa o parágrafo que finaliza a seção

Comentário:

A forma como os comandos FIND são colocados no programa rede não permite que dois registros, pertencentes a um mesmo *record-type* membro de um set-type, sejam recuperados antes que um registro de cada *record-type* contido no grafo o seja. Assim, a cada vez recupera-se um único valor de cada um dos atributos existente nas listas de atributos (<A_i-list>), formando um registro que pode ser gravado em um arquivo auxiliar.

Exemplo: Considere o seguinte comando GORDAS associado a uma chamada de abertura do cursor C1 (tipo de operação = OC)

GET (<matricula,nome-empreg> OF empregados,
<nome-cargo> OF cargo OF empregados) OF Departamento
WHERE cod-depto OF Departamento = "DEP1"

Passo A - Inclui-se na parte de declaração de arquivos e variáveis do programa o arquivo OC-C1-ARQ e as variáveis FLAG-ERRO e FLAG-FIM.

Passo B - Obtém-se
T1 (RT1: Departamento, PT1: (cod-depto = "DEP1"))

Passo C - Obtém-se
T2 ((RT2 : Empregado, ST2: Dep-Emp, AT2: (matricula,nome-empreg)),
(RT2 : Cargo, ST2: Cargo-Emp, AT2: (nome-cargo)))

Passo D - Mantém-se T1 inalterada

Passo E - Constrói-se o grafo G=(V,A), V=(v_1,v_2,v_3) e A=(a_1,a_2), onde:

v_1 = (RT: Departamento, P-list: (cod-depto = "DEP1"))
v_2 = (RT: Empregado, A-list:(matricula,nome-empreg))
v_3 = (RT: Cargo, A-list: (nome-cargo))
a_1 = (ST: Dep-Emp, NO: 1)
a_2 = (ST: Cargo-Emp, NO: 2)

Passo F - Inclui-se no programa:

```
OC-C1 SECTION.
OC-C1-0.
      PERFORM OC-C1-1
      GO TO OC-C1-EXIT.
```

Passo G - Visita-se o vértice v_1

Passo H - Inclui-se no programa:

```
OC-C1-1.
```

Passo I - Inclui-se no programa:

```
MOVE ''DEP1'' TO  cod-depto
```

Passo J - Verifica-se que o vértice visitado é o vértice fonte e inclui-se no programa:

```
FIND ANY Departamento USING cod-depto
IF DB-STATUS NOT EQUAL 0
   MOVE 1 TO FLAG-ERRO
ELSE
```

Passo K - Verifica-se que *cod-depto* identifica univocamente os registros de *Departamento* (LOCATION MODE CALC USING cod-depto)

Passo K1 - Inclui-se no programa:

```
GET Departamento
```

Passo K2 - Verifica-se que a lista de atributos do vértice visitado é vazia e nenhuma ação deve ser executada

Passo K3 - Verifica-se que existem mais vértices a visitar no grafo

Passo K3.2 - Inclui-se no programa:

```
PERFORM OC-C1-2.
```

Passo G - Percorre-se a aresta a_1 e visita-se o vértice v_2

Passo H - Inclui-se no programa:

```
OC-C1-2.
```

Passo I - Nenhuma ação é executada já que a lista de predicados do vértice é vazia

Passo J - Verifica-se que o *record-type Empregado* representado pelo vértice visitado é membro no *set-type Dep-Emp* representado pela aresta percorrida para alcançá-lo. Inclui-se no programa:

```
FIND FIRST Empregado WITHIN Dep-Emp
IF DB-STATUS NOT EQUAL 0
   MOVE 1 TO FLAG-ERRO
ELSE
```

Passo K - Verifica-se que o comando do passo **J** não seleciona um único registro (<param-list> é vazia e o *record-type* não é mestre do *set-type* percorrido)

Passo K1 - Inclui-se no programa:

```
      MOVE 0 TO FLAG-FIM.
      PERFORM OC-C1-3 UNTIL FLAG-FIM = 1
      MOVE 0 TO FLAG-FIM.
OC-C1-3.
      GET Empregado
```

Passo K5 - Verifica-se que a lista de predicados de seleção do vértice visitado é vazia

Passo K5.1 - Verifica-se que a lista de atributos do vértice visitado é não vazia e inclui-se no programa

```
MOVE matricula TO matricula OF OC-C1-REG
MOVE nome-empreg TO nome-empreg OF OC-C1-REG
```

Passo K5.2 - Verifica-se que existem mais vértices a visitar no grafo e inclui-se no programa:

```
PERFORM OC-C1-4.
```

Passo K6 - Verifica-se que o vértice visitado não é o vértice fonte

Passo K6.2 - Inclui-se no programa:

```
FIND NEXT Empregado WITHIN DEP-EMP
```

```
IF DB-STATUS = END-OF-SET
   MOVE 1 TO FLAG-ERRO
             FLAG-FIM.
```

Passo G - Percorre-se a aresta a_2 e visita-se o vértice v_4

Passo H - Inclui-se no programa:

```
OC-C1-4.
```

Passo I - Nenhuma ação é executada já que a lista de predicados do vértice é vazia

Passo J - Verifica-se que o *record-type Cargo* representado pelo vértice visitado é mestre no *set-type Cargo-Emp* representado pela aresta percorrida para alcançá-lo. Inclui-se no programa:

```
FIND OWNER WITHIN Cargo-Emp
IF DB-STATUS NOT EQUAL 0
   MOVE 1 TO FLAG-ERRO
ELSE
```

Passo K - Verifica-se que o comando do passo J seleciona um único registro (*record-type* é mestre do *set-type* percorrido)

Passo K1 - Inclui-se no programa:

```
GET Cargo
```

Passo K5 - Verifica-se que a lista de predicados de seleção do vértice visitado é vazia

Passo K2 - Verifica-se que a lista de atributos do vértice vistado é não vazia e inclui-se no programa

```
MOVE nome-cargo TO nome-cargo OF OC-C1-REG
```

Passo K3 - Verifica-se que não existem mais vértices a visitar no grafo

Passo K3.1 - Inclui-se no programa:

```
WRITE OC-C1-REG.
```

Passo L - Inclui-se no programa

```
OC-C1-EXIT. EXIT.
```

5.4.2 Transações GORDAS para atualização de dados

Os algoritmos usados para tratamento das transações GORDAS de atualização de dados se assemelham ao algoritmo apresentado para as transações de seleção de dados. Essa semelhança se deve ao fato de que para ambos os tipos de transação é necessária a construção de um caminho de navegação no esquema local que permita satisfazer os predicados contidos na cláusula WHERE e os critérios de seleção de ocorrências nos *set-type* (SET SELECTION).

A diferença essencial entre o algoritmo para transações de seleção de dados e os algoritmos para transações de atualização de dados é que nos últimos não precisam ser armazenados os dados existentes nos *record-type* do grafo. Esse fato permite que sejam excluídos do algoritmo da seção 5.4.1 os passos que se referem à criação do arquivo de sa'ida (passo **A**), os passos que se referem à tabela T2 (passos **C** e **E2**) e os passos que alimentam o registro do arquivo de sa'ida (passos **K2**, **K5.1** e **K5.4**). A outra modificação genérica que precisa ser feita no algoritmo da seção 5.4.1 é a substituição do comando de gravação do registro de sa'ida (passo **K3.1**) por outros comandos que executam as operações de atualização no SBD componente rede.

A seguir serão apresentados as transações de atualização de dados mais comuns e as modificações específicas que cada uma delas impõe ao algoritmo da seção anterior, além daquelas alterações genéricas descritas no último parágrafo.

I) Exclusão de entidades

Transação GORDAS:

DELETE FROM <E>
[WHERE <P_1> AND <P_2> ... AND <P_n>]

Algoritmo:

1: Modificar o passo **E** para incluir no grafo de navegação o *record-type* <RT> do esquema local que corresponde ao tipo de entidade <E> do esquema componente, caso a cláusula WHERE não apareça na transação GORDAS.

2: Substituir o comando do passo **K3.1** pelos seguintes comandos:

FIND CURRENT <RT>
ERASE <RT>

```
IF DB-STATUS NOT EQUAL 0
   MOVE 1 TO FLAG-ERRO.
```

Comentários:

1: O comando FIND CURRENT <RT> é usado para tornar o registro do *record-type* <RT> o registro corrente da unidade de execução (*current of run-unit*), satisfazendo assim a exigência do comando ERASE. Isto é necessário porque durante a navegação no grafo o último registro recuperado pode não ter sido o registro desejado.

II) Inclusão de entidades

Transação GORDAS:

```
INSERT INTO E
        ATTRIBUTES <a1> = <param-1>
                          :
                   <an> = <param-n>

[ADD TO RELATIONSHIP <R1> WHERE
        <E> : <E.P-list>
        <E1> : <E1.P-list>
        :
ADD TO RELATIONSHIP <Rn> WHERE
        <E> : <E.P-list>
        <En> : <En.P-list>]
```

Algoritmo:

1: Modificar o passo **E** construindo para cada relacionamento $<R_i>$ um grafo de navegação a partir dos predicados de seleção que se referem ao tipo de entidade $<E_i>$. Esse tipo de entidade corresponde ao *record-type* mestre $<M_i>$ no *set-type* $<S_i>$ que corresponde ao relacionamento $<R_i>$.

2: Modificar o passo **G** para percorrer cada um dos grafos construídos no passo **E**

3: Substituir o comando do passo **K3.1** pelos seguintes comandos:

FIND CURRENT <M_i>

4: Incluir antes do passo **L** os seguintes passos:

P1: Incluir no programa gerado os seguintes comandos:

MOVE <param-1> TO <a_{I_1}>
⋮
MOVE <param-n> TO <a_{I_m}>
INSERT <RT>

P2: Para cada relacionamento <R_i> em que <E> não tem participação total faça:

CONNECT <RT> TO <S_i>
IF DB-STATUS NOT EQUAL 0
MOVE 1 TO FLAG-ERRO.

Comentários:

1: O aparecimento de comandos para inclusão de instâncias de relacionamentos na transação GORDAS é conseqüência da presença de chaves estrangeiras na relação que foi objeto da atualização no programa original do usuário construído sobre um esquema externo relacional.

2: Cada grafo construído para um relacionamento no passo **E** representa o caminho de navegação que satisfaz o SET SELECTION do *set-type* que representa o relacionamento no esquema local.

3: <RT> é o *record-type* do esquema local que corresponde ao tipo de entidade <E> do esquema componente.

4: Ao inserir o *record-type* <RT>, o SGBD rede automaticamente o inclui em uma *set-occurrence* de cada *set-type* em que ele participa obrigatoriamente (SET MEMBERSHIP MANDATORY/FIXED AUTOMATIC).

5: A inclusão do *record-type* <RT> nos *set-type* em que ele participa como membro opcionalmente (SET MEMBERSHIP OPTIONAL MANUAL) é feita no passo **P2**.

6: Os atributos $<a_{L_1},\ldots,a_{L_n}>$ são os atributos do esquema local definidos no programa e correspondem aos atributos $<a_1,\ldots,a_n>$ do esquema componente

III) Alteração de atributos de entidades

Transação GORDAS:

```
MODIFY <a1,a2,...,an> OF <E> TO
          <param-1,param-2,...,param-n>
[WHERE <P1> AND <P2> ... AND <Pn>]
```

Algoritmo:

1: Modificar o passo **E** da mesma forma que no algoritmo para exclusão de entidades.

2: Substituir o comando do passo **K3.1** pelos seguintes comandos:

```
FIND CURRENT <RT>
MOVE <param-1> TO <aL1>
MOVE <param-2> TO <aL2>
        :
MOVE <param-n> TO <aLn>

MODIFY <RT>
IF DB-STATUS NOT EQUAL 0
   MOVE 1 TO FLAG-ERRO.
```

IV) Exclusão de instâncias de relacionamentos

Transação GORDAS:

```
REMOVE FROM RELATIONSHIP <R> WHERE
          <E1> : <E1.P-list>
          [<E2> : <E2.P-list>
                :
          <En> : <En.P-list>]
```

Algoritmo:

1: Incluir um passo antes do passo **B** para transformar o formato da cláusula WHERE da transação GORDAS para o formato da cláusula WHERE da transação de seleção de dados. Para tal, toma-se como tipo de entidade base um dos tipos de entidade contidos na cláusula WHERE e substitui-se os atributos contidos nos predicados de seleção dos demais tipos de entidade pelos mesmos atributos concatenados com os nomes de conexão que permitem referenciar esses tipos de entidade a partir do tipo de entidade base

2: Incluir após o passo **E** um passo para verificar o tipo de construtor do esquema local usado para representar o relacionamento <R>. Se o relacionamento <R> for representado por um *set-type* <S> e o *record-type* (<RT>) membro desse *set-type* não aparece no grafo, então deve-se incluir um vértice no grafo para representá-lo. Se existir no grafo um vértice que representa o *record-type* mestre de <S>, então deve-se incluir também uma aresta partindo desse vértice e chegando no vértice incluído. Por outro lado, se o relacionamento <R> estiver representado por um *record-type* conector <RC> e esse *record-type* não aparece no grafo, então deve-se incluir no grafo um vértice para representá-lo e uma aresta partindo do vértice fonte do grafo, que representa o tipo de entidade base, e chegando no vértice incluído.

3: Substituir o passo **K3.1** pelo seguinte passo:

P: Se o relacionamento <R> corresponde a um *set-type* <S> no esquema local
Então

P1: Incluir os seguintes comandos no programa:

```
FIND CURRENT <RT>
DISCONNECT <RT> FROM <S>
IF DB-STATUS NOT EQUAL 0
   MOVE 1 TO FLAG-ERRO.
```

Senão

P2: Incluir os seguintes comandos no programa:

```
FIND CURRENT <RT>
ERASE <RC>
IF DB-STATUS NOT EQUAL 0
   MOVE 1 TO FLAG-ERRO.
```

Comentários:

1: A transformação do formato da cláusula WHERE é importante no caso de relacionamentos representados através de registros conectores no esquema local. Essa transformação garante que os passos **B** e **E** incluirão no grafo o vértice que representa o *record-type* conector e as arestas que ligam esse vértice a cada um dos outros vértices que representam os *record-type* mestres dos *set-type* em que esse conector participa como membro. Suponha, por exemplo o seguinte comando GORDAS:

REMOVE FROM Particip-Projeto WHERE
Empregado : matricula = 3515
Projeto : cod-projeto = "PROJ1"

Transformando o formato da cláusula WHERE obtemos os seguintes predicados:

matricula OF Empregado = 3515
cod-projeto OF projetos OF Empregado

Aplicando os passo **B** e **E** do algoritmo obtemos o grafo G=(V,A), V=(v_1,v_2,v_3) e A=(a_1,a_2), onde:

v_1 = (RT: Empregado, P-list: (matricula = 3515))
v_2 = (RT: Particip-Projeto)
v_3 = (RT: Projeto, P-list: (cod-projeto : "PROJ1"))
a_1 = (ST: Emp-Proj, NO: 1)
a_2 = (ST: Proj-Emp, NO: 2)

2: A introdução do vértice, que representa o *record-type* membro do *set-type* correspondente ao relacionamento do comando GORDAS, acontece quando não aparecem predicados de seleção envolvendo esse *record-type* na cláusula WHERE. Isto se dá, por exemplo, nos seguintes comandos (*Dep-Emp* corresponde ao *set-type* em que *Departamento* é o *record-type* mestre e *Empregado* é o *record-type* membro)

REMOVE FROM Dep-Emp WHERE
Departamento : cod-depto : "DEP3"

REMOVE FROM Dep-Emp

3: A introdução do vértice, que representa o *record-type* conector correspondente ao relacionamento do comando GORDAS, acontece quando na cláusula WHERE aparecem predicados de seleção se referindo a no máximo um tipo de entidade presente no relacionamento. Isto acontece, por exemplo nos seguintes comandos:

REMOVE FROM Particip-Projeto WHERE
Empregado : matricula = 3515

REMOVE FROM Particip-Projeto WHERE
Projeto : cod-projeto = "PROJ3"

REMOVE FROM Particip-Projeto

V) Inclusão de instâncias de relacionamentos

Transação GORDAS:

ADD TO RELATIONSHIP <R> WHERE
<E_1> : <E_1.P-list>
[<E_2> : <E_2.P-list>
⋮
<E_n> : <E_n.P-list>]

[ATTRIBUTES <a_1> = <param-1>,...,<a_n> = <param-n>]

Algoritmo:

1: Modificar o passo **E**, construindo um grafo de navegação para cada lista de predicados contida na cláusula WHERE.

2: Modificar o passo **G** para percorrer cada um dos grafos construídos no passo **E**.

3: Excluir o passo **K3.1**.

4: Incluir antes do passo **L** os seguintes passos:

P: Se o relacionamento <R> corresponde a um *set-type* <S> no esquema local cujo *record-type* membro é <RT>

Então

P1: Incluir os seguintes comandos no programa:

```
FIND CURRENT <RT>
CONNECT <RT> FROM <S>
IF DB-STATUS NOT EQUAL 0
   MOVE 1 TO FLAG-ERRO.
```

Senão

P2: Se existem predicados de atribuição (<a_i> = <param-i>) envolvendo atributos do *record-type* conector que representa o relacionamento <R>

Então

P2.1: Incluir os seguintes comandos no programa:

```
MOVE <param-1> TO <a_I1>
MOVE <param-2> TO <a_I2>
        :
MOVE <param-n> TO <a_In>

INSERT <RC>
IF DB-STATUS NOT EQUAL 0
   MOVE 1 TO FLAG-ERRO.
```

Senão

P2.2: Incluir os seguintes comandos no programa:

```
INSERT <RC>
IF DB-STATUS NOT EQUAL 0
   MOVE 1 TO FLAG-ERRO.
```

Comentários:

1: Cada grafo de navegação construído no passo **E** satisfaz uma parte dos critérios de seleção da set-occurrence que será incluída.

VI) Alteração de atributos de instâncias de relacionamentos

Transação GORDAS:

MODIFY <a_1,a_2, ...,a_n> OF *R* TO
<param-1,param-2,...,param-n>
[WHERE <E_1> : <E_1.P-list>
[<E_2> : <E_2.P-list>
⋮
<E_n> : <E_n.P-list>]

Algoritmo:

1: Incluir um passo antes do passo **B** para transformar o formato da cláusula WHERE do comando GORDAS para o formato da cláusula WHERE da transação de seleção de dados.

2: Modificar o passo **E** para que, após a construção do grafo, seja verificada a presença no grafo do *record-type* conector (<RC>) que representa o relacionamento <R> no esquema local. Se <RC> não estiver contido no grafo, incluí-lo no mesmo.

3: Substituir o comando do passo **K3.1** pelos comandos:

```
MOVE <param-1> TO <a_L1>
MOVE <param-2> TO <a_L2>
      ⋮
MOVE <param-n> TO <a_Ln>

MODIFY <RC>
IF DB-STATUS NOT EQUAL 0
   MOVE 1 TO FLAG-ERRO.
```

VII) Transações GORDAS para atualização com mais de um comando

Os algoritmos para tratamento de transações GORDAS de atualização que contêm mais de um comando podem ser obtidos através da combinação dos algoritmos apresentados anteriormente. Um exemplo desse tipo de transação é a transação da figura 5.2 que substitui as instâncias do relacionamento *Particip-Projeto* em que o empregado com matrícula

100 participa. Para esse tipo de transação aplica-se o algoritmo do item IV da seção 5.1.2 para o primeiro comando e o algoritmo do item V da mesma seção para o segundo comando, tomando-se o cuidado de não considerar no segundo algoritmo o predicado de seleção *matricula = 100* que já foi tratado pelo primeiro algoritmo.

```
REMOVE FROM Particip-Projeto WHERE
            Empregado : matricula = 100
            Projeto   : cod-projeto = ''PROJ1''

ADD    TO   Particip-Projeto WHERE
            Empregado : matricula = 100
            Projeto   : cod-projeto = ''PROJ2''
```

Figura 5.2: Exemplo de transação de atualização GORDAS contendo mais de um comando

Capítulo 6

Conclusões

6.1 Considerações finais

Transparência de modelos de dados em um SBDH é a propriedade que garante a cada usuário a possibilidade de visualizar e manipular os dados através de um único modelo de dados e de uma única linguagem de manipulação de dados (LMD), independentemente do fato de existirem diferentes modelos e linguagens nos SBDs que compõem o ambiente integrado. Tal propriedade é obtida através dos mapeamentos entre os esquemas e entre as LMDs dos SBDs componentes.

Nesta dissertação foram propostas algumas soluções para garantir a transparência de modelo de dados em SBDHs construídos a partir da integração dos dados contidos em SBDs autônomos que seguem o modelo de dados rede ou o modelo de dados relacional.

A abordagem adotada para o mapeamento entre os esquemas e as LMDs dos SBDs componentes foi aquela que prevê o mapeamento do tipo *muitos-para-muitos* [Hsi92]. Nesse tipo de mapeamento, os esquemas dos SBDs componentes (*esquemas locais*) são convertidos em esquemas que seguem um mesmo modelo de dados, o modelo de dados comum (MDC). Os esquemas obtidos (*esquemas componentes*) são integrados em um ou mais esquemas virtuais globais (*esquemas federados*) que, por sua vez, são convertidos em *esquemas externos* nos modelos de dados originais para serem usados pelos usuários dos SBDs componentes. As operações contidas nos programas de aplicação, construídos sobre um esquema externo, que se referem a SBDs não locais, são transformadas em operações equivalentes, primeiramente na LMD associada ao modelo de dados comum e posteriormente na LMD nativa do SBD componente.

Esse tipo de abordagem se contrapõe ao mapeamento do tipo *um-para-muitos* usado

na maioria dos trabalhos publicados na literatura correlata, em que todos os usuários do SBDH "enxergam" os dados segundo o mesmo modelo de dados e utilizam exclusivamente a LMD associada a esse modelo unificado para construir suas aplicações.

A utilização de um modelo de dados comum e de uma linguagem intermediária se justifica por dois motivos importantes. Primeiro, o modelo de dados comum permite a uniformização da representação dos dados facilitando a integração dos mesmos. Segundo, à medida que a variedade de modelos de dados e LMDs presentes no SBDH cresce, o uso de um modelo de dados e uma LMD intermediários permite uma sensível diminuição do número de mapeamentos necessários, em comparação com o mapeamento direto entre cada par de modelos e linguagens.

Optou-se por adotar um modelo de dados semântico como MDC por acreditar-se na melhor adequação desse tipo de modelo de dados ao objetivo pretendido. O modelo de dados e a linguagem intermediária escolhidos foram o modelo ECR e a linguagem GORDAS associada a ele. O modelo ECR é uma extensão do modelo Entidade-Relacionamento que introduz um novo construtor (*categoria*) o qual permite a representação de abstrações de generalização/especialização. Além disso, o modelo ECR apresenta uma definição funcional dos atributos e uma especificação mais completa do tipo de participação das entidades nos relacionamentos.

As soluções propostas na dissertação incluíram:

- metodologias para conversão de esquemas locais em esquemas componentes;
- metodologias para conversão de esquemas federados em esquemas externos;
- arquiteturas de processadores para permitir o acesso aos dados do SBDH via programas de aplicação construídos sobre esquemas externos;
- algoritmos para transformar operações na LMD associada ao esquema externo em operações GORDAS sobre os esquemas componentes;
- algoritmos para transformar operações GORDAS sobre o esquema componente em operações equivalentes na LMD nativa do SBD componente.

6.2 Contribuições

Uma importante contribuição desta dissertação foi a proposição de soluções concretas para permitir a utilização de mais de um modelo de dados e mais de uma LMD para o acesso aos dados de um SBDH, mantendo-se a transparência de modelos de dados. Embora a maioria

dos trabalhos publicados na literatura correlata não adotem essa abordagem, acredita se que a adequação do modelo de dados e da linguagem de acesso ao perfil do usuário garante uma maior transparência ao SBDH, pois os usuários não precisam se familiarizar com um outro modelo de dados e com uma nova LMD para utilizar as facilidades providas pelo ambiente integrado.

As metodologias propostas para tradução de esquemas, embora possam ser utilizadas em outros contextos, apresentam características específicas que permitem a sua aplicação em SBDHs e as diferenciam de outras metodologias já apresentadas. Uma dessas características é a construção e armazenamento de mapeamentos entre os objetos dos esquemas presentes nos diferentes níveis que compõem a arquitetura do SBDH. Tais mapeamentos são essenciais para o processo de transformação de operações do usuário em operações que possam ser executadas pelos SGBDs dos SBDs componentes.

As arquiteturas e algoritmos apresentados nos capítulos 4 e 5, por sua vez, possibilitam o acesso aos dados de um SBDH através de programas de aplicação comuns, em que comandos de uma LMD são embutidos em uma linguagem de programação hospedeira. Assim permite-se:

1. O aproveitamento das facilidades fornecidas pelas linguagens hospedeiras de propósito geral;
2. O acesso repetitivo aos dados do SBDH sem a necessidade de se construir interfaces especiais ou de se executar somente operações *ad-hoc*;
3. A utilização de LMD procedimental, como a LMD rede, para manipular os dados do SBDH;
4. O aproveitamento de recursos computacionais providos pelos sistemas onde estão incluídos os SBDs componentes.

Além disso, as premissas adotadas para a integração de esquemas possibilitam a preservação das aplicações locais já existentes e conseqüentemente a manutenção da autonomia de projeto e associação dos SBDs componentes.

Uma outra contribuição relevante do trabalho foi a avaliação do uso de um modelo de dados semântico como MDC. O modelo ECR mostrou-se bastante adequado ao papel de MDC porque conseguiu representar de maneira clara e precisa os construtores dos modelos de dados relacional e rede. Além disso, modelos semânticos, como o modelo ECR, apresentam uma outra vantagem: a capacidade de capturar e modelar restrições de integridade não contidas nos esquemas locais. Essa capacidade permite a inclusão pelo próprio sistema de rotinas para verificação das restrições de integridade nos programas

construídos para atuar sobre os SBDs componentes. Desse modo, o modelo de dados comum pode servir como uma espécie de interface semântica [Oli92] para os modelos de dados dos SBDs componentes, tornando os seus dados menos vulneráveis às aplicações dos usuários.

6.3 Sugestões para extensões e trabalhos futuros

Algumas extensões podem ser propostas para tratar de forma mais detalhada determinados pontos citados na dissertação. Essas extensões são:

- a definição de mecanismos para inclusão automática de rotinas para validação de restrições de integridade no processo de transformação de operações e construção de programas que atuam sobre os SBDs componentes;
- o desenvolvimento de uma metodologia para integração de esquemas que obedeça às premissas citadas na seção 3.4 e que resolva os conflitos estruturais e semânticos existentes entre os esquemas componentes;
- a proposição de soluções para suportar a decomposição e processamento de transações globais GORDAS, obtidas a partir da transformação de comandos de LMDs não procedimentais usadas pelos usuários do SBDH;
- a definição de algoritmos para mapeamento entre a linguagem GORDAS e outras LMDs utilizadas pelos SBDs componentes como, por exemplo, linguagens de quarta geração;
- o tratamento de outras variações mais sofisticadas de comandos SQL;
- a proposição de metodologias para conversão de esquemas e algoritmos para transformação de operações entre o nível federado e o nível externo da arquitetura do SBDH que permitam a inclusão do conceito de visões nos esquemas externos.

Além das extensões citadas anteriormente, seria interessante o desenvolvimento de pesquisas que avaliassem novas tendências propostas na literatura como a utilização de modelos de dados orientados a objetos no papel de modelo de dados comum e de sistemas especialistas para suportar a tradução e integração de esquemas.

Espera-se, por fim, que as soluções propostas nesta dissertação, bem como as extensões sugeridas, possam ser implementadas futuramente, obtendo-se um conjunto de ferramentas que possa servir como base para um projeto de maior porte, envolvendo aspectos

operacionais dos SBDHs como: modelos para controle de concorrência, tolerância à falha e prevenção de bloqueios perpétuos (*deadlocks*); protocolos e políticas para administração do ambiente integrado; e mecanismos para incrementar o desempenho das aplicações.

Bibliografia

[ABD+89] M. Atkinson, F. Bancilhon, D. Dewitt, K. Dittrich, D. Maier, and S. Zdonik. The object-oriented database manifesto. In *Proceedings of the First Int. Conference on Deductive and Object-Oriented Databases*, Kyoto, Japan, December 1989.

[ADD+91] R. Ahmed, P. DeSmedt, W. Du, W. Kent, M. Ketabchi, W. Litwin, A. Rafii, and M. Shan. The PEGASUS heterogeneous multidatabase system. *Computer*, pages 19–27, December 1991.

[AP87] N. Azar and G. Pichat. Translation of an extended entity-relationship model into the univeral relation with inclusion formalism. In S. Spaccapietra, editor, *Entity-Relationship Approach*, pages 253–268. North-Holland, 1987.

[BDH+88] H. Briand, C. Ducateau, Y. Hebrail, D. Herin-Aime, and J. Kouloumdjian. From minimal cover to entity-relationship diagram. In S. T. March, editor, *Entity-Relationship Approach*, pages 287–301. North-Holland, 1988.

[BDK+88] V. Belcastro, A. Dutkowski, W. Kominski, M. Kowalewski, C. Mallamaci, and S. Mezyic. An overview of the distributed query system DQS. In *Lecture Notes in Computer Science*, pages 170–189. Springer-Verlag, 1988. Volume 303.

[BHP92] M. Bright, A. Hurson, and S. Pakzard. A taxonomy and current issues in multidatabase systems. *Computer Magazine*, pages 50–59, mar 1992.

[BL84] C. Batini and M. Lenzerini. A methodology for data schema integration in entity-relationship model. *IEEE Transactions on Softwar Engineering*, SE-10(6):650–664, nov 84.

[BLN86] C. Batini, M. Lenzerini, and S. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):323–364, December 1986.

[BOT86] Y. Breitbart, P. Olson, and G. Thompson. Database integration in a distributed heterogeneous database system. In *IEEE Proceedings of the 2nd Int. Conference on Data Engineering*, pages 301–310, 1986.

[Car87] A. Cardenas. Heterogeneous distributed database management: The HD-DBMS. *Proceedings of the IEEE*, 75(5):588–600, May 1987.

[CBTY89] A. Chen, D. Brill, M. Templeton, and C. Yu. Distributed query processing in mermaid: a frontend system for multiple database systems. *J. Selected Areas Commun.*, 7(3):390–398, April 1989.

[Che76] P. Chen. The entity-relationship model: Toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.

[Chu90] C. Chung. DATAPLEX: An access to heterogeneous distributed databases. *Communications of ACM*, 33(1):70–80, January 1990.

[CNC83] I. Chung, F. Nakamura, and P. Chen. A decomposition of relations using the entity-relationship approach. In P. P. Chen, editor, *Entity-Relationship Approach to Information Modeling and Analysis*, pages 149–171. North-Holland, 1983.

[CP84] S. Ceri and G. Pelagatti. *Distributed Databases - Principles and Systems*. McGraw-Hill, 1984.

[CRE87] B. Czejdo, M. Rusinkiewicz, and D. Embley. An approach to schema integration and query formulation in federated database systems. In *IEEE Proceedings of the 3rd Int. Conference on Data Engineering*, pages 477–484, February 1987.

[CS83] M. A. Casanova and J. E. Sa. Design entity-relationship schemas for conventional information systems. In *Entity-Relationship Approach to Software Engineering*, pages 265–277. North-Holland, 1983.

[DA83] S. Dumpala and S. Arora. Schema translation using the entity-relationship approach. In P. P. Chen, editor, *Entity-Relationship Approach to Information Modeling and Analysis*, pages 337–356. North-Holland, 1983.

[DA88] K. Davis and A. Arora. Converting a relational database model into an entity-relationship model. In S. T. March, editor, *Entity-Relationship Approach*, pages 271–285. North-Holland, 1988.

[Dat86] C.J. Date. Introdução a sistemas de bancos de dados, 1986. Capítulo 1, páginas 26-47, Editora Campus ltda., terceira edição.

[DAT87] S. Deen, R. Amin, and M. Taylor. Data integration in distributed databases. *IEEE Transactions on Software Engineering*, SE-13(7):860 864, July 1987.

[DC83] A. Dogac and P. Chen. Entity-relationship model in the ANSI/SPARC framework. In P. P. Chen, editor, *Entity-Relationship Approach to Information Modeling and Analysis*, pages 357 371. North-Holland, 1983.

[Dem83] B. Demo. Program analysis for convertion from a navigational to a specification database interface. In *Proceedings of the 9th Int. Conference on Very Large DataBases*, pages 387 398, Florence, Italy, October 1983.

[DH81] U. Dayal and H. Hwang. View definition and generalization for database integration in a multidatabase system. *IEEE Transactions on Software Engineering*, SE-10(6):628 644, November 1981.

[DH87] S. Demurjian and D. Hsiao. The multilingual database system. In *IEEE Proceedings of the 3rd Int. Conference on Data Engineering*, pages 44 51, 1987.

[DH88] S. Demurjian and D. Hsiao. Towards a better understanding of data models through the multilingual database system. *IEEE Transactions on Software Engineering*, 14(7):916 858, July 1988.

[DL87] P. Dwyer and J. Larson. Some experiences with a distributed database testbed system. *Proceedings of the IEEE*, 75(5):633 647, May 1987.

[dO92] Ronaldo Lopes de Oliveira. Sistemas de bancos de dados heterogêneos, 1992. Estudo Dirigido em Banco de Dados, DCC-IMECC-UNICAMP, Campinas.

[DPSM93] Carlos H.C. Duarte, Esther Pacitti, Sidney D. Silva, and Rubens N. Melo. HEROS: Um sistema de bancos de dados heterogêneos orientado a objetos, 1993. Anais do VIII Simpósio Brasileiro de Bancos de Dados.

[DS82] L. Dorsey and S. Su. The decompilation of COBOL-DML programs for the purpose to DBMS migration. In *Proceedings COMPSAC 82, IEEE Computer Software and Application Conference*, pages 495 504, 1982.

[EN89] R. Elmasri and S. Navathe. *Fundamentals of Database Systems*. The Benjamin/Cummings Publishing Company Inc., 1989.

[EP90] A. Elmargamid and C. Pu. Guest editors introduction to the special issue on heterogeneous databases. *ACM Computing Surveys*, 22(3):175 178, September 1990.

[EW83] R. Elmasri and G. Wiederhold. GORDAS: a formal high level query language for the entity-relationship model. In P. P. Chen, editor, *Entity-Relationship Approach to Information Modeling and Analysis*, pages 49–72. North-Holland, 1983.

[EWH85] R. Elmasri, J. Weeldreyer, and A. Hevner. The category concept: An extension to entity-relationship model. In H. J. Schneider, editor, *Data and Knowledge Engineering*, pages 75–116. North-Holland, June 1985.

[Fon92] J.S. Fong. Methodology for schema translation from hierarqpical or network into relational. *Information and Software Technology*, 31(3):159–171, March 1992.

[HB91] A. Hurson and M. Bright. Multidatabase systems: An advanced concept in handling distributed data. In M. Yovits, editor, *Advances in Computers*, pages 149–200. Academic Press, 1991.

[HK87] R. Hull and R. King. Semantic database modeling: Survey, applications, and research issues. *ACM Computing Surveys*, 19(3):15–62, September 1987.

[HK89] D. Hsiao and M. Kamel. Heterogeneous databases: Proliferation, issues, and solutions. *IEEE Transactions on Knowledge Data Engineering*, 1(1):45–62, 1989.

[HM85] D. Heimbigner and D. McLeod. A federated architecture for information management. *ACM Transactions on Office Systems*, 3(3):253–278, July 1985.

[HR90] S. Hayne and S. Ram. Multi-user view integration system (MUVIS): An expert system for view integration. In *IEEE Proceedings of the 6th Int. Conference on Data Engineering*, pages 402–409, February 1990.

[Hsi92] D. Hsiao. Tutorial on federated databases and systems (Part 1). *The VLDB Journal*, 1(1):127–179, jul 1992.

[JK90] P. Johannesson and K. Kalman. A method for translating relational schemas into conceptual schemas. In F. H. Lochovsky, editor, *Entity-Relationship Approach to Database Design and Querying*, pages 271–285. North-Holland, 1990.

[KBGW91] W. Kim, N. Ballou, J. Garza, and D. Woelk. A distributed object oriented database system suporting shared and private databases. *ACM Transactions on Database Systems*, 9(1):31–51, January 1991.

[KDN90] M. Kaul, K. Drosten, and E. Neuhold. Viewsystem: Integrating heterogeneous information bases by object-oriented views. In *IEEE Proceedings of the 6th Int. Conference on Data Engineering*, pages 2–10, February 1990.

[Ken79] W. Kent. Limitations of record based information models. *ACM Transactions on Database Systems*, 4(1):107–131, 1979.

[LA86] W. Litwin and A. Abdellatif. Multidatabase interoperability. *Computer*, 19(12):10–18, December 1986.

[LA87] W. Litwin and A. Abdellatif. An overview of multidatabase manipulation language MDSL. *Proceedings of the IEEE*, 75(5):621–632, May 1987.

[Lar83] J. Larson. Bridging the gap between network and relational management systems. *Computer*, pages 82–92, September 1983.

[LDJS80] H. Lam, T. Dodd, D. Jefferson, and S. Su. Program analysis and conversion due to DBMS migration. In *Proceedings COMPSAC 80, IEEE Computer Software and Application Conference*, pages 97–108, 1980.

[Lie81] Y. Lien. Hierarquical schemata for relational databases. *ACM Transactions on Database Systems*, 4(1):107–131, 1981.

[Lin89] T. W. Ling. External schemas of entity-relationship based database management systems. In C. Batini, editor, *Entity-Relationship Approach*, pages 195–508. North-Holland, 1989.

[LMR90] W. Litwin, L. Mark, and N. Roussopoulos. Interoperabilty of multiple autonomous databases. *ACM Computing Surveys*, 22(3):267–293, September 1990.

[LNE89] J. Larson, S. Navathe, and R. Elmasri. A theory of attribute equivalence in databases with applications to schema integration. *IEEE Transactions on Software Engineering*, 15(4):449–463, 1989.

[LR82] T. Landers and R. Rosenberg. An overview of multibase. In H. J. Schneider, editor, *Distributed DataBases*, pages 153–184. North-Holland, December 1982.

[Man88] F. Manola. Distributed object management technology. Technical Report TM-0014-06-88-165, GTE Laboratories Incorporated, Waltham, Ma., 1988.

[MR91] P. Muth and T. Rakow. Atomic commitment for integrated databases systems. In *IEEE Proceedings of the 7th Int. Conference on Data Engineering*, pages 296–301, Kobe, Japan, April 1991.

[NA88] S. Navathe and A. Awong. Abstracting relational and hierarquical data with a semantic data model. In S. T. March, editor, *Entity-Relationship Approach*, pages 305–333. North-Holland, 1988.

[NEL86] S. Navathe, R. Elmasri, and J. Larson. Integrating user views in database design. *Computer*, 19(1):50–62, January 1986.

[Oli92] Ricardo Oliveira. S-SQL: uma interface semântica. Master's thesis, DCC-IMECC-UNICAMP, 1992.

[ORA85] ORACLE Corporation, Belmont, California. *ORACLE Overview and Introduction to SQL*, 1985.

[PLC91] C. Pu, A. Leff, and S. Chen. Heterogeneous and autonomous transaction processing. *Computer*, pages 64–72, December 1991.

[PM88] J. Peckham and R. Maryanski. Semantic data models. *ACM Computing Surveys*, 20(3):153–189, September 1988.

[Set86] W. Setzer. Projeto lógico e projeto físico de banco de dados, 1986. V Escola de Computação, Belo Horizonte, Capítulos 3 e 4.

[SL90] A. Sheth and J. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3):183–236, September 1990.

[SLCN90] A. Sheth, J. Larson, A. Cornellio, and S. Navathe. A tool for integrating conceptual schemas and user views. In *IEEE Proceedings of the 6th Int. Conference on Data Engineering*, pages 102–109, February 1990.

[SLW88] A. Sheth, J. Larson, and E. Watkins. TAILOR: A tool for updating views. In *Lecture Notes in Computer Science*, pages 190–213. Springer-Verlag, 1988. Volume 303.

[Sou86] J. Souza. SIS: a schema integration system. In *Proceedings of the BNCOD5 Conference*, pages 167–185, 1986.

[SPY92] S. Spaccapietra, C. Parent, and Dupont Y. Model independent assertions for integration of heterogeneous schemas. *The VLDB Journal*, 1(1):81–126, July 1992.

[SS77] J. M. Smith and D. C. P. Smith. Database abstractions: aggregation and generalization. *ACM Transaction on Database Systems*, 2(2):105–133, 1977.

[SS86] H. Schek and M. Scholl. The relational model with relation-valued atributes. *Information Systems*, 11(2), 1986.

[Sto88] M. Stonebraker. Distributed databases systems introduction. In Michael Stonebraker, editor, *Readings in Database Systems*, pages 189 195. Morgan-Kaufman, 1988.

[TBC+87] M. Templeton, D. Brill, A. Chen, S. Dao, E. Lund, R. McGregor, and P. Ward. MERMAID: a front-end to distributed heterogeneous databases. *Proceedings of the IEEE*, 75(5):695 708, May 1987.

[TF76] R. Taylor and R. Frank. CODASYL database management systems. *ACM Computing Surveys*, 8(1):67 103, March 1976.

[TFS+79] R. Taylor, J. Fry, B. Schneiderman, D. Smith, and S. Su. Database programa conversion: A framework to research. In *Proceedings of The VLDB Conference*, pages 299 312, 1979.

[TFC+90] G. Thomas, G. Thompson, C. Chung, E. Barkmeyer, F. Carter, M. Templeton, S. Fox, and B. Hartman. Heterogeneous distributed databases for production use. *ACM Computing Surveys*, 22(3):237 266, September 1990.

[TYF86] T. Teorey, D. Yang, and J. Fry. A logical design metodology for relational databases using the extended entity-relationship model. *ACM Computing Surveys*, 18(2):197 222, June 1986.

[vdL89] Rick F. van der Lans. *The SQL Standard*. Prentice Hall International, 1989.

[VL80] Y. Vassiliou and F. Lochovsky. DBMS transaction translation. In *Proceedings COMPSAC 80, IEEE Computer Software and Application Conference*, 1980.

[YC84] C. Yu and C. Chang. Distributed query processing. *ACM Computing Surveys*, 16(1):399 433, December 1984.

[Zan79] C. Zaniolo. Design of relational views over network schemas. In *Proceedings of ACM SIGMOD Int. Conference on Management of Data*, pages 179 190, 1979.