UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA
DEPARTAMENTO DE ENGENHARIA DE COMPUTAÇÃO E AUTOMAÇÃO INDUSTRIAL

Este exem[illegible] final da tese
defendi[illegible] Farid Nourani
[illegible] pela Comissão
Julgador[illegible] 31 . 8 . 92.
Orientador

TRATAMENTO DE RESTRIÇÕES DE CONSISTÊNCIA EM SISTEMAS DE BANCO DE DADOS PARA APLICAÇÕES DE PROJETO

Autor: Farid Nourani
Orientador: Prof. Dr. Léo Pini Magalhães

Dissertação apresentada à Faculdade de Engenharia Elétrica da Universidade Estadual de Campinas como parte dos requisitos para obtenção do título de Mestre em Engenharia Elétrica.

Agosto de 1992

UNICAMP
BIBLIOTECA CENTRAL

"Em verdade, o conhecimento é um autêntico tesouro para o homem e uma fonte de glória, de benção, de contentamento, de exaltação, de alegria e de felicidade."

Bahá'u'lláh

Aos meus pais que me deram a vida, mas principalmente, o adorno da minha alma.

Aos meus filhos Hugo e Anis que contribuíram com a sua quota de sacrifício.

AGRADECIMENTOS

Especialmente, ao Prof. Dr. Léo Pini Magalhães, pela oportunidade que me concedeu de trabalhar consigo, pela amizade e apoio nos momentos mais difíceis e pela orientação na realização deste trabalho.

À minha querida esposa Farideh, pelo apoio incondicional, compartilhando todos os momentos.

Ao Prof. Dr. Odelar Leite Linhares, pela amizade, apoio e exemplo de dedicação.

Aos meus amigos Armando L. N. Delgado e Regina C. Ruschel, pela inestimável colaboração, amizade e compreensão.

Aos meus amigos Marcos Lordello Chaim, Antonio Fernando F. C. Branco e Aleardo Manacero Junior, pela amizade e agradável convivência e pelas discussões interessantes e enriquecedoras.

Aos amigos de Pós-Graduação na UNICAMP, pela amizade e discussões extrovertidas do dia-a-dia, nas quais muito aprendi.

Aos demais amigos que direta ou indiretamente colaboraram para a realização deste trabalho.

RESUMO

Em Ciências de Computação, uma das linhas de pesquisa na área de Sistemas de Banco de Dados que vem se destacando há mais de uma década trata de desenvolver mecanismos adequados às necessidades das aplicações ditas não-convencionais. Em particular, por causa da inadequação e insuficiência dos mecanismos tradicionais de controle de transações e de integridade existentes nos SGBDs convencionais, nos últimos anos, inúmeras propostas têm sido apresentadas para o controle de consistência em aplicações de engenharia.

A proposta deste trabalho é estudar os mecanismos de controle de consistência e de integridade em sistemas de banco de dados, particularmente os de tratamento de restrições em ambientes de projeto. Assim, são identificadas as principais características de restrições de consistência para aplicações de PAC, sendo apresentado um sistema de software (denominado Subsistema de Tratamento de Restrições - STR) para tratamento de restrições no contexto GERPAC/UNICOSMOS, possibilitando definição, manipulação e verificação dinâmica de restrições a nível do GERPAC.

O STR é desenvolvido em ambiente UNIX das estações de trabalho SUN, na linguagem de programação C, constituindo um Subsistema de Tratamento de Restrições para o SGBD GERPAC, um SGBD para aplicações de PAC.

ABSTRACT

One direction of research in Data Base Systems, where a lot of effort has been applied since more than one decade is the development of proper mechanisms for *non-conventional* applications. Particulary, in recent years, innumerous propositions have been presented for consistency control in engeneering applications, due to the inadequancy of traditional mechanisms of consistency, integrity and transaction control in *Conventional Data Base Systems*.

The purpose of this work is to study the consistency and integrity control mechanisms of Data Base Systems, more particulary, in the context of design environments. Major features of consistency constraints for CAD applications are discussed; a software system (STR) for constraints control in the GERPAC/UNICOSMOS context is presented.

This software permits the dynamic definition, manipulation and checking of constraints. STR has been developped in C, in the Laboratório de Computação e Automação (LCA) of FEE/UNICAMP, in a UNIX environment; it is the Constraints Management Subsystem of the GERPAC DBMS (a DBMS for CAD applications).

CONTEÚDO

LISTA DE FIGURAS

"Considerai o homem como uma mina rica em jóias de inestimável valor. A educação, tão somente, pode fazê-la revelar seus tesouros e habilitar a humanidade a tirar dela algum benefício."

Bahá'u'lláh

CAPÍTULO 1

INTRODUÇÃO

A tecnologia de Sistemas de Banco de Dados (SBD) surgiu basicamente com o objetivo de proporcionar aos sistemas computacionais a capacidade de armazenar e gerenciar grandes quantidades de dados de maneira confiável e consistente, permitindo que estes dados sejam compartilhados entre diversos usuários. Inicialmente, foram os programadores das aplicações comerciais que sentiram a necessidade de compartilhamento de Bases de Dados (BD). Este fato explica, em parte, porque a tecnologia de banco de dados evoluiu, primeiramente, em direção aos sistemas de banco de dados orientados para aplicações comerciais e administrativas.

Consequentemente, os sistemas de banco de dados comerciais e/ou administrativos representam a maioria dos SBDs existentes. As estatísticas mostram que somente em 1985 30% das aplicações desenvolvidas utilizaram algum Sistema Gerenciador de Banco de Dados (SGBD) (Bat85). Estes sistemas são normalmente chamados de *sistemas de banco de dados convencionais*.

Nos últimos anos, outras áreas de aplicação (projetos de engenharia, inteligência artificial, automação de escritórios, representação de conhecimento, computação gráfica, etc.) têm recebido a atenção dos pesquisadores da área

de banco de dados. Os SBDs desenvolvidos para estas aplicações são chamados, frequentemente, de *sistemas de banco de dados não-convencionais*.

1.1 PRINCIPAIS OBJETIVOS DOS SBDs

As principais preocupações dos programadores de aplicações comerciais constituiram os objetivos fundamentais dos sistemas de banco de dados. Com o avanço da tecnologia de computadores e dos sistemas de banco de dados e o crescente uso destes sistemas em aplicações denominadas não-convencionais (por serem diferentes da tradicional comercial), os seguintes objetivos fortaleceram o uso de sistemas de banco de dados: controle de redundância; compartilhamento de dados; facilidades de recuperação; imposição de restrições de integridade e controle de segurança de dados.

Outros objetivos adicionais também contribuíram para a difusão da tecnologia de banco de dados. Entre eles podemos mencionar: redução do tempo de desenvolvimento de aplicações; estabelecimento de normas e regulamentos para desenvolvimento de aplicações; disponibilidade de informações sempre atualizadas e economia em recursos humanos.

A seguir descrevemos os principais objetivos que motivaram o surgimento dos sistemas de banco de dados e sustentam a sua utilização no desenvolvimento de aplicações.

1.1.1 Controle de Redundância

Na prática, um item de dado lógico aparece em diversas visões de diferentes usuários. Porém, esta dispersão de dados não é conveniente a nível físico de um sistema de informação. De acordo com a abordadgem de Sistemas de Banco de Dados,

durante o projeto de um sistema aplicativo diversas visões de diferentes grupos de usuários devem ser integradas de tal forma que cada item de dado lógico seja armazenado somente em um único lugar. Isto evita inconsistências e também oferece uma otimização do espaço de armazenamento.

Entretanto, em alguns casos, uma redundância controlada pode ser muito útil. Por exemplo, em sistemas de banco de dados relacionais, quando se deseja criar um relacionamento entre várias relações introduz-se a chave primária de uma relação como chave estrangeira nas demais relações. Desta forma, uma referência ao atributo utilizado é armazenado redundantemente em todos os registros de várias relações. Porém, sem esta facilidade a eficiência na recuperação de informações sofreria uma redução significativa.

1.1.2 Compartilhamento de dados

Uma vez que, nos sistemas de banco de dados, os dados utilizados por diversas aplicações são integrados e mantidos em uma base de dados, os SGBDs devem permitir que vários usuários tenham acesso simultâneo a estes dados. Para isto, um SGBD multiusuário deve oferecer mecanismos de controle de concorrência a fim de garantir que diversos usuários possam atualizar um mesmo dado de maneira controlada e segura para não gerar inconsistências no resultado das atualizações.

Como diversos tipos de usuários, com conhecimento técnico variado, utilizam uma base de dados, os SGBDs devem oferecer, também, uma variedade de interfaces para o usuário. Os tipos de interfaces em geral incluem linguagens de consulta para usuários casuais, linguagem de programação para programadores, sistema de menus e interfaces de linguagens naturais para usuários leigos.

1.1.3 Facilidades de Recuperação

Um SGBD confiável deve oferecer mecanismos de recuperação em casos de ocorrência de falhas de software ou hardware. Por exemplo, se o sistema falha durante a execução de um programa de atualização, o subsistema de recuperação é responsável por assegurar que a base de dados seja restaurada para o seu estado anterior à execução do programa.

1.1.4 Imposição de Restrições de Integridade

A maioria das aplicações de banco de dados exige que certas restricões de integridade sejam impostas sobre os dados. O tipo mais simples de uma restrição de integridade é a especificação do tipo de dado de um item de dado. Existem outros tipos de restrições mais complexas. Por exemplo, especificar que um registro em um arquivo deve ser relacionado com alguns registros em outros arquivos, constitui uma restrição de integridade relativamente complexa.

As restrições de integridade, em sua maioria, são derivadas da semântica dos dados e da realidade que eles representam. É responsabilidade dos projetistas especificar as restrições de integridade durante o projeto do banco de dados. Algumas restrições podem ser especificadas através do SGBD e são impostas automaticamente. Outras restrições devem ser verificadas através dos programas de atualização ou no momento da entrada de dados. Atualmente a maioria dos SGBDs comerciais é relativamente limitada quando se trata de imposição automática de diversos tipos de restrições.

1.1.5. Controle de Segurança

Quando vários usuários compartilham uma base de dados, alguns não são autorizados a acessar todas as informações da base de dados. Na realidade, cada usuário é autorizado a acessar uma determinada porção da base de dados, pré-estabelecida pelo administrador do sistema. Além disso, o tipo de acesso também pode ser restringido (só leitura, só escrita, leitura e escrita). Para tal, os SGBDs devem possuir um subsistema de autorização e segurança que é utilizado pelos administradores de banco de dados para criar e especificar as restrições de segurança e de autorizações. Estas restrições devem ser observadas toda vez que for realizado um acesso a banco de dados. Em caso de violação de alguma restrição o subsistema de segurança do SGBD deve bloquear o acesso e realizar as devidas ações pré-estabelecidas.

1.2 PRINCIPAIS CARACTERÍSTICAS DOS SBDs

Uma base de dados é uma coleção logicamente coerente de dados interrelacionados com significados semânticos inerentes, através dos quais tenta-se representar alguns aspectos do mundo real. Diversas características distinguem os sistemas de banco de dados dos métodos tradicionais de programação com arquivos. A seguir são discutidas algumas entre as mais importantes destas características.

1.2.1 Natureza de Auto-Manutenção

Uma característica fundamental dos sistemas de banco de dados é que eles contém não somente os dados propriamente ditos, mas também uma definição ou descrição completa da estrutura da base de dados. Esta descrição é armazenada num subsistema chamado *Dicionário de Dados* ou *Catálogo de Dados* ([All82], [Wer86]), sendo composta de informações sobre a estrutura de cada arquivo, o tipo e a forma de armazenamento de cada item de dado, além de várias restrições sobre estes dados. Assim, o dicionário de dados é utilizado para manter a coerência da estrutura de dados e facilitar o uso e a manutenção do sistema.

1.2.2 Independência de Dados

A Independência de Dados representa a capacidade dos sistemas de banco de dados de esconder os detalhes da implementação do sistema dos programas de aplicação. Em processamento com sistemas de arquivos tradicionais, a estrutura de dados é embutida nos programas que contém as rotinas de acesso. Desta forma, qualquer mudança na estrutura de um arquivo pode exigir modificações em todos os programas que acessam este arquivo. Ao contrário, as rotinas de acesso de um SGBD são escritas independentemente de qualquer um dos arquivos do sistema. A estrutura dos arquivos de dados é armazenada no dicionário de dados do SGBD, separadamente das rotinas de acesso. Assim, o SGBD permite modificações na base de dados sem provocar grandes consequências ou alterações nos programas de aplicação.

1.2.3 Abstração de Dados

A abordagem da tecnologia de Banco de Dados proporciona ao usuário uma representação conceitual dos dados sem entrar em detalhes de seu armazenamento. Um modelo de dados é uma forma de abstração de dados utilizada para oferecer esta representação conceitual. Os modelos de dados utilizam conceitos lógicos, como por exemplo, objetos e suas propriedades e relacionamentos , os quais podem ser mais fáceis de compreender do que conceitos de armazenamento em computadores.

Em sistemas de banco de dados a estrutura detalhada e a organização de cada arquivo de dados são armazenadas no dicionário de dados. Os usuários interagem com a representação conceitual dos arquivos e o SGBD extrai do dicionário de dados, quando necessário, os detalhes destes arquivos.

1.2.4 Múltiplas Visões de Dados

Um sistema de banco de dados tipicamente tem vários usuários, cada um pode necessitar uma perspectiva ou uma visão diferente da base de dados. Uma visão pode ser um subconjunto da base de dados ou então conter dados virtuais, que são derivados a partir de outros dados existentes nos arquivos e que não estão explicitamente armazenados na base de dados.

1.3 SBDs CONVENCIONAIS VERSUS SBDs NÃO-CONVENCIONAIS

Um sistema de banco de dados deve ser bem projetado a fim de refletir adequadamente a parte da realidade que pretende representar. Isto depende, em parte, do modelo de dados sobre o qual o sistema é projetado. Assim, o modelo de dados utilizado deve ser expressivo o suficiente para capturar a semântica da aplicação. Em geral, os modelos de dados utilizados em SBDs convencionais têm poucos recursos para concretizar este fato ([Tsi82],[Pot88]). Estes modelos apresentam basicamente três elementos primitivos fundamentais:

* Tipos de dados básicos (Inteiro, Lógico, String, etc.) e construtores de tipos compostos (Registro);
* Operadores para criar e manipular instâncias desses tipos (operadores aritméticos, lógicos e de strings);
* Restrições de integridade implícitas (exemplo: chave primária não pode ser nula no Modelo Relacional).

Eventualmente o modelo pode também permitir, ainda que de forma rudimentar, definição de novos operadores em termos de operadores primitivos e de novas restrições de integridade explícitas.

As informações existentes nas aplicações não-convencionais são difíceis de serem capturadas e representadas pelos modelos de dados e SGBDs convencionais pelos seguintes motivos:

* Os SGBDs clássicos são ótimos para representar dados escalares, cadeias curtas de caracteres e registros de tamanho fixo. As aplicações não-convencionais possuem cadeias muito longas de caracteres, registros de tamanho variável ou informações textuais e vetores e matrizes de dados.

* As aplicações não-convencionais são caracterizadas por um grande número de tipos de objetos com poucas ocorrências {Pri89}. Os BDs convencionais têm uma posição oposta.

* Em atividades não-convencionais os esquemas de banco de dados evoluem constantemente.

* Nas aplicações não-convencionais, é necessário preservar as versões antigas e criar novas versões de um mesmo objeto. Em uma base de dados de projetos, por exemplo, o "log" deve ser mantido para poder traçar a evolução de um projeto e possibilitar um eventual retrocesso.

* As transações em SBDs não-convencionais são de longa duração, pois um usuário pode trabalhar sobre um determinado objeto por semanas antes de salvá-lo na base de dados. Enquanto isso, as aplicações convencionais são caracterizadas por suas transações curtas, tanto em sistemas "batch" como em sistemas "on-line". Enquanto nos sistemas "batch" uma transação pode processar grandes volumes de dados, nos sistemas "on-line" em geral manipula somente poucos dados. Porém, em qualquer caso, a execução de uma transação não demora mais do que poucos segundos. Além disso, nestas aplicações, as transações representam um grau elevado de acesso local aos dados, isto é, a maioria delas acessa praticamente a mesma porção da base de dados.

* Enquanto as aplicações convencionais são geralmente caracterizadas por acessos altamente localizados (dados comuns são acessados frequentemente por diversos usuários), em aplicações não-convencionais a base de dados é particionada entre vários grupos de usuários. Consequentemente, os conflitos em acessar os dados são muito mais frequentes em sistemas de banco de dados convencionais.

Modelos de dados utilizados para projetar bases de dados de aplicações não-convencionais devem ter capacidade de representar a maior parte da semântica dessas aplicações ({Pec88},{Pot88}), permitindo definição e manipulação de inúmeros tipos de objetos complexos.

Embora a tecnologia de SBDs convencionais represente uma solução adequada para o problema de armazenamento e gerenciamento confiável e seguro de grandes quantidades de dados, para as aplicações não-convencionais ela não cobre adequadamente os outros requisitos, as vezes mais importantes. Nestes sistemas, há uma maior necessidade em mecanismos que proporcionam definição e verificação de maior número de tipos de restrições. Além disso, o suporte a estas restrições deve ser de forma mais flexível e mais dinâmica do que nos SGBDs convencionais.

Esta necessidade tem motivado o desenvolvimento de diversos projetos de pesquisa, principalmente em sistemas de base de conhecimento e também em aplicações de projeto. Entre as tentativas de integrar aos SGBDs não-convencionais mecanismos de definição e de controle de restrições podemos mencionar, entre outras, as seguintes: CONMAN {Wal89} - um gerenciador de restrições que suporta uma representação consistente e um gerenciamento eficiente de restrições em sistemas de base de conhecimento; Beta {Bro80} - uma linguagem para especificação de esquemas para sistemas de banco de dados, com facilidades de definição de restrições de integridade; System G {Che86} - um sistema de programação lógica orientado a objetos com recursos para gerenciamento de regras; ADABTPL {Cro87} - um modelo de dados baseado em definição de tipos de restrições para aplicações em Automação de Escritórios; KRISYS ({Deb89b}, {Mat88}) - um Sistema Gerenciador de Bases de Conhecimento (KBMS), baseado na integração das tecnologias de SBDs e IA, com recursos adequados para sistemas avançados de PAC; o subsistema gerenciador de regras do SGBD POSTGRES, sucessor do SGBD relacional INGRESS ({Sto88},{Sto86}); ETM (Event/Trigger Mechanism) - um subsistema genérico de definição e controle de restrições e gatilhos para o sistema DAMASCUS, projetado para aplicações de projetos de VLSIs ({Kot88},{Dit86a}); MARVEL ({Kai88a},{Kai88b} - um ambiente de bases de conhecimento para aplicações de engenharia, que suporta diversas entidades e ferramentas de forma integrada, baseado em conceito de restrições do tipo "pré-condicão, ação, pós-condição".

1.4 OBJETIVOS DESTE TRABALHO

Em ambientes de Projeto Auxiliado por Computador (PAC), a necessidade de um SGBD é uma realidade indiscutível. Nestes ambientes, devido a intensa interação de diversos usuários, o SGBD requer recursos flexíveis de controle da consistência e coerência de dados. Estes recursos não se resumem, apenas, em mecanismos convencionais de controle de consistência existentes na maioria dos sistemas de banco de dados comerciais, tais como, recuperação e backup, definição de transações e controle de concorrência. Definição dinâmica e controle simplificado de restrições de integridade são mecanismos adicionais que devem ser acrescidos aos mecanismos tradicionais de controle de consistência, nos SGBDs destinados às aplicações de PAC.

A intenção de proporcionar ao SGBD GERPAC (ver Capítulo 4) mecanismos de gerenciamento de restrições de forma genérica, foi uma das preocupações dos idealizadores e implementadores deste sistema. Desde o início do seu desenvolvimento, foram criados mecanismos e estruturas que podessem facilitar a realização desta idéia.

O objetivo principal do presente trabalho foi especificar e implementar mecanismos para a definição e o gerenciamento de alguns tipos específicos de restrições de integridade no contexto GERPAC/UNICOSMOS. Consequentemente, foram estudadas as técnicas de controle da consistência existentes em SGBDs convencionais e também os mecanismos propostos por alguns protótipos de SGBDs não-convencionais. Além disso, foi feito um estudo das restrições de consistência e suas principais características em ambientes PAC.

1.5 ORGANIZAÇÃO DA DISSERTAÇÃO

Os próximos capítulos deste trabalho são organizados da seguinte forma. No capítulo 2 são descritas as questões de integridade em sistemas de banco de dados convencionais. Assim, primeiramente são analisados os diversos tipos de restrições de integridade. Em seguida são abordados diversos mecanismos de controle de consistência geralmente existentes em sistemas de banco de dados, tais como, recuperação e backup, controle de concorrência e segurança de dados.

O capítulo 3 apresenta um estudo dos ambientes de PAC e suas necessidades em termos de controle de restrições. Neste capítulo, também são descritas algumas características principais de alguns protótipos de SGBDs para aplicações não-convencionais, e os recursos de gerenciamento de restrições propostos por eles.

No capítulo 4, inicialmente são apresentados o SGBD GERPAC e o seu núcleo UniCOSMOS. Em seguida, é descrito o STR (Sistema de Tratamento de Restrições) que constitui o núcleo desta dissertação. Finalmente, no capítulo 5 são relacionadas as conclusões e alguns comentários sobre este trabalho. O capítulo 6 apresenta uma extensa bibliografia que serviu de referência ao presente trabalho.

"Tão poderosa é a luz da Unidade que pode iluminar a Terra inteira."

Bahá'u'lláh

CAPÍTULO 2

INTEGRIDADE EM SISTEMAS DE BANCO DE DADOS

2.1 INTRODUÇÃO

Neste capítulo são abordados os principais aspectos da questão de *integridade* nos Sistemas Gerenciadores de Bancos de Dados (SGBD). Em particular, são analisados alguns mecanismos fornecidos pelos SGBDs para garantir a consistência dos dados armazenados no Sistema.

A fim de manter a integridade de um sistema de banco de dados é necessário definir um conjunto de *restrições de integridade*. Estas *restrições de integridade* geralmente residem em algum lugar do sistema e são utilizadas para verificar, a qualquer momento, se alguma operação do sistema compromete a sua integridade [Hon81].

Em geral, os sistemas que oferecem algum grau de controle de integridade, fornecem basicamente uma simples proteção contra interferência, além de uma validação rudimentar das transações individuais [Dat83].

O primeiro aspecto a ser analisado, neste capítulo, é o de como propiciar uma maior garantia da consistência das informações, armazenadas na base de dados, no que diz respeito à sua semântica. Por exemplo, a idade de uma pessoa deve ter valores inteiros no intervalo de 0 a 120 ou, então, o nível de estoque

de um determinado produto não pode ter um valor negativo. Neste caso estamos falando de **restrições de integridade**.

O segundo aspecto é o da **recuperação de falhas** em Sistemas de Banco de Dados. Estes sistemas precisam incorporar não apenas várias verificações e controles para reduzir a probabilidade de ocorrência de falhas mas, também, e mais significativamente, um extenso conjunto de procedimentos para a recuperação destas falhas que inevitavelmente irão ocorrer {Dat83}. Mais adiante, observaremos que *transações* constituem um mecanismo essencial para possibilitar o processo da recuperação.

Em terceiro lugar é analisada, de forma sucinta, a questão de **controle da concorrência**. Nos sistemas multiusuários, um SGBD deve garantir que as ações de um usuário não tenham influências ou interferências nas ações de outro.

Finalmente o quarto aspecto da integridade de dados diz respeito à **segurança** do sistema. As informações devem ser protegidas contra acessos não autorizados ou contra tentativas de destruição de dados, sejam parciais ou totais.

2.2 RESTRIÇÕES DE INTEGRIDADE

Independentemente do modelo de dados no qual um SGBD é baseado, as informações armazenadas na base de dados devem ser consistentes. Isto significa que elas sempre devem estar em conformidade com a realidade supostamente representada pelo sistema.

Uma restrição de integridade é geralmente um predicado que deve ser verificado por um subsistema do SGBD a fim de que a base de dados possa ser considerada consistente. Este *subsistema de integridade* deverá ter as seguintes responsabilidades:

* **Monitoração da integridade da base de dados.** Se a base de dados e o ambiente do mundo real que a modelou fossem estáticos, então uma vez que fosse atingido um estado consistente e íntegro, sempre se manteria a sua integridade. Porém, os ambientes do mundo real são sujeitos a mudanças e os sistemas devem acompanhar estas mudanças. Cada mudança pode violar a integridade da base de dados. Na maioria dos sistemas de banco de dados esta responsabilidade se reduz a uma simples monitoração das transações e, mais especificamente, às operações de atualização {Dat83}.

* **Tomar uma ação apropriada no caso de violação de uma restrição.** Por exemplo, rejeitar a operação, informar a violação ou até mesmo corrigir a anormalidade.

2.2.1 Representação das Restrições de Integridade

Uma restrição de integridade consiste basicamente de três partes: um *evento*, um *predicado* e uma *ação* {McC89}. O evento, também chamado de *condição-gatilho* {Dat83}, é o que dispara a verificação da restrição. O predicado é uma coleção de requisitos que são avaliados quando a restrição é disparada. A ação é uma série de instruções que são executadas quando a restrição é disparada e o predicado é satisfeito {McC89}. Para uma restrição de integridade, em {Dat83} é sugerida uma estrutura que contém uma cláusula AFTER ou BEFORE, determinando o evento, um predicado e uma ação contendo uma cláusula ELSE. Assim, podemos representar uma restrição de integridade da seguinte forma:

```
RI1 : AFTER UPDATING S.STATUS :
            S.STATUS > 0
      ELSE
            DO
                set return to "rule RI1 violated";
                REJECT;
            END;
```

A cláusula AFTER indica que a verificação da restrição deve ser feita após a atualização do campo S.STATUS. O predicado **S.STATUS > 0** indica a natureza da verificação, ou seja, a restrição, e sugere que o novo valor de S.STATUS deve ser maior do que zero. A cláusula ELSE, por sua vez, indica o que deve ser feito se a verificação falhar, no caso rejeitar a atualização, retornando uma mensagem.

2.2.2 Classificação das Restrições de Integridade

As restrições de integridade podem ser classificadas em diversas categorias segundo critérios diferentes. Do ponto de vista funcional, as restrições de integridade podem ser classificadas em restrições **Estruturais** e restrições **Comportamentais** (Gar89). Além disso, as restrições de integridade podem ser consideradas **Estáticas** ou **Dinâmicas** (Del85), **Imediatas** ou **Adiadas** ((Del85), (Dat83),(Laf82)), e finalmente restrições de **Domínio** ou restrições de **Relação** (Dat83). A seguir é descrita cada uma destas categorias.

2.2.2.1 Restrições de Integridade Estrutural

Do ponto de vista funcional, as restrições de integridade estrutural são aquelas inerentes aos modelos de dados, que expressam propriedades semânticas específicas para estruturas básicas do sistema, utilizando o modelo de dados. Uma restrição estrutural é geralmente especificada como uma parte do esquema, na definição da base de dados (Gar89).

Os seguintes tipos de restrições, existentes nos sistemas relacionais, são exemplos de restrições de integridade estrutural: restrições de chave única;

restrições de integridade referencial; restrições de domínio e restrições sobre "valores nulos" {Gar89}.

2.2.2.2 Restrições de Integridade Comportamental

Estas, são as restrições que modelam as propriedades semânticas da base de dados regulando o comportamento das aplicações. Uma restrição de integridade comportamental é geralmente introduzida no esquema como uma parte adicional ao mesmo {Gar89}.

No modelo relacional, podem ser introduzidas diversas formas de restrições de integridade comportamentais. Elas podem expressar relacionamentos internos a uma relação {Gar89}, tais como:

* **Dependências funcionais**
  Uma dependência funcional existe quando cada valor de um atributo *B* é determinado pelo valor de um atributo *A*, se este não for um atributo composto, e pelo valor do conjunto inteiro dos componentes de *A* (e não pelo valor de um subconjunto de *A*), se este for um atributo composto. Com uma dependência funcional existe uma relação *um-para-um* entre o atributo determinante e o conjunto de atributos dependentes.

* **Dependências multivaloradas**
  Uma dependência multivalorada é aquela na qual um atributo pode determinar mais de um valor para outro atributo. Neste caso, não existe uma relação de *um-para-um* entre o atributo determinante e o conjunto de atributos dependentes {Smi87}.

* **Dependências de agregação**
  São aquelas que restringem os possíveis valores de uma função agregada sobre os atributos em uma relação. Por exemplo, a soma dos emprestimos de um determinado cliente, em um sistema bancário, não deve exceder 1.000.000.

As restrições de integridade comportamental podem também definir relacionamentos entre as relações, tais como (Gar89):

* **Dependências de Inclusão**
  Estas restrições especificam que o valor de um ou mais atributos de uma relação deve aparecer em uma outra relação. Por exemplo, cada valor de NUM.DEPTO na relação de EMPREGADOS deve aparecer em alguma tupla da relação de DEPARTAMENTOS.

* **Dependências Equacionais**
  Estas, são aquelas que impõem a igualdade ou a desigualdade de duas expressões aritméticas, a partir de valores dos atributos ou de funções sobre os atributos das relações. Por exemplo, a quantidade disponível de um determinado item, em um sistema de controle de estoque, sempre deve ser igual a quantidade de aquisição menos a quantidade vendida daquele item.

2.2.2.3 Restrições Estáticas

Restrições estáticas são aquelas que verificam a validade de um determinado estado da base de dados, independentemente de todos os demais estados. Uma restrição estática descrita por uma linguagem simbólica seria:

RI1 : VERIFYON SOCIOS
SOCIOS.IDADE > 0

Isto significa que na relação SOCIOS, o valor do atributo IDADE deve ser positivo. Estas restrições são chamadas, também, de *restrições de estado*.

2.2.2.4 Restrições Dinâmicas

Restrições dinâmicas ou *restrições de transição*, ao contrário das restrições estáticas, correspondem à verificação da transformação de um estado

da base de dados em outro. Ou seja, são restrições que envolvem a comparação de estados da base de dados, antes e depois de um certo evento. Por exemplo, a restrição:

RI2 : BEFORE UPDATING S.STATUS FROM NEW-STATUS :
NEW-STATUS > S.STATUS

indica que sempre, na ocasião da atualização do atributo S.STATUS, o seu novo valor deve ser maior do que o valor anterior. Isto envolve a comparação dos estados anterior e posterior do atributo, na ocasião da atualização.

2.2.2.5 Restrições Imediatas

Quando a verificação de uma restrição é exigida no momento de sua declaração e, posteriormente, no momento de cada alteração dos objetos envolvidos nesta restrição, ela é chamada de restrição imediata. As restrições RI1 e RI2, citadas anteriormente como exemplos, são consideradas restrições imediatas.

2.2.2.6 Restrições Adiadas

Em contraste com as restrições imediatas, esta categoria de restrição tem a sua verificação adiada para um momento posterior à sua declaração. Como exemplo podemos citar uma transação de transferência de valores em um sistema bancário, onde para cada agência há um campo TOTAL que expressa a soma de saldos de todas as contas daquela agência. Isto pode ser representado através da seguinte restrição:

RI3 : SET SUM (CLIENTE.SALDO) = UNIQUE (AGENCIA.TOTAL)

Agora considere uma transação de transferência, que transfere uma quantia em valores de uma conta para outra e assim atualiza dois saldos distintos.

Claramente, se a restrição acima (RI3) for satisfeita antes da transação ser executada, então ela ainda será satisfeita após a execução da transação. Porém, durante a transação a restrição é temporariamente violada. Como não é desejável que a restrição RI3 seja verificada depois da primeira alteração (debitar a quantia da primeira conta) e antes da segunda alteração (creditar a quantia na segunda conta), a condição-gatilho deverá ser adiada para um momento posterior, ou seja, para o momento do término bem sucedido da transação.

Uma restrição adiada, então, é uma restrição de integridade para a qual a condição-gatilho é especificada como sendo o término, bem sucedido, da transação (WHEN COMMITING) {Dat83}. Estas restrições não possuem parâmetros explícitos, sendo aplicadas como parte do processamento do comando COMMIT para as transações que modificam qualquer variável capaz de afetar a avaliação da restrição {Dat83}.

2.2.2.7 Restrições de Domínio

A definição de um domínio *D*, que explícita ou implicitamente determina todos os valores em *D*, constitui uma restrição de integridade simples, mas importante {Dat83}. Esta restrição deve ser verificada em todas as inserções ou atualizações. Assim, a restrição de domínio, para um determinado domínio, nada mais é do que a definição daquele domínio {Ham75} {Dat83}. As condições-gatilho das restrições de domínio não aparecem na declaração da restrição, pois sempre são as mesmas, a saber: atualização de um campo ou a inserção de um registro que contém um campo definido no domínio em questão.

2.2.2.8 Restrições de Relação

As restrições de relação preocupam-se em admitir ou não uma determinada tupla como candidata à inserção em uma dada relação. Além disso, elas podem verificar um eventual relacionamento existente entre tuplas de várias relações.

2.2.3 Exemplos de Restrições de Integridade

Na maioria dos sistemas de banco de dados atuais, podemos encontrar diversos tipos de restrições de integridade que, de certa forma, podem ser classificadas em alguma das categorias descritas anteriormente. A seguir mencionaremos algumas destas restrições, tais como: restrições de *chave única*, restrições de integridade *individuais*, restrições de integridade de *conjunto*, restrições *intra-relação* e restrições *inter-relações*.

2.2.3.1 Restrições de Chave Única

Uma das restrições estáticas mais importantes é a restrição que se preocupa com as chaves de uma relação. Um atributo ou um conjunto de atributos pode ser utilizado para qualificar univocamente uma tupla dentro de uma relação. Neste caso o SGBD fornece mecanismos para a declaração das chaves de uma relação e a cada inserção verifica se o valor da chave é único dentro da relação.

2.2.3.2 Restrições de Integridade Individual

Estas são restrições que devem ser verificadas a nível de cada tupla, individualmente. Por isso, são chamadas, também, de restrições de integridade de registros. Por exemplo, uma restrição entre atributos de uma mesma tupla como:

$$\text{QTDE-ESTOQE} \geq \text{QTDE-PEDIDO}$$

é considerada uma restrição de integridade individual. Este grupo de restrições talvez sejam as restrições mais comuns e mais simples a definir e a verificar em sistemas de banco de dados [Del85].

2.2.3.3 Restrições de Integridade de Conjunto

Este grupo de restrições envolve um conjunto inteiro de registros, ao invés de apenas um registro particular. A seguinte restrição escrita em SQL é um exemplo de uma restrição de integridade de conjunto:

```
ASSERT R15 ON EMPREGADOS :
        100.000 > (SELECT AVG (SALARIO)
                   FROM EMPREGADOS
                   WHERE DEPTO.NOME = "PRODUÇÃO")
```

Esta restrição indica que o salário médio do pessoal do departamento de PRODUÇÃO, da relação de EMPREGADOS, não deve ser superior a 100.000.

2.2.3.4 Restrições de Integridade Intra-Relações

No caso das restrições de integridade de registros não há referências a valores armazenados em outros registros da relação. Porém, isto é geralmente útil para controlar os valores de atributos de uma tupla em relação aos valores do mesmo atributo em outras tuplas da relação. Isto constitue uma restrição intra-relação *vertical* (Del85). As vezes é desejável controlar o valor de um atributo em relação aos valores de outros atributos da mesma tupla. Isto pode constituir uma restrição intra-relação *horizontal* (Del85).

2.2.3.5 Restrições de Integridade Inter-Relações

Até agora, a maioria das restrições descrita neste capítulo atuava em uma única relação. Porém, geralmente um sistema de banco de dados relacional é constituído de diversas relações. Portanto, deve haver meios de estabelecer

regras entre estas relações para manter a integridade das informações inter-relacionadas, armazenadas em diversas relações distintas.

Estas restrições, conhecidas também como restrições de *integridade referencial*, são casos especiais e importantes de integridade de sistemas de banco de dados relacionais. As restrições de integridade inter-relações geralmente se fazem necessárias quando uma relação faz referências a outra {Dat86}.

Esta abordagem é baseada no conceito de *chave estrangeira* do modelo relacional {Dat86}. Um atributo A de uma relação R_1 é considerado *chave estrangeira* se ele é definido em um domínio primário D, e em qualquer momento cada valor de A em R_1 é *nulo* ou igual a V, onde V é o valor da chave primária de alguma tupla em alguma relação R_2 (R_1 e R_2 não necessáriamente distintas), com chave primária definida em D {Dat77}. Desta forma, uma relação como a R_1 é considerada uma *relação de referência* e uma relação como a R_2 é considerada uma *relação referenciada*. O atributo $R_1.A$, por sua vez, é considerado um *atributo referencial* {Dat86}.

2.2.4 Restrições de Integridade em alguns Sistemas Existentes

2.2.4.1 Sistema R

Tanto a Linguagem de Manipulação de Dados (DML) como a de Definição de Dados (DDL) do Sistema R são baseadas nos comandos da linguagem SQL. Em termos de restrições de integridade este sistema basicamente oferece os seguintes recursos ({Dat77},{Dat83}):

* Restrições de domínio através da verificação de tipo de dados.

* Permite recusar valores "nulos" para determinados campos. Neste caso no comando CREATE TABLE será inserida a cláusula opcional NONULL, a nível de cada campo.

* Garantia de unicidade para determinados campos ou combinações de campos, para definição de chaves, através da criação de um índice utilizando a cláusula UNIQUE.

exemplo:

```
SELECT UNIQUE F.NOME
FROM F,FP
WHERE F.#F AND
      FP.#P = "P2"
```

OBS: P representa a relação de Peças, F a relação de Fornecedores e FP a relação de Fornecimento.

Porém, os recursos mais importantes da SQL, em termos de controle da integridade não foram implementados no Sistema R. Estes recursos incluem (Dat83):

* Definição de *assertivas*, através do comando ASSERT, que de certa forma permite a especificação de todos os tipos de restrições de integridade e seus eventos correspondentes. Para a eliminação de uma assertiva usa-se o comando DROP ASSERTION.

* Declaração de *gatilhos*, através do comando DEFINE TRIGGER. Os eventos relacionados a estes gatilhos são restritos às clausulas dos comandos SELECT, INSERT, UPDATE e DELETE. O comando DROP TRIGGER elimina um determinado gatilho.

* As restrições do tipo adiadas podem ser verificadas, a qualquer momento, pelo usuário, através do comando ENFORCE INTEGRITY.

2.2.4.2 Query By Example

A linguagem Query By Example oferece os seguintes tipos de restrições de integridade ({Ull82},{Dat83}):

* Uma forma limitada de suporte aos domínios, através de declaração e verificação de tipos de dados.

* Suporte para as chaves primárias.

* Criação de uma tabela de restrições para cada relação, através da instrução:

I.CONSTR (<condition-list>).I.

Na instrução acima, o primeiro **I.** corresponde a inserção da própria restrição na tabela e o segundo **I.** corresponde a inserção das partes da restrição que seguem na mesma linha da tabela. A lista de condições pode conter qualquer um ou todos os caracteres **I.**, **U.** e **D.**, os quais definem as condições-gatilho de cada restrição, a saber: inserção (Insert), atualização (Update) e eliminação (Delete), respectivamente.

Exemplo:

Imagine que queremos estabelecer, em um sistema bancário, onde temos a relação CLIENTES (nome, endereço, saldo), a seguinte restrição: "Nenhum cliente pode ter um saldo negativo superior a 10.000".

Esta restrição pode ser representada da seguinte maneira:

CLIENTES	nome	endereço	saldo
I.CONSTR (I.,U.).I.			≤ - 10.000

Uma entrada D.CONSTR permite eliminar (Delete) uma restrição e uma entrada P.CONSTR permite imprimir (print) uma restrição.

2.2.4.3 INGRES

O sistema INGRES permite que o usuário defina certas restrições em uma relação (Smi87). Como por exemplo:

```
DEFINE INTEGRITY ON EMPLOYEE IS EMPLOYEE.SALARY < 100.000
```

Esta restrição indica que o valor do atributo SALARY da relação EMPLOYEE deve ser menor que 100.000. Entretanto, o INGRES não oferece mais do que limitados recursos de suporte a restrições de integridade. Por exemplo, este sistema não suporta explicitamente o conceito de chaves primárias nem a construção de domínios, exceto os "domínios de base" (integer, float e char). Além disso, o INGRES não suporta restrições de transição ou restrições adiadas, nem permite a especificação de condições-gatilho (Date 83).

Porém, o INGRES possui uma técnica interessante para a implementação das restrições de integridade. Esta técnica resume-se na modificação de todas as transações do usuário que envolvem a atualização dos dados (Dat83). Por exemplo, a instrução:

```
REPLACE EMPLOYEE (SALARY = EMPLOYEE.SALARY * 1,1)
```

será modificada, pelo sistema, para:

```
REPLACE EMPLOYEE (SALARY = EMPLOYEE.SALARY * 1,1)
        WHERE EMPLOYE.SALARY * 1,1 < 100.000
```

o que garantirá que a restrição acima especificada sempre seja obedecida (Smi87).

2.2.4.4 DBTG

As restrições de integridade fazem parte das instruções da Linguagem de Definição de Dados do DBTG {Dat83}. Consequentemente, não há muita flexibilidade nas suas declarações. O DBTG oferece basicamente quatro classes de restrição {Dat83}:

* Suporte a definição de vários tipos de dados, embora não exista a noção de domínio.

* Restrições de chave única, declaradas através da cláusula DUPLICATES ARE NOT ALLOWED, tanto dentro de uma descrição de registro, onde pode definir uma chave candidata, como dentro de um *SET*.

* Conceitos de "classes de inserção" (MANUALL ou AUTOMATIC) e de "classes de retenção" (OPTIONAL ou MANDATORY ou FIXED). Estas cláusulas são declaradas na ocasião da definição do tipo do registro ou do *set*, e influenciam os procedimentos de inserção e de remoção de elementos nestas estruturas.

* O DBTG fornece uma cláusula CHEK para a especificação de restrições de integridade individuais e certas restrições referenciais.

Exemplos:

```
1)RECORD NAME IS S
  :
  :
  STATUS ... CHEK IS STATUS > 0

2)SET NAME IS S-SP
  OWNER IS S
  MEMBER IS P
  :
  :
  CHEK IS #S IN SP = #S IN S
```

2.3 RECUPERAÇÃO DE FALHAS

Todos os sistemas de computação em geral e os sistemas de banco de dados em particular são sujeitos a falhas. Estas falhas podem ser provocadas pelos próprios sistemas, pelos sistemas operacionais, pelo "hardware" ou por um evento externo ao sistema. O desafio, principalmente para os sistemas de banco de dados, é, portanto, reduzir a probabilidade da ocorrência de falhas e, mais importante, possibilitar a recuperação dos dados quando ocorre uma falha {Dat83}.

Assim, os sistemas de banco de dados, por um lado, devem oferecer mecanismos de verificação e controle para reduzir as falhas e, por outro lado, fornecer procedimentos para recuperar estas falhas. Em geral, a redução de falhas pode envolver questões como controle de qualidade, tanto de "hardware" como de "software", controles físicos e até mesmo questões de política de operação.

Porém, a questão de recuperação constitui um esquema integrante dos Sistemas Gerenciadores de Banco de Dados, sendo responsável pela deteção de falhas e a restauração do sistema para um estado consistente que existia antes da ocorrência da falha {Kor86}.

Independentemente da natureza da falha, os princípios em que se baseia a recuperação de falhas são simples e podem se resumir em uma única palavra: **redundância**. Isto significa que, a recuperação consiste em assegurar que qualquer informação, existente no banco de dados, possa ser reconstruída a partir de alguma outra informação armazenada redundantemente em algum outro lugar do sistema {Dat83}.

Existem basicamente duas técnicas de manter réplicas dos dados de um sistema, a fim de possibilitar a recuperação dos mesmos após a ocorrência de uma falha:

1. Periodicamente, é feita uma cópia de todo o banco de dados, que normalmente reside em fitas magnéticas. Esta técnica é conhecida como a técnica de *back-up* e a periodicidade de sua execução depende essencialmente da natureza do sistema. Ela é utilizada geralmente para a recuperação de falhas provocadas pelo "hardware" que danifiquem todo o banco de dados, ou comprometam a validade de suas informações. Neste caso, carrega-se a cópia mais recente do sistema e tenta-se recuperar um estado consistente mais recente, utilizando os arquivos de *log* (arquivos que guardam o histórico de todas as transações feitas na base de dados).

2. Toda vez que se fizer uma alteração no estado do banco de dados, grava-se um registro em um arquivo especial denominado *log*, com todos os detalhes da alteração, inclusive os valores antigos dos itens alterados. Este arquivo é utilizado pelo sistema, depois da ocorrência de uma falha, para desfazer e/ou refazer as atualizações feitas no banco de dados até o momento da falha.

Neste trabalho focalizamos a segunda técnica, por ser uma tarefa, quase que exclusivamente, do SGBD e por conter diversos procedimentos interessantes que fazem parte dos recursos oferecidos por quase todos os SGBDs.

2.3.1 Transações

A fim de possibilitar o controle das atualizações nos sistemas de banco de dados, foi introduzido o conceito de *transação*. Uma transação é uma unidade de programa (um conjunto de operações) cuja execução preserva a consistência do banco de dados (Kor86). Uma transação acessa e possivelmente atualiza diversos itens de dados. Cada um destes itens é lido, pela transação, precisamente uma

única vez e é, possivelmente, gravado também uma única vez, se a transação atualizar aquele item de dado.

Uma transação é definida pelo programador. Ou seja, é o programador quem define onde começa e onde termina uma transação e quais são as ações que ela realiza. Em outras palavras, a transação é uma unidade de programa, escrita pelo programador dentro do seu programa aplicativo. Logo, é responsabilidade do programador definir apropriadamente as diversas transações de modo que cada uma preserve a consistência do banco de dados (Kor86).

2.3.1.1 Estrutura de uma transação

Uma transação consiste na execução de uma sequência de operações especificadas pela aplicação, começando com uma operação especial BEGIN TRANSACTION e terminando com uma operação COMMIT TRANSACTION ou com uma operação ROLLBACK TRANSACTION.

A operação BEGIN TRANSACTION serve como uma mensagem para notificar o começo da transação ao SGBD, para que este dê início aos procedimentos de controle das atualizações feitas durante a transação. O COMMIT TRANSACTION é utilizado para sinalizar o término bem-sucedido de uma transação, o que significa que todas as instruções e as ações da transação foram executadas com sucesso. A instrução ROLLBACK TRANSACTION é usada para indicar o término mal-sucedido de uma transação. Esta instrução pode ser usada pelo programador, quando ele prevê a ocorrência de uma situação exepcional.

O próprio SGBD, também pode executar um ROLLBACK TRANSACTION quando detecta a ocorrência de uma falha. A Figura 2.1 mostra a estrutura de uma transação e a Figura 2.2 mostra a execução de um programa aplicativo com as transações nele contidas.

```
.
.
.
BEGIN-TRANSACTION
   read
   .
   .
   write
   .
   .
   if A not exist
         then ROLLBACK-TRANSACTION
   else  .
         .
         .
   endif
COMMIT-TRANSACTION
.
.
.
```

Figura 2.1 - A estrutura de uma transação.

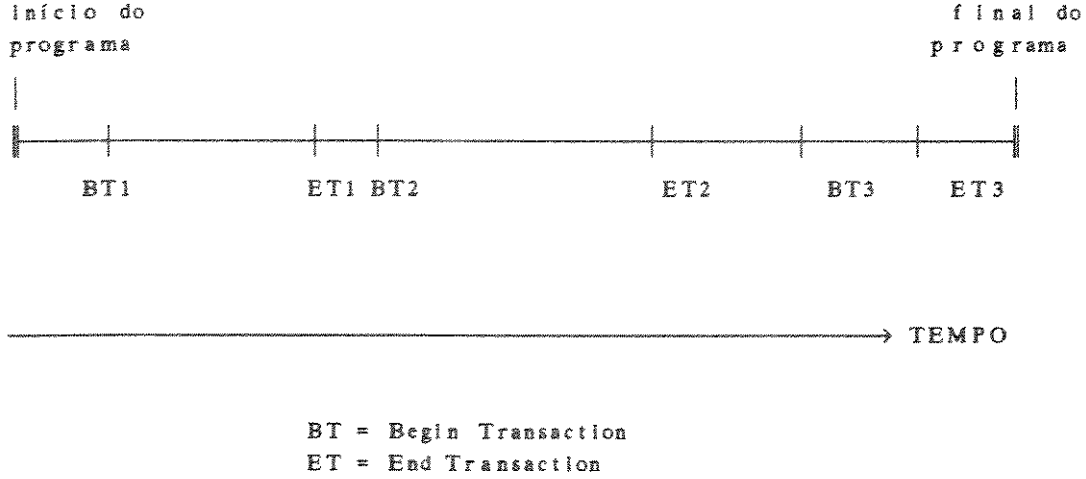

Figura 2.2 - Execução de transações durante um programa.

2.3.1.2 Propriedades de uma transação

Como já foi visto, uma transação é uma sequência de operações que quando executadas não afetam a integridade do banco de dados, mantendo-o em um estado consistente (Gra81). Esta característica é baseada nas seguintes propriedades:

a) Uma transação, do ponto de vista do usuário, é uma operação *atômica*. Ou seja, as transações são proposições do tipo *tudo-ou-nada* (Dat83). Isto significa que quando uma transação é executada, todas as instruções nela contidas devem ser executadas com sucesso ou, então, nada é executado de modo algum. Além disso, os resultados de uma transação, executada corretamente, devem sobreviver às falhas do sistema.

b) Se a execução de uma transação for iniciada, então ela é de fato executada exatamente uma vez. Sabe-se que, devido às falhas do sistema, há situações onde as transações devem ser refeitas, até mesmo várias vezes. Contudo, o efeito líquido deve ser, sempre, como se elas tivessem sido executadas apenas uma única vez. Isto implica que quaisquer atualizações produzidas por uma transação devem ser aplicadas no banco de dados exatamente uma vez, e que quaisquer mensagens de saída, produzidas pela transação, devem ser transmitidas apenas uma vez (Dat83). As transações não devem ser perdidas, ou feitas parcialmente, ou então feitas mais de uma vez.

c) Cada transação deve ser suficientemente isolada das outras. Isto significa que, uma transação não pode fornecer resultados parciais a outras transações, quando executadas concorrentemente (Cas85). Esta propriedade é bastante importante para os sistemas distribuídos e/ou multiusuários que, além do mais, exigem a *serialização* das transações. Um conjunto de transações é dito como serializável se, e somente se, o resultado da execução intercalada dessas transações, em um determinado estado do banco de dados, for o mesmo da execução seriada das mesmas transações, operando no mesmo estado inicial do banco de dados (Dat83).

2.3.1.3 Ciclo de vida de uma transação

Quando um programa de aplicação executa uma instrução de *begin-transaction*, esta sinaliza o início de uma nova ocorrência da referida transação, a nível do SGBD. Nesta hora, é criado um identificador interno e alocado um espaço de trabalho associado à transação [Del85]. Uma série de mecanismos de controle também é ativada. A partir deste momento, no *ciclo de vida* desta particular ocorrência da transação, pode ocorrer uma das seguintes situações:

* **A transação é completada com sucesso**

  Neste caso, a transação executa com sucesso todas as operações pertinentes a ela, processando o *commit-transaction*.

* **A transação é interrompida**

  Esta situação pode ocorrer quando a execução de uma transação em andamento é interrompida por algum evento externo. Isto pode acontecer por causa de uma falha do sistema ou de uma decisão voluntária do próprio SGBD. Esta última pode ocorrer, por exemplo, em casos onde surge um *deadlock* entre várias transações.

* **A transação é cancelada**

  Este é o caso em que a transação contém uma instrução *rollback-transaction*, prevista pelo programador e então ocorre uma condição desfavorável que causa a execução desta instrução.

A Figura 2.3 mostra as três possibilidades de execução durante o ciclo de vida de uma ocorrência de uma determinada transação.

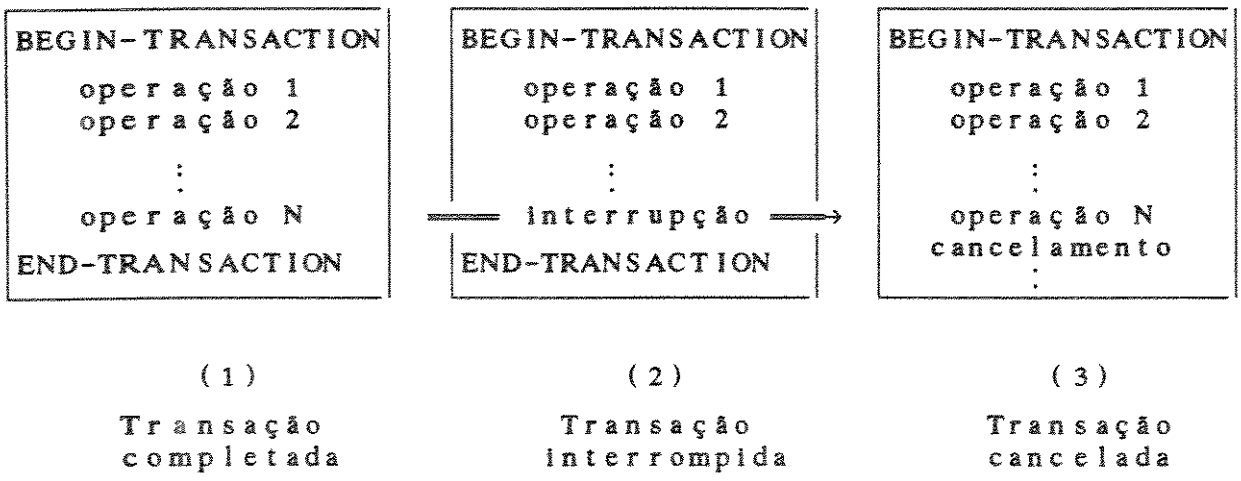

Figura 2.3 - Três possibilidades para execução de uma transação

2.3.2 Log com protocolo de atualizações imediatas

A fim de possibilitar a recuperação dos dados de uma eventual falha, todas as modificações realizadas no banco de dados através de uma transação são registradas num arquivo especial chamado *log*. Em caso da ocorrência de uma falha, este arquivo é utilizado para desfazer e/ou refazer as operações de todas as transações efetuadas anteriormente, tantas vezes quantas forem necessárias, para atingir um estado consistente, no qual se encotrava o banco de dados antes da ocorrência da falha (Kor86).

Uma maneira de tornar as modificações, provocadas pelas transações, efetivas é aplicar todas as atualizações diretamente no banco de dados e manter um "log" incremental de todas as modificações no estado do sistema (Kor86). Desta forma, torna-se possível desfazer todas as atualizações necessárias que foram efetuadas no banco de dados pelas transações.

2.3.3 Log com protocolo de atualizações adiadas

Durante a execução de uma transação todas as operações "write" são atrasadas até que a execução da transação se complete com sucesso. Neste momento, ou num momento posterior (chekpoint), o log é utilizado para a efetivação de todas as atualizações associadas à transação no banco de dados. Neste caso, quando ocorrer uma falha, a informação do log é simplesmente ignorada e a transação é re-executada.

2.4 CONTROLE DE CONCORRÊNCIA

Em um ambiente multiusuário, várias transações podem ser executadas concorrentemente. Portanto, é necessário que o sistema controle a interação entre as transações concorrentes, de modo a evitar que elas destruam a consistência do banco de dados. Este controle é conseguido através de uma variedade de mecanismos.

Neste trabalho, ao mesmo tempo apresentamos alguns dos principais problemas que podem surgir em uma execução concorrente de um conjunto de transações (*Interferência* e *Deadlock*) e alguns dos principais mecanismos utilizados para solucionar estes problemas (*Bloqueio* e *Timestamping*).

2.4.1 Interferência

Mesmo se todas as transações, de um determinado conjunto de transações, estiverem individualmente corretas (isto é, se executadas individualmente, não produzirão um estado inconsistente no banco de dados), ainda é possível, que quando executadas concorrentemente, interfiram uma na outra, de modo a produzir um resultado global incorreto.

Se uma transação acessa um dado somente para leitura, tal interferência não irá ocorrer. Mas se uma ou mais transações acessam um dado para atualizá-lo, então poderão surgir os problemas relacionados à interferência. Tipicamente, em um sistema com concorrência os processos ativos recebem frações intermitentes de tempo do CPU. Desta forma, cada processo é geralmente executado em uma série de intervalos de tempo descontínuos. Portanto, nenhum processo pode ser considerado indivisível e assim as execuções de vários processos podem ser sobrepostas (Smi87).

Suponhamos que *T0* e *T1* sejam duas transações, de um sistema bancário, que transferem fundos de uma conta para outra. A transação *T0* transfere 50 da conta *A* para a conta *B* e a transação *T1* transfere 10% do saldo da conta *A* para a conta *B*. Considere, também, o valor corrente das contas *A* e *B* como 1.000 e 2.000 respectivamente. Suponha que as duas transações sejam executadas individualmente uma seguida da outra. Estas sequências de execução estão representadas na Figura 2.4. Como se pode observar, na sequência de execução *T0*, *T1*, o valor final das contas *A* e *B*, depois da execução das duas transações, será de 855 e de 2.145 respectivamente. Assim, a soma *A+B* também é preservada depois da execução das transações. Da mesma forma, se a sequência da execução fôr *T1*, *T0*, o valor final das contas *A* e *B* será de 850 e de 2.150 respectivamente, preservando assim o valor total da soma das contas *A* e *B*. Isto significa que a consistência do banco de dados é mantida no caso da execução sequencial das transações.

No entanto, este requisito de consistência não é conseguido no caso de uma execução paralela das transações *T0* e *T1*, mostrada na Figura 2.5. Para maiores detalhes sobre o problema da interferência, recomendamos as referências (Cas85) e (Dat83).

T 0	*T 1*

`read (A)`
`A := A - 50`
`write (A)`
`read (B)`
`B := B + 50`
`write (B)`
	`read (A)`
	`temp := A * 0,1`
	`A := a - temp`
	`write (A)`
	`read (B)`
	`B := B + temp`
	`write (B)`

T 0	*T 1*

`read (A)`
`A := A - 50`
`write (A)`
`read (B)`
`B := B + 50`
`write (B)`
	`read (A)`
	`temp := A * 0,1`
	`A := A - temp`
	`write (A)`
	`read (B)`
	`B := B + temp`
	`write (B)`

Figura 2.4 - Execução sequencial das transações *T0* e *T1*.

T 0	*T 1*

`read (A)`
`A := A - 50`
 	
`write (A)`
`read (B)`
`B := B + 50`
`write (B)`
	
	`read (A)`
	`temp := A * 0,1`
	`A := A - temp`
	`write (A)`
	`read (B)`
	
	`B := B + temp`
	`write (B)`

T 0	*T 1*

`read (A)`
`A := A - 50`
`write (A)`
 	
`read (B)`
`B := B + 50`
`write (B)`
	
	`read (A)`
	`temp := A * 0,1`
	`A := A - temp`
	`write (A)`
	
	`read (B)`
	`B := B + temp`
	`write (B)`

Figura 2.5 - Uma execução paralela das transações *T0* e *T1*.

Uma grande preocupação é que as execuções concorrentes das transações sejam equivalentes às suas execuções sequenciais em alguma ordem (Kor86). Esta preocupação é resolvida através da serialização das transações, discutida anteriormente. Várias técnicas são empregadas para alcançar uma serialização desejável das transações. Dentre elas analisamos, aqui, a técnica de *bloqueio*.

2.4.1.1 Bloqueio

O bloqueio é uma técnica de controle de concorrência. Isto é, uma técnica para regular o acesso concorrente a objetos compartilhados. O método mais comumente usado, para implementar o bloqueio, é permitir que uma transação acesse um item de dado somente se ela estiver fazendo corretamente um bloqueio adequado naquele item de dado (Del85).

Existem vários modos pelos quais um item de dado pode ser bloqueado. Neste trabalho analisamos dois destes modos: o *bloqueio exclusivo* e o *bloqueio compartilhado*. Outros tipos de bloqueios são discutidos em (Dat83).

* **Bloqueio Exclusivo**

  Se uma transação T contiver um bloqueio exclusivo (denotado por LX) em algum objeto do banco de dados, então ela pode realizar operações de leitura e gravação naquele objeto e, enquanto durar este bloqueio, nenhuma outra transação distinta T' pode fazer um bloqueio daquele objeto, até que T libere o seu bloqueio (operação UN - unlock).

* **Bloqueio Compartilhado**

  Se uma transação T contiver um bloqueio compartilhado (denotado por LS) em algum objeto do banco de dados, então ela pode realizar somente operações de leitura naquele objeto e, durante este bloqueio, uma outra

transação distinta *T'* também pode fazer um bloqueio compartilhado sobre aquele objeto. Porém nenhuma transação pode fazer um bloqueio exclusivo sobre este objeto, até que todos os bloqueios compartilhados sobre ele sejam liberados.

O exemplo anterior, da transferência de valores entre duas contas em um sistema bancário, agora pode ser escrita de modo que garanta a execução serial das transações, através das operações de bloqueio exclusivo nas contas A e B. Entretanto, o uso das operações de bloqueio pode provocar uma situação onde nenhuma transação possa sequer prosseguir com a sua execução normal. Esta situação, conhecida como *deadlock* (impasse) é representada na Figura 2.6.

T2	*T3*
LX (B)	
read (B)	
B := B - 50	
write (B)	
	LS (A)
	read (A)
	LS (B)
LX (A)	

Figur 2.6. Deadlock entre as transações *T2* e *T3*

Quando ocorre uma situação de impasse, o sistema deve cancelar uma das transações para que as outras possam prosseguir a sua execução. Alguns mecanismos de implementação de vários tipos de bloqueio são discutidos em (Ben82) e (Mag86).

2.4.2 Deadlock

Como foi visto anteriormente, a técnica de bloqueio exclusivo leva as vezes a uma situação de impasse, na qual cada transação fica a espera de recursos que outra transação está processando. Por exemplo, se as transações *T1* e *T2* requerem o controle dos objetos *A* e *B* respectivamente, e em algum momento *T1* requer o *B* e *T2* requer o *A* para prosseguirem a sua execução, então ambas as transações ficarão em uma espera permanente, o que caracteriza um *deadlock*. Em geral, existem duas classes de técnicas utilizadas para solucionar este problema: prevenir um "deadlock" ou detectar um "deadlock".

1. **Prevenção de Deadlock**

   Quando a aquisição de um recurso não pode ser satisfeita, as duas transações envolvidas são testadas, a transação *Ti* que pretende bloquear um determinado objeto e a transação *Tj* que atualmente tem o controle daquele objeto. Este teste, aplicado a *Ti* e *Tj*, deve dar a garantia de que se *Ti* for colocada na lista de espera por *Tj*, o "deadlock" não irá ocorrer. Assim, se ambas as transações *Ti* e *Tj* passarem no teste, então a transação *Ti* poderá esperar pela transação *Tj*, até que esta libere o bloqueio do recurso requerido pela transação *Ti*. Caso contrário, uma das transações será cancelada e re-executada posteriormente {Del85}. Outro método, de prevenir um "deadlock", é atribuir uma prioridade a cada transação. Desta forma, a transação *Ti* poderá esperar pela transação *Tj* se, e somente se, a prioridade da transação *Ti* fôr menor que da transação *Tj* {Del85}.

2. **Deteção de Deadlock**

   Esta técnica permite que as transações fiquem esperando uma pela outra, sem um controle em particular, mas cancela aquelas que são envolvidas em um "deadlock". Isto é, quando um "deadlock" é detectado, uma das

transações é escolhida como a vítima e é cancelada para eliminar o "deadlock". Em geral, é escolhida aquela transação que tiver feito o menor número de atualizações, ou aquela que foi iniciada mais recentemente, ou então aquela que tiver feito o menor número de bloqueios (Del85).

Todavia, estas técnicas da eliminação de "deadlock" podem causar outro problema. Uma transação pode ser cancelada diversas vezes em seguida e a sua execução ser impedida definitivamente. Para remediar esta situação foram desenvolvidas as técnicas de *"timestamping"* (Dat83).

2.4.2.1 Timestamping

As técnicas de "timestamping", conhecidas também como técnicas de pré-ordenação (Cas85), consistem em conceder um identificador único às transações. Por exemplo, uma senha ou um número de protocolo ou até mesmo a hora interna do sistema. Neste último caso, as transações mais velhas sempre terão prioridade sobre as mais jovens. Com este mecanismo, sempre será executada a transação que estiver com a posse do identificador, que será repassada a outra transação quando esta terminar a sua execução ou liberar o recurso solicitado.

Uma diferença fundamental entre as técnicas de "timestamping" e as de bloqueio é que, enquanto as técnicas de bloqueio tentam sincronizar a execução intercalada de um conjunto de transações de modo que o resultado seja equivalente a alguma execução serial daquelas transações, as técnicas de "timestamping" tentam sincronizar a mesma execução intercalada de modo que o resultado seja equivalente a uma execução serial das mesmas transações. Por exemplo, a execução serial definida pela ordem cronológica dos "timestamps" (Dat83).

2.5 SEGURANÇA

A maioria dos usuários exige que seus dados sejam protegidos contra acessos ou atualizações não autorizadas. Mas, até que ponto esta proteção deve chegar? Isto, depende da natureza da informação armazenada no banco de dados. Se uma informação tem pouca importância, então não deve haver medidas de segurança fortemente elaboradas. Porém, como um propósito geral dos sistemas de banco de dados, deve-se assumir que todos os dados são valiosos. Portanto, qualquer SGBD deve oferecer aos seus usuários mecanismos confortáveis e confiáveis para garantir a proteção e a privacidade de suas informações.

2.5.1 Granularidade de Segurança

Medidas de segurança em sistemas de banco de dados podem ser classificadas de acordo com a sua granularidade. Isto é, qual é a menor unidade do sistema que pode ou deve receber uma proteção individual (Smi87). Tipicamente, um proprietário de um conjunto de dados pode controlar o acesso aos campos do registo, individualmente. Por exemplo, um determinado usuário pode receber a permissão de ler o campo nome ou o campo salário de um registro da relação de empregados, mas nunca ambos os campos conjuntamente. É de grande utilidade que um SGBD ofereça controles condicionais. Por exemplo, o usuário X tem permissão de acessar o nome e o salário de todos os empregados, mas é autorizado a modificar somente o salário dos empregados sob sua gerência. Finalmente, os sistemas sofisticados podem oferecer controles temporais ou espaciais (Smi87). Por exemplo, somente o departamento de pessoal pode ser autorizado a modificar o nome e o endereço, ou então, somente o departamento de almoxarifado tem a autorização de modificar a quantidade disponível de cada peça.

2.5.2 Imposição de Segurança

Um possível método de impor as medidas de segurança é ter um módulo do SGBD como árbitro, que examine cada transação e determine se a segurança será ou não violada, caso ela seja executada. O sistema INGRES impõem a segurança de maneira semelhante a que usa para a implementação das restrições de integridade. Ou seja, as instruções (em QUEL) são modificadas antes de serem executadas. Algumas violações da segurança podem ser acidentais ou então, podem fazer parte de procedimentos de ajustes de controles. Por estas razões o SGBD deve registrar, no log, as transações que foram rejeitadas por razões de segurança {Smi87}.

2.5.3 Técnicas para Garantir a Segurança

Existem basicamente quatro classes principais de técnicas designadas para garantir a segurança de dados {Del85}, durante a sua manipulação, em sistemas de banco de dados:

1. Controle de Acesso ou Autorização
2. Controles de Fluxo de Dados
3. Controles de Inferência
4. Criptografia de Dados

2.5.3.1 Controle de Acesso: Autorização

O contole de acesso é feito através da verificação da identidade dos usuários, quando tentam acessar o sistema. O sistema, então, deve permitir que eles acessem certas partes do banco de dados, de acordo com os critérios de privacidade e prioridade, estabelecidos previamente pelo administrador.

Portanto, o sistema deve incorporar mecanismos dinâmicos de autorização, capazes de analisar direitos de acesso a qualquer momento, permitindo ou recusando o acesso aos usuários que o requeriram (Del85).

Todos os usuários do banco de dados devem ter condições de criar seus objetos e manipulá-los da maneira que desejarem e eventualmente conceder permissão de acessá-los, aos outros usuários. Porém, se um usuário decidir eliminar um determinado objeto, então o SGBD deve cancelar automaticamente os direitos de acesso àquele objeto que foram dados aos outros usuários (Del85).

Para cada usuário, portanto, o sistema terá de manter um registro, normalmente denominado o *perfil do usuário* (Dat83), especificando os objetos que o usuário está autorizado a acessar e as operações que ele pode realizar sobre estes objetos. De modo semelhante, o sistema poderia manter um registro para cada objeto, um *perfil do objeto* (Dat83), especificando os usuários que têm permissão para acessá-lo.

2.5.3.2 Controle do Fluxo de Dados

Este problema está ilustrado na Figura 2.7. Paulo e João são dois usuários do SGBD. Paulo não tem a permissão de acessar o objeto *R*, mas ele pede ao João, o qual tem este direito, para fazer uma copia do *R* e transmitir a ele.

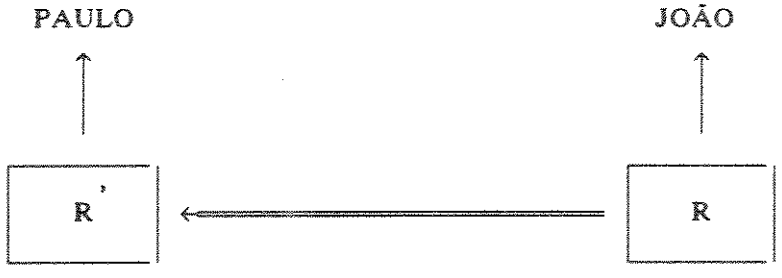

Figura 2.7 - Controle de Fluxo de dados.

Para estudar este problema, devemos primeiro definir em termos gerais o conceito de "fluxo". Um "fluxo" existe entre um objeto *X* e um objeto *Y*, quando um programa lê o objeto *X* e o grava no objeto *Y*. Copiar um arquivo ou um objeto *X* para um arquivo ou um objeto *Y* é considerado um "fluxo". O objeto *X* é a *origem* do fluxo e o objeto *Y* é o *destino* (Del85) (Figura 2.8).

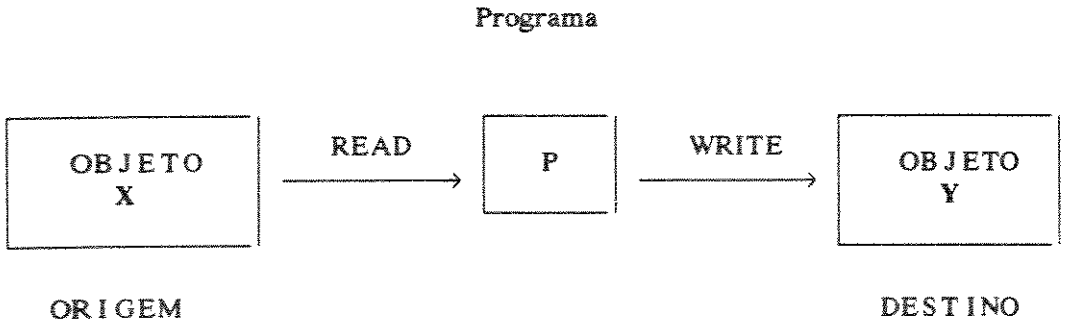

Figura 2.8 - Um fluxo de X para Y.

A maioria dos métodos empregados para a implementação do controle de fluxo de dados utilizam o conceito de níveis de segurança (Dat83). Baseado neste conceito, os dados são classificados em:

* altamente secretos,
* secretos,
* confidenciais,
* não confidenciais.

Assim, cada objeto de dados recebe um nível de segurança. Cada nível é mais restritivo do que aqueles que o seguem. Os usuários, por sua vez, recebem um nível de autorização e são impostas regras simples em que:

* O usuário *i* pode ver o objeto *j* apenas se o nível de autorização de *i* fôr maior ou igual ao nível de segurança de *j*;

* O usuário *i* só pode modificar o objeto *j* se o nível de autorização de *i* fôr igual ao de *j* {Dat83}.

2.5.3.3 Controle de Inferência

As vezes é necessário distribuir algumas informações confidenciais a um grupo de usuários. Para fazer isto, o controle de fluxo de dados deve ser relaxado ou temporariamente suspenso {Del85}. Este é o caso dos sistemas de banco de dados estatísticos. Por exemplo, um banco de dados de um censo, que contém um grande número de registros individualmente sensíveis, só pode fornecer informações de resumos estatísticos a seus usuários e não informações referentes a algum indivíduo específico. O problema é que, como os resumos contém vestígios da informação original, um investigador poderia ser capaz de construir ou reconstruir esta informação com o processamento de recursos suficientes. Esta questão é conhecida como a *dedução da informação confidencial por inferência* ({Dat83},{Del85}).

Eliminar este problema é quase impossível. Porém, existem diversas técnicas que tentam minimizar, ao máximo, este problema, entre as quais algumas são analisadas aqui, de forma bastante resumida.

* **Controle dos resultados sobrepostos**
  Neste caso, o sistema não permite resposta a certas consultas, onde os resultados sobrepõem parcialmente um determinado número de elementos das perguntas previamente realizadas.

* **Particionamento do banco de dados**
  Neste método, os items de dados são armazenados em grupos, cada um

contendo um número pré-determinado de elementos. Consultas só podem ser realizadas sobre o grupo em si, e não sobre algum subconjunto do grupo.

* **Distorção voluntária dos resultados**
  Aqui, os resultados são alterados voluntariamente antes de serem enviados aos usuários. Geralmente, um número gerado de forma "pseudo-aleatória", que depende do próprio item de dado, é acrescentado à resposta.

* **Escolha aleatória da amostragem**
  As consultas do usuário só podem ser aplicadas a um conjunto de elementos cuja composição é escolhida aleatóriamente. Assim, o usuário não pode isolar cada elemento escolhido.

2.5.3.4 Criptografia de Dados

Muito dificilmente os controles de acesso ou de fluxo ou, então, da inferência são suficientes para prevenir completamente um sistema contra acessos não autorizados. O único meio efetivo de realizar esta terefa é a codificação das informações, de tal forma que só possam ser decifradas através de processadores de códigos secretos.

Criptografar é uma maneira segura de armazenar e transportar dados. Usando um código secreto C, geralmente composto por dígitos, uma informação I é transformada em um criptograma $C(I)$, antes de ser armazenada na memória ou transmitida através da rede. Posteriormente, quando fôr necessário adquirir esta informação, então ela é decodificada por um decodificador $DC(I)$.

2.6 Comentários finais

Integridade e consistência em Sistema de Bancos de Dados devem ser assegurados através de uma série de comandos apropriados que facilitem a declaração e a verificação de restrições de integridade e de consistência. Entretanto, são poucos os SGBDs que, de fato, oferecem um conjunto compreensível de facilidades e mecanismos para isto ([Yan88],[Dat83]).

Manter a integridade e a consistência de um sistema de banco de dados não é trivial. Em sistemas mais complexos pode perceber-se quão difícil é esta tarefa. Geralmente, exige-se que estes sistemas tenham mecanismos que os façam capazes de:

* Inicializar uma transação e monitorar o seu progresso durante a sua execução. Se durante a execução, não ocorrer nenhuma anormalidade, então a transação completada deve ser efetivada no banco de dados. Caso contrário, deve proceder-se o seu cancelamento, retrocedendo e recuperando os valores originais dos itens de dados modificados pela transação.

* Controlar a concorrência e sincronizar as execuções paralelas de transações, a fim de evitar que o sistema perca a sua integridade ou que ocorram situações de impasse.

* Garantir que após cada falha o sistema possa recuperar-se e atingir um estado consistente mais recente.

* Permitir que todos os usuários tenham, cada um, um grau de privacidade e segurança, confortável e confiante.

Além destas tarefas serem de difícel realização, fatalmente terão uma influência negativa no desempenho dos sistemas. Por exemplo, diversas restrições de integridade, como as intra-relacionais, são geralmente onerosas para o SGBD.

Durante cada modificação no banco de dados, seja uma alteração, uma inserção ou uma exclusão, requer-se um grande esforço computacional, o que pode comprometer o desempenho do sistema. Por outro lado, nenhum SGBD pode privar os seus usuários destes recursos, tão benéficos e úteis. Pelo contrário, cada vez mais os fornecedores tentam implementar o maior número possível de facilidades de controle de integridade, consistência e segurança em seus SGBDs.

Este capítulo procurou apresentar e analisar problemas e soluções para a questão de tratamento de integridade nos sistemas de banco de dados, à luz da literatura existente, como forma de situar a questão em sistemas convencionais, servindo como introdução ao capítulo 3, que analisará a mesma questão no contexto de sistemas não-convencionais.

"A Terra é um só país e a humanidade seus cidadãos.

Bahá'u'lláh

CAPÍTULO 3

CONTROLE DE RESTRIÇÕES DE CONSISTÊNCIA EM SISTEMAS DE BANCO DE DADOS NÃO-CONVENCIONAIS

3.1 INTRODUÇÃO

O desenvolvimento acelerado dos computadores tem oferecido facilidades crescentes de organizar e administrar informação, um recurso cada vez mais vital de qualquer atividade humana. Consequentemente, as aplicações da tecnologia de banco de dados atualmente se encontram em contínua expansão. Entre estas novas áreas de aplicação, a de engenharia e a de inteligencia artificial têm recebido uma atenção especial dos pesquisadores.

Como foi visto anteriormente, de forma resumida, diversas características distinguem os SBDs convencionais e os SBDs utilizados nestas novas áreas de aplicação. Neste capítulo, procuraremos identificar algumas das características específicas dos sistemas de banco de dados utilizados nos ambientes de PAC e nos sistemas de base de conhecimentos, principalmente as que dizem respeito ao controle de restrições. Neste sentido, alguns protótipos de SGBDs, desenvolvidos para estas aplicações, também serão analisados.

3.2 SISTEMAS DE BANCO DE DADOS PARA AMBIENTES DE PROJETO

As atividades nos ambientes de projeto auxiliado por computador envolvem grandes quantidades de objetos, geralmente compartilhados por projetistas que não estão interessados em detalhes sobre a forma de seu armazenamento. A principal preocupação dos projetistas é em torno das facilidades de acesso e o grau de confiabilidade dos dados.

Em aplicações de PAC, a maioria dos objetos (circuitos VLSI, peças mecânicas, edifícios, etc.) podem ser conceitualmente divididos em componentes menores, que por sua vez também podem ser recursivamente subdivididos (Mel88). Para apoiar as atividades de projeto, as ferramentas de PAC, inclusive o SGBD, devem refletir esta estrutura hierárquica.

Em ambientes de PAC, como podem existir diversas ferramentas de projeto sob a forma de programas escritos em diferentes linguagens de programação, é importante que haja interfaces adequadas para acesso ao banco de dados de projeto, a partir de linguagens de programação padrão.

A interface de usuário é um componente de grande impacto na avaliação de um sistema de PAC e deve se valer, entre outras, das técnicas de computação gráfica interativa (Toz91) e de programação orientada a objetos, para oferecer um resultado agradável e eficiente ao usuário.

Um sistema de projeto auxiliado por computador consiste em três classes de ferramentas (Kat85): **ferramentas de síntese**, **ferramentas de análise** e **ferramentas para gerenciamento da informação**. As ferramentas de síntese apoiam o projetista na criação de objetos a serem projetados. As ferramentas de análise lhe dão subsídios para verificar a exatidão do projeto. As ferramentas de gerenciamento da informação são responsáveis pela criação, manutenção e visualização de uma base de dados consistente contendo a descrição do projeto. Uma base de dados integrada é o eixo principal, através do qual todos os dados do projeto podem ser compartilhados entre as ferramentas do ambiente de projeto.

Os requisitos necessários para gerenciamento de dados em um ambiente de engenharia incluem, entre outras coisas, uma abordagem hierárquica do projeto, suportes para representações múltiplas do projeto, controle de configuração e de versões, atualização automática de todas as partes do projeto quando uma é modificada, e finalmente, uso de restrições de consistência para apoiar todas estas atividades.

Outra característica marcante dos processos de projeto é a sua natureza iterativa {Ket87}. Por este motivo, os projetistas não podem fornecer prontamente uma descrição completa do projeto. Eles criam uma descrição parcial do projeto que será completada posteriormente através de refinamentos sucessivos ({Enc83},{Vet82}). Em cada iteração, o projetista pode obter diversos refinamentos aparentemente corretos. Porém, o que pode aparecer correto em um determinado momento poderá tornar-se incorreto posteriormente.

De qualquer forma, os estados anteriores do projeto devem estar sempre disponíveis para um trabalho posterior do projetista. Isto leva a necessidade de mecanismos de controle de versões nos ambientes de projeto. Estes mecanismos estão, por exemplo, descritos em ({Kat84},{Kat86a}, {Kat86b},{Cho86}). Os projetistas geralmente escolhem uma alternativa e prosseguem até completar o projeto. Se aquela alternativa se tornar inadequada, eles recuam e adotam outra alternativa.

Um banco de dados de projeto tem tipicamente uma estrutura complexa de dados, visto que ele deve organizar os dados do projeto em múltiplas representações, com suas versões evolucionárias e implementações alternativas. Esta natureza iterativa e tentativa dos processos de projeto e o controle de versões e configurações implicam na necessidade de verificações da consistência do projeto.

Consequentemente, nestas aplicações, o controle de consistência depende, essencialmente, das facilidades que o SGBD fornece para a descrição e a definição dinâmica de diversas restrições de consistência sobre o projeto, na base de dados. Em outras palavras, a definição e a verificação de restrições de consistência devem ser aplicadas como transações de projeto via um subsistema de

gerenciamento ou via uma interface de usuário adequados para garantir o desenvolvimento de um projeto correto.

Em resumo, os requisitos necessários para o gerenciamento de dados em ambientes de projeto devem incluir (Kat85):

* Estruturação hierárquica de dados;
* Representações múltiplas de projeto;
* Verificação da consistência de representações cruzadas;
* Controle de versões;
* Interfaces para múltiplas linguagens de programação;
* Integração com gráficos e desenhos;
* Manipulação de entidades estruturadas;
* Manipulação de bases de dados convencionais;
* Suporte para processamento de dados distribuídos e,
* Integração do gerenciamento de projetos com de manufatura.

3.2.1 Transações em Ambientes de Projeto

Apesar das aplicações de PAC geralmente acessarem os dados de projetos organizados em objetos complexos, os sistemas de gerenciamento de dados necessários para estas aplicações devem continuar a fornecer mecanismos para suportar as transações convencionais de sistemas de banco de dados. Isto significa que as transações em SGBDs utilizados em ambientes de PAC devem ter, além das inúmeras características das transações convencionais, propriedades que facilitem a interação do projetista com todos os dados do projeto durante todos as fases do processo de projeto.

Esta nova categoria de transações, usualmente chamada de transações de projeto (Kat85), em geral são mecanismos através dos quais os projetistas criam novas versões consistentes de objetos do projeto. As principais diferenças entre

as transações convencionais e as transações de projeto estão na necessidade de mecanismos especiais de verificação da consistência para transações de projeto. A seguir descrevemos algumas destas diferenças ([Kim84],[Ban85], [Kor88]).

1. As Transações de Projeto têm longa duração

Os projetistas interagem com os dados de projeto durante longos períodos de tempo, talvez dias ou mesmo semanas, enquanto que as transações convencionais são de curta duração, no máximo alguns minutos. Portanto, os mecanismos que viabilizam acessos exclusivos aos dados compartilhados, forçando as transações a esperarem enquanto os dados não estão disponíveis, não são adequados nos ambientes de projeto. Ao contrário das transações convencionais, nas transações de longa duração os estados intermediários devem ser tolerados. Isto significa que devem ser permitidas inconsistências temporárias.

2. As Transações de Projeto manipulam grandes volumes de dados

As transações de projeto acessam grandes coleções de registros normalmente distribuídos em diversos arquivos. Assim, o grande volume de dados envolvidos proíbe que o SGBD seja requisitado para acessos individuais aos registros. Desta forma, os bancos de dados de projeto são melhor utilizados como repositório de dados compartilhados, de onde os dados devem ser extraídos quando há necessidade de acessos intensivos.

3. As Transações de Projeto precisam mais do que uma simples serialização

Uma execução correta de transações concorrentes (em sistemas convencionais) é assegurada através da serialização das mesmas. Entretanto, uma simples serialização de transações de projeto é insuficiente para assegurar a integridade do sistema e garantir que os dados de um projeto sejam consistentes após uma atualização. Pois, nestes sistemas há transações substancialmente diferentes

das transações tradicionais, podendo ser aninhadas ou de longa duração, o que dificulta a sua serialização. Assim, são necessários programas especiais de validação para verificar a consistência dos dados de projeto.

4. As Transações de Projeto não são atômicas

Nas aplicações convencionais, o processo de recuperação de falhas exige que algumas transações sejam refeitas, e outras desfeitas completamente, até que a base de dados atinja um estado consistente. Como nestas aplicações o conjunto de operações de uma transação é relativamente pequeno, a perda de trabalho realizado correspondente às transações desfeitas não chega a ser muito significativa.

Porém, em ambientes de projeto como as transações são longas e representam grandes esforços de trabalho, em caso de falhas no sistema, deve ser recuperada a maior porção possível do trabalho realizado. Portanto, nestes ambientes deve se evitar que uma transação seja desfeita completamente. Neste caso, o sistema deve desfazer a menor porção possível da transação e recuperar o restante. Desta forma, as transações de projeto não podem ser proposições do tipo *tudo-ou-nada*, ou seja, elas não podem ser consideradas operações atômicas, permitindo recuperação parcial das mesmas.

5. As Transações de Projeto podem ser aninhadas

Ao contrário das transações convencionais, as transações em ambientes de projeto tendem a ser aninhadas ({Hae87},{Mos87}). Ou seja, uma transação pode conter várias subtransações e cada subtransação também pode conter outras subtransações. O conceito de transações aninhadas oferece uma estrutura de controle dinâmica que permite a distribuição da tarefa, dentro da transação, em pequenas partes que são executadas de forma paralela e podem ser desfeitas, caso ocorra uma falha, sem afetar outras partes.

3.2.2 Restrições de Consistência em Ambientes de PAC

Na área de projeto auxiliado por computador e outras aplicações de banco de dados em engenharia, existe um grande número de restrições de consistência. Todavia, a maioria destas restrições ou está embutida na própria aplicação, ou então é deixada a critério dos projetistas. É muito importante que estas restrições possam ser expressas como parte do processo de definição de dados de projeto, para que os SGBDs possam verificar se elas continuam satisfeitas durante mudanças de projetos. Neste trabalho, para efeito de simplificação, muitas vezes as restrições de consistência são chamadas simplesmente de restrições.

Como já abordado no item anterior, a forma clássica de controle de consistência através de gerência de transações, utilizada em sistemas de banco de dados convencionais, não é suficiente para ambientes de aplicações em engenharia. Devido à complexidade de suas estruturas e aos tipos de relacionamentos entre seus objetos, estas aplicações exigem mecanismos mais completos e flexíveis para manter a consistência do sistema. Por exemplo, dados de um projeto devem satisfazer critérios abrangendo desde os métodos e a tecnologia utilizados pelos projetistas até as leis físicas da realidade projetada. Portanto, além de *integridade referencial* (Dat86), *cardinalidade* e o conceito de *identificadores* ou propriedades de *chaves*, outras restrições devem ser consideradas em SGBDs utilizados nos ambientes de PAC.

Katz (Kat83) menciona três classes de restrições para aplicações PAC: **restrições de conformidade** (a consistência entre a especificação e a implementação de um objeto de projeto); **restrições de composição** (cooperação correta entre subobjetos para formarem um superobjeto) e **restrições de equivalência** (porções de diferentes representações de um objeto de projeto devem satisfazer determinadas equivalências de relações). Dittrich em (Dit86b) acrescenta uma quarta classe de restrições: **restrições ambientais** (certos valores internos de objetos encontram limitações a fim de refletir corretamente os fatos do mundo real).

3.2.3 Características das Restrições de PAC

Em {Mor89} o autor descreve que o *propósito* de uma restrição é estabelecer um relacionamento desejado (substancialmente diferente dos relacionamentos tradicionais em sistemas de banco de dados) entre diversos itens ou objetos de dados do sistema e, que a *operação* de uma restrição limita-se a combinação de valores que simultaneamente satisfazem os itens de dados ou os objetos participantes da restrição.

De acordo com a literatura ({Kot88}, {Kor88}, {Dit86b}, {Neu82}), quando as restrições de consistência são observadas do ponto de vista das aplicações PAC, várias questões devem ser levadas em consideração:

* Nestas aplicações podem existir inúmeras restrições complexas, através das quais os projetistas tentam aproximar ao mundo real, a estrutura e os relacionamentos complexos existentes na representação de um projeto.

* As restrições podem ter de ser violadas deliberadamente durante um longo período de tempo. Isto ocorre porque a consistência global dos dados é atingida através de um processo gradual ao longo de diferentes estágios de consistência local. Durante estes estágios intermediários uma porção da base de dados pode estar semanticamente correta enquanto outra parte pode não estar consistente, ou até mesmo, os dados podem estar ainda ausentes. Inconsistências temporárias, porém longas, devem ser toleradas em diferentes níveis, sem perda da exatidão semântica dos dados.

* O momento em que os dados devem atingir a consistência não pode ser fixado uma vez e para sempre. Na maioria das vezes é o programa de aplicação ou o próprio usuário quem decide o momento da verificação da consistência.

* Em caso da violação de restrições, as reações do sistema devem ser

flexíveis e também, em alguns casos, sob controle externo. Nos SGBDs convencionais uma transação é desfeita quando, no final da mesma, detecta-se uma inconsistência. Isto não é aceitável em aplicações onde as transações podem demorar tipicamente por várias horas ou até dias e envolvem vastos esforços de trabalho sobre grandes volumes de dados. Portanto, para estas aplicações devem ser consideradas outras formas de reação. As vezes, uma simples notificação ao usuário, dando-lhe eventualmente informações adicionais, tais como, onde e porque os dados estão inconsistentes, pode ser a solução mais apropriada.

* Do ponto de vista tanto da eficiência de processamento como da facilidade de definição, algumas restrições não devem ser definidas explicitamente, isto é, em forma declarativa, sendo mais eficiente sua definição em forma algoritmica em programas de verificação, os quais posteriormente serão executados em determinados momentos para assegurar a consistência dos dados. Porém, embutidas nestes programas, as restrições não devem permanecer desconhecidas para o SGBD. Embora, o SGBD não esteja explicitamente a par do conteúdo destas restrições, pode desejar impor a sua execução, provocando assim reações corretas de acordo com a situação.

* Especificar que ação (ou programa) deve ser executada e em que momento, constitui um dos problemas semânticos especiais em SGBDs, particularmente em aplicações PAC, onde há planos de ação predefinidos.

* As novas áreas de aplicação de SGBDs demonstram ser áreas mais dinâmicas do que as convencionais. Assim, o conjunto de restrições deve ser manipulado e modificado mais frequentemente do que nas aplicações tradicionais. Ele pode variar de acordo com a tecnologia, com o desenvolvimento de novas ferramentas, com decisões de gerentes, com a equipe de projetistas e até mesmo com os hábitos de cada projetista individualmente.

3.3 SISTEMAS DE BASE DE CONHECIMENTO

A idéia de que a informação é algo mais do que meramente um conjunto de dados, cada vez mais têm estimulado as pesquisas na área de ciências de computação. Uma visão mais abrangente do conceito de informação aproxima-o de **conhecimento**, que pode ser expresso através de regras, assertivas lógicas, redes semânticas e frames (Elm89). As regras são utilizadas para dedução de novos fatos, que explicitamente não fazem parte do sistema, baseados nos dados existentes neste.

Os SGBDs convencionais têm uma capacidade limitada de representar conhecimento e deduzir novos fatos. Durante os últimos anos, tem surgido um esforço considerável, por parte dos pesquisadores, para desenvolvimento de uma nova geração de sistemas, capazes de suprir as necessidades de diversas aplicações de engenharia no que diz respeito à manipulação de conhecimento e bases de conhecimento.

Estes sistemas, chamados de Sistemas de Base de Conhecimento (SBC) ou simplesmente Sistemas de Conhecimento (SC), surgiram a partir de extensões de sistemas de banco de dados que passaram a incorporar recursos da Inteligência Artificial e técnicas de inferência (Mel88).

Os resultados das diversas pesquisas nesta área levaram ao surgimento de duas categorias de protótipos (Mel88): **Bancos de Dados Dedutivos** e **Sistemas Gerênciadores de Bases de Conhecimento** (SGBC).

Em sistemas de banco de dados dedutivos estão presentes todas as características funcionais dos sistemas de banco de dados convencionais, acrescidas de um mecanismo de inferência, que possibilita a dedução de novos fatos a partir dos dados já existentes. Porém, os domínios de atuação dos bancos de dados dedutivos são os mesmos que os dos bancos de dados convencionais, ou seja, os dados.

Por sua vez, em SGBCs são necessárias novas formas de representação de conhecimento e consequentemente são requeridas novas características funcionais, para a manutenção de conhecimento. Definir e qualificar conhecimento é uma tarefa muito difícil. Conhecimento é normalmente produzido por especialistas, dentro de um domínio particular de especialização, sendo utilizado para definir, controlar e interpretar os dados, em uma base de dados.

Em geral, em sistemas de informação o conhecimento pode aparecer em diversas formas {Wie84}: *Conhecimento estrutural* - conhecimento sobre as dependências e restrições entre os dados; *Conhecimento procedural genérico* - conhecimento que pode ser descrito somente através de um procedimento ou um método; *Conhecimento específico da aplicação* - conhecimento determinado pelas regras e regulamentos aplicáveis em um domínio específico; *Conhecimento empresarial* - uma forma mais apurada de conhecimento que permite uma empresa tomar suas decisões.

Porém, do ponto de vista técnico e com enfoque para os sistemas de bases de conhecimento, o conhecimento pode ser classificado em três categorias {Elm89}:

Conhecimento intencional - é definido como o conhecimento que existe por trás do conteúdo real da base de dados. Este conhecimento pode ser totalmente especificado antes que a base de dados seja estabelecida.

Conhecimento extencional - é o conhecimento especificado pelas informações, existentes nas bases de dados, sobre os dados propriamente ditos, os meta-dados.

Conhecimento derivado - é a forma de conhecimento na qual o conhecimento intencional e o extencional são agregados, normalmente em forma de regras que são mantidas na base de regras.

A representação de conhecimento envolve questões mais complexas do que a representação de dados através dos modelos de dados {Sow88}. Além disso, a manipulação de conhecimento exige, assim como nas aplicações de BDs, suportes

eficientes para operações de armazenamento e recuperação de conhecimento, porém, com ênfase sobre mecanismos para desenvolvimento de raciocínio {Deb90a}. Consequentemente, os SGBCs devem prover mecanismos não somente para a representação de conhecimento, mas também para a manipulação de conhecimento e a dedução de novos fatos. As deducões são realizadas através de avaliação de regras utilizando um interpretador de regras que faz parte do sistema de base de conhecimento.

A Figura 3.1 {Kel86} mostra uma arquitetura genérica para sistemas de base de conhecimento. Como pode ser observado, os componentes básicos de um SBC são dois: o Gerenciador de Conhecimento e a Base de conhecimento. O Gerenciador de Conhecimento, por sua vez, é composto de dois módulos: o módulo de inferência e o módulo de busca. Na Base de Conhecimento são armazenados os dados, que constituem a base de dados, e as regras, que constituem a base de regras.

Um processador dedutivo proporciona diversos benefícios para um sistema de bases de conhecimento {Kel86}. O primeiro é que ele permite a extração de informações que não estão armazenadas explicitamente na base de dados. Este aumento da função de recuperação da informação pode ser especialmente importante para muitas aplicações de banco de dados.

Além disso, um processador dedutivo pode gerar não somente uma série de respostas para determinadas consultas, mas também um determinado número de evidências (linhas de raciocínio), que podem ser usadas para confirmar ou negar as respostas.

Outro benefício consiste no fato de ser possível responder às perguntas do tipo "suponhamos que", "o que aconteceria se" e outras, que aparecem em muitas consultas de alto nível. Enquanto que, nos sistemas de banco de dados responder este tipo de perguntas é extremamente difícil, se não impossível.

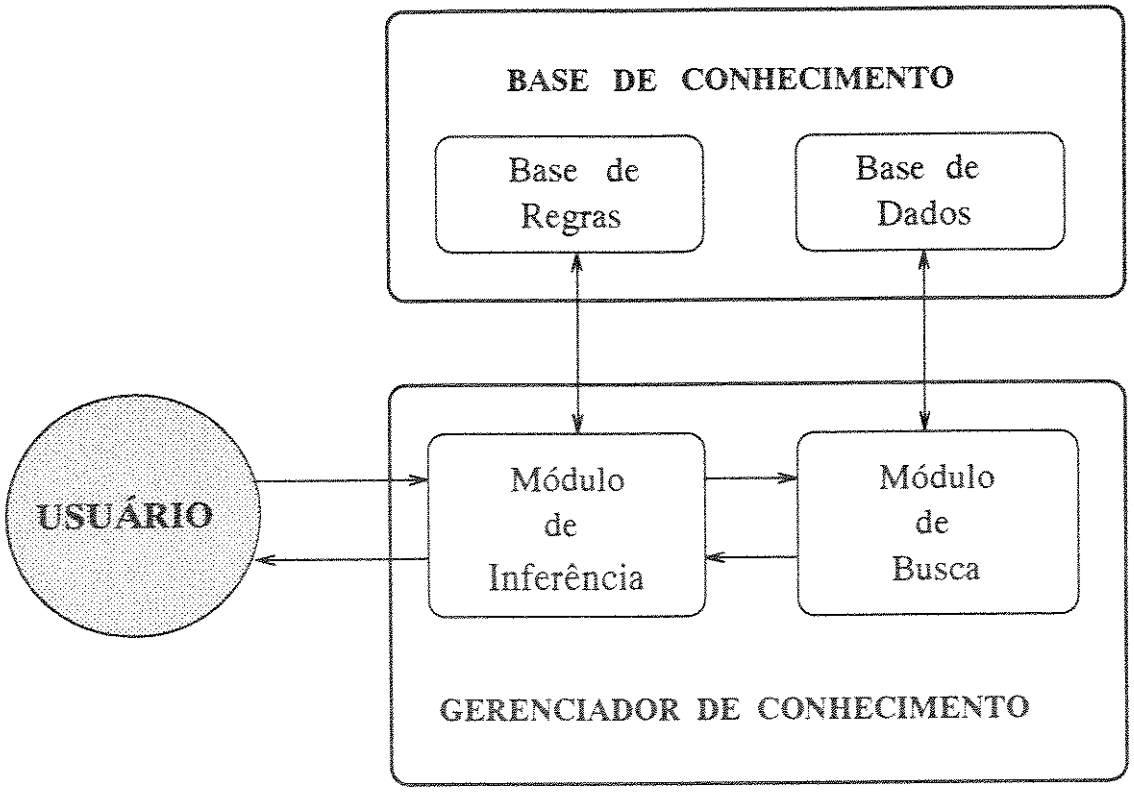

Figura 3.1 - Arquitetura Genérica de um SGBC

3.4 REPRESENTAÇÃO DE RESTRIÇÕES EM ALGUNS PROTÓTIPOS PROJETADOS PARA APLICAÇÕES NÃO-CONVENCIONAIS

3.4.1 POSTGRES

O POSTGRES (POST inGRES) foi desenvolvido no início dos anos 80 como sucessor do SGBD relacional INGRES, implementado ao final da década de 70 na Universidade de California em Berkeley ([Sto76], [Sto87]). O POSTGRES é considerado um SGBD extensível e suporta objetos complexos, típicos das aplicações de projetos de engenharia.

O modelo de dados do POSTGRES é uma extensão do modelo relacional que permite realizar algumas construções dos modelos de dados semânticos [Row87]. O POSTGRES suporta objetos complexos, definição e operação de tipos abstratos de dados, uso de procedimentos definidos por usuário e a herança de atributos entre tipos de objetos [Ioc89].

A linguagem de definição e manipulação de dados do POSTGRES é o POSTQUEL, uma versão estendida da QUEL. Apesar de suas estruturas básicas serem muito semelhantes à QUEL, as inúmeras extenções feitas fazem com que a POSTQUEL possa suportar objetos complexos, tipos de dados definidos por usuário, controle de versões e dados históricos, consultas iterativas, definição e manipulação de gatilhos e de restrições. Estas facilidades são descritas em [Sto86].

A arquitetura do sistema POSTGRES consiste basicamente de dois componentes: o **POSTMASTER** e o **Servidor POSTGRES** [Sto86]. O POSTMASTER (ver Figura 3.2) é responsável por gerenciar o processo de armazenamento na base de dados, implementar o sistema de paginação e controlar o gerenciamento de transações. Uma unidade de POSTMASTER pode ser instalada em cada nó do sistema onde o SGBD

POSTGRES está instalado. Muitas das funções realizadas pelo POSTMASTER são executadas em processos paralelos aos do usuário.

O Servidor POSTGRES é a camada superior do sistema e é suportada pelo POSTMASTER. Ele é responsável por todo o processo de comunicação dos programas do usuário com o POSTMASTER, além do armazenamento temporário dos resultados das transações. Existe um Servidor POSTGRES associado a cada programa do usuário que esteja ativo dentro do sistema. Os servidores são disparados como processos independentes e executam as operações da base de dados, de forma paralela, para o programa de aplicação. A Figura 3.2 mostra a arquitetura do sistema POSTGRES.

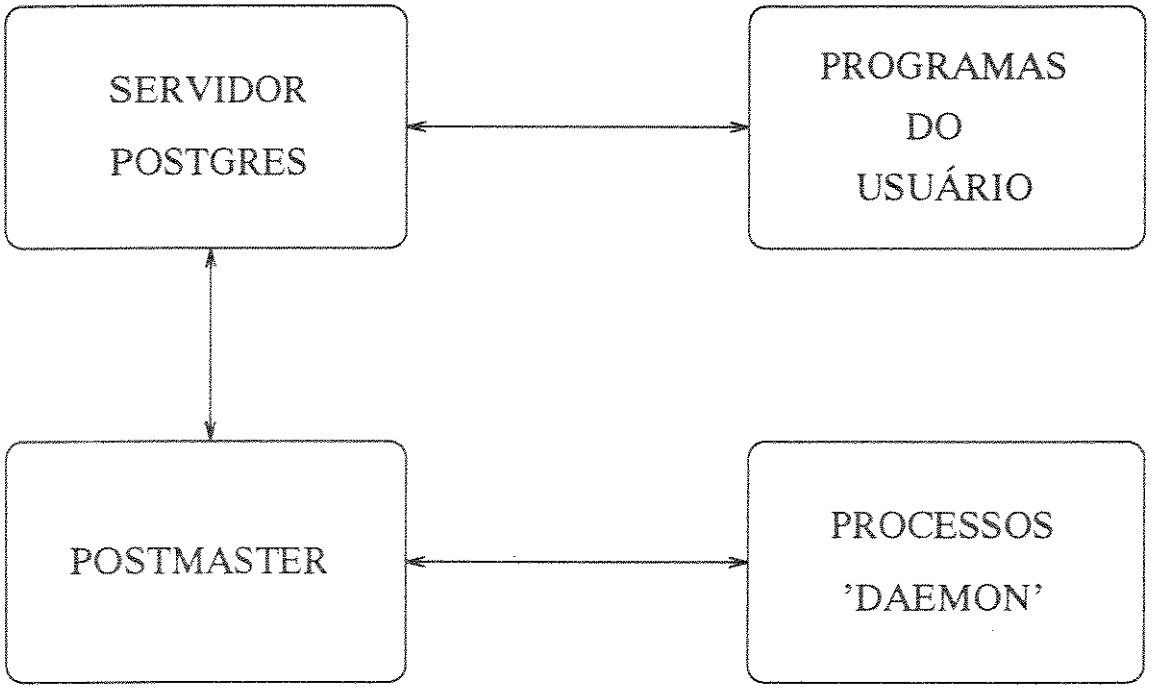

Figura 3.2 - A Arquitetura do Sistema POSTGRES

3.4.1.1 Gerenciador de Restrições do POSTGRES

O gerenciador de restrições do POSTGRES consiste em comandos da linguagem de consulta (POSTQUEL), precedidos por uma cláusula. POSTGRES permite que qualquer comando POSTQUEL seja rotulado com três cláusulas diferentes que modificam o significado do comando. Assim, os comandos modificados constituem os diversos tipos de restrições (basicamente três) que podem ser utilizadas em diferentes situações (Sto88). Desta forma, cria-se uma complexidade adicional para usuário, uma vez que ele é obrigado a aprender a linguagem de consulta POSTQUEL.

A primeira cláusula é a cláusula "**always**". A semântica desta cláusula é que o comando associado a ela será executado sempre. Por exemplo, as seguintes linhas de comando constituem uma restrição que modifica o salário do João de forma que sempre quando um usuário tenta acessá-lo veja um valor igual ao salário do José.

```
always replace EMP (salário = E.salário)
using E in EMP
where EMP.nome = "João" and E.nome = "José"
```

A segunda cláusula que pode ser acoplada a qualquer comando POSTQUEL é a cláusula "**refuse**". A semântica de "refuse" é que o comando associado a ela nunca deve ser executado. Por exemplo, o comando "retrieve" pode ser modificada da seguinte forma:

```
refuse retrieve (EMP.salário)
where EMP.nome = "João"
```

Assim, se ocorrer qualquer tentativa de acesso ao salário do João, o sistema recusará este acesso. Isto significa que quando usamos a cláusula "refuse" associada a qualquer comando, a operação indicada não poderá ser efetuada para nenhuma tupla que satisfaça a condição. Estes tipos de restrições,

construídas com a cláusula "refuse", são geralmente úteis para fins de proteção dos dados.

A terceira e última cláusula utilizada para rotular os comando POSTQUEL é a cláusula **"one-time"**. Por exemplo:

```
one-time replace EMP (salário = E.salário)
using E in EMP
where EMP.nome = "João" and E.nome = "José"
```

O significado deste comando é que a operação deve ser executada uma única vez, quando a condição for satisfeita. Neste caso, o efeito é exatamente o mesmo se o comando fosse executado diretamente pelo usuário sem a cláusula "one-time". Maior detalhes destes mecanismos estão descritos em {Sto88}.

O POSTGRES ativa as restrições em momentos apropriados, construindo processos específicos. Quando uma restrição é definida na base de dados para uma validação imediata ou adiada, o POSTGRES executa os comandos correspondentes à restrição de modo especial e avalia todos os itens de dados que são lidos ou serão escritos. Em cada um destes itens de dados ou na sua correspondente coluna, será aplicado um determinado tipo de bloqueio {Sto88}. Os bloqueios são tipicamente aplicados a nível de atributos. Porém, como podem existir situações onde sejam necessários bloqueios progressivos, o POSTGRES permite diversos níveis de bloqueios.

O sistema de restrições pode ser usado também para criar dois tipos de visões, *parciais* e *só-leitura*, diferentes das visões padrões do POSTGRES. Para criar uma visão só-leitura, primeiro é especificada uma relação e depois sobre esta relação define-se uma restrição. As visões parciais são relações que têm uma coleção de atributos e adicionalmente um conjunto de dados extraídos por quantas restrições forem necessárias.

O gerencidor de restrições do POSTGRES processa as tuplas inseridas, excluídas, recuperadas ou substituídas e retorna uma tupla revisada ou uma mensagem de erro para a rotina de execução.

3.4.2 DAMASCUS

O sistema de banco de dados DAMASCUS é um SGBD projetado para aplicações de engenharia. O núcleo central deste sistema foi projetado de tal forma que possa ser utilizado para qualquer SGBD destinado às aplicações não convencionais (Dit86a). Este sistema foi projetado e implementado a partir de meados dos anos 80 na Faculdade de Informática da Universidade de Karlsruhe, na Alemanha.

Projetado e implementado num ambiente UNIX, o DAMASCUS é composto de dois módulos principais: o **Sistema Gerenciador de Projeto** (DMS - Design Management System) e o **Núcleo do Sistema Gerenciador de Base de Dados** (KDBMS - Kernel Data Base Management System) (Dit86a). A Figura 3.3 mostra a arquitetura deste sistema.

O gerenciamento dos objetos de projeto é uma parte específica da aplicação de sistemas de banco de dados. No caso de DAMASCUS, o Sistema Gerenciador de Projeto é construído para aplicação em projetos de VLSI. Baseado no modelo de dados DODM (Design Object Data Model (Dit86a)), o DMS é um sistema orientado a objetos, representando as semânticas da aplicação através de suas estruturas e operadores. Além das facilidades oferecidas pelo modelo de dados, o DMS pode também realizar controle de consistência e gerenciamento de transações.

O Núcleo do SGBD é constituído de duas camadas: o sistema interno de gerenciamento de objetos (IOMS - Internal Object Management System) e o sistema de armazenamento de objetos (OSS - Object Storage System).

Este Núcleo inclui facilidades supostamente comuns a todas as aplicações de projeto (projetos de VLSI, projeto de peças mecânicas, engenharia de software, etc.) e até a outras aplicações não convencionais (processamento de imagem, BD geográficos, etc.) (Dit86a). Por esta razão, este núcleo foi desenvolvido de tal forma que fosse suficientemente genérico e poderoso para permitir que diversos Sistemas Gerenciadores de Projeto pudessem ser construídos sobre ele, possibilitando assim, o seu uso em diferentes áreas de aplicações não convencionais (Dit86a).

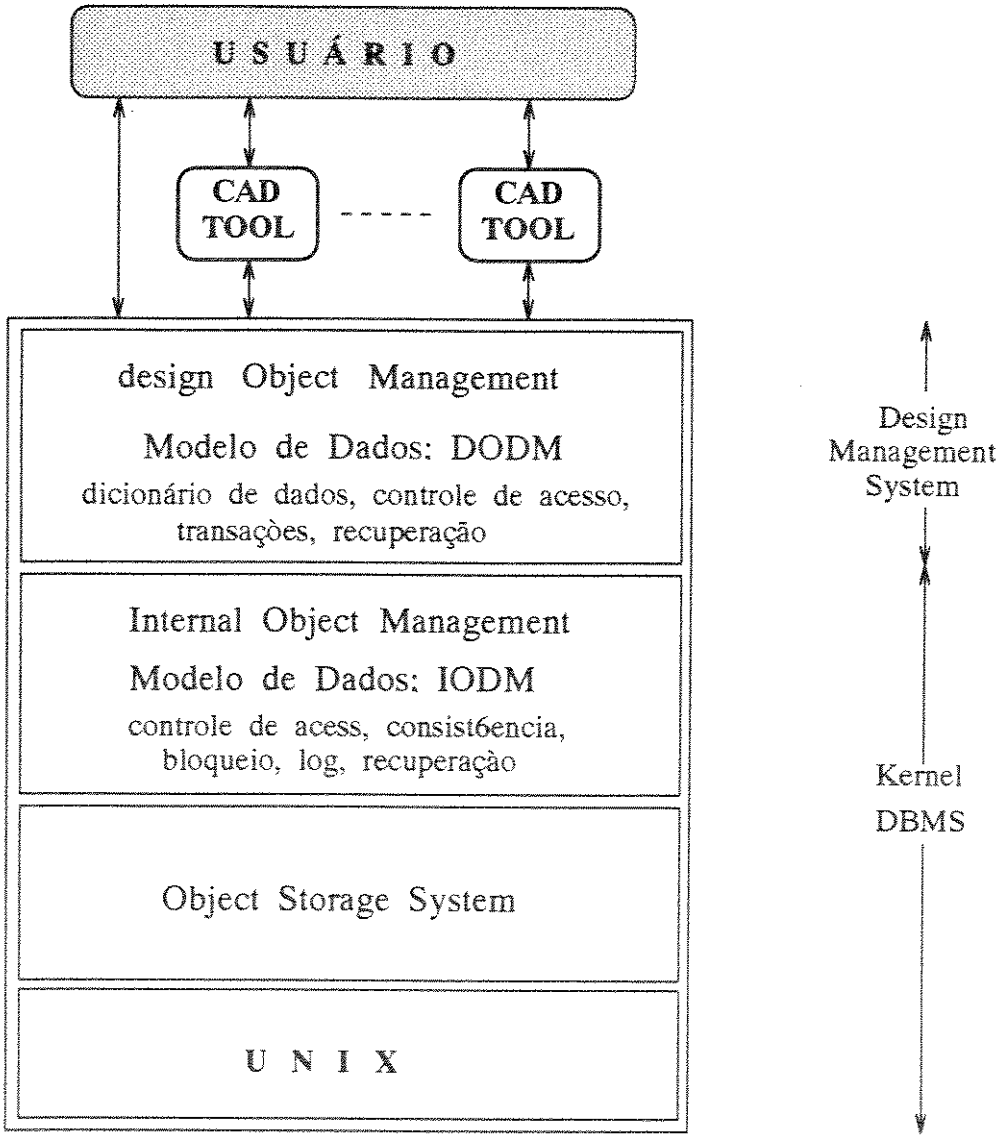

Figura 3.3 - A Arquitetura do Sistema DAMASCUS (Dit86a)

O IOMS é baseado no modelo de dados IODM (Dit86a), e oferece mecanismos básicos para todos os serviços relevantes de um SGBD. Estes mecanismos incluem facilidades de bloqueio (locking) e de monitoração (logging) e também recursos básicos para controle de consistência.

O OSS é o componente do SGBD que realiza um armazenamento eficiente dos objetos, permitindo um acesso rápido aos mesmos. O gerenciador de "buffer", bem como os mecanismos que influenciam o armazenamento e as estrtégias de acesso, também são partes deste módulo do sistema.

Uma das características importantes do DAMASCUS é a sua concepção de controle de consistência, implementado num sistema chamado ETM (Event/Trigger Mechanism) que será analisado na próxima seção. Outras características do DAMASCUS estão descritas em ([Dit86a],[Dit85]).

3.4.2.1 Controle de Consistência em DAMASCUS

O controle de consistência em DAMASCUS baseia-se na idéia de ter um gatilho para cada evento e uma ou mais ações associadas a cada gatilho [Kot88]. Um evento é um indicador que sinaliza a ocorrência de uma determinada situação. Os eventos podem ser tanto definidos pelo usuário como predefinidos pelo sistema (eventos padrão). Uma ação corresponde a um determinado módulo executável, escrito em alguma linguagem de programação de alto nível e contém algumas operações do banco de dados. Um gatilho é um par composto de um evento e uma ação. Sua semântica representa a execução da ação quando ocorre o evento [Dit86b].

Nesta concepção, as rotinas de verificação das restrições de consistência são tratadas como ações. Uma rotina de verificação pode ser obtida de duas formas:

1. A restrição de consistência é definida como um predicado lógico, transformada automaticamente, pelo SGBD, em uma rotina de verificação que interpreta um resultado boleano.

2. A restrição de consistência é sempre especificada pelo usuário de forma algorítmica. Neste caso, o programa pode ser aceito diretamente como rotina de verificação. Assim, programas existentes de verificação da consistência podem ser integrados do sistema, associados a um gatilho.

O ETM possue uma interface para definição e remoção dinâmica de eventos, ações e gatilhos. Quando o ETM é utilizado como um componente do SGBD, esta interface pode ser explorada de duas formas:

- torná-la disponível para usuário (ou para os programa de aplicação), dentro da interface do SGBD,

- usá-la de dentro dos outros componentes do SGBD.

Além disso, há também uma linguagem de definição de restrições (CDL) que apesar de não fazer parte do ETM, utiliza suas operações para definição de restrições tanto em sua forma algorítmica como em sua forma declarativa.

Os elementos do ETM são acessados através de várias tabelas hash que permanecem na memória principal. Existem três tabelas: a *tabela de eventos*, a *tabela de ações* e a *tabela de gatilhos*. As ações (os programas de verificação) são compiladas e suas formas executáveis são armazenadas em uma biblioteca de códigos objetos, enquanto os seus fontes são mantidos em uma biblioteca de arquivos fontes.

Todos os tipos de eventos, inclusive os eventos padrão, são inseridos na tabela de eventos. As informações sobre qual é o gatilho definido para um determinado evento, que ação deve ser executada e, se o gatilho foi ativado corretamente podem ser recuperadas a partir desta tabela. Como é permitida definição de múltiplos gatilhos para um determinado evento, a implementação da tabela de eventos permite que as suas entradas tenham tamanho variável, mantendo uma lista de gatilhos correntemente associados a cada evento.

Algumas características proporcionam ao ETM uma performance aceitável. Entre estas características podemos identificar a existência de diversos ponteiros entre as diferentes tabelas para aumentar a velocidade de ativação e/ou desativação dos gatilhos. As ações são executadas em processos independentes que se comunicam com o processo do SGBD. Maiores detalhes do ETM são encontrados em ([Dit86b],[Kot88]).

3.4.3 KRISYS

KRISYS (Knowledge Representation and Interface SYStem) é um sistema gerenciador de bases de conhecimento, projetado e desenvolvido na Universidade de Kaiserslautern. Este sistema integra conceitos da área de Inteligência Artificial com os conceitos dos Sistemas de Banco de Dados para proporcionar, de forma mais adequada, os suportes necessários e desejados para a representação, manipulação e o gerenciamento de conhecimento. O modelo de conhecimento utilizado oferece um espectro rico de construtores que permitem uma representação precisa de domínio da aplicação [Mat88].

O modelo de conhecimento do KRISYS, KOBRA (KRISYS Object-Centered RepresentAtion), oferece uma representação orientada a objetos. Isto significa que, qualquer elemento, no domínio da aplicação, é expresso como um objeto do modelo KOBRA, chamado *esquema*, no qual são integrados os aspectos descritivos, funcionais e organizacionais do mundo real. Desta forma, cada entidade do universo da aplicação corresponde diretamente a um objeto da base de conhecimento. Cada esquema, que de forma geral é análoga a um frame ou a qualquer outra unidade em outros sistemas de representação, pode conter atributos para descrição da entidade do mundo real.

A filosofia que sustenta este protótipo está baseada em três classes de requisitos essenciais. Em primeiro lugar, os SGBCs devem satisfazer os requisitos de suas aplicações. Em segundo lugar, eles devem ter suportes para as

necessidades dos projetistas de bases de conhecimento, que representam um papel muito importante neste contexto. E finalmente, para um gerenciamento eficiente de conhecimento devem ser considerados alguns aspectos de implementação, tais como estruturas de dados e algorítmos. Assim, um SGBC deve ter características provenientes dos três pontos de vista diferentes. Desta forma, a arquitetura do SGBC é dividida em três níveis diferentes: Nível da Implementação, Nível da Engenharia e o Nível da Aplicação, que respectivamente suportam cada uma das classes de requisitos acima mencionadas (Figura 3.4).

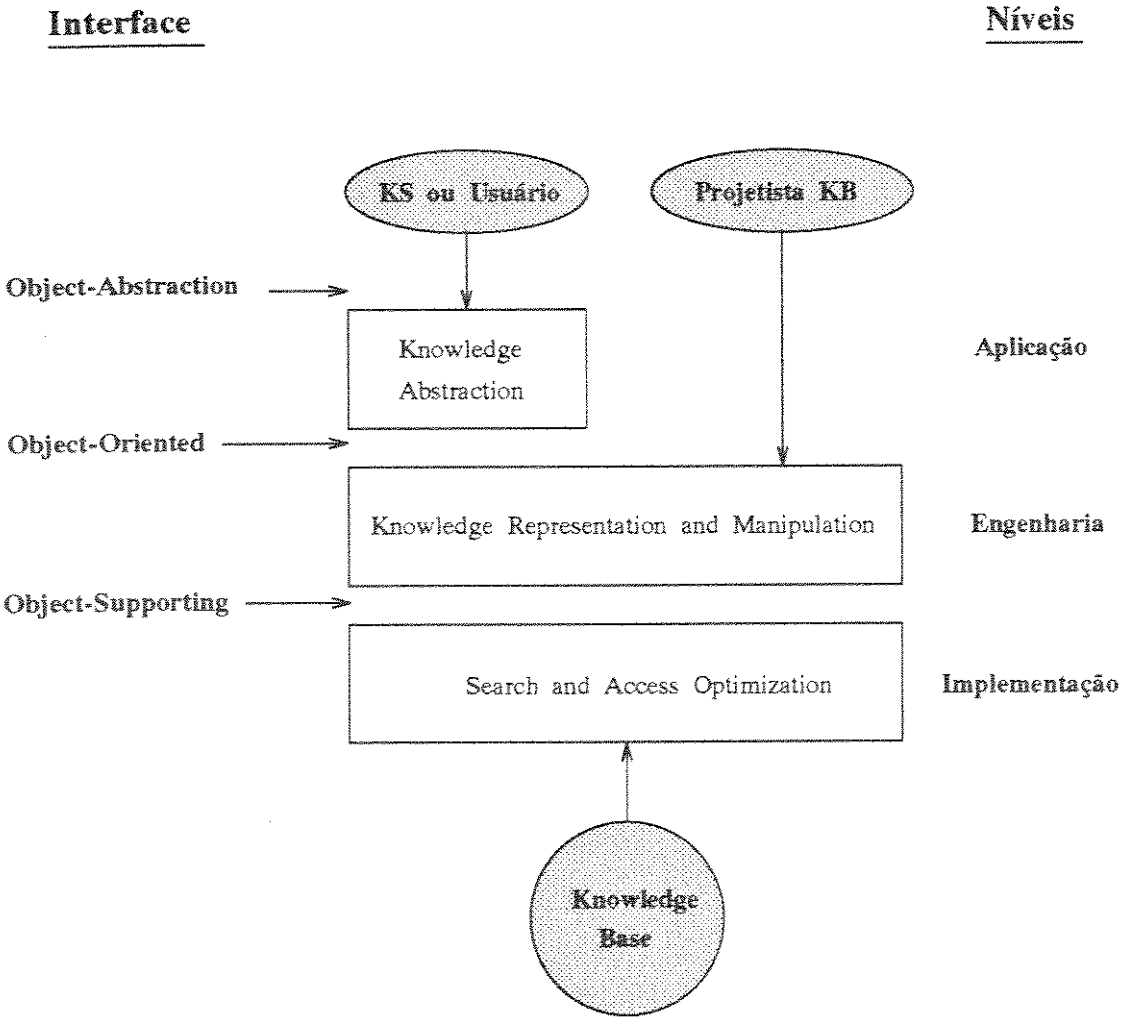

Figura 3.4 - Arquitetura sugerida em (Mat88) para um SGBC

Visando a *Independência de conhecimento* {Mat88} (análoga a independência de dados em BD), o Nível da Aplicação trata o conhecimento de uma maneira abstrata, independente das questões de eficiência do Nível da Implementação ou das questões de modelagem do Nível da Engenharia. Portanto, a nível da interface de "object-abstraction" (Figura 3.4) a base de conhecimento pode ser comparada com um tipo abstrato de dado que interage com o usuário final ou com o Sistema de Conhecimento, através de um conjunto de operações específicas. Assim, a maneira pela qual o conhecimento é representado ou modificado está oculta do Sistema de Conhecimento. Por esta razão, não pode haver distinção entre o conhecimento explicitamente armazenado e o conhecimento intencionalmente adquirido.

Em outras palavras, é responsabilidade do Nível da Aplicação selecionar estruturas simbólicas apropriadas para representar o conhecimento, e também selecionar os mecanismos apropriados de recuperação ou de dedução para responder às perguntas, bem como assimilar novos conhecimentos, ambos expressos em forma de uma linguagem de consulta oferecida pela interface de "object-abstraction" (KOALA - KRISYS Object Abstraction LAnguage {Deb89a}).

O Nível de Engenharia é focalizado do ponto de vista dos projetistas de base de conhecimento. Aqui, o SGBC é visto em termos de como o conhecimento de uma aplicação pode ser representado, organizado e manipulado. Em outras palavras, neste nível trata-se dos aspectos de descrição, organização e operação de um modelo de representação de conhecimento, o qual é utilizado pelos projetistas através da interface do Nível de Engenharia (object-oriented interface).

Neste nível, também são propostos mecanismos de verificação das restrições de consistência. A fim de definir restrições, neste nível, o projetista da base de conhecimento deve ser capaz de definir procedimentos que são associados a atributos de objetos. Posteriormente estes procedimentos serão automaticamente ativados antes ou depois que cada atributo é acessado ou atualizado, mantendo a base de conhecimento em um estado consistente ({Deb90a},{Deb90b}).

O Nível da Implementação tem como objetivo cobrir, com eficiência, as tarefas ligadas ao armazenamento de conhecimento e, consequentemente, dar

suporte aos dois níveis. Por esta razão, a maioria das características deste nível estão relacionadas às questões tradicionais dos SBDs, porém, aplicadas para bases de conhecimento, tais como, estruturas de armazenamento, técnicas de busca, eficiência de acesso, controle de concorrência e mecanismos de recuperação.

A arquitetura do KRISYS é dividida em três módulos diferentes, que refletem os aspectos dos três níveis discutidos acima (Figura 3.5 (Mat89)). O módulo de gerenciamento e processamento de consulta (Query-Processing Manager - QPM) corresponde ao Nível da Aplicação. Para implementar o Nível da Engenharia foi escolhido o KRISYS Frame System (KFS), que suporta a interface para o projetista da base de conhecimento e para o módulo QPM (object-oriented interface). Os requisitos do Nível da Implementação são preenchidos através do módulo de Working-Memory and Context Manager (WMCM) e um núcleo de SGBD. O WMCM inclui mecanismos que garantem a localidade das aplicações, isto é, a preservação do processamento de objetos da aplicação muito próximo à própria aplicação. O núcleo do SGBD é responsável pelo armazenamento e pelo gerenciamento da base de conhecimento.

O Nível da Aplicação do KRISYS pode ser melhor caracterizado através da descrição de sua interface, a linguagem KOALA, que define todas as interações entre um usuário ou uma aplicação e o sistema. Estas interações podem ocorrer através de dois tipos de operações: ASK e TELL.

A operação ASK é utilizada para questionar o KRISYS se uma expressão formulada é verdadeira, o que resulta na recuperação de um conhecimento. O comando ASK é composto de duas partes: uma projeção e uma seleção (ou qualificação). A parte de qualificação especifica a expressão a ser examinada, a qual é estruturada de acordo com cálculos de predicados de primeira ordem ($\neg$, $\vee$, $\wedge$, $\exists$, $\forall$, etc.).

A operação TELL serve para informar novos fatos e conhecimentos ao KRISYS. Assim, afirma-se que uma expressão formulada é verdadeira, então, a base de conhecimento se modifica de forma apropriada. Este comando também contém duas partes: a parte assertiva e a parte seletiva. A primeira parte contem a especi-

ficação das afirmativas e é similar à parte seletiva, porém, é muito mais restrita. Por exemplo, não é permitido formular assertivas usando conectores lógicos. Basicamente, existe um número limitado de assertivas pré-determinadas a serem especificadas, que são interpretadas independentemente. Maiores detalhes da linguagem KOALA se encontram, principalmente, em ([Mat88],[Mat89] e [Deb89a]).

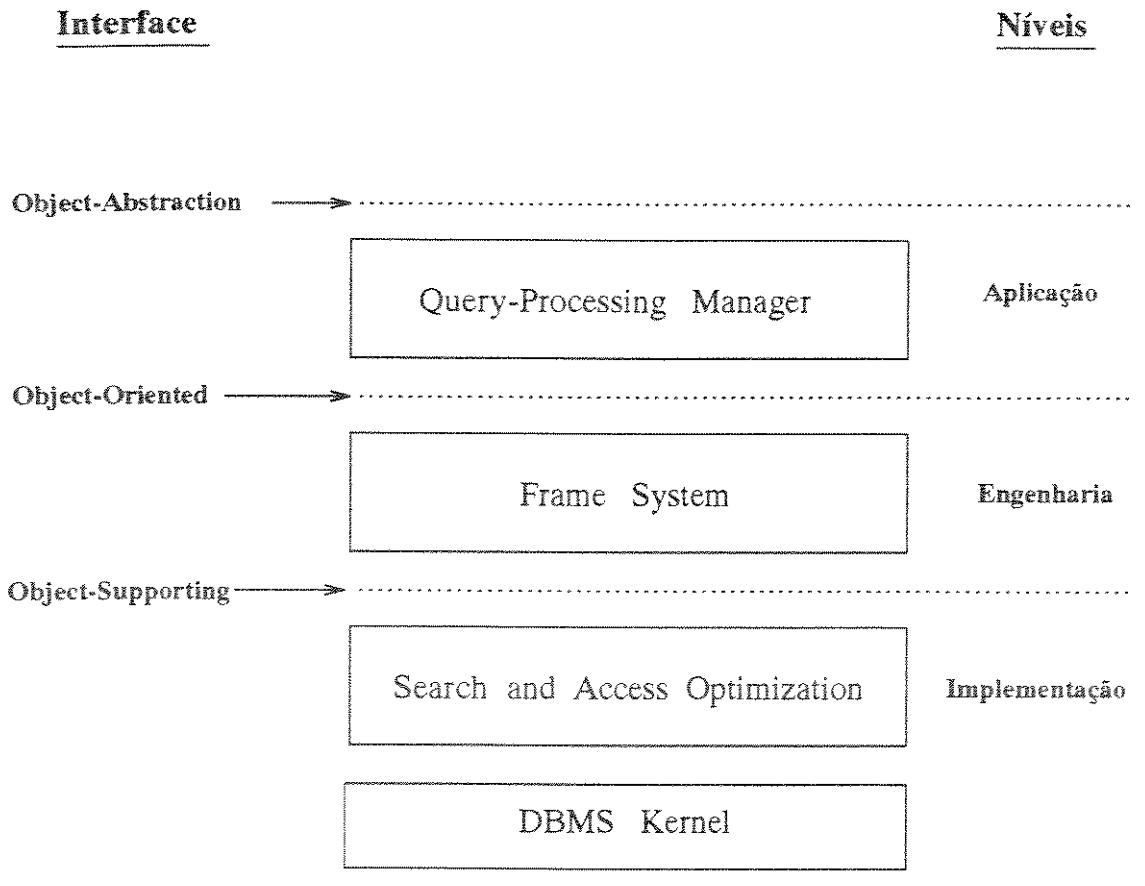

Figura 3.5 - A Arquitetura do Sistema KRISYS [Mat89]

O principal objetivo do WMCM é reduzir o número de chamadas ao SGBD e reduzir o caminho de acesso aos objetos da base de conhecimento, permitindo que a aplicação referência os objetos diretamente, sempre que possível. Isto é realizado através do armazenamento temporário dos objetos necessários em uma estrutura especial de memória principal chamada "memória de trabalho" (working-memory). Assim, não há necessidade de extrair os objetos da base de conhecimento, através do núcleo do SGBD, toda vez que a aplicação tenta acessá-los. A "memória de trabalho" é, portanto, uma forma de "buffer" que proporciona um acesso muito rápido aos objetos armazenados na base de conhecimento.

O núcleo de SGBD utilizado em KRISYS é o PRIMA ([Dad86],[Har87]), um protótipo de implementação do modelo MAD (Molecule-Atom Data model) [Pau87], que permite definição e manipulação dinâmica de objetos. Desenvolvido para suportar as aplicações não convencionais, o PRIMA dispõe de mecanismos de armazenamento para uma variedade de objetos com tamanhos diferentes, além de técnicas flexíveis de representação e acesso e características básicas de integridade, tais como, mecanismos de bloqueio e recuperação.

3.4.3.1 Controle de Integridade Semântica em KRISYS

Diversos conceitos de abstração, como classificação, generalização, associação e agregação são suportados pelo KOBRA, o modelo de conhecimento do KRISYS. Cada um destes conceitos é representado através de um par específico de atributos simétricos do tipo "slot" (por exemplo, "subclass-of" e "has-subclasses" para representação da generalização). Os atributos do tipo "slot" expressam as propriedades de um objeto ou os seus relacionamentos com os outros objetos. Outro tipo de atributo, chamado "method", é usado para descrever os aspectos operacionais e o comportamento dos objetos. Desta forma, no caso de modificações na hierarquia dos relacionamentos (generalização, associação, agregação, etc.), a sua simetria, expressa através dos "slots", é automatica-

mente mantida pelo sistema, ou seja, a cláusula inversa correspondente a cada par de "slot" é automaticamente atualizada (Deb90b).

Além destas restrições de integridade, impostas pelo KOBRA e representadas através dos "slots", outras restrições de integridade, associadas com os valores dos atributos, como a cardinalidade e o domínio, também são suportadas pelo KRISYS. Estas restrições são especificadas através de duas cláusulas, utilizadas na definição de atributos: "cardinality" e "possible-values". Se uma operação do usuário violar alguma destas restrições ela é rejeitada.

O KRISYS permite também definição de outras restrições mais complexas. que podem envolver cálculos ou dependências entre vários atributos de um ou mais objetos. Estas restrições, classificadas anteriormente como restrições de comportamento, são especificadas através de um mecanismo chamado "*daemon*" (Deb90b).

O mecanismo de "daemon" é similar, em muitos dos seus aspectos, ao mecanismo de gatilho em sistemas de banco de dados. Cada "daemon" é um procedimento, escrito em alguma linguagem de programação de alto nível (por exemplo, Common Lisp), contendo as instruções especificadas pelo usuário, que definem uma restrição relacionada a atributos de objetos, bem como as condições a as ações desta restrição.

Porém, do ponto de vista conceitual, existem várias diferenças entre os gatilhos e os "daemons", por exemplo, um gatilho é capaz de reagir somente a um único determinado evento (vários gatilhos podem ser necessários para manter uma restrição de integridade), enquanto que um "daemon" reune todos os eventos que envolvem um atributo e as suas ações em um único objeto conceitual que guarda todas as operações realizáveis sobre o atributo (Deb90b).

Outro mecanismo de integridade envolve construtores denominados "methods". Semelhante ao conceito de transação em SGBDs, em KRISYS um "method" é visto como uma unidade de integridade. Esta noção é fundamentada por mecanismos que permitem que as ações de um "method" sejam desfeitas, como uma forma de reação no caso de uma violação da restrição. Como a ativação dos "methods" pode ser arbitrariamente aninhada (um "method" pode ativar outros "methods"), diferentes

níveis de consistência podem ser definidos. Semelhante às transações aninhadas, ao desfazer os efeitos de um "method" , serão canceladas também todas as ações relativas aos outros "methods" chamados por ele. Em casos onde este procedimento não é desejado, o mecanismo pode ser alterado, introduzindo uma operação dentro do "method", permitindo que a violação da restrição seja tratada de uma forma mais apropriada.

Apesar de todos estes mecanismos, em algumas situações é desejável uma representação mais declarativa das restrições de integridade. Nestes casos, o KRISYS oferece mecanismos que possibilitam uma representação mais apropriada destas restrições em forma de *regras*. As regras podem ser utilizadas de duas formas: ou a sua ativação resulta em dedução de uma informação intencional (através de um processo de retrocessos sucessivos) ou a sua ativação modifica o estado da base de conhecimento, usando um mecanismo de sucessivos avanços. Ambas as alternativas podem ser empregadas para preservar a consistência da base de conhecimento. Maiores detalhes sobre o controle de consistência em KRISYS são descritos em [Deb90b].

3.5 COMENTÁRIOS FINAIS

Com este capítulo completamos a revisão bibliográfica iniciada nos capítulos anteriores, situando o leitor nas atividades em desenvolvimento no contexto de controle de integridade em SGBDs convencionais e não-convencionais.

No contexto de sistemas de banco de dados convencionais, foram abordados diversos mecanismos tradicionais de controle de integridade, consistência e segurança, tais como, controle de transações e "logs", controle de concorrência através de serialização de transações e outros mecanismos.

Na área de aplicações não-convencionais, foram destacadas as aplicações de banco de dados em projetos de engenharia e em sistemas de base de conhecimento. Tentou-se mostrar a necessidade que estas aplicações têm de restrições de consistência adequadas. Além disso, foram analisadas as características particulares destas restrições em cada área de aplicação.

Nesta tentativa, as restrições de consistência e suas características em ambientes de projeto foram abordadas de forma mais detalhada pelo fato de estarem relacionadas diretamente com o objetivo deste trabalho. No próximo capítulo situa-se a principal contribuição deste trabalho, através da especificação e implementação do STR, Subsistema de Tratamento de Restrições do GERPAC.

Sois todos folhas de uma mesma árvore e gotas de um mesmo mar."

Bahá'u'lláh

CAPÍTULO 4

TRATAMENTO DE RESTRIÇÕES NO SGBD GERPAC

4.1 INTRODUÇÃO

As dificuldades em utilizar SBDs convencionais, projetados para atender as necessidades da área comercial, nas aplicações de engenharia tem estimulado o desenvolvimento de novos modelos de dados ({Cro87},{Del87},{Mat89}) e arquiteturas de SBD ({Mat88},{McC89},{Sow88}) para atenderem as exigências das aplicações desta área, particularmente em ambientes de projeto ({Ric87a}, {Sto86}, {Dit86a}).

Uma das características e vantagens marcante dos sistemas de banco de dados é o controle de consistência e de integridade ({Elm89},{Kor86},{Smi87}), oferecido pela maioria dos SGBDs. Entretanto os mecanismos existentes que proporcionam esta facilidade não são suficientes nem totalmente adequados para as aplicações não-convencionais ({Ban85},{Dit85}{McC89}{Pri89}).

Diversas frentes de pesquisa, principalmente na área de Banco de Dados, têm dedicado grandes esforços no desenvolvimento de novos métodos e mecanismos de controle de consistência para sistemas de banco de dados utilizados nas aplicações de engenharia ({Deb89b},{Del87},{Kat83},{Kai88a},{Kot88}). Em particular,

têm-se destacado o desenvolvimento de sistemas ou subsistemas de controle e gerenciamento de restrições ou regras ({Kot88},{Dit86},{Deb89a}, {Nou91}).

Uma proposta, em desenvolvimento no Departamento de Engenharia de Computação e Automação Industrial (DCA) da Faculdade de Engenharia Elétrica (FEE) da UNICAMP, engloba o GERPAC, um SGBD para aplicações de projeto, e um sistema de armazenamento associativo, o UNICOSMOS, utilizado como o núcleo do GERPAC. Nos próximos itens serão relacionadas as principais características destes dois sistemas e apresentada uma proposta de tratamento de restrições no contexto GERPAC/UNICOSMOS.

4.2 AMBIENTE GERPAC/UNICOSMOS

Este ambiente é constituido por três componentes principais: o Sistema de Armazenamento Associativo UNICOSMOS (UNICAMP Object Storage Management System), o SGBD GERPAC (Sistema de GERência de Banco de Dados para Aplicações PAC) e o modelo de dados MER/PAC (Modelo Entidade Relacionamento para Aplicações PAC). O UNICOSMOS, um sistema independente de um modelo de dados, como veremos a seguir, é utilizado como núcleo para o SGBD GERPAC, implementado sobre os conceitos do modelo MER/PAC, uma extensão do Modelo Entidade Relacionamento para aplicações de PAC. A seguir são descritas, resumidamente, as principais características de cada um destes componentes.

4.2.1 UNICOSMOS

UNICOSMOS, originado do SIGA (Sistema de Gerência de Armazenamento Associativo {Ric87a}), é uma alternativa a outros sistemas associativos que manipulam objetos e conjuntos, como é o caso de CORAS {Enc 83}, a partir do qual

teve a sua implementação. Neste sistema, a flexibilidade de estruturas de armazenamento e a possibilidade de representação de mais de um tipo de informação por objeto permite representar uma quantidade de informações semânticas muito maior que a obtida em sistemas de armazenamento de dados convencionais (Ric87a). Este fato possibilita a utilização do UNICOSMOS em diversos tipos de aplicações distintas, em particular como núcleo de SGBDs.

O UNICOSMOS é um sistema voltado para a manipulação de objetos[1] (Ric87a), portanto, os seus elementos primitivos são constituidos de objetos. No contexto UNICOSMOS, cada objeto é uma estrutura definida pelo usuário para armazenamento de informações que representem as entidades do mundo real. Os objetos contém basicamente três tipos de informações:

nome - que permite sua identificação;

atributos - que caracterizam as propriedades de classe por eles representados;

ocorrências - que representam suas instâncias, cada uma delas com uma combinação de valores diferentes associados a atributos.

Os objetos do UNICOSMOS têm por função representar classes, que internamente descreverão os elementos do modelo suportado pelo SGBD implementado sobre o UNICOSMOS. Desta forma, este sistema se torna independente do modelo de dados permitindo que os objetos representem relações, tipos de registros, tipos de conjuntos, tipos de relacionamentos, tipos de entidades, dependendo do modelo utilizado. Os objetos do UNICOSMOS podem ser de três tipos:

* **Simples** - representa uma classe, à qual podem estar associados atributos;

* **Conjunto** - define agrupamentos de outros objetos, que podem ser simples, conjuntos ou mesmo relações;

[1] O nome objeto aqui é usado em contexto próprio do UNICOSMOS, como descrito em seguida.

* **Relação** - permite estabelecer relacionamentos binários entre outros objetos que não sejam do tipo relação.

A nível do UNICOSMOS, as informações são armazenadas em diversos Arquivos Internos:

1. **Arquivo Interno de Nomes:** Este arquivo, a partir do qual os demais arquivos são acessados, constitui a interface de acesso entre o usuário e as informações armazenadas sobre cada objeto. Este acesso é estabelecido através do nome do objeto.

2. **Arquivo Interno de Objetos:** Este arquivo contém todas as informações armazenadas no sistema sobre cada objeto, como suas estruturas de atributos centralizando a comunicação com todos os demais arquivos do sistema.

3. **Arquivo Interno de Tríades:** O Arquivo de Tríades mantém informações sobre os relacionamentos binários (as tríades) entre os objetos.

4. **Arquivo Interno de Valores:** Constitui o depósito dos conjuntos de valores definidos para os atributos de objetos. Para cada atributo definido no Arquivo Interno de Objetos existe uma entrada no Arquivo de Valores. Os valores estão associados a ocorrências. A informação sobre quais valores estão associados a quais ocorrências também está armazenada neste arquivo.

5. **Arquivo Interno de Ocorrências:** Ocorrências são instâncias associadas a objetos. A cada ocorrência está associada uma entrada neste arquivo. Uma ocorrência de um objeto é definida através da associação de valores aos atributos deste objeto.

6. **Arquivo Interno de Índices:** Este arquivo foi proposto para permitir uma otimização no acesso a valores de atributos. Se um atributo é definido como indexado, o identificador associado a ele no Arquivo Interno de

Objetos indica não uma entrada para o conjunto de valores no Arquivo Interno de Valores, como no caso normal, mas sim uma entrada a uma estrutura de índice definida no Arquivo Interno de Índices.

7. **Arquivo Interno de Acesso:** O Arquivo Interno de Acesso define quais ocorrências, entre todas definidas no Arquivo Interno de Ocorrências, estão associadas a um objeto específico.

O UNICOSMOS oferece inúmeras funções para a definição e manipulação de suas primitivas de forma que o usuário não tenha necessidade de acessar a estrutura interna dos arquivos e manipular o Sistema de Paginação. Estas funções são definidas internamente em três níveis:

Nível Terciário: As funções deste nível constituem a interface externa do UNICOSMOS e englobam as funções de definição de bases de dados (*"Sistema de Arquivos"*) e as rotinas de definição e manipulação dos elementos primitivos, disponíveis ao usuário.

Nível Secundário: É o nível de visão lógica dos dados, onde são definidas e manipuladas as estruturas internas de dados referentes aos diversos Arquivos Internos do sistema.

Nível Primário: Contêm as funções que manipulam o nível físico. Estas funções definem a forma de acesso aos dados em disco, controlando também o sistema de paginação.

Outra característica do UNICOSMOS é a sua capacidade de controlar diversas bases de dados, onde cada uma delas constitue um Sistema de Arquivos. Esta é uma das necessidades básicas em aplicações PAC, onde é necessário haver o controle de informações a níveis global e local simultaneamente. Este fato permite a integração entre dados de diversas aplicações e é suportado por um dos Sistemas de Arquivo, o Catálogo de Projetos.

Uma das vantagens que o Catálogo de Projetos proporciona ao sistema é a possibilidade de controle de versões de projeto. Para isto, ele permite que a cada um dos Sistemas de Arquivos, sob seu controle sejam associadas diversas versões, cada uma constituindo uma base de dados distinta.

Para controlar as associações existentes entre duas versões são utilizados dois objetos relação. O primeiro especifica qual base de dados desempenha o papel de "geradora" e o segundo especifica o papel da base de dados "gerada". Todas as operações são realizadas através de funções UNICOSMOS que mantém atualizado o Catálogo de Projetos.

A partir de uma nova extensão do UNICOSMOS (Sil91), este sistema pode também suportar alguns conceitos do paradigma de orientação a objetos. São eles: *Mecanismos de Herança*, *Mecanismos de Abstração* (Classificação, Generalização, Agregação, etc.) e *Tipos Abstratos de Dados* (TAD).

4.2.2 Modelo de dados MER/PAC

O modelo de dados MER/PAC, proposto pelo grupo de Banco de Dados do DCA/FEE/UNICAMP em 1986 (Del87), é uma extensão ao Modelo de Entidade-Relacionamento (MER) proposto originalmente por Chen (Che76). Tanto o MER quanto o MER/PAC, permitem uma abordagem unificada para o processo de projeto de banco de dados, independente do modelo suportado pelo SGBD para a implementação a nível organizacional.

Uma das principais características do MER/PAC diz respeito ao suporte ao tratamento dos diferentes níveis do processo PAC, que podem ser encarados sob dois pontos de vista, os chamados micro-aspecto e macro-aspecto (Ric87a). A representação do aspecto micro do processo PAC envolve os dados referentes aos objetos e suas associações, enquanto que o aspecto macro corresponde ao controle que deve ser exercido para garantir a consistência entre os diversos níveis de processos de projeto, que são gerados a partir de um processo inicial. A

execução deste controle caracteriza uma Transação Macro, um conceito particular ao GERPAC, diferindo do conceito de transação em SBD convencionais.

Apesar de uma parte dos dados, utilizados para a representação do aspecto micro de processos PAC, possa ser representada através das primitivas originais do MER, tipos de entidades (objetos) e tipos de relacionamentos (associações) e informações associadas (atributos de entidades e relacionamentos), outros conceitos fundamentais em PAC não podem ser representados em MER, como o conceito de objeto complexo e de relacionamentos exclusivos, além de conceitos de abstração, destacando-se em particular os de Classificação, Generalização e Agregação.

Através das primitivas da proposta MER/PAC pode-se representar os dados envolvidos tanto no aspecto micro como no aspecto macro de processos PAC. Estas primitivas tentam solucionar as deficiências do Modelo Entidade-Relacionamento original (MER), permitindo o uso de diversos métodos de abstração para a representação de informações e a representação de características temporais da realidade modelada. As primitivas MER/PAC se resumem basicamente em: Tipos de Entidades, Tipos de Relacionamentos, Agrupamentos e Esquemas.

Tipos de Entidades representam classes que contém entidades, que são as unidades de informação do Modelo. Ao tipo de entidade podem ser associados atributos que definem as propriedades da classe. Um tipo de entidade pode ser regular, quando a existência de entidades componentes da classe é definida independentemente das entidades de outras classes, ou fraco, quando há dependência em relação a entidades de outras classes.

Tipos de Relacionamentos representam classes que contém relacionamentos, que associam elementos dos conjuntos ligados através do tipo do relacionamento. Atributos podem também estar associados ao tipo de relacionamento, definindo suas propriedades. Tipos de relacionamentos podem também ser regulares, quando a remoção de um relacionamento não afeta os elementos envolvidos na associação, ou fracos, quando a remoção de um relacionamento pode acarretar a remoção de parte dos elementos envolvidos na associação.

Agrupamentos contém conjuntos de elementos primitivos do MER/PAC (tipos de

entidades e/ou tipos de relacionamentos), permitindo diferentes níveis de abstração (classificação, generalização, agregação, etc.). A representação de tão diferentes conceitos através de uma mesma primitiva só é possível através da definição de um outro nível de informação a ela associada. Estas informações são as restrições associadas às primitivas MER/PAC. Através de restrições é possível especificar a função do agrupamento (agregação, generalização, objeto complexo, relacionamento exclusivo, etc.). Assim como tipos de entidades, agrupamentos podem ser também regulares ou fracos.

C-esquemas são os elementos primitivos que permitem a representação do macro-aspecto do projeto. Através de c-esquemas é possível a representação de versões do projeto. Assim, cada representação de um micro-aspecto de projeto (uma expansão), em conjunto com os dados operacionais associados (versão), representa uma ocorrência do c-esquema ao qual a representação está associada. Esquemas também podem ser classificados quanto a sua dependência em regulares ou fracos.

4.2.3 GERPAC

GERPAC, Sistema de Gerência de Banco de Dados para aplicações PAC, é uma proposta de um SGBD, implementado sobre o UNICOSMOS, que oferece mecanismos e facilidades voltadas para as aplicações PAC (Mag89). Esta proposta baseia-se em um modelo de dados mais próximo ao nível conceitual, a fim de proporcionar tais mecanismos e facilidades. Assim, este SGBD foi projetado para suportar os elementos primitivos do modelo MER/PAC (Del88), com conceitos de transação, integridade e consistência adequados com o seu propósito e essencialmente distintos dos SGBDs convencionais.

Os elementos primitivos do GERPAC são aqueles referentes ao MER/PAC, ou seja, Tipos de Entidades, Tipos de Relacionamentos, Agrupamentos e Esquemas. Além destes elementos primitivos, uma ferramenta importante na representação das

informações no GERPAC é a possibilidade de especificação de Restrições, que podem ser associadas a qualquer um dos elementos primitivos do GERPAC.

As restrições assumem papel fundamental na descrição das características semânticas adicionais àquelas representáveis diretamente pelo modelo, como ocorre no caso de agrupamentos. Podem também ser utilizadas para suportar diversos aspectos de controle de consistência, como será destacado posteriormente.

A principal característica do GERPAC que define sua adequação a aplicações PAC é a possibilidade de se gerenciar o processo de projeto (o macro-aspecto) através das mesmas ferramentas com que se manipulam os elementos do projeto (seu micro-aspecto), o que permite uma continuidade de visão e tratamento do usuário.

4.2.4 Mapeamento GERPAC/UNICOSMOS

No GERPAC, o conceito de classe primária (definida a partir de unidades de informação e não a partir de outras classes) é representado através de dois elementos primitivos, o tipo de entidade e o tipo de relacionamento. Estes dois elementos são representados no UNICOSMOS como objetos simples, sendo suas instâncias representadas por ocorrências do objeto simples correspondente.

Os objetos Relação e Tríades do UNICOSMOS permitem representar os relacionamentos do GERPAC, do tipo binário. A representação de relacionamentos não binários é possível através de um elemento específico (um objeto relação) denominado "papel". O papel não tem atributos definidos pelo usuário, mas apenas os pré-definidos pelo sistema, que permitem representar o mapeamento e a dependência dos objetos associados.

Agrupamentos são mapeados como objetos conjunto do UNICOSMOS. A definição de como são constituídas as ocorrências de cada agrupamento em particular não é pré-definida pelo sistema, sendo suportada através de definição de restrições associadas a cada agrupamento.

C-esquemas têm representação diferenciada de acordo com o nível em que o usuário está trabalhando. Se um usuário está definindo a representação de um c-esquema, então este c-esquema será encarado como um Sistema de Arquivos do UNICOSMOS, uma base de dados distinta. Ao definir uma nova expansão, uma nova base de dados será criada, e o mesmo ocorre para a criação de novas versões de cada expansão. Em outras palavras, cada instância de um c-esquema corresponde a uma base de dados UNICOSMOS.

4.2.5 Transações no Contexto GERPAC/UNICOSMOS

Devido às características peculiares das aplicações de PAC, o conceito de transações no contexto GERPAC/UNICOSMOS difere do conceito tradicional de transações em SGBDs convencionais. Aqui são considerados três tipos diferentes de transações: *transações micro*, *transações meta-micro* e *transações macro* ({Olg90},{Olg90a}).

4.2.5.1 Transação Micro

Uma transação micro, no contexto GERPAC/UNICOSMOS, é vista como uma transação tradicional em SGBDs convencionais. Em outras palavras uma transação micro consiste em uma sequência de operações cuja execução mantém a consistência da base de dados. Todas as propriedades de transações, descritas no capítulo 2, são válidas para esta categoria de transações do GERPAC.

4.2.5.2 Transação Meta-Micro

O conceito de transação meta-micro pode ser visto como uma transação de longa duração que não contém nenhuma transação micro, ou como uma transação de longa duração que aninha diversas transações micro. Quando uma transação meta-micro falha, a base de dados é restaurada para o estado em que se encontrava antes do começo da transação micro corrente e não para o estado anterior da base de dados, no início de execução da transação meta-micro.

4.2.5.3 Transação Macro

A transação macro corresponde a mecanismos de controle e recuperação das falhas semânticas, em analogia à transação micro que corresponde às falhas de sistema. Assim, as transações macro estão associadas ao controle de consistência, ou seja, elas verificam se restrições de consistência a nível de objetos de projeto são observadas.

4.3 SUBSISTEMA DE TRATAMENTO DE RESTRIÇÕES DO GERPAC

Neste ítem situa-se a principal contribuição do presente trabalho. Inicialmente são analisadas as características das restrições de consistência no contexto GERPAC/UNICOSMOS. Em seguida é analisado o Arquivo Interno de objetos do UNICOSMOS, que será utilizado no suporte à definição da estrutura do STR. Ao final o STR é especificado, sendo proposta uma solução de implementação que

busca aproveitar, ao máximo, os recursos existentes no ambiente GERPAC/UNICOSMOS, contribuindo para a simplicidade do sistema e facilitando o uso do GERPAC nas aplicações de engenharia.

4.3.1 Restrições no Contexto GERPAC/UNICOSMOS

Restrições no contexto GERPAC possuem uma semântica mais ampla do que nos SGBDs convencionais. No GERPAC restrições desempenham um papel importante na representação da informação, inclusive na definição de ocorrências em agrupamentos e na manutenção de consistência entre vários c-esquemas. Isto ocorre porque as restrições podem ser definidas tanto sobre objetos, através de seus atributos, como também, sobre as suas instâncias, envolvendo objetos distintos (Nou91).

No GERPAC, o conceito de Restrições assemelha-se ao conceito de operações sobre um conjunto de primitivas, sendo no entanto mais flexível. Restrições podem ser associadas a objetos (tipos de entidades e tipos de relacionamentos) e também a agrupamentos e c-esquemas. No caso de objetos, as restrições podem ser definidas sobre seus atributos, especificando aspectos individuais, como por exemplo, faixas de valores válidos para cada atributo, ou interrelacionamentos entre valores de diferentes atributos. Particularmente, para tipos de relacionamentos elas podem ser utilizadas para representar condições que permitam definições e/ou exclusões de ocorrências. No entanto, é em agrupamentos que as restrições assumem um papel fundamental.

A nível (interno) do UniCOSMOS, as restrições são implementadas e tratadas como atributos (de um objeto), podendo ser definidas, atualizadas, consultadas ou removidas. Este fato possibilita uma definição e manipulação dinâmica de restrições.

Uma das necessidades marcantes das aplicações de PAC é justamente uma definiçao dinâmica de restrições de consistência. Pois, em um processo de

projeto, por causa de sua natureza iterativa e tentativa, o usuário não dispõe de informações suficientes para definir, no momento de sua especificação, todas as restrições necessárias.

As restrições são definidas gradualmente, acompanhando a evolução do projeto, dando condições ao usuário de verificar a consistência dos dados em cada etapa do projeto. Sem a facilidade de definição dinâmica de restrições, o usuário pode ficar obrigado a modificar a especificação do projeto, em detrimento à flexibilidade do processo.

Outra necessidade das aplicações de PAC, suprida pelo GERPAC, é a definição e a manipulação de grande número de restrições. No GERPAC, como veremos posteriormente, cada restrição é armazenada como uma instância de um conjunto de atributos. Desta forma, o usuário tem a total liberdade em declarar, para cada objeto, tantas restrições quantas desejar. O *Subsistema de Tratamento de Restrições* do GERPAC possibilita uma manutenção dinâmica e fácil de todas as restrições definidas no sistema.

Como foi visto anteriormente, através da primitiva *c-esquema* do MER/PAC o usuário pode modelar e representar as características globais dos processos de projeto. Da mesma forma, as características particulares de cada objeto do projeto podem ser representadas através da declaração de cada *objeto simples* do MER/PAC e da declaração de seus atributos.

É também importante notar que dentro do GERPAC as restrições podem ser definidas a nível de qualquer um dos objetos do MER/PAC. Portanto, é possível definir tanto restrições locais como globais. Desta forma, o usuário tem condições de definir, por exemplo, eventuais restrições de composição que não foram cobertas pelo MER/PAC. Ou então, definir novas restrições de composição sobre qualquer esquema da base de dados.

Torna-se necessário lembrar que nas aplicações de PAC as restrições globais podem não estar satisfeitas durante um longo período de tempo. Portanto, o GERPAC permite a existência de inconsistências temporárias durante um processo de projeto. Porém, após o acompanhamento de vários estágios de consistências locais, o usuário pode desejar solicitar, a seu critério, a verificação de

algumas restrições que possam garantir um estado de consistência global, para a base de dados.

A verificação de restrições por parte do usuário, e em momentos determinados por ele, constitui uma das formas mais frequentes e usuais da verificação de consistência em SGBDs utilizados nos ambientes de PAC.

Outra característica das restrições no contexto GERPAC é a sua representação algorítmica. Em outras palavras, uma restrição é um programa escrito em alguma linguagem de alto nível, no qual são utilizadas as operações e as funções do GERPAC. Este programa, depois de ser compilado, é executado no momento em que se deseja verificar a restrição. Dependendo do resultado da verificação, serão realizadas as ações apropriadas, que são determinadas e codificadas no próprio programa que constitui a restrição.

Desta forma, o GERPAC proporciona flexibilidade para o usuário definir restrições adequadas de acordo com cada tipo de aplicação. Além disso, o usuário tem a liberdade de escolher a forma e a ferramenta mais adequada para o desenvolvimento do algoritmo correspondente a cada restrição.

4.3.2 Suporte a Implementação de Restrições

Como foi mencionado anteriormente, a nível do UniCOSMOS as informações são armazenadas em diversos **Arquivos Internos** ({Fip88},{Ric87a}). No GERPAC, as restrições são consideradas informações relacionadas aos objetos do sistema, sendo definidas como atributos destes objetos. Consequentemente, estas informações são armazenadas no **Arquivo Interno de Objetos**.

Todas as informações de cada objeto, inclusive as de restrições, são armazenadas em células (estruturas de 14 bytes), que ligadas entre si constituem as listas internas do Arquivo Interno de Objetos (Figura 4.1). Estas listas são cinco:

1. **Lista de Nomes:** Contém todos os nomes definidos para o objeto, ou seja, o nome inicial e os sinônimos definidos posteriormente;

2. **Lista de Atributos:** Contém diversas máscaras de atributos, definidas pela aplicação. Mantém também a informação sobre a quantidade de ocorrências associadas a cada máscara de atributos. As máscaras de atributos contém todas as informações sobre os atributos e as restrições definidos no objeto.

3. **Lista de Elementos de Conjuntos:** Esta lista armazena os identificadores internos dos objetos que pertencem ao objeto conjunto que contém esta lista;

4. **Lista de Conjuntos Associados ao Objeto:** Nesta lista são armazenados os identificadores internos dos objetos tipo *conjunto*, aos quais o objeto que contém esta lista pertence. Através desta lista, e da lista de elementos de conjuntos, são implementados os conceitos de hierarquia de objetos, tais como, generalização, agregação, agrupamento, etc. (Sil91);

5. **Lista de Entradas ao Arquivo de Tríades:** Esta lista armazena apontadores semelhantes aos identificadores internos, que são definidos para o Arquivo de Tríades, definindo a posição naquele arquivo onde estão armazenadas as informações sobre as tríades nas quais o objeto tem participação.

O *cabeçalho* das listas internas do Arquivo de Objetos contém informações sobre a quantidade de elementos em cada uma destas listas. A flexibilidade na definição do tamanho de cada lista é garantida através de uma estrutura intermediária (os *subcabeçalhos*) que, além dos apontadores ao início de cada lista na página, permitem armazenar apontadores a uma possível continuação da lista em outra página (Figura 4.2).

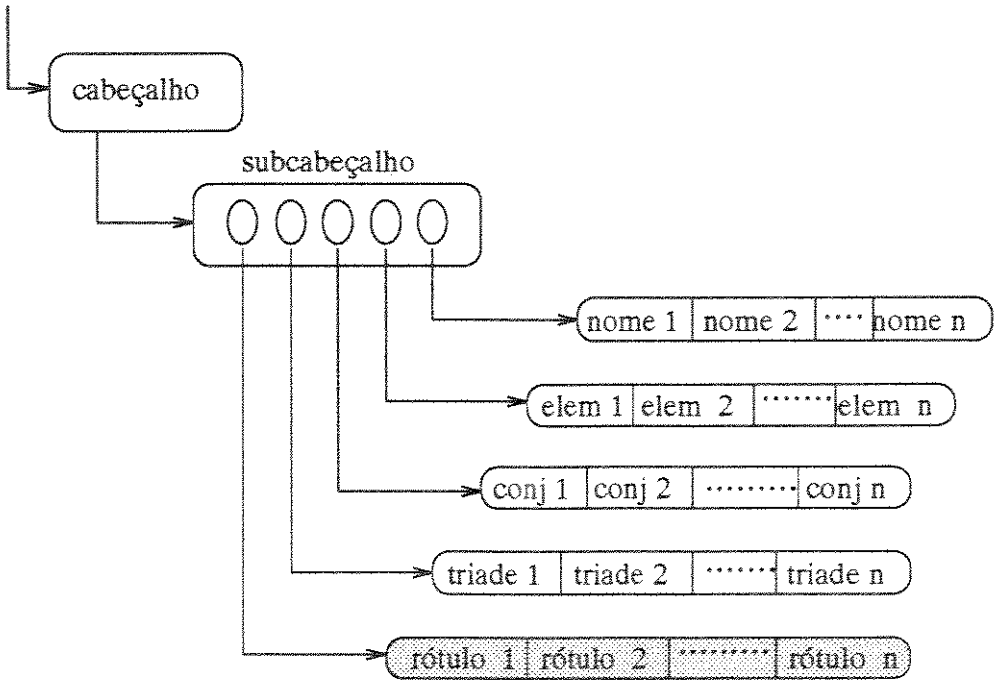

Figura 4.1 - Listas Internas do Arquivo de Objetos

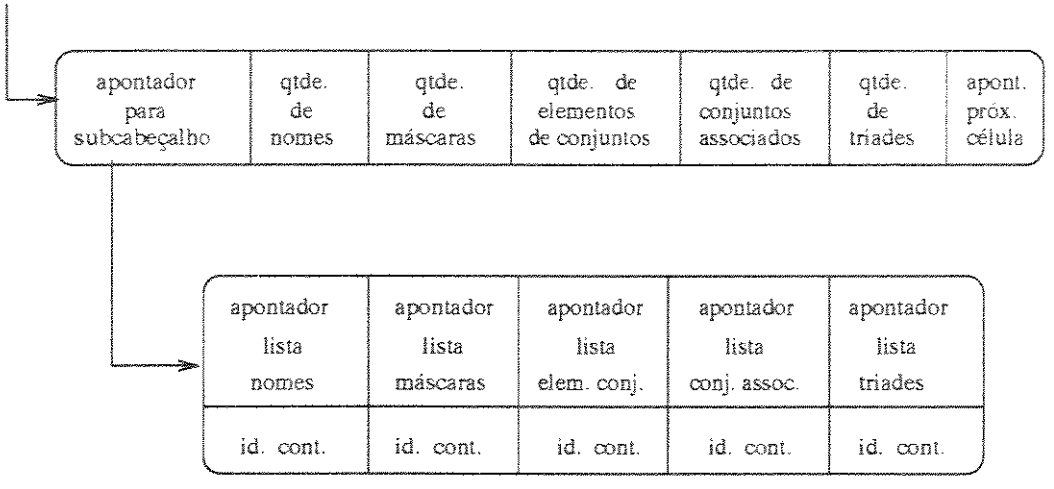

Figura 4.2 - Estrutura do Cabeçalho e do Subcabeçalho

4.3.2.1 Máscaras de Atributos

A criação de uma lista de máscaras para a lista de atributos proporciona ao GERPAC a facilidade de definir diversos conjuntos de atributos para um mesmo objeto. Desta forma, um único objeto pode assumir diversos papeis, com características diferentes, para aplicações distintas.

As características que um objeto assume para uma determinada aplicação são representadas através de atributos normais do objeto, definidos em uma máscara. Este mesmo objeto pode apresentar outras características, totalmente distintas, para outra aplicação, representadas através de atributos definidos em outra máscara. As máscaras também podem ser utilizadas para a definição de restrições de consistência em cada objeto, pois, como iremos observar posteriormente, as restrições de consistência sobre cada objeto são armazenadas como instâncias de um conjunto definido de atributos deste objeto.

Cada máscara de atributos possui um rótulo, pelo qual é identificada dentro do sistema. Por isso, a lista de máscaras também é chamada de *lista de rótulos*. Cada rótulo é utilizado para definir um conjunto de atributos que representam características distintas do objeto. Assim, no rótulo sempre existe a informação que determina a sua função. Esta informação, armazenada no *subrótulo*, indica se o rótulo é utilizado para definir atributos normais do objeto ou para definir restrições sobre o objeto. O GERPAC tem mecanismos para saber quais rótulos do objeto estão utilizados para definição de atributos normais e quais estão utilizados para definição de restrições.

4.3.2.2 A Lista de Máscaras de Atributos

A Figura 4.3. mostra a estrutura da Lista de Máscaras de Atributos. Como pode ser observado, cada célula da Lista de Máscaras possui os seguintes dados, de 2 bytes cada:

- apontador para o subrótulo;
- rótulo da máscara, que permite identificar cada conjunto de atributos;
- quantidade de atributos definidos na máscara (no caso em que a máscara é utilizada para a definição de restrições, este valor é igual a 5 - ver item 4.3.2.3);
- apontador para a página de continuação da lista de atributos;
- apontador para a máscara na página de continuação;
- apontador para o início da lista de atributos;
- apontador para a próxima célula da lista de máscaras.

O *subrótulo* é constituido de uma única célula, com as seguites informações:

- apontador para o Arquivo de Acesso;
- quantidade de ocorrências definidos em cada máscara (quando a máscara é utilizada para a definição de restrições, esta informação representa o número de restrições definidas naquela máscara);
- código interno do último atributo da lista de atributos;
- identificador da função da máscara, que especifica se a máscara é utilizada para a definição de atributos normais ou para a definição de restrições do objeto;
- identificador do gatilho, quando a máscara é utilizada para a definição de restrições;
- 2 bytes não utilizados;
- apontador para a próxima célula.

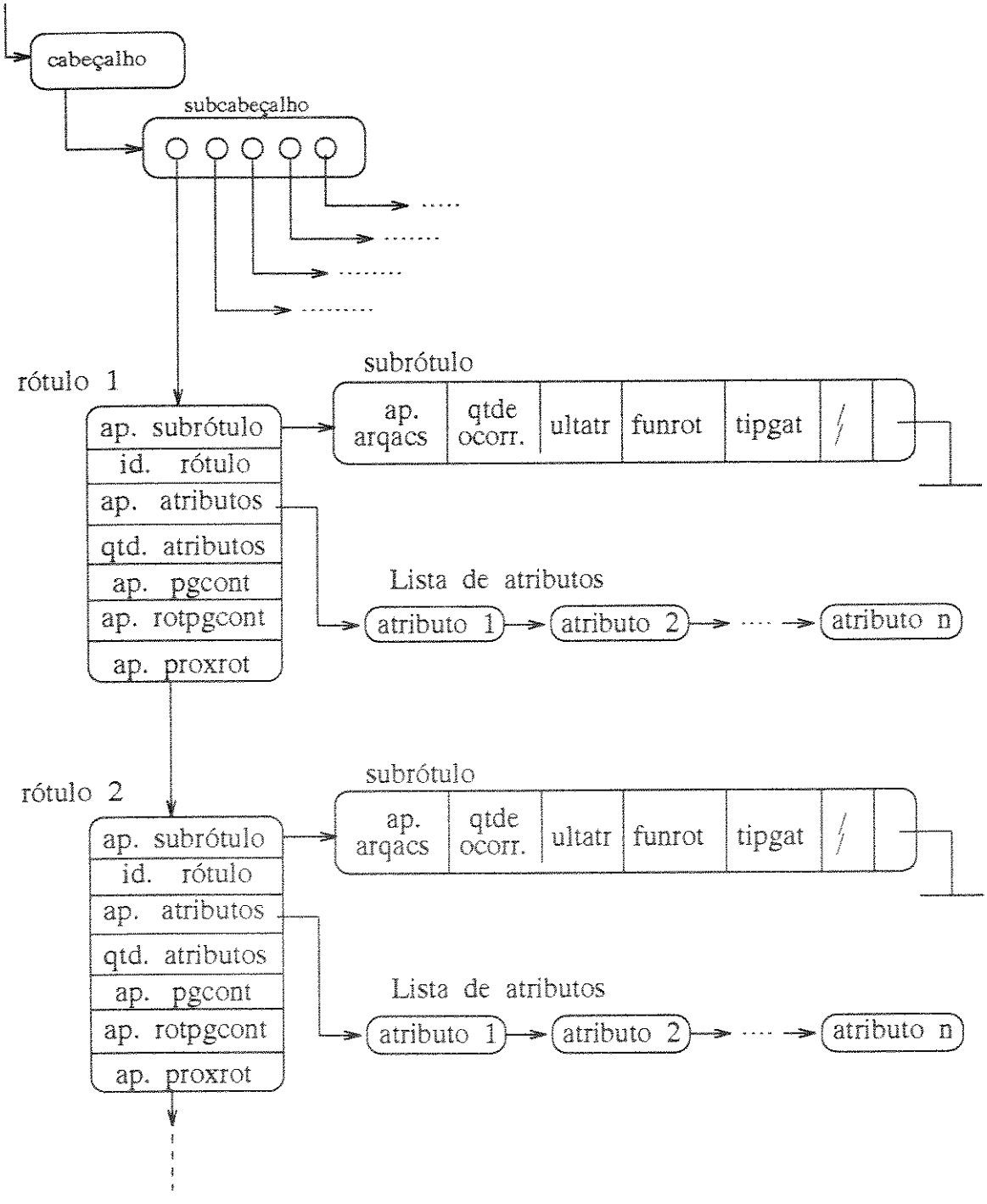

Figura 4.3 - Lista de Máscaras de Atributos

Cada máscara é associada a um determinado tipo de gatilho. No presente protótipo são considerados apenas dois tipos de gatilho. O primeiro destina-se a verificação de uma determinada restrição na ocasião da atualização dos objetos e/ou atributos por ela restringidos. No contexto deste trabalho, as restrições com este tipo de gatilho são chamadas de **Restrições Imediatas**.

Assim, toda vez que o GERPAC atualizar qualquer objeto ou atributo, o Subsistema de Tratamento de Restrições verifica, imediatamente, se existe alguma restrição associada a este objeto ou a este atributo. Caso positivo, todas as restrições associadas serão verificadas para manter a consistência da base de dados, de acordo com as especificações do usuário (as restrições).

O segundo tipo de gatilho representa uma decisão do usuário em verificar, a qualquer momento, um conjunto de restrições ou uma determinada restrição. As restrições com este tipo de gatilho são chamadas de **Restrições Adiadas**.

Este tipo de restrição, pelo fato de possibilitar verificações totalmente aleatórias e flexíveis da consistência dos dados em qualquer etapa do processo de projeto, talvez seja o tipo de restrição mais necessário e utilizado nas aplicações de engenharia em geral.

Porém, sempre as restrições do mesmo tipo, isto é, com o mesmo gatilho, serão armazenadas em uma mesma máscara. Assim, todas as restrições imediatas, definidas para um objeto, serão armazenadas em uma máscara, e todas as restrições adiadas, definidas para o mesmo objeto, serão armazenadas em uma outra máscara. As máscaras são identificadas através da informação existente na célula do seu subrótulo (o identificador de gatilho).

Outros tipos de restrição de consistência, eventualmente definidos pelo usuário, associados a outros tipos de gatilho, posteriormente suportados pelo sistema, serão armazenados normalmente em outras máscaras, sem necessidade de modificação das estruturas do Arquivo Interno de Objetos ou das funções do sistema que manipulam estas estruturas.

Este fato, proporciona ao GERPAC uma manutenção fácil das restrições de consistência. Além disso, os procedimentos de consulta às restrições também

ficam mais fáceis e mais flexíveis, oferecendo ao usuário um controle e manipulação das restrições mais confortável.

Uma nova máscara será criada, para um determinado objeto, somente no momento em que estiver sendo definida a primeira restrição nesta máscara. Em outras palavras, nenhum objeto, em nenhum momento, terá máscaras vazias, ou seja, máscaras que ainda não tenham nenhum atributo nelas definido.

Isto significa que as estruturas relativas às máscaras de atributos são criadas dinamicamente, a medida que o projeto evolui através das diversas etapas do processo de projeto. Isto contribui significativamente para a otimização dos espaços de memória primária e/ou secundária, utilizados no armazenamento das estruturas do Arquivo Interno de Objetos, diminuindo a complexidade do mesmo e melhorando consequentemente o desempenho do sistema.

4.3.2.3 A Lista de Atributos

Esta lista contém a definição de todos os atributos associados à máscara a qual esta lista está ligada. Os dados de cada atributo ocupam 3 células da lista. Assim, uma máscara com 6 atributos associadaos a ela, tem uma lista de atributos contendo 18 células. Cada atributo possui as seguintes informações:

- nome do atributo;
- código interno do atributo;
- tipo de dado associado ao atributo;
- apontador interno ao Arquivo de Valores (IC) ou ao Arquivo de Indices (II), conforme o atributo seja ou não indexado (Ric87a);
- indicador de atributo indexado;
- indicador de atributo múltivalorado.

A definição de restrições de consistência envolve a definição de um conjunto de 5 atributos, associados a uma máscara. Cada instância deste conjunto de atributos representa uma restrição.

Da mesma forma que uma ocorrência de um determinado objeto é representada pelas instâncias correlacionadas de todos os seus atributos normais (atributos que normalmente representam as características de um objeto), uma restrição é representada pela instância correlacionada do conjunto de atributos definidos na máscara correspondente. Estes atributos representam:

* **nome** da restrição;
* **representação** da restrição (o arquivo fonte da restrição);
* **ação** da restrição (o arquivo executável da restrição);
* **status** da restrição, indicando se a restrição foi satisfeita ou não (na última verificação);
* **domínio** da restrição, indicando os atributos que participam da restrição (podem ser atributos de objetos diferentes);

A Figura 4.4 mostra, de forma simplificada, a estrutura do Arquivo Interno de Objetos e suas listas internas. Nesta figura, a máscara com rótulo 1 é utilizada para a definição de atributos normais do objeto e a máscara de rótulo 2 é utilizada para a definição de restrições do objeto, através do conjunto de 5 atributos que definem cada restrição.

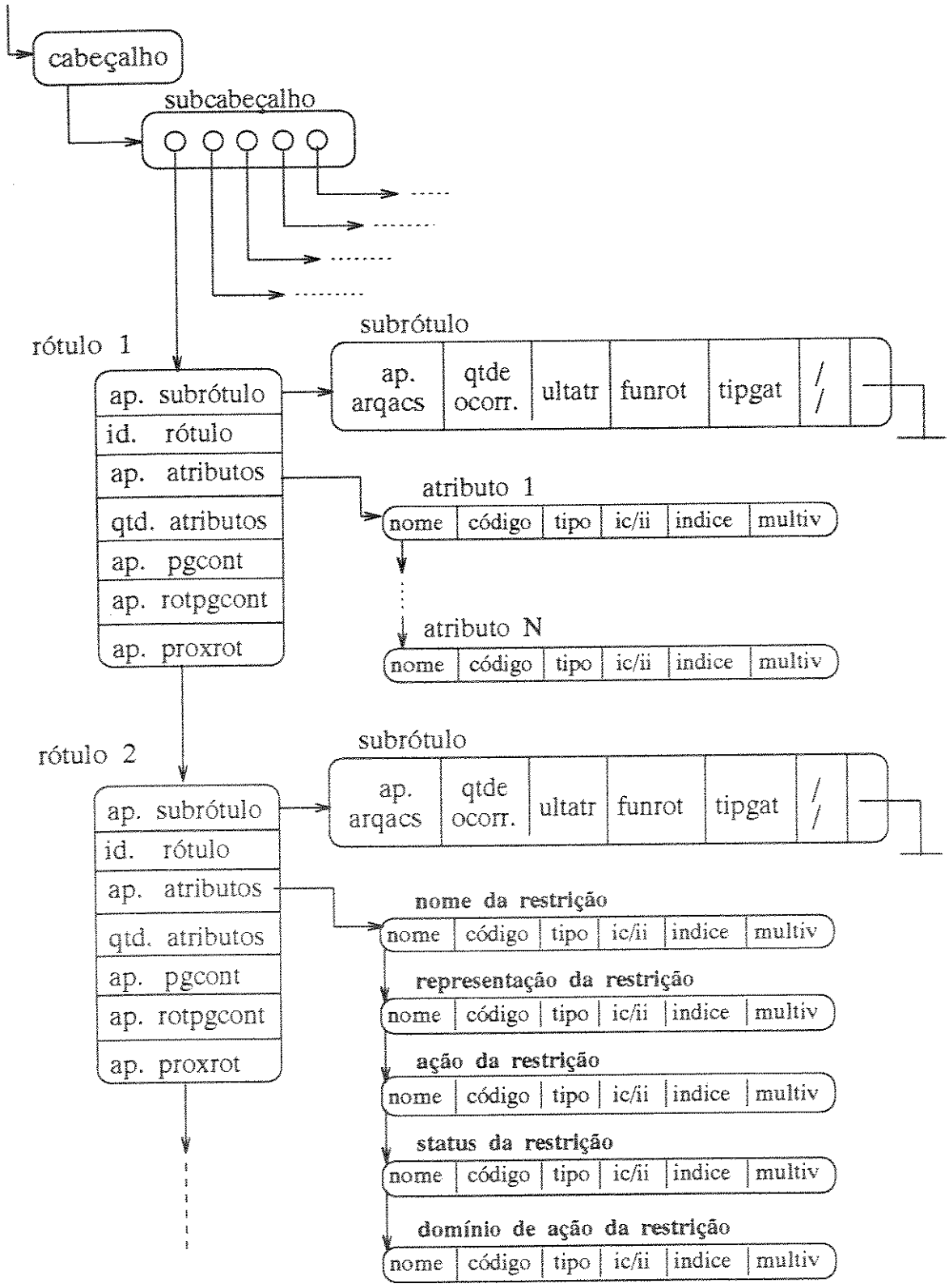

Figura 4.4 – Estrutura do Arquivo de Objetos e a Lista de Máscaras

4.3.3 Especificação do Sistema

O Subsistema de Tratamento de Restrições (STR) é o sistema responsável pela definição, manipulação e controle de todas as restrições de consistência de aplicação. O STR é composto de cinco componentes: **Interface de Paradigmas de Programação** (IPP), **Módulo de Manutenção de Restrições** (MMR), **Módulo de Consulta às Restrições** (MCR), **Módulo de Verificação de Restrições** (MVR) e a **Tabela de Restrições Imediatas** (TRI).

4.3.3.1 Interface de Paradigmas de Programação (IPP)

Nos ambientes de PAC, devido à natureza das aplicações, o uso de diferentes paradigmas de programação é uma característica bastante comum. Assim, o GERPAC pode suportar vários paradigmas de programação, definidos em uma biblioteca mantida pelo SGBD, de forma a possibilitar maior flexibilidade e eficiência na definição de restrições de consistência.

Para cada paradigma válido no GERPAC, o STR terá um módulo de interface. Cada módulo possibilita que as restriçõs definidas pelo usuário sejam suportadas pelo STR, facilitando a interação do usuário com o sistema. Assim, o usuário fica liberado da obrigação de ter conhecimento detalhado das funções do GERPAC. Desta forma, a partir de um programa fonte de cada restrição, é gerado um arquivo executável, mantido e utilizado posteriormente pelo STR para verificar a consistência da base de dados.

No presente protótipo existem dois módulos, voltados para as linguagens de programação COMMON LISP e C, sendo possível somente a definição de restrições codificadas através destas linguagens. A Figura 4.5 mostra a comunicação entre IPP, GERPAC e os usuários.

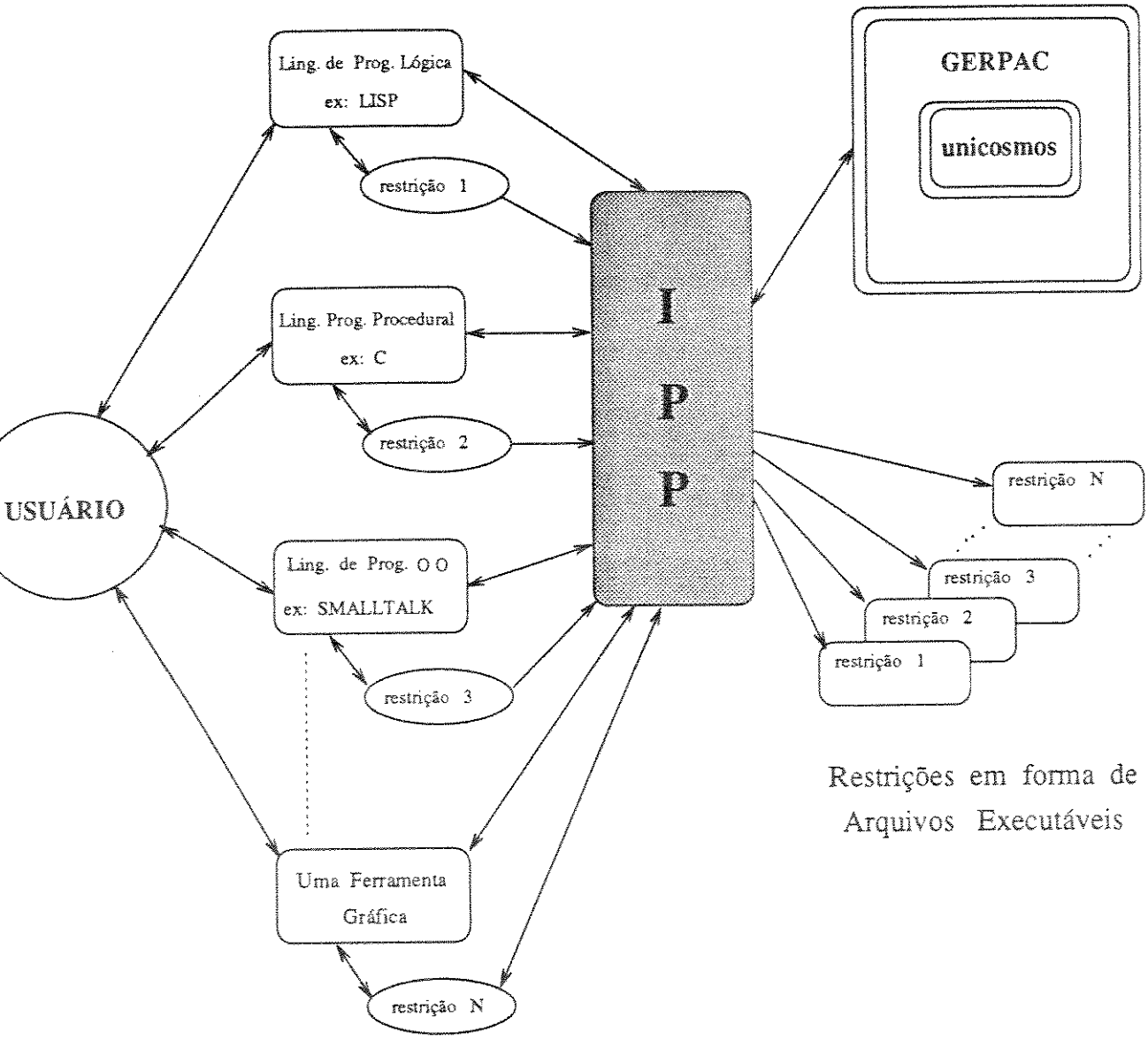

Figura 4.5 - Interface de Paradigmas de Programação (IPP)

4.3.3.2 Módulo de Manutenção de Restrições (MMR)

O Módulo de Manutenção de Restrições (MMR) é responsável por todo o processo de gerência da definição e do armazenamento de restrições no GERPAC. Uma restrição, como um atributo normal, pode ser inserida, removida ou atualizada. Porém, como uma restrição é uma ocorrência de um conjunto de 6 atributos, a sua inserção ou remoção implica na inserção ou remoção da ocorrência deste conjunto de atributos.

Como foi visto anteriormente, as restrições com mesmo tipo de gatilho são armazenadas em uma mesma máscara. Portanto, na ocasião da inserção de uma restrição, o usuário deve informar, além dos valores dos 5 atributos, o tipo de gatilho da restrição. O MMR sempre pesquisa as células dos subrótulos existentes no objeto, se existir algum rótulo definido para este tipo de gatilho, a restrição é inserida como uma nova ocorrência do conjunto de atributos existentes, com valor zero para o atributo *status* (restrição não satisfeita). No entanto, se o rótulo não existir, isto é, a restrição é a primeira a ser definida com este tipo de gatilho, então será criada para o objeto toda a estrutura necessária para definir um novo rótulo e uma nova lista de atributos.

Da mesma forma, quando ocorre uma remoção, o MMR verifica se a restrição a ser removida é a última da lista. Caso positivo, toda a estrutura da lista de máscara e do conjunto de atributos será também removida. Caso contrário, será removida somente a ocorrência do conjunto de atributos associada com a restrição.

De qualquer forma, tanto na ocasião de inserção de uma restrição (seja a primeira da lista ou não), como na ocasião da remoção de uma restrição (se ela não é a última da lista), as informações da célula de subrótulo são atualizadas.

No momento de inserção de uma restrição do tipo imediato, será automaticamente inserida na TRI uma entrada associada a ela. Em seguida, a restrição será executada para a sua verificação, após a qual, e dependendo do resultado da verificação, o valor do atributo *status* da restrição será atualizado.

O sistema não permite a alteração dos valores do conjunto de atributos de uma restrição. Se o usuário deseja alterar qualquer um destes valores, ele deve primeiro remover a restrição existente e em seguida inserir uma nova ocorrência da mesma com os novos valores desejados.

Para manter a consistência dos dados da própria restrição, são verificadas as datas de criação do arquivo fonte e do arquivo executável da restrição. Esta verificação é efetuada tanto no momento da inserção como no momento da verificação da restrição. Assim, na ocasião da inserção de uma restrição se a data do seu arquivo executável for anterior à data do seu arquivo fonte, ela será recusada.

Além disso, o domínio de ação de cada restrição (o valor do atributo *domínio* da restrição) é também verificado. Para isto, no momento da inserção da restrição, todos os valores de pares de objeto e atributo (o atributo *domínio* da restrição é um atributo multivalorado) que o usuário informa como sendo o domínio de ação da restrição, são verificados para garantir que tais objetos e atributos sejam válidos para o sistema.

4.3.3.3 Módulo de Consulta às Restrições (MCR)

Este módulo é responsável pelo fornecimento de qualquer informação a respeito das restrições de consistência definidas no sistema. Através das funções deste módulo, incorporadas ao GERPAC e ao UNICOSMOS, o usuário pode realizar diversos tipos de consulta às restrições. As informações fornecidas pelas funções deste módulo auxiliam o usuário em manter melhor controle sobre as restrições de integridade definidas no sistema.

4.3.3.4 Módulo de Verificação de Restrições (MVR)

Este módulo é responsável em verificar se o estado da base de dados satisfaz as exigências de uma determinada restrição ou de um conjunto de restrições. A verificação de uma restrição consiste na execução do arquivo executável correspondente, cujo nome está armazenado no atributo *ação* da restrição.

Se, no momento da verificação de uma restrição, for observada pelo sistema qualquer irregularidade entre as datas de criação dos arquivos fonte e objeto da restrição, o processo é imediatamente descontinuado e o usuário notificado.

A execução de uma restrição se dá através de um processo independente, o que contribui para a melhoria do desempenho do GERPAC. Este processo é iniciado após uma solicitação do usuário ou do próprio sistema sendo gerenciado pelo STR. A comunicação entre este processo e o GERPAC é feita através do *Roteador* (o módulo responsável pela comunicação entre diversos processos do GERPAC), utilizando os recursos do sistema operacional UNIX.

A execução de restrições de consistência imediatas é efetuada através da Tabela de Restrições Imediatas. Após a execução de uma restrição o valor da ocorrência do atributo *status* correspondente à restrição será atualizado de acordo com o resultado da sua verificação.

4.3.3.5 Tabela de Restrições Imediatas (TRI)

Esta tabela é uma estrutura que contém o nome de todos os atributos normais (de objetos) que participam em alguma restrição em conjunto com o nome do arquivo executável de cada restrição associada (Figura 4.6). Através desta

tabela é possível disparar o processo da verificação de uma restrição cujo gatilho é a atualização dos atributos normais nela envolvidos.

A TRI é montada a partir das informações existentes nas máscaras de atributos definidas para armazenamento de restrições imediatas e armazenada em disco. Ela é sempre carregada para a memória principal na ocasião da abertura da base de dados em uma sessão de trabalho, e ali permanece até que a base de dados seja fechada (encerramento da sessão de trabalho), quando é gravada de volta para o disco. Entretanto, se no momento da abertura da base de dados a TRI não for encontrada em disco, ela será imediatamente montada e carregada para a memória principal. Desta forma, não será necessária a montagem da TRI toda vez que a base de dados for aberta, melhorando assim o desempenho do sistema. A Figura 4.7 mostra genericamente o interrelacionamento entre os módulos do STR, o GERPAC e o Roteador.

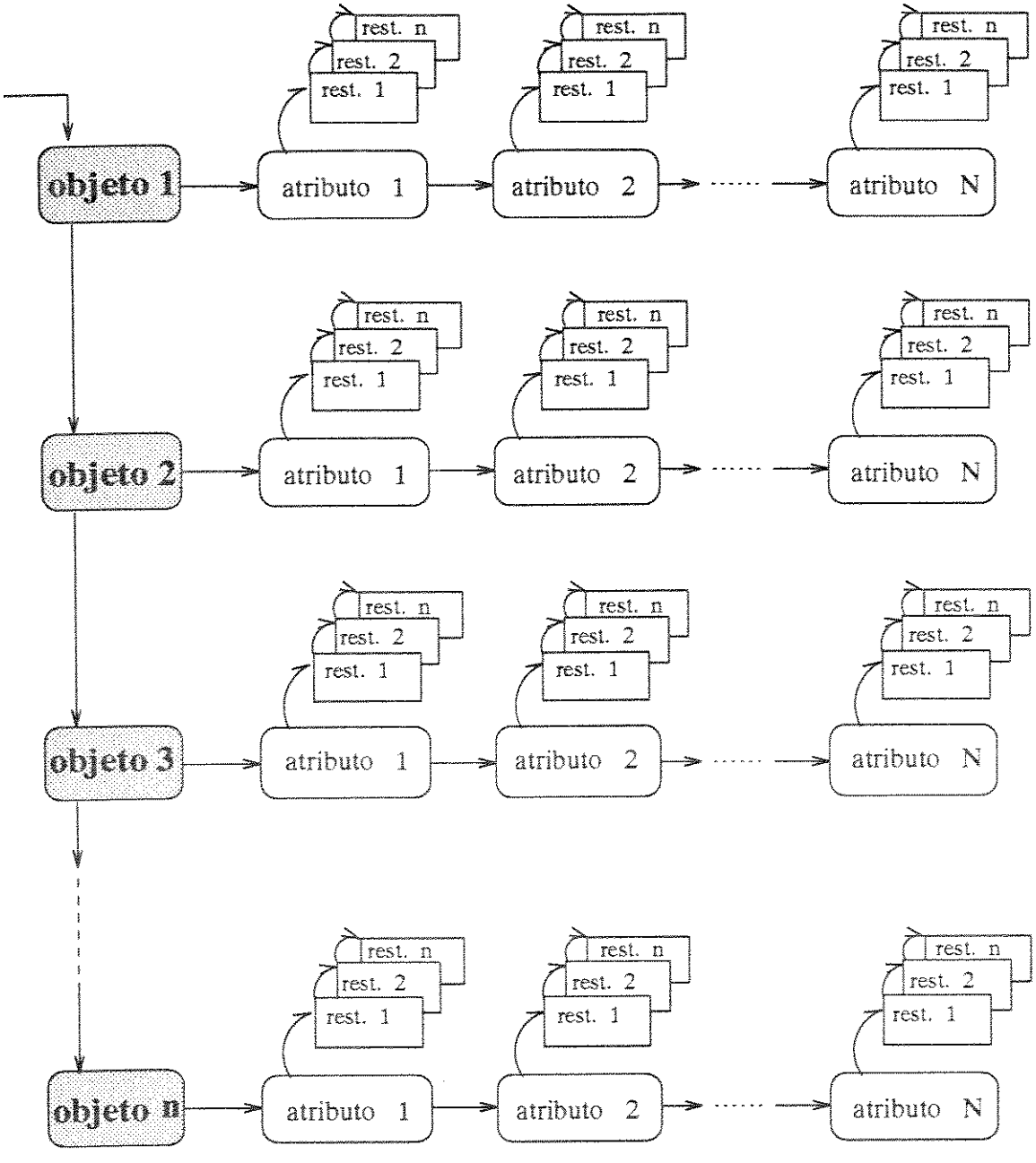

Figura 4.6 - Estrutura da Tabela de Restrições Imediatas (TRI)

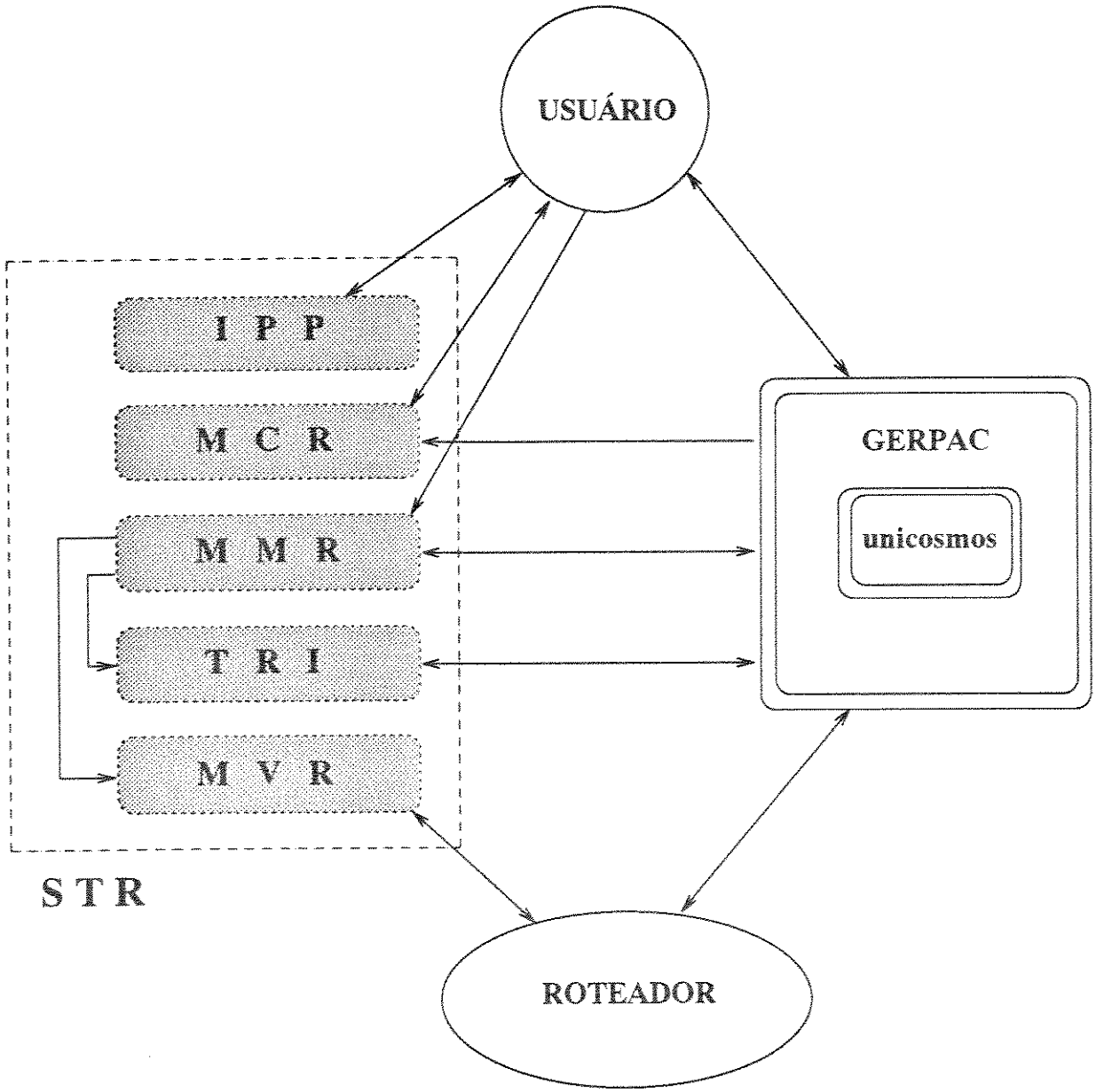

Figura 4.7 - Esquema de comunicação entre STR e GERPAC

4.3.4 Funções do Subsistema de Tratamento de Restrições

Todas as funções do STR são desenvolvidas em linguagem C no ambiente UNIX das estações de trabalho da SUN. A maioria destas funções estão incorporadas aos módulos do GERPAC ou do UNICOSMOS. A seguir são relacionadas as funções de cada módulo do STR.

4.3.4.1 Funções do MMR

a) **Funções incorporadas ao GERPAC**

1. **dfojrl_ger_def (objeto, rótulo, restrição)**

   Esta função armazena uma restrição em um determinado rótulo de um objeto. O parâmetro <restrição> é um buffer que contém todas as informações necessárias para armazenar uma restrição (o nome, a representação, a ação, o status e o domínio de ação da restrição).

2. **vfddrl_ger_def (restrição)**

   Verifica a consistência dos dados de buffer de entrada da restrição no momento de sua declaração.

3. **vfaqrl_ger_str (restrição)**

   Verifica a consistência dos arquivos fonte e objeto da restrição.

4. **dfrllt_ger_def (objeto, rótulo)**

   Cria a lista de atributos de restrições no rótulo especificado de um determinado objeto.

5. **dfojtg_ger_def (objeto, rótulo)**
   Cria um rótulo para um objeto.

6. **dftgrl_ger_def (objeto, rótulo, tipo-gatilho)**
   Marca um rótulo para uso de restrições e grava no subrótulo do mesmo o tipo de gatilho associado a ele.

7. **rmojrl_ger_rem (objeto, rótulo, restrição)**
   Remove uma determinada restrição de um determinado rótulo do objeto.

8. **rmalrl_ger_rem (objeto, rótulo)**
   Remove todas as restrições associadas a um determinado rótulo do objeto.

9. **rmisrl_ger_rem (objeto, rótulo, clausula)**
   Remove todas as restrições de um rótulo que satisfizerem as condições expressas na clausula.

10. **rmtglt_ger_rem (objeto, rótulo)**
    Remove um determinado rótulo de um objeto.

b) **Funções incorporadas ao UNICOSMOS**

Nível Terciário

1. **mcrore_uni_grt (objeto, rótulo, tipo-gatilho)**
   Marca um rótulo como sendo de uso exclusivo das restrições de consistência, atualizando os valores de função e de gatilho no subrótulo.

Nível Secundário

1. **azfuro_uni_drt (pagobj, celobj, rlótulo, função-rótulo)**
   Armazena na página e célula correspondentes a função do rótulo de um objeto.

2. **azgaro_uni_drt (pagobj, celobj, rlótulo, tipo-gatilho)**
   Armazena na página e célula correspondentes o tipo de gatilho associado a um rótulo de um objeto.

3. **srevfu_uni_dst (subrótulo, função-rótulo)**
   Escreve a função do rótulo na célula do subrótulo.

4. **srevga_uni_dst (subrótulo, tipo-gatilho)**
   Escreve o tipo de gatilho associado ao rótulo na célula do subrótulo.

4.3.4.2 Funções do MCR

a) **Funções incorporadas ao GERPAC**

1. **obojrl_ger_con (objeto, rótulo, nome_restrição)**
   Obtém uma determinada restrição de um rótulo do objeto.

2. **obalrl_ger_con (objeto, rótulo)**
   Obtém todas as restrições de um determinado rótulo do objeto.

3. **obstrl_ger_con (objeto, rótulo, clausula)**
   Obtém todas as restrições definidas em um determinado rótulo do objeto que satisfizerem as condições especificadas na clausula.

4. **vftglt_ger_con (objeto, rótulo)**
   Verifica se um determinado rótulo existe em um objeto.

5. **vftgrl_ger_con (objeto, rótulo)**
   Verifica se um determinado rótulo de um objeto é para definição de restrições.

6. **oblttg_ger_con (objeto, lista-rótulos)**
   Obtém o código de todos os rótulos existentes em um objeto.

7. **obrltg_ger_con (objeto, lista-rótulos)**
   Obtém o código de todos os rótulos de um objeto utilizados para definição de restrições.

8. **obtrtg_ger_con (objeto, rótulo)**
   Obtém o tipo de gatilho associado a um determinado rótulo de um objeto.

b) **Funções incorporadas ao UNICOSMOS**

Nível Terciário

1. **obrooj_uni_grt (objeto, lista-rótulos)**
   Obtém a lista de códigos de todos os rótulos de um objeto.

2. **obrore_uni_grt (objeto, lista-rótulos)**
   Obtém a lista de códigos de todos os rótulos de um objeto, utilizados para definição de restrições.

3. **vfrore_uni_grt (objeto, rótulo)**
   Verifica se um rótulo do objeto é para definição de restrições.

4. **porolt_uni_grt (objeto, rótulo)**
   Procura por um determinado rótulo na lista de rótulos de um objeto.

5. **cngaro_uni_grt (objeto, rótulo)**
   Obtém o tipo de gatilho associado a um determinado rótulo de um objeto.

Nível Secundário

1. **tdrore_uni_drt (pagobj, celobj, lista-rótulos)**
   Obtém todos os códigos de todos os rótulos de um objeto, utilizados para definição de restrições, a partir de número da página e da célula do objeto.

2. **obtdro_uni_drt (pagobj, celobj, lista-rótulos)**
   Obtém todos os códigos de todos os rótulos de um objeto a partir de número da página e da célula do objeto.

3. **obfuro_uni_drt (pagobj, celobj, rótulo)**
   Obtém a função de um rótulo a partir de número da página e da célula do objeto.

4. **obgaro_uni_drt (pagobj, celobj, rótulo)**
   Obtém o tipo de gatilho de um rótulo a partir de número da página e da célula do objeto.

5. **srlrfu_uni_drt (subrótulo)**
   Lê a função de um rótulo na célula do subrótulo.

6. **srlrga_uni_drt (subrótulo)**
   Lê o tipo de gatilho de um rótulo na célula do subrótulo.

4.3.4.3 Funções do MVR

a) Funções incorporadas ao GERPAC

1. **exojrl_ger_str (objeto, rótulo, restrição)**
   Executa a "ação" (o arquivo executável) de uma determinada restrição de um rótulo do objeto.

2. **exnmrl_ger_str (objeto, rótulo, nome_restrição)**
   Executa a "ação" de uma restrição especificada pelo seu nome.

3. **exalrl_ger_str (objeto, rótulo)**
   Executa todas as restrições definidas em um determinado rótulo do objeto.

4. **exisrl_ger_str (objeto, rótulo, clausula)**
   Executa todas as restrições definidas em um determinado rótulo do objeto que satisfizerem as condições expressas na clausula.

5. **exacrl_ger_str (ação_restrição)**
   Inicia o processo de execução do arquivo executável da restrição.

6. **upstrl_ger_upd (objeto, rotulo, restrição)**
   Atualiza o atributo *status* de uma determinada restrição de um rótulo do objeto.

4.3.4.4 Funções da TRI

a) Funções incorporadas ao GERPAC

1. **montri_ger_str (base_de_dados)**
   Monta a TRI na memória, na ocasião da inicialização do sistema, a partir da informações existentes nos rótulos dos objetos da base_de_dados.

2. **lodtri_ger_str (arquivo_tri)**
   Carrega a TRI na memória a partir do arquivo_tri existente em disco.

3. **rectri_ger_str ()**
   Recupera a TRI. Se ela existir em disco chama a função que carrega-a na memória, caso contrário chama a função que monta-a na memória.

4. **savtri_ger_str ()**
   Salva a TRI na memória permanente do sistema.

5. **psqtri_ger_str (objeto, rótulo, atributo)**
   Pesquisa na TRI por um atributo de um determinado rótulo do objeto. Caso o atributo exista na TRI, inicia o processo de verificação das restrições associadas a este atributo.

6. **instri_ger_str (objeto, rótulo, restrição)**
   Insere uma determinada restrição na TRI, juntamente com os atributos de seu domínio de ação.

7. **remtri_ger_str (objeto, rótulo, restrição)**
   Remove uma determinada restrição da TRI.

8. **rmltri_ger_str (objeto, rótulo, lista-restrições)**
   Remove da TRI todas as restrições da lista_restrições de um rótulo do objeto.

4.4 COMENTÁRIOS FINAIS

Neste capítulo foram inicialmente apresentados o GERPAC, o UNICOSMOS e o modelo de dados MER/PAC. Em seguida foi descrito e especificado o Subsistema de Tratamento de Restrições (STR) do GERPAC, sendo analisadas as características de restrições de consistência no contexto GERPAC/UNICOSMOS. Para melhor situar o leitor no desenrolar da implementação do sistema, algumas das estruturas, principalmente o Arquivo Interno de Objetos, utilizadas para implementação da nossa proposta, foram abordados mais detalhadamente.

O STR, através dos seus componentes básicos (IPP, MMR, MCR, MVR e TRI), proporciona ao GERPAC a capacidade de definição, manipulação e verificação dinâmica de restrições de consistência com facilidade e flexibilidade. Na sua implementação procurou-se utilizar, ao máximo, as funções já existentes do GERPAC, o que contribuiu para sua simplicidade e melhor desempenho. As funções dos seus cinco módulos foram incorporadas aos módulos do GERPAC e do UNICOSMOS, causando pequenas modificações em algumas funções destes, quando necessário. O detalhamento das funções do STR, suas listagens e exemplos podem ser encontrados em [Nou92].

"Quem se dedica, hoje, ao serviço da inteira raça humana, é, em verdade, um homem."

Bahá'u'lláh

CAPÍTULO 5

CONCLUSÕES

O principal objetivo deste trabalho foi munir o GERPAC de mecanismos de tratamento de restrições de consistência que possibilitassem a definição e a verificação de diversos tipos de restrições, suprindo muitas das necessidades das aplicações PAC.

Para isso, inicialmente foram introduzidos os conceitos básicos de sistemas de banco de dados tanto para as aplicações convencionais como para as aplicações não-convencionais, como forma de situar o leitor no contexto deste trabalho.

No capítulo 2, discutiu-se a questão de controle de integridade em sistemas de banco de dados convencionais, destacando o tratamento de diversos tipos de restrições de integridade com alguns exemplos em diversos sistemas existentes.

O capítulo 3 tratou a mesma questão no contexto de sistemas de banco de dados não-convencionais, em particular, nas aplicações em projetos de engenharia e em sistemas de base de conhecimento. O capítulo quatro comentou as restrições no contexto GERPAC/UNICOSMOS e descreveu a implementação de mecanismos, propostos neste trabalho, para o tratamento de restrições no SGBD GERPAC.

De um modo geral, do ponto de vista funcional, as restrições de integridade são classificadas em duas categorias: *restrições de integridade estrutural* e *restrições de integridade comportamental* ({Gar89},{Elm89}). As restrições de integridade estrutural são aquelas inerentes às características dos modelos de dados que moldam a estrutura do sistema, enquanto as restrições de integridade comportamental são aquelas que tratam da semântica dos dados do sistema.

As restrições de integridade ainda podem ser classificadas como *restrições estáticas* ou *restrições dinâmicas* ({Del85},{Dat83}), levando em conta a integridade temporal da base de dados. As restrições de integridade estáticas verificam a validade de um determinado estado do sistema, enquanto as dinâmicas verificam a consistência do sistema durante a sua transformação de um estado em outro. Elas podem também ser classificadas em *restrições imediatas* ou em *restrições adiadas* ({Laf82},{Dat83}), quanto ao momento de sua verificação.

Nas aplicações de Projeto Auxiliado por Computador, as restrições de consistência podem ser agrupados em quatro classes: *restrições de conformidade*, *restrições de composição*, *restrições de equivalência* e *restrições ambientais* ({Kat83}, {Dit86}). As restrições de cada uma destas classes, por sua vez, podem ser diferenciadas de acordo com a classificação anterior.

Esta taxonomia leva em consideração a consistência entre as diversas atividades típicas dos ambientes PAC, como por exemplo, especificação e modelagem do projeto, criação e manipulação de diferentes representações do projeto, manipulação e recuperação dos dados de diversos esquemas e versões do projeto.

As restrições de integridade comportamental estáticas são, até agora, as mais estudadas, devido a facilidade de serem expressas através da lógica de predicados de primeira ordem. Consequentemente, a evolução dos sistemas baseados em regras, como é o caso de bases de conhecimento, tem contribuido significantemente para prover mecanismos adequados de tratamento de restrições em sistemas de banco de dados.

O controle da grande maioria de outras restrições de integridade, as quais não se enquadram na categoria de restrições comportamental estáticas, é feito ou

através dos mecanismos oferecidos pelo modelo de dado ou embutido na implementação da aplicação. Desta forma, a sua modificação ou a definição de novas restrições desta natureza, após a implementação do sistema, se torna muito complexa e custosa.

Como foi visto anteriormente, as restrições de consistência no contexto GERPAC/UNICOSMOS são definidas de forma procedural, isto é, o usuário define procedimentos que constituem as restrições, as quais são em seguida declaradas como atributos dos objetos do sistema. Por esta razão, um usuário do GERPAC, a partir dos resultados do presente trabalho, pode definir uma restrição de qualquer uma das categorias mencionadas anteriormente.

Por exemplo, o usuário pode definir um procedimento que restringe a estrutura de um determinado objeto, quanto a quantidade de seus atributos, ou no caso de objetos complexos limita o número ou até mesmo o tipo de seus componentes. Nestes casos o usuário estaria definindo restrições de integridade estrutural ou restrições de composição.

Por outro lado, ele pode também definir uma restrição em um determinado objeto que verifique a cardinalidade de seus relacionamentos com os outros objetos, ou que simplesmente limita os valores de alguns atributos a valores de determinados conjuntos pré-estabelecidos. Neste caso estamos falando de restrições de integridade comportamental ou restrições ambientais.

As restrições dinâmicas, por sua vez, podem ser definidas através de procedimentos que utilizem os arquivos de log das transações do sistema, para manter a consistência dos objetos na sua transformação de um estado em outro. Outras classes de restrições, como restrições de conformidade e restrições de equivalência também podem ser facilmente definidas no GERPAC.

Além disso, o fato de restrições de consistência serem declaradas e tratadas como atributos dos objetos do sistema, proporciona manipulação e manutenção flexíveis e dinâmicas das mesmas, o que caracteriza uma das necessidades das aplicações de PAC. Isto demonstra a flexibilidade dos mecanismos de tratamento de restrições do GERPAC.

Assim, a nossa proposta apresenta diversas características importantes para

controle de restrições de consistência nos ambientes PAC, corroboradas em outros trabalhos na mesma área. Na nossa proposta, como também no HIPAC {McC89} e no SGBD O2 {Med91}, as restrições podem ser consideradas como objetos, tendo atributos e podendo ser criadas, modificadas ou excluídas do sistema. Semelhante ao sistema de regras do POSTGRES {Sto88}, no GERPAC também as restrições podem servir para outros propósitos, como por exemplo, controle de segurança e proteção. Como foi observado, no capítulo anterior, na nossa proposta, como também no SGBD DAMASCUS, as restrições podem ser declaradas e manipuladas dinâmicamente.

Em resumo, dos mecanismos necessários para controle de restrições de consistência em aplicações de engenharia, discutidos em inúmeros trabalhos, os seguites são suportados por este protótipo:

* Gerenciamento de grande número de restrições;
* Facilidade de definição e manipulação dinâmica;
* Definição de restrições de consistência local e global;
* Tolerância a inconsistências temporárias, porém longas;
* Flexibilidade quanto ao momento de verificação;
* Flexibilidade quanto ao tipo de ação;
* Definição algorítmica de restrições;
* Execução e verificação de restrições através de processos independentes.

Diversos trabalhos ainda podem ser realizados para melhorar o estado atual do sistema. Uma das melhorias pode ser a definição e a implementação de um modelo flexível e até dinâmico de definição de gatilhos, o que poderá oferecer recursos para manutenção de integridade do sistema. Neste caso, estes mecanismos poderiam servir também para o controle de segurança do sistema. Pois, desta forma, as normas de segurança poderiam ser implementadas através de definição de restrições sobre acessos aos objetos do sistema.

Outra linha de trabalho poderia se preocupar em manter a consistência das próprias restrições, o que atualmente é bastante rudimentar, limitando-se a simples verificação da integridade entre arquivos de fonte e de objeto das restrições. A integridade semântica entre as restrições também poderia ser abordada neste trabalho, o que não seria entretanto uma tarefa trivial.

Outra possível contribuição diz respeito a definição declarativa de restrições, o que envolve a criação de uma linguagem de manipulação de dados e uma linguagem de consulta para o SGBD GERPAC.

"Esforsai-vos para que vossas ações dia a dia sejam belas orações."

'Abdu'l-Bahá

CAPÍTULO 6

Referências Bibliográficas:

{All82} Allen, Frank W.; Loomis, Mary E. S. & Mannino, Michael V., "The Integrity Dictionary/Directory System", ACM Computing Surveys, Vol. 14, No. 2, Junho 1982.

{Ban85} Bancilhon, François; Kim, Won & Korth, Henry F., "A Model of CAD Transactions", Proc. of 11th Int. Conf. on Very Large Data Bases, Stockholm, Agosto 1985.

{Ben82} Ben-Ari, M., "Principle of Concorrent Programing", Prentice-Hall, USA, 1982.

{Bat85} Batini, C. & Ceri, S., "Database Design: Methodologies, Tools, and Environments", ACM, 1985.

{Bro80} Brodie, Michael L., "The Application of Data Types to Database Semantic Integrity", Inform. Systems, Vol. 5, 1980.

{Cas85} Casanova, M. A. & Moura, A. V., "Princípios de Sistemas de Banco de Dados Distribuídos", Campus, Rio de Janeiro, 1985.

{Che76} Chen, P. P. S., "The Entity Relationship Model - Toward a Unified View of Data", ACM Transactions on Database, Vol. 1, No. 1, Março 1976.

{Che86} Chen, Giming, "A Rule-Based Object/Task Modeling Approach", ACM SIGMOD RECORD, Vol. 15, No. 2, 1986.

{Cho86} Chou, Hong-Tai & Won Kim, "A Unifying Framework For Version Control in a CAD Environment", Proc. of 12th Int. Conf. on Very Large Data Bases, Kyoto, Agosto 1986.

{Cro87} Croft, W. B. & Stemple, D. W., "Supporting Office Document Architectures with Constrained Types", ACM SIGMOD RECORD, Vol. 16, No. 3, Dez. 1987.

{Dad86} Dadam, P., et ali., "A DBMS Prototype to Support Extended NF2-Relations: An Integrated View on Flat Tables and Hierarchies", Proc. of ACM SIGMOD Conf. Washington D.C., 1986.

{Dat77} Date, C. J., "An Introduction to Database Systems - Volume I", Addison-Wesley, USA, 1977.

{Dat83} Date, C. J., "An Introduction to Database Systems - Volume II", Addison-Wesley, USA, 1983.

{Dat86} Date, C. J., "Relational Database", Addison-Wesley, USA, 1986.

{Deb89a} Debloch, Stefan; Leick, F. J. & Mattos, N. M., "A State-oriented Approach to the Specification of Rules and Queries in KBMS", Research Report, University of Kaiserslautern, 1989.

{Deb89b} Debloch, Stefan; Harder, T.; Mattos, N. & Mitschang, B., "KRISYS: KBMS Support for Better CAD Systems", em: Proc. of the 2nd Int. Conf. on Data and Knowledge Systems for Manufactoring and Engineering, Gaithersburg - Maryland, Out. 1989.

{Deb90a} Debloch, Stefan, "Enforcing Integrity in the KBM KRISYS", Proc. of the Second Int. Workshop on Fundation of Models and Languages for Data and Objects, Austria, Set. 1990.

{Deb90b} Debloch, Stefan, "Handling Integrity in a KBMS Architecture for Workstation/Server Environments", Research Report, University of Kaiserslautern, 1990.

{Del85} Delobel, Claude & Adiba, Michel, "Relational Database Systems", Nort-Holland, Netherlands, 1985.

{Del86} Delgado, A. L. N.; Magalhães, L. P. & Ricarte, I. L., Sistemas de Gerenciamento de Banco de Dados para PAC, Anais do I Simpósio Brasileiro de Banco de Dados, Abr. 1986.

{Del87} Delgado, A. L. N., Bancos de dados no contexto de projeto auxiliado por computador, Tese de Mestrado, DCA-FEE-UNICAMP, Jan. 1987.

{Del88} Delgado, A. L. N.; Magalhães, L. P. & Ricarte, I. L., Supporting design environments through the E-R Model, 12th IMACS Word Conference on Scientific Computation - Paris, França, Julho 1988.

{Dit85} Dittrich, Klaus R.; Kotz, Angelika M. & Mulle, Jutta A., "A Multilevel Approach to Design Database Systems and its Basic Mechanisms", Proc. of IEEE COMPINT, Montreal, 1985.

{Dit86a} Dittrich, Klaus R.; Kotz, Angelika M. & Mulle, Jutta A., "Database Support for VLSI Design: The DAMASCUS System", em: CAD Interface and Data Transfer Formats in Electronics, Springer-Verlag, 1986.

{Dit86b} Dittrich, K. R.; Kotz, A. M. & Mulle, J. A., An Event/Trigger Mecanism to Enforce Complex Consistency Constraints in Design Databases, SIGMOD RECORD, Vol. 15, No. 3, Set. 1986.

{Elm89} Elmasri, Ramez & Navathe, S. B., "Fundamentals of Database Systems", Benjamin/Cummings Publishing Company, Inc., USA, 1989.

{Enc83} Encarnação, J. & Schlechtendahl, E. G., "Computer Aided Design - Fundamentals and System Architectures", Springer-Verlag, 1983.

{Fip88} Fipec, Sisteams de banco de dados no contexto de projeto auxiliado por computador, Relatório Final, Processo 1.1731-0, Jan. 1988.

{Gar89} Gardarin, Georges & Valduriez, P., "Relational Databases and Knowledge Bases", Addison-Wesley, USA, 1989.

{Gra81} Gray, Jim, "The Transaction Concept: Virtues and Limitations", Proc. of 7th Int. Conf. on Very Large Data Bases, Canes, França, Set. 1981.

{Hae87} Haerder, Theo & Rothermel, K., "Concepts for Transaction Recovery in Nested Transactions", Proc. of ACM SIGMOD Conf., 1987.

{Ham75} Hammer, Michael M. & McLeod, Dennis J. "Semantic Integrity in a Relationl Data Base System", Research Report, Massachusetts Institute of Technology, 1975.

{Har87} Harder, T.; Mattos, N. & Mitschang, B., "PRIMA - A DBMS Prototype Supporting Engineering Aplicarion", Proc. of 13th Very Large Data Base Conf., Brighton, 1987.

{Hon81} Hong, Y. C. & Su, Stanley Y. W. "Associative Hardware and Software Techniques for Integrity Control", ACM Transactions on Database Systems, Vol. 6, No. 3, Set. 1981.

{Ioc89} Iochpe, Cirano, "Database Recovery in the Design Environment: Requirements Analysis and Performance Evaluations", Tese de Doutorado, Universidade de Karlsruhe, Alemanha, Dez. 1989.

{Kai88a} Kaiser, Gail E.; Barghouti, Naser S.; Feiler, Peter H. & Schwanke, Robert W., "Database Support for Knowledge-Based Engineering Environment", IEEE Expert, Summer 1988.

{Kai88b} Kaiser, Gail E.; Feiler, Peter H. & Propovich, S. S., "Intelligent Assistance For Software Development and Maintenance", IEEE Software, Maio 1988.

{Kat83} Katz, R. H., "Managing the Chip Design Database", IEEE Computer, Dez. 1983.

{Kat84} Katz, Randy H. & Lehman, Tobin J., "Database Support for Versions and Alternatives of Large Design Files", IEEE Tran. on Software Engineering, Vol. SE-10, No. 2, Mar. 1984.

{Kat85} Katz, Randy H., "Information Management for Engineering Design", Springer-Verlag, Alemanha, 1985.

{Kat86a} Katz, Randy H., M. Anwarrudin & E. Chang, "A Version Server for Computer-Aided Design Data", Proceedings 23rd ACM/IEEE Design Automation Conference, Las Vegas, NV, Junho 1986.

{Kat86b} Katz, Randy H.; Chang, Ellis & Bhateja, Rajiv, "Version Modeling Concepts for Computer-Aided Design Databases", ACM SIGMOD Int. Conf. on Management of Data, 1986.

{Kel86} Kellogg, Charles, "From Data Management to Knowlege Management", Computer, Vol. 19, No. 1, Jan. 1986.

{Ket87} Ketabchi, Mohammad A. & Berzins, Valdis, "Modeling and Managing CAD Databases", IEEE Computer, Fev. 1987.

{Kim84} Kim, Won; Lorie, R.; McNabb, D. & Plouffe, W., "A Transaction Mechanism for Engineering Design Databases", Proc. of 10th Int. Conf. on Very Large Data Bases, Singapore, 1984.

{Kor86} Korth, Henry F. & Silberschatz, Abraham "Database System Concepts", McGraw-Hill, USA, 1986.

{Kor88} Korth, H. F., "On Long-Duration CAD Transactions", Information Scienses - An International Journal, vol. 46, Nos. 1/2, 1988.

{Kot88} Kotz, A. M; Dittrich, K. R. & Mulle, J. A., "Supportin Semantic Rules by a Generalized Event/Trigger Mechanism", International Conference on Extending Database Technology, Venice, Itália, Mar. 1988.

{Laf82} Lafue, Gilles M. E., "Semantic Integrity Dependencies and Delayed Integrity Checking", Proc. of 8th Int. Conf. on Very Large Data Bases, Mexico City, Set. 1982.

{Mag86} Magalhães, Maurício F. "Software para Tempo Real", UNICAMP, Campinas, 1986.

{Mag89} Magalhães, L. P.; Delgado, A. L. N.; Ricarte, I. L.; Ruschel, R. C. & Olguin, C. J. M., "Implementação de um Banco de Dados Não Convencional: A esperiência GERPAC/UniCOSMOS", Anais do IV Simpósio Brasileiro de Banco de Dados, Abr. 1989 e Anais das 18 Jornadas Argentinas de Informática e Investigacion Operativa, Buenos Aires, Argentina, Ago. 1989.

{Mat88} Mattos, N., "KRISYS - A Multi-layered Prototype KBMS Supporting Knowledge Independence, em: Proc. Int. Computer Science Conf. - Artificial Inteligence: Theorie and Applications, Hong Kong, Dez. 1988.

{Mat89} Mattos, Nelson M., "An Approach to Knowledge Base Management: requirements, knowledge representation and design issues", Tese de Doutorado, Unversidade de Kaiserlautern, Alemanha, Abril 1989.

{McC89} McCarthy, Dennis R. & Dayal, Umeshwar "The Architecture of an Active Data Base Managemente System", SIGMOD RECORD, Vol. 18, No. 2, Jun. 1989.

{Med91} Medeiros, Claudia B. & Pfeffer, P., "Transformação de um Banco de de Dados Orientado a Objetos em um BD Ativo", Anais do VI Simpósio Brasileiro de Banco de Dados, Manaus, AM, 22 a 24 de Maio de 1991.

{Mel88} Melo, Rubens Nascimento, "Banco de Dados Não Convencionais: A Tecnologia do BD e suas Novas Áreas de Aplicação", VI Escola de Computação, Campinas, SP, 1988.

{Mor89} Morgenstern, M., "Constraint-Based Systems: Knowledge About Data", Proceedings from Second International Conference on Expert Database Systems, 1989.

{Mos87} Moss, J. Eliot B., "Log-Based Recovery for Nested Transactions", Proc. of 13th Int. Conf. on Very Large Data Bases, Brighton, 1987.

{Neu82} Neumann, T. & Hornung, C., "Consistency and Transactiond in CAD Database, Proceedings from the 8th International Conference on Very Large Data Bases, Mexico-City, 1982.

(Nou91) Nourani, Farid & Magalhães, Léo Pini, "Um Sistema de Tratamento de Restrições para Aplicações de Projeto no Contexto GERPAC/UNICOSMOS", Anais do VI Simpósio Brasileiro de Banco de Dados, Manaus, AM, 22 a 24 de Maio de 1991.

(Nou92) Nourani, Farid, "STR - Subsistema de Tratamento de Restrições do SGBD GERPAC", Relatório Técnico, DCA/FEE/UNICAMP, Agosto de 1992.

(Olg90) Olguin, C. J. M. & Magalhães, L. P., Um Gerenciador de Transações no Contexto do GERPAC/UniCOSMOS, Anais do V Simpósio Brasileiro de Banco de Dados, 1990.

(Olg90a) Olguin, C. J. M., "Gerenciamento de Transações no Contexto do GERPAC/UniCOSMOS", Tese de Mestrado, DCA-FEE-UMICAMP, Maio 1990.

(Pau87) Paul, H. B.; Schek, H. J.; Scholl, M. H.; Weikum, G. & Deppisch, U., "Architecture and Implementation of the Darmstadt Databse Kernel System", Proc. of ACM SIGMOD Conf., San Francisco, 1987.

(Pec88) Peckham, Joan & Maryanski, Fred, "Semantic Data Models", ACM Computing Surveys, Vol. 20, No. 3, Set. 1988.

(Pot88) Potter, Walter D. & Trueblood, Robert P., "Traditional, Semantic, and Hyper-Semantic Approaches to Data Modeling", IEEE Computer, Jun. 1988.

(Pri89) Price, Roberto Tom & Golendziner, Lia G., "Bancos de Dados para Aplicações Não-Convencionais", IV Escola Brasileira e Argentina de Informática, Termas de Rio Hondo, Argentina, Jan. 1989.

(Ric87) Ricarte, I. L. & Delgado, A. L. N., "GERPAC - Um SGBD para PAC", Anais do II Simpósio Brasileiro de Banco de Dados, Maio 1987.

(Ric87a) Ricarte, I. L., "Sistemas de gerência de dados para projeto auxiliado por computador", Tese de Mestrado, DCA-FEE-UNICAMP, Jan. 1987.

(Row87) Row, L. A. & Stonebraker, M. R., "The POSTGRES Data Model", Proc. of 13th Int. Conf. on Very Large Data Bases, Brighton, 1987.

{Sil91} Silveira, Mamede A. M., "Extensão do Núcleo UNICOSMOS para Suporte à Orientação por Objetos", Tese de Mestrado, DCA, FEE, UNICAMP, Abril 1991.

{Smi87} Smith, Peter D. & Barnes, G. Michael "Files & Databases: An Introduction", Addison-Wesley, USA, 1987.

{Sow88} Sowa, John F., "Knowledge Representation in Database, Expert Systems, and Natural Language", in Proc. of Working Conference on the Role of Artificial Inteligence in Databases and Information Systems, Guangzhou, China, Julho 1988.

{Sto76} Stonebraker, M., "The Design and Implementation of INGRES", ACM-TODS, Set. 1976.

{Sto86} Stonebraker, M., "The Design of POSTGRES", SIGMOD Record, Vol. 15, No. 2, Fev. 1986. e em: Proc. of Int. Conf. on Management of Data, Washington D.C., Maio 1986.

{Sto87} Stonebraker, M., "The Design of POSTGRES Storage System", Proc. of 13th Int. Conf. on Very Large Data Bases, Brighton, 1987.

{Sto88} Stonebraker, Michael; Hanson, Eric N. & Potamianos S., "The POSTGRES Rule Manager", IEEE Transaction on Software Engineering, Vol. 14, No. 2, Jul. 1988.

{Toz91} Tozzi, Clésio Luis, "PAC - Projeto Auxiliado por Computador", Editora UNICAMP, Campinas, 1991.

{Tsi82} Tsichritzis, Dionysios C. & Lochovsky, Frederick H., "Data Models", Prentice-Hall, Inc., USA, 1982.

{Ull82} Ullman, Jeffrey D. "Principle of Database Systems", Computer Science Press Inc., USA, 1982.

{Vet82} Vetter, M. & Maddison, R. N., "Database Design Methodology", Prentice-Hall Int. Inc., 1982.

{Wal89} Wald, Joseph A., "Implementing Constraints in a Knowledge Base", em: Proc. of the 2nd Int. Conf. on Expert Database Systems, 1989.

{Wer86} Wertz, Charles J., "The Data Dictionary: Concepts and Uses", Inf. Science, Inc., USA, 1986.

{Wie83} Wiederhold, Gio, "Database Design", McGraw-Hill, 1983.

{Wie84} Wiederhold, G. "Knowledge and Database Management", IEEE Software, Jan. 1984.

{Yan88} Yannaroudakis, E. J. "The Architectural Logic of Database Systems", Springer-Verlag, UK, 1988.