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

Este exemplar corresponde a redação final da tese defendida por Carlos José Maria Olguín e aprovada pela Comissão Julgadora em 04/maio/1990.

31/7/90

GERENCIAMENTO DE TRANSAÇÕES NO CONTEXTO DO GERPAC/UNICOSMOS

por: *Lic. Carlos José Maria Olguín*
orientador: *Prof. Dr. Léo Pini Magalhães*

Tese apresentada à Faculdade de Engenharia Elétrica FEE - UNICAMP como parte dos requisitos exigidos para a obtenção do título de MESTRE EM ENGENHARIA.

4 de Maio de 1990

Este trabalho contou com o apoio financeiro do
Conselho Nacional de Desenvolvimento Científico e Tecnológico -
CNPq

Dedico este trabalho aos meus pais,
ALFREDO e SUSANA,
que foram o começo de tudo.

C.J.M.OLGUÍN

AGRADECIMENTOS

Aos meus irmãos Alfredo, Ignacio, Paola e Marcelo. Simplesmente por serem.

À Conceição, presença fundamental nestes últimos tempos.

Ao Prof. Dr. Léo Pini Magalhães. Não só pela orientação neste trabalho como também pela amizade e confiança em mim depositados.

Às grandes amigas Ana e Adriana que souberam me ajudar e me deram ânimo quando dele precisei.

Ao Ivan, Armando e Regina, pelo apoio e amizade.

Aos colegas do grupo PAC/BD, pelo companheirismo.

Aos amigos Luiz, Roberto, Thomas, Aluizio por me terem "aturado" (não é fácil) e compartilhado bons momentos.

Aos amigos Fernando, Mirta, Mónica, Juan, Vicente, Carlos, Mario, Alejandro e tantos outros que seria difícil mencionar.

À Universidade Estadual de Campinas por ter me dado a possibilidade de fazer este trabalho.

Ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), pelo apoio financeiro.

E a todos aqueles que colaboraram, direta ou indiretamente, para a realização deste trabalho.

RESUMO

O presente trabalho define, especifica e implementa uma primeira versão do Sistema de Controle de Transações do GERPAC. Transações a dois níveis, assim como definido no MER/PAC - Modelo Entidade-Relacionamento aplicado a projeto, são abordadas. Assim, há transações internas a um objeto de projeto (transações micro, no jargão MER/PAC) e transações envolvendo diversos objetos de projeto (transações meta-micro). Transações a nível micro, são transações tradicionais, ou seja transações no contexto de operações atômicas. Transações a nível meta-micro, já são transações complexas, podendo abrigar o controle de operações entre diferentes níveis de abstração.

ABSTRACT

This work defines, specifies and implements the first version of the transaction management system for GERPAC. Transactions at two different levels, as defined in MER/PAC - Entity-Relationship Model for CAD, are considered. In this way, there exist transactions related to just one project object (micro transactions) and transactions considering many object projects (meta-micro transactions). At the micro level, transactions are standard ones, that is, transactions in the context of atomic operations. On the other hand, at the meta-micro level, transactions are more complex: they include the operating control between different abstraction levels.

ÍNDICE

CAPÍTULO 2 - CONCEITOS BÁSICOS

CAPÍTULO 3 - APRESENTAÇÃO DA SOLUÇÃO

CAPÍTULO 4 - CONCLUSÕES

ANEXOS

Lista de Figuras

"FABRICO um elefante
de meus poucos recursos.
Um pouco de madeira
tirado a velhos móveis
talvez lhe dê apoio..."

"O elefante"
Carlos Drummond de Andrade

CAPÍTULO 1

CAPÍTULO 1 - INTRODUÇÃO

Conteúdo

1.1 - Introdução

A utilização de sistemas computacionais em Engenharia, notadamente na área de auxilio ao desenvolvimento de projetos, motivou diversas pesquisas conjuntas entre Ciência de Computação e Engenharia. Um exemplo clássico é o desenvolvimento alcançado na área de Computação Gráfica, não apenas na representação visual de objetos como também como ferramenta de integração entre usuários e sistemas computacionais.

Uma outra linha de pesquisa envolvendo estas duas áreas, é a do gerenciamento de estruturas para o armazenamento e manipulação de dados. Na área administrativo-comercial, o interesse pelo desenvolvimento de tais estruturas é antigo, culminando com o desenvolvimento de Sistemas de Bancos de Dados, sistemas de software que armazenam, manipulam e controlam grandes quantidades de dados.

Em PAC - Projeto Auxiliado por Computador, a necessidade de se utilizar Sistemas de Gerência de Banco de Dados (SGBD) está identificada devido, entre outros, à necessidade de se ter um controle sobre o fluxo de informação existente entre os vários processos de um projeto, como também sobre o fluxo de informações necessário para um processo em particular, aspectos estes muito importantes em sistemas PAC. Assim, sistemas de banco de dados passam a ter um papel importante como uma ferramenta de suporte também para os sistemas PAC (veja figura 1.1) [/ENCA 80/, /ENCA 82/ e /ENCA 83/], tanto pela sua capacidade de gerir informações com eficácia, como por possibilitarem um

controle dos fluxos de informações entre os diversos processos de projeto que existem em um ambiente PAC.

Em sistemas de projeto em geral e em sistemas PAC em particular, podem-se identificar dois aspectos [/DELG 87/] :

- aspecto **micro**, correspondente aos aspectos envolvidos na definição e manipulação das informações de um objeto de projeto isolado (um aspecto do projeto total);

- aspecto **macro**, correspondente aos aspectos decorrentes da forma como o processo de projeto total evolui e como os processos existentes em um sistema PAC interagem, impondo, eventualmente, novas restrições aos objetos de projeto.

Cada um destes aspectos lida com informações de características diferentes, tendo assim exigências diferentes com relação à forma de manipulação das informações.

A existência destes dois aspectos bem diferenciados no ambiente PAC, impõe tratamentos diversos no que se refere a questões de controle de dados como por exemplo, questões relativas à recuperação da informação (i.e., o processo de levar a um estado correto um banco de dados que foi levado a um estado incorreto ou suspeito após alguma falha de processamento) e à integridade da informação (i.e., a propriedade de um sistema de banco de dados a qual assegura que os valores armazenados são corretos).

É a partir dos aspectos micro e macro próprios dos sistemas PAC que se pode reconhecer dois tipos de transação em relação ao aspecto de recuperação de uma falha, e um terceiro tipo, quando consideramos questões de integridade. Os tipos de transação relacionados com estas questões são especificamente, as micro transações (Mi), meta-micro transações (mMi) e macro transações (Ma).

O presente trabalho vem propor uma alternativa no desenvolvimento de um gerenciador de transações - a componente de um SGBD responsável pelo controle de execução das transações - para SGBD's orientados a aplicações de

engenharia, aproveitando parte da experiência adquirida na área administrativo-comercial (aplicações ditas "*convencionais*").

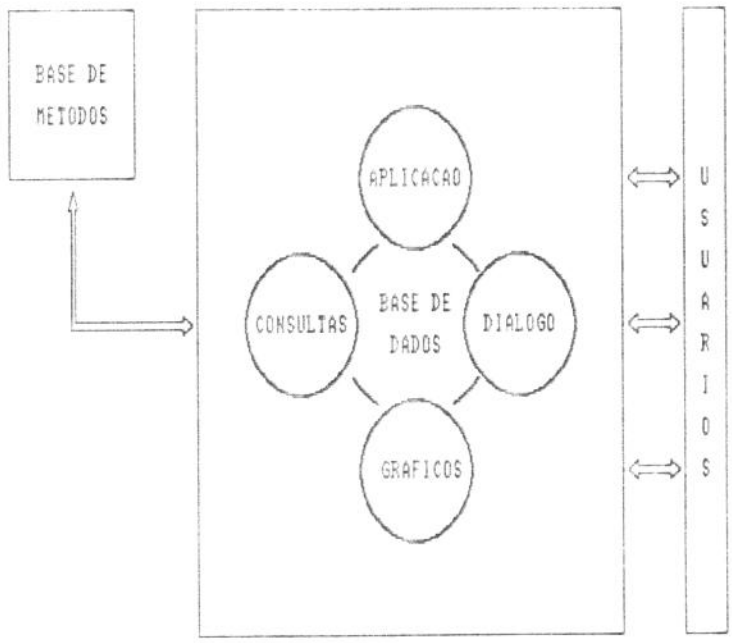


FIGURA 1.1. Sistema PAC

Trabalhos anteriores levantaram as necessidades particulares de bancos de dados para tais sistemas, propondo ferramentas para a modelagem de dados (MER/PAC - Modelo Entidade Relacionamento orientado a projeto) [/DELG 87/] e mostraram uma proposta de implementação de um SGBD com as características desejadas (GERPAC/UniCOSMOS) [/RICA 87/]. O atual trabalho vem apresentar um gerenciador de transações no contexto do GERPAC/UniCOSMOS que contempla, a nível de recuperação da informação, as características que se manifestam em sistemas PAC.

Resumidamente o presente documento é composto pelos seguintes capítulos:

O capítulo 1 constitui esta introdução, onde foi colocado de forma sucinta o problema que será discutido nos próximos capítulos.

No capítulo 2 apresenta-se uma visão geral de sistemas de banco de dados, dando-se ênfase a conceitos associados com transação.

No capítulo 3 é apresentado o problema com o qual se está trabalhando de forma a definir detalhadamente o escopo do mesmo. Finalmente, é apresentada a solução proposta.

No capítulo 4 são apresentadas as conclusões do trabalho, seguidas das sugestões e comentários das possíveis linhas de continuidade para o presente trabalho.

Finalmente, encerrando este trabalho, são apresentados os anexos que documentam a implementação.

1.2 - Referências

/DATE 83/ DATE C.J.
"An Introduction to Database Systems - II"
Addison-Wesley - 1983

/DELG 87/ DELGADO A.L.N.
"Bancos de Dados no Contexto PAC"
Dissertação de Mestrado, FEE/UNICAMP
Janeiro 1987

/ENCA 80/ ENCARNAÇÃO J.
"Computer Aided Design Modelling, Systems Engineering, CAD Systems"
Lecture Notes in Computer Sciences, 89
Springer-Verlag - 1980

/ENCA 82/ ENCARNAÇÃO J. e KRAUSE F. (Eds.)
"File structures and Data Bases for CAD"
North-Holland - 1982

/ENCA 83/ ENCARNAÇÃO J. e SCHLECHTENDAHL E.G.
"Computer Aided Design - Fundamentals and System Architectures"
Springer-Verlag - 1983

/RICA 87/ RICARTE I.L.M.
"Sistemas de Gerência de Dados para Projeto Auxiliado por Computador"
Dissertação de Mestrado, FEE/UNICAMP
Junho 1987

"...Corro atrás do tempo
Vim de não sei onde
Devagar é que
Não se vai longe ..."

"Bom conselho"
Chico Buarque

CAPÍTULO 2

CAPÍTULO 2 - CONCEITOS BÁSICOS

Conteúdo

2.1 - Sistemas de banco de dados (conceitos básicos)

Os sistemas de informação têm sofrido um processo evolutivo considerável. De simples sistemas de gerência de arquivos, estes têm passado a ser (grandes) coleções de dados que servem concorrentemente a um grupo de usuários e a várias aplicações distintas. Estas coleções de dados recebem o nome de *Bases de Dados*.

O processo de projeto de um sistema de banco de dados consiste em, a partir de requerimentos de informação e processamento de um sistema de informação existente, construir uma representação da aplicação que consiga refletir as propriedades *estáticas* e *dinâmicas* necessárias para suportar as transações e consultas requeridas. Num nível de generalização maior, o sistema de banco de dados deveria poder representar as propriedades estáticas e dinâmicas comuns a todas as aplicações sendo, portanto, independente da aplicação. Por outro lado, a representação resultante deve ser capaz de

suportar alterações aos requerimentos das aplicações ou ser facilmente modificada para tais efeitos.

O objetivo principal de um sistema de banco de dados é conseguir *independência de dados*, a qual pode ser entendida como sendo "a imunidade que os programas de aplicação têm em relação às mudanças na estrutura dos dados ou nos métodos de acesso" [/DATE 83/].

As propriedades estáticas de uma aplicação comprendem objetos, propriedades desses objetos (i.e., atributos) e as relações entre os objetos. As proprjedades dinâmicas, por sua vez, incluem operações de modificação sobre os objetos e as relações entre as operações. Aquelas propriedades que não podem ser representadas como operações ou objetos são expressas como *condições de integridade semântica*, as quais podem ser consideradas condições lógicas. Alguns exemplos de condições de integridade semântica (assertivas) são:

- todo produto deve possuir um número único de identificação;
- a superficie de um dos lados de um objeto não pode exceder os 40 m2;
- o comprimento da asa de um avião não pode ser menor do que 4 m.

Como resultado de projetar um sistema de banco de dados é obtido, por um lado, um esquema e, pelo outro, especificações de consultas e transações. Os esquemas definem as propriedades estáticas da base de dados enquanto as consultas e transações definem as propriedades dinâmicas.

Um esquema consiste de definições de todos os tipos de objetos da aplicação (atributos, relações e condições).

Associado ao conceito de esquema encontram-se os conceitos de base de dados, subesquema e integridade lógica de uma base de dados.

O conceito de subesquema está associado à parte que um determinado processo de uma aplicação precisa enxergar de um esquema, isto é, o processo pode precisar operar com um subconjunto prédeterminado dos objetos definidos. Finalmente, diz-se que uma base de dados possui integridade lógica se os valores da base de dados são instâncias válidas dos tipos definidos no esquema e se todas as condições de integridade semântica são satisfeitas.

Um dos objetivos de um sistema de gerenciamento de banco de dados é responder consultas e suportar transações. Uma consulta pode ser entendida

como sendo uma expressão lógica sobre os objetos e relações definidas no esquema que produz, como resultado, a identificação de um subconjunto da base de dados. Uma transação consiste de várias operações de consulta e atualização sobre objetos em um subesquema (que pode coincidir com o esquema) e é usada para definir operações ou eventos da aplicação.

Um modelo de dados é um conjunto de conceitos, formalmente definidos, que ajudam a considerar e expressar as propriedades estáticas e dinâmicas e as condições de integridade semântica de uma aplicação. Um modelo de dados fornece as bases sintáticas e semânticas para ferramentas e técnicas usadas para projetar e empregar um sistema de banco de dados. Ferramentas associadas com modelos de dados são as linguagens para definir, manipular, consultar e suportar a evolução das bases de dados.

A maior parte dos sistemas de gerenciamento de banco de dados (SGBD ou DBMS na sua versão inglesa) possuem uma linguagem de definição de dados (DDL, Data Definition Language), para definir esquemas e subesquemas; uma linguagem de manipulação de dados (DML, Data Manipulation Language), para escrever os programas que manipulam a base de dados e uma linguagem de consulta (QL, Query Language) para escrever consultas.

Um sistema de gerência de bases de dados é um sistema que implementa as ferramentas associadas a um modelo de dados.

O emprego de sistemas de gerenciamento de banco de dados nas áreas comerciais (sistemas bancários, sistemas de estoque, etc.) tem tido um sucesso muito grande já que, através da utilização deste tipo de sistemas, é possível refletir as propriedades estáticas e dinâmicas, mencionadas anteriormente, que se apresentam nas aplicações da área comercial.

2.2 - Aspectos de controle de um SGBD

O controle pelo qual o SGBD é responsável diz respeito, entre outros, à questão da consistência, ou seja, a garantia de que os dados armazenados correspondem a um estado correto em relação à realidade. Neste item será analisado um dos principais aspectos relativos ao controle de dados:

- *recuperação*, abordando o problema de falhas em sistemas computacionais.

2.2.1 - Introdução

Dois são os fatores que induziram à criação do conceito de transação, como entendido dentro do contexto de sistemas de banco de dados.

Por um lado, temos o fato da *existência de falhas* que podem acontecer durante a execução do sistema computacional. As falhas que são consideradas dentro deste contexto podem ter sua origem a partir de erros de programação, erros de hardware, erros de operação, etc. [/DATE 83/].

Esta situação obriga os projetistas de sistemas de banco de dados a considerar estas anomalias de forma tal que seja possivel recuperar o estado inicial do sistema frente à presença de uma falha. Os princípios nos quais se baseia a recuperação são muito simples e podem ser resumidos em uma simples palavra: *redundância*.

Outro dos fatores que contribuíram ao desenvolvimento do conceito de transação, consideramos tenha sido a própria *evolução dos sistemas de gerência de banco de dados*. Nos primeiros tempos, sistemas de gerência de banco de dados eram vistos como sistemas de monoprocessamento em um contexto de trabalho por lotes (batch), onde as consultas eram processadas uma após a outra e sem nenhum tipo de intervenção do usuário, além da ordem de começo por ele fornecida. Esta visão teve uma evolução natural, até chegarmos hoje à existência de sistemas de banco de dados com características de multiprocessamento, dentro de um contexto on-line, onde se pensa na incorporação de interfaces amigáveis entre o usuário e o sistema e, por outro lado, na possibilidade de distribuição dos dados o que significa ter que considerar sistemas de gerenciamento de dados distribuídos.

Nestas condições de processamento o que se tem é a possibilidade de tarefas executando concorrentemente, o que significa por um lado, que estas tarefas podem acessar os mesmos itens de dados e, por outro, que podem ser gerados resultados parciais que são lidos por outras tarefas podendo ocasionar erros nos resultados finais.

Uma situação possível é detalhada a seguir (ver figura 2.1). Vamos imaginar que, em um momento dado, temos uma tarefa T1 que começa seu

processamento no mesmo instante em que outra tarefa, denominada aqui como T2, começa também a sua execução. As duas tarefas executam suas ações concorrentemente. Suponhamos que a tarefa T1 depois de ter lido um item de dados I0, continua com a sua execução normal e grava após um intervalo de tempo, um item de dados I1 como resultado de alguma transformação do item lido anteriormente (item I0). Por outro lado a tarefa T2, após a gravação do item de dados I1 pela tarefa T1, lê o conteúdo do item de dados I1 e o emprega para realizar algumas operações concluindo sua execução antes da finalização da tarefa T1. A tarefa T1, que continua com a sua execução, detecta que o valor do item de dados I0 estava errado, o que determina que todo o processamento executado seja invalidado decidindo-se pela anulação do mesmo.

O problema que aqui se apresenta é claro. É necessário de alguma forma, avisar às outras tarefas da anulação das suas ações de forma que situações como as descritas anteriormente possam ser identificadas e tomadas as operações necessárias para sua correção. Desafortunadamente isto não é sempre possível já que as pessoas que estavam executando estas tarefas podem ter abandonado a sessão de trabalho.

A solução proposta para tentar eliminar este tipo de problema tem a sua origem com os trabalhos feitos por Bjork e Davies no ano de 1973 sobre as, assim chamadas, *esferas de controle*. Neste trabalho eles propunham uma forma de isolamento dos dados empregados por uma tarefa de maneira que outros usuários pudessem conhecer o grau de confiabilidade daqueles dados [/BJOR 73/,

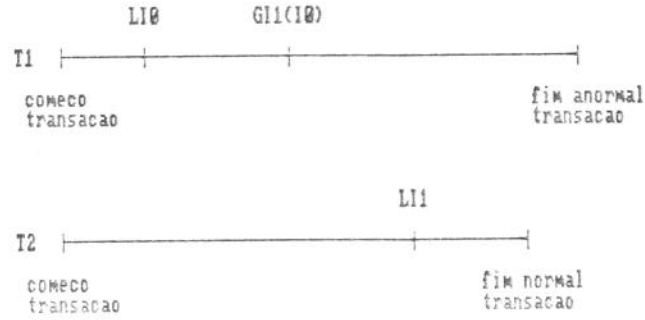


FIGURA 2.1. Possivel execucao de duas transacoes

/DAVI 73/]. Estes conceitos foram restringidos por Eswaran para serem usados na área de banco de dados e receberam o nome de *transação* [/ESWA 76/].

Nas seções seguintes daremos uma definição do conceito de transação, bem como também serão abordados os aspectos de quais as propriedades que as mesmas devem possuir, sua estrutura, os tipos de falhas que podem ocorrer durante a execução de um sistema e as formas de recuperação com que se conta para reconstruir o estado do sistema no ponto anterior ao começo da execução.

2.2.2 - Conceito de transação

Uma base de dados pode ser considerada como uma coleção de objetos, os quais possuem um valor determinado em um momento dado. Por outro lado são definidas operações/ações sobre as entidades as quais realizarão uma transformação do estado das entidades.

Encontram-se definidas também uma série de assertivas sobre os valores que uma entidade dada pode receber como também sobre as operações que se realizam. O conjunto de assertivas é conhecido pelo nome de restrições de consistência.

Uma base de dados que cumpre todas as restrições de consistência é definida como sendo consistente. Por outro lado, se as operações que são executadas sobre uma base de dados consistente cumprem também as restrições de consistência para elas definidas, diz-se que a base de dados passa de um estado consistente a outro estado consistente.

Uma transação pode ser vista então como "... as ações de um processo que são agrupadas em sequências que são unidades de consistência." [/ESWA 76/].

A partir desta definição inicial do conceito de transação, o mesmo foi visto de vários ângulos diferentes dando origem a uma série de definições equivalentes. O importante a ressaltar é que uma transação reflete basicamente a idéia de que as "atividades de um usuário são isoladas de todas as atividades concorrentes" [/HAER 83/]. Do ponto de vista de um sistema de banco de dados uma transação pode ser vista como uma sequência de iterações com a base de dados, que representa uma atividade significativa no ambiente do usuário transformando o estado da base de dados, dito consistente, em outro

estado o qual é também considerado consistente.

Uma observação interessante que pode ser feita em relação à idéia de isolamento das atividades de um usuário é que o tempo de duração da transação vai depender do grau de concorrência que se quer alcançar.

Temos falado até agora que uma transação é um conjunto de atividades próprias de um usuário que são isoladas de outras atividades que estão executando concorrentemente.

Uma pergunta natural de ser feita neste ponto seria, por exemplo, de que forma reconhecemos que uma atividade dada pertence a uma transação, ou então, como podemos saber quando uma transação começa ou termina? Na seção seguinte veremos de que forma uma transação é estruturada e com que mecanismos contamos para identificar o começo ou o fim de uma transação.

2.2.3 - Estrutura de uma transação

Assume-se que uma transação começa com uma instrução *BEGIN_TRANSACTION* e finaliza com uma instrução *COMMIT_TRANSACTION* ou *ROLLBACK* dependendo do tipo de finalização alcançado. No caso da execução de uma instrução *COMMIT_TRANSACTION* considera-se que a transação executou com sucesso, alcançando seu ponto de terminação normal. Agora no caso da execução de uma instrução *ROLLBACK* o que pode ser assumido é que alguma condição de exceção foi encontrada durante a execução das operações incluídas na transação (e.g. dados não válidos), provocando a terminação da execução e a consequente eliminação das operações executadas.

Na figura a seguir mostra-se a estrutura básica de uma transação. (Figura 2.2).

A visão que se tem neste sentido no trabalho apresentado sobre recuperação em /DATE 83/ é que todas as transações, em geral, podem ser consideradas como tendo a seguinte estrutura:

- aceitar uma mensagem de entrada (I);
- executar algum processamento na base de dados (P) (conjunto de ações);
- enviar uma/várias mensagem/s de saída (O).

FIGURA 2.2. Estrutura de uma transacao

2.2.4 - *Propriedades de uma transação*

O conceito de transação, visto no contexto de sistemas de banco de dados como uma sequência de interações com ele, é caracterizado pelas seguintes propriedades:

- *Atomicidade*, ou todas as ações da transação são executadas ou nenhuma o é. A propriedade de atomicidade obriga à eliminação das operações executadas pela transação no caso que ela seja interrompida por uma falha;
- *Consistência*, uma transação que alcança o seu ponto de finalização normal preserva a consistência da base de dados. É assumido que uma transação entrega, na sua finalização, valores consistentes, isto é, que cumprem as restrições de consistência definidas para a base de dados;
- *Isolamento*, uma transação não pode mostrar resultados parciais a outras executando concorrentemente (as técnicas empregadas para alcançar isolamento são conhecidas como sincronização);
- *Persistência*, o sistema deve garantir que os resultados gerados por uma transação que fez "*commitment*" não serão perdidos na presença de uma falha posterior. Considerando que os resultados de uma transação que

devem ser preservados pelo sistema são armazenados na base de dados, a atividade de manter a persistência das transações é chamada de recuperação da base de dados ("*database recovery*").

As propriedades apresentadas até agora são basicamente as que uma transação deve cumprir para possuir a característica de *indivisibilidade* na execução de suas operações, i.e., ou todas as ações são propriamente refletidas na base de dados ou nenhuma o é. Nenhum tipo de modificação é feita na base de dados caso em algum ponto anterior ao ponto de "*commit*" seja encontrada uma condição de exceção.

Algumas observações podem ser feitas com relação às propriedades. Uma das razões pelas quais a propriedade de isolamento é necessária é pelo fato de tentar eliminar o problema de cancelamentos em cascata ("*cascading aborts*") ou seja, a necessidade de eliminar todas as transações que viram resultados parciais de uma transação que foi cancelada posteriormente.

Finalmente, uma outra propriedade que é mencionada na literatura sobre recuperação é [/CERI 84/]:

- *Seriabilidade*, se várias transações são executadas concorrentemente, o resultado deve ser o mesmo caso elas executassem serialmente mantendo alguma ordem. A atividade de garantir a seriabilidade de uma transação é conhecida com o nome de controle de concorrência ("*concurrency control*"). Esta característica dos sistemas possibilita que os programadores codifiquem transações como se elas fossem executadas isoladamente.

2.2.5 - *Tipos de falhas*

Como mencionado anteriormente, uma transação pode ser vista como tendo uma estrutura típica onde o seu começo é identificado através da instrução *BEGIN_TRANSACTION* seguido de uma série de operações próprias da aplicação à qual pertence a transação e, finalmente, a execução da instrução *COMMIT_TRANSACTION* ou *ROLLBACK* dependendo da terminação normal ou não da transação.

Podemos então definir três formas de terminação de uma transação em função das condições de processamento que se apresentam durante a execução (ver figura 2.3):

- a transação alcança seu ponto de finalização normal, isto é, seu *COMMIT_TRANSACTION*;
- a transação detecta um erro na informação de entrada ou algum outro tipo de violação nas restrições de consistência e solicita a execução de uma instrução *ROLLBACK*;

FIGURA 2.3. Formas de terminacao de uma transacao

- o sistema detecta que a transação não está executando dentro de condições de processamento normais para ele (tempo expirado ou "*deadlock*") e cancela a transação.

Nos casos em que a transação não alcança seu ponto de finalização normal é considerado que esta situação foi ocasionada pela presença de uma falha (condição anormal de processamento). Falhas podem ser classificadas segundo a origem, isto é:

- *falhas de transação*, por alguma condição de erro não considerada a transação não alcança seu "*commit*" e as operações executadas devem ser desfeitas;
- *falhas do sistema*, este tipo de falha é originado geralmente por um erro de código no SGBD, uma falha no sistema operacional ou uma falha de hardware onde o conteúdo da memória principal é perdido;
- *falhas do dispositivo de armazenamento*, o que acontece neste tipo de

falha é que há perda total ou parcial da memória secundária que contém a base de dados. Este tipo de falha é provocado normalmente por um erro no sistema operacional, erro de hardware, queda dos cabeçotes ou perda de informação por desmagnetização.

Estes tipos de falhas provocam que o estado do sistema não seja consistente em um dado momento. É definida então uma série de ações, chamadas de recuperação, que possibilitam levar o estado do sistema ao estado inicial considerado como consistente.

2.2.6 - Formas de recuperação

Como dito anteriormente, o estado de uma base de dados é visto como sendo consistente se os valores que cada uma de suas entidades cumpre as restrições de consistência definidas. Por outro lado, pode-se dizer que a base de dados é consistente se ela contém os resultados de transações executadas com sucesso.

Consideramos que uma transação entrega sempre valores confiáveis, isto é, leva a base de dados de um estado consistente a outro estado consistente [/DELG 87/]. Porém, durante a execução das ações de uma transação o estado pode encontrar-se temporariamente inconsistente. Esta situação pode ter consequências desagradáveis na presença de falhas já que a inconsistência pode se tornar permanente. Por esta razão é necessário executar uma série de operações, chamadas de recuperação, para reconstruir o estado da base de dados, ou seja, levá-lo ao estado de consistência inicial.

Tomemos como exemplo a execução de uma transação que realiza a transferência de fundos entre contas. A sequência lógica de operação seria, uma vez obtidos os valores iniciais, verificar se a conta a debitar tem fundos, debitar o valor indicado pela transação, creditar a quantia na conta destino e gerar uma mensagem para indicar a finalização da transação. Vamos imaginar por um instante que uma falha ocorra entre as operações de debitar de uma conta e creditar na outra provocando desta forma o cancelamento da execução havendo, portanto, deixado a base de dados inconsistente já que a quantia debitada a uma conta não foi creditada na outra.

Esta situação deve ser modificada de forma a obter novamente um estado consistente da base de dados. Com o objetivo de levar a base de dados ao estado inicial (considerado como consistente) é empregada uma série de ações, UNDO (desfazer) e REDO (refazer), que prevêm diferentes situações.

Definiremos a seguir estas ações considerando a situação prevista.

- *UNDO parcial*, este tipo de ação é executada devido a uma falha de transação. Por definição a ação UNDO desfaz todos os efeitos da transação e não influencia nenhuma outra;
- *UNDO global*, esta ação de recuperação surge a partir de uma falha do sistema. O efeito que ela tem é o de desfazer todas as ações que foram executadas por transações não completadas no momento em que se produziu a falha;
- *REDO parcial*, quando recuperamos o estado do sistema como consequência de uma falha do sistema, devido à qual a execução terminou de uma forma não comum, os resultados de algumas transações completadas podem não ter sido gravados na base de dados. Assim elas devem ser repetidas, se necessário, pelo componente de recuperação;
- *REDO global*, neste caso o que assumimos é que existe perda física da base de dados. Deve-se recuperar então a partir do arquivo "*back-up*" e refazer todas as ações que foram executadas desde a data de back-up até o momento da falha. Este tipo de situação é provocado na presença de uma falha do dispositivo de armazenamento.

Uma observação importante neste ponto, é que tanto uma ação de UNDO (desfazer) quanto uma de REDO (refazer) tem que ter a característica de *idempotência*, isto é,

(UNDO (UNDO (... (UNDO (X)) ...))) = UNDO (X)

e

(REDO (REDO (... (REDO (X)) ...))) = REDO (X)

Este fato deve-se basicamente à possibilidade de existir uma falha no momento em que se está recuperando uma falha anterior.

Como dito anteriormente, os princípios nos quais se baseiam as ações de recuperação são muito simples e podem ser resumidos em uma simples palavra:

redundância. O significado deste conceito é o de que uma parte de informação possa ser recuperada a partir de outra parte de informação, armazenada redundantemente em algum lugar do sistema [/DATE 83/]. A redundância é alcançada neste contexto, através de um arquivo "*back-up*" o qual é uma cópia da base de dados e também a partir do que se chama de arquivo "*log*" que é um arquivo onde se mantém a informação necessária para poder, em situações especiais, eliminar ou refazer as ações que foram executadas sobre a base de dados. Isto é, toda vez que se faz uma mudança na base de dados, um registro contendo os valores antigo e novo do item modificado é escrito no arquivo "*log*". Na nossa abordagem a informação que é mantida é a informação associada às páginas, as quais são o elemento de manipulação do sistema.

Uma maneira de agilizar o processo de recuperação é a tomada de "*checkpoints*". Uma tomada de "*checkpoint*" consiste do processo de efetivar, ou seja, transferir de memória principal para memória secundária, em intervalos pré-especificados, as informações referentes ao arquivo "*log*" e às alterações na base de dados, armazenando informações sobre o estado do sistema no momento desta operação em um arquivo de "*checkpoint*". Desta forma pode-se saber quais transações devem ser desfeitas ou refeitas no momento do reinício da operação do sistema. Para este propósito, pode-se dividir transações em cinco categorias [/GRAY 81a/]. Sejam Tc o momento da tomada de "*checkpoint*" e Tf o momento (posterior a Tc) em que ocorreu a falha do sistema (veja figura 2.4):

T1. Transação iniciada e encerrada antes de Tc;

T2. Transação iniciada antes de Tc e encerrada antes de Tf;

T3. Transação iniciada antes de Tc mas não encerrada até Tf;

T4. Transação iniciada após Tc e encerrada antes de Tf;

T5. Transação iniciada após Tc mas não encerrada até Tf.

As transações do tipo T1 não são afetadas por processos de recuperação, pois a tomada de "*checkpoint*" garantiu que as alterações realizadas foram efetivadas. As transações do tipo T3 e T5 (não encerradas até o momento da falha) devem ser desfeitas, utilizando o arquivo de "*log*". As transações do tipo T2 e T4 (encerradas antes do momento da falha) devem ser refeitas, pois como não houve tomada de "*checkpoint*" entre o momento final da transação e o

momento da falha, não há garantias que as alterações tenham realmente sido efetivadas.

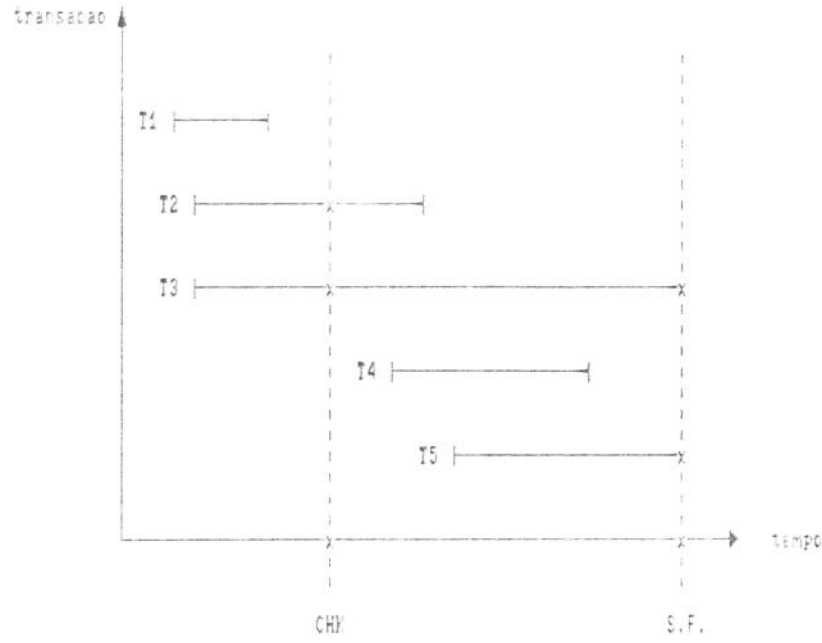


FIGURA 2.4. Tipos de transacoes

Os registros gravados no arquivo "*log*" devem poder ser recuperados de uma maneira seletiva, no lugar de puramente sequencial. É portanto conveniente que o arquivo "*log*" possa ser gravado em um dispositivo de acesso direto. Porém, como sugerido por Gray, um sistema operacional pode gerar cerca de 200 milhões de bytes com informação de "*log*" todo dia [/GRAY 78/]. Por esta razão é inviável manter o arquivo inteiro permanentemente em disco. Em geral o que se faz é o que a seguir se detalha.

Primeiro, o "*log*" ativo[1] é verdadeiramente gravado em disco. Quando o disco está sem espaço físico disponível o gerenciador de "*log*" deriva a outro disco e grava o conteúdo do primeiro em fita magnética. Logicamente, o processo de gravação pode ser feito em paralelo com o uso on-line do novo disco de "*log*". O arquivo de "*log*" total consiste então da parte ativa em uso

[1] Esta denominação é utilizada aqui para significar a parte que está sendo empregada, isto é, aquela que se encontra em disco.

mais um número arbitrário de partes em fita magnética.

O processo de cópia não é realmente tão simples como parece; na verdade o que acontece é que o novo arquivo "*log*" é aberto antes de completado o antigo arquivo "*log*", digamos quando a quantidade gravada do arquivo supera 95% da capacidade do mesmo. Os registros "*log*" de transações que foram iniciadas depois do novo arquivo ser aberto são gravados no novo arquivo, os outros registros continuam sendo gravados no 5% restante. Neste ponto duas coisas podem acontecer; a primeira, o arquivo de "*log*" antigo tem uma condição de overflow em cujo caso as transações que estão empregando o arquivo são terminadas anormalmente e suas ações desfeitas; a segunda coisa que pode acontecer é que o processamento das transações termine normalmente havendo gravado todos os registros de "*log*" necessários no arquivo com espaço sobrando com o que se pode fechar o arquivo.

2.3 - Sistemas de banco de dados para PAC - O GERPAC

2.3.1 - Introdução

Uma vez que o desenvolvimento de um sistema de banco de dados representa um grande esforço e já que o uso de tecnologia de banco de dados surge como necessidade em um grande número de aplicações de engenharia, tenta-se fazer uso de pacotes comerciais disponíveis no mercado em aplicações desta classe [/VERN 84/]. Infelizmente, estes sistemas falham em suportar muitos dos aspectos de aplicações de engenharia [/LORI 82/, /FOIS 82/].

A razão disto é que estes sistemas não foram projetados para propósitos de engenharia, mas para resolver problemas administrativos. As características de banco de dados são bem distintas nestes dois domínios, diferenças estas que são frequentemente mencionadas na literatura [/ENCA 83/].

Um banco de dados administrativo contém poucos tipos de entidades com um grande número de ocorrências, ao contrário de bancos de dados de engenharia, que contêm muitos tipos de entidades com poucas ocorrências. Paralelamente a isto, em engenharia existe a necessidade de modelar e manipular associações

complexas, o que não é o caso em administração, onde as associações são mais simples.

Para ilustrar, tomando-se PAC (Projeto Auxiliado por Computador) como exemplo, o modelamento geométrico de objetos é uma atividade importante. Assim, sistemas de bancos de dados para PAC devem ser capazes de manipular dados geométricos (linhas, círculos, polígonos, etc.). Isto leva à constatação de que os tipos de dados convencionais (inteiros, caráter, etc.), não são suficientes para aplicações PAC, exigindo tipos de dados de estrutura mais complexa de forma a manter a propriedade de independência dos dados.

Além disso, recursos adicionais devem ser oferecidos em sistemas de banco de dados para PAC de forma a garantir a integridade semântica e a consistência de dados, já que variantes múltiplas ou modificações do mesmo objeto devem poder coexistir. Associado a isto, tem-se que em aplicações PAC é necessária a definição de múltiplas visões dos dados, pois o ambiente de projeto consiste de múltiplos processos que acessam os mesmos dados em diferentes estados de desenvolvimento.

De fato, um processo de projeto envolve basicamente o desenvolvimento de um esquema (estrutura de dados), de forma que este se modifica continuamente durante a vida do projeto, até transformar-se em um produto. Tal característica não é considerada em aplicações de administração, onde o esquema tem um tempo de vida relativamente longo.

Estas características, aliadas à dinâmica geral de um processo de projeto, têm consequências importantes no que diz respeito aos períodos em que um sistema de banco de dados para PAC pode vir a estar inconsistente. De um modo geral, um banco de dados pode ser visto como uma coleção de entidades que podem assumir determinados valores, sendo que o conjunto de todos estes valores definem um estado do sistema de banco de dados. Além disso, tem-se associado um conjunto de restrições de consistência e integridade, de modo que um estado de banco de dados que satisfaz estas restrições é considerado consistente.

Como apresentado anteriormente, uma sequência de ações sobre uma base de dados que transfere esta de um estado consistente para outro estado consistente define uma transação. Durante este processo de transferência de

estado, a base de dados pode estar temporariamente em um estado inconsistente.

Outra característica fundamental a ser considerada em sistemas de banco de dados para PAC é a duração dos períodos de inconsistência. Um banco de dados convencional é inconsistente somente durante uma transação, que dura tipicamente apenas alguns segundos. Um banco de dados PAC, pelo contrário, torna-se consistente apenas no final de processo de projeto. Assim, como se verá a seguir, o conceito de transação no ambiente PAC necessita de um tratamento especial.

Ao final de uma transação convencional, as atualizações são validadas usando-se restrições de consistência. Somente atualizações que não violarem as restrições são colocadas à disposição do banco de dados. No entanto, ao final de uma transação no ambiente PAC, duas classes de restrições de consistência devem ser verificadas durante a validação dos dados: restrições locais e restrições globais, em decorrência de projetos possuirem características globais (por exemplo, projetar um avião) e locais (para um avião projetar a asa).

Para compreender melhor esta última observação é bom lembrar os conceitos apresentados na introdução desta monografia sobre os aspectos micro e macro de PAC. Dentro do aspecto micro, um esquema referente a um objeto de projeto é definido em termos de objetos, associações e restrições de formação que devem ser obedecidas por estes elementos. Do ponto de vista macro, consideram-se reflexos da atividade de projeto sobre os vários esquemas definidos no ambiente PAC. Assim, ao se planejar o processo global de projeto, definem-se todos os subprocessos e os esquemas associados a cada um, bem como as precedências entre subprocessos.

Para representar tais aspectos a nível de banco de dados, tem-se que as restrições de formação dos elementos de um objeto de projeto (aspecto micro) são correspondentes a restrições de integridade e consistência de um esquema de banco de dados, restrições estas que são aqui denominadas de restrições locais. Por outro lado, o processo de projeto que atua sobre um determinado esquema pode estar subordinado a um processo de nível superior, que pode ter outros subprocessos subordinados atuando sobre outros esquemas de banco de

dados. Assim, tem-se um conjunto de representações de objetos de projeto (esquemas) que estão interrelacionados. O processo de nível superior segue, portanto, um determinado conjunto de restrições de integridade e consistência de banco de dados que se referem a este interrelacionamento, restrições estas que se denominam restrições globais em relação aos diversos esquemas.

Uma transação é dividida assim, em duas fases que evoluem paralelamente: uma que verifica as restrições locais, e outra que verifica as restrições globais para a validação dos dados de projeto. É importante notar que estas duas fases devem ser tratadas diferentemente, já que dados invalidados globalmente, não devem ser necessariamente perdidos, pois isto não é admissível em um processo de projeto, onde os dados só atingem consistência ao final deste processo.

GERPAC - Sistema Gerenciador de Dados para Aplicações PAC - [/RICA 87/] é uma proposta do grupo de banco de dados do DCA/FEE/UNICAMP de um SGBD que suporta os conceitos do modelo de dados MER/PAC [/DELG 87/], que foi proposto para englobar as necessidades de representação de informações PAC. Desta forma, tal sistema permite a definição e manipulação de tipos de entidades, tipos de relacionamentos, agrupamentos e esquemas, entidades, relacionamentos e seus atributos. Permite ainda o suporte à definição de restrições, sendo adequado tanto para utilização na gerência do aspecto micro como do aspecto macro do processo PAC.

2.3.2 - Comentários finais

O primeiro trabalho desenvolvido pelo grupo de banco de dados do DCA/FEE/UNICAMP foi a implementação do Núcleo de Armazenamento Associativo CORAS (sigla para "*Core System for Associative Storage*"). A primeira implementação de CORAS foi desenvolvida na T.H.Darmstadt (RFA) em linguagem de tempo real PEARL, sendo sua especificação derivada de outros sistemas associativos (LEAP, SAM, DATAS) [/WU 83/]. Entre os trabalhos desenvolvidos com CORAS em Darmstadt, incluem-se extensões para manipulações de dados geométricos para aplicações PAC, mostrando a viabilidade da utilização deste sistema nesta área [/BARO 82/]. A implementação do grupo UNICAMP foi

desenvolvida em FORTRAN, havendo versões para os sistemas PDP/45, VAX 780 e compatíveis com IBM-PC. A utilização do sistema CORAS-UNICAMP como núcleo de um SGBD foi inicialmente proposta em /WU 83/ para o suporte a uma extensão do MER desenvolvida para aplicações em tempo real.

Da experiência adquirida destas propostas, as principais deficiências de CORAS nestas aplicações foram identificadas, levando à definição de um novo núcleo para SGBD, o Sistema de Gerência de Armazenamento Associativo (SICA, agora renomeado UniCOSMOS). A proposta UniCOSMOS absorve os principais conceitos de CORAS, introduzindo outros com a finalidade de otimizar sua utilização em SGBDs não convencionais. A implementação do UniCOSMOS foi desenvolvida em linguagem C em um ambiente UNIX System V.

O GERPAC é a primeira proposta de SGBD que utiliza UniCOSMOS como núcleo. UniCOSMOS é independente de modelos de dados, de modo que as características particulares ao modelo, no caso o MER/PAC, são suportadas através de uma interface entre rotinas do núcleo e as funções GERPAC.

Neste capítulo tentou-se apresentar uma visão geral sobre os conceitos associados a Sistemas de Banco de Dados tanto da área administrativo-comercial ("*convencional*") quanto da área de engenharia ("*não convencional*") como também apresentou-se o conceito de transação e alguns aspectos associados ao mesmo. Os pontos que foram abordados abrangem principalmente aqueles que constituirão a base para o desenvolvimento de capítulos posteriores, não havendo a pretensão de se ter tratado aqui todas as questões referentes ao assunto ou mesmo detalhar todos os pontos expostos.

2.4 - Referências

/BARO 82/ BARON N. et alii
"An Approach to the Integration of Geometrical Capabilities into a Database for CAD Applications"
em /ENCA 82/

/BJOR 73/ BJORK L.A.
"Recovery scenario for a DB/DC system"
Proc.of the ACM 73 National Conference - Ago.73

/CERI 84/ CERI S.
"Distributed databases, principles and systems"
Mc.Graw-Hill Co. - 1984

/DATE 83/ DATE C.J.
"An Introduction to Database Systems - Volume II"
Addison-Wesley - 1983

/DAVI 73/ DAVIES C.T.
"Recovery semantics for a DB/DC system"
Proc.of the ACM 73 National Conference - Ago.73

/DELG 87/ DELGADO A.L.N.
"Bancos de dados no contexto de projeto auxiliado por computador"
Tese de mestrado DCA-FEE-UNICAMP - Jan.87

/EAST 81/ EASTMAN C.M.
"Database Facilities for Engineering Design"
Proceedings of the IEEE - 69 (10) - 1981

/ENCA 82/ ENCARNAÇÃO J. e KRAUSE F. (Eds.)
"File Structures and Data Bases for CAD"
North-Holland - 1982

/ENCA 83/ ENCARNAÇÃO J. e SCHLECHTENDHAL E.G.
"Computer Aided Design - Fundamentals and System Architectures"
Springer-Verlag - 1983

/ESWA 76/ ESWARAN K.P., GRAY J.N., LORIE R.A. e TRAIGER I.L.
"The notions of consistency and predicate locks in a database system"
Communications of the ACM - Nov.76

/FOIS 82/ FOISSEAU J. e VALETTE F.R.
"A Computer Aided Design Data Model: FLOREAL"
em /ENCA 82/

/GRAY 78/ GRAY J.N.
"Notes on database operating systems"
Lecture notes in computer sciences, Vol.60
Springer-Verlag - 1978

/GRAY 80/ GRAY J.N.
"A transaction model"
Lecture notes in computer sciences, Vol.85
Springer-Verlag - 1980

/GRAY 81/ GRAY J.N.
"The Transaction Concept: Virtues and Limitations"
Proceedings of the International Conference on Very Large Data Bases - 1981

/GRAY 81a/ GRAY J.N., McJONES P., BLASGEN M., LINDSAY B., LORIE R., PRICE T., PUTZOLU F. e TRAIGER I.
"The Recovery Manager of the System R Database Manager"
ACM Computing Surveys - Jun.1981

/HAER 83/ HAERDER T. e REUTER A.
"Principles of transaction-oriented database recovery"
ACM Computing Surveys - Dez.83

/LORI 82/ LORIE R.A.
"Issues in Database for Design Applications"
em /ENCA 82/

/NEUM 80/ NEUMANN T.
"CAD Databases Requirements and Architectures"
Lecture Notes in Computer Science - 89
Springer-Verlag - 1980

/RICA 87/ RICARTE I.L.M.
"Sistemas de gerência de dados para Projeto auxiliado por computador"
Tese de mestrado DCA-FEE-UNICAMP - Jun.1987

/VERN 84/ VERNADAT F.B.
"A Commented and Indexed Bibliography on Data Structuring and Data Management in CAD/CAM: 1970 to Mid-1983"
National Research Council of Canada No. 23373
Março 1984

/WU 83/ WU S.T.
"Implementação de um Núcleo de Banco de Dados baseado na Filosofia de CORAS"
Documentação Preliminar, DCA/FEE/UNICAMP - 1983

"En la diversidad de los hombres
y de las razas, reside el secreto
del arte, de la ciencia,
del avance de la civilización."

"Si todos los hombres..."
José Narosky

CAPÍTULO 3

CAPÍTULO 3 - APRESENTAÇÃO DA SOLUÇÃO

Conteúdo

3.1 - Introdução - Transações no contexto PAC

Em ambientes PAC são manipulados objetos de projeto os quais são normalmente construídos a partir de outros objetos que por sua vez, podem ser compostos de objetos menores; qualquer modificação de uma parte pode levar à

alteração dos objetos de projeto que possuem aquele objeto como uma componente. Assim, pode-se ver que neste ambiente é importante considerar uma atualização não só como uma operação envolvendo um objeto de projeto (*nível micro de projeto*) mas também de que maneira isto se reflete nos objetos de projeto associados (*nível macro de projeto*). Esta peculiaridade das aplicações de projeto exige uma abordagem de transação diferente da tradicional. Em nossa abordagem consideraremos três tipos de transação diferentes, chamadas aqui de *transação micro, transação meta-micro e transação macro*, correspondendo as duas primeiras à questão de recuperação a falhas a nível micro e macro de projeto e a terceira à questão do controle de consistência dentro do ambiente de projeto.

Sendo o tratamento da recuperação de falhas o aspecto central deste trabalho, os conceitos de transação micro e meta-micro serão considerados em maior detalhe, considerando-se em menor profundidade as transações macro, que tratam do controle de consistência.

3.1.1 - Transações micro

Uma transação micro é vista como uma transação conforme o modelo popularizado por Jim Gray [/GRAY 78/] no qual a visão que se tem da transação é a de uma sequência atômica de operações de leitura e escrita executada sobre uma base de dados. Em outras palavras, quando uma transação finaliza com sucesso ("*commit*") todas as atualizações são tornadas permanentes na base de dados e, quando a transação falha, o efeito das atualizações até o ponto da falha é completamente eliminado da base de dados.

As propriedades das transações definidas em /GRAY 80/ e /HAER 83/ de *atomicidade, consistência, isolamento e persistência* permanecem válidas para transações micro.

Este modelo de transações tem funcionado efetivamente em aplicações de processamento de dados chamadas convencionais (ou seja, sistemas bancários, aplicações comerciais, etc.) onde a duração de uma transação é considerada curta. Neste tipo de aplicação é razoável prevenir o acesso aos dados que estão sendo atualizados por uma transação forçando a espera de outras

transações, já que a espera será curta. É razoável também, se uma transação falha, eliminar todas as atualizações realizadas pela mesma, já que relativamente pouco trabalho será perdido.

3.1.2 - Transações meta-micro

Em ambientes de projeto, a unidade de execução implícita no conceito de transação (aqui denominada transação micro) não é suficiente para atender as características de operações que envolvam mais de um objeto isolado.

Assim, o GERPAC/UniCOSMOS introduziu o conceito de transação meta-micro, a qual pode ser vista como:

- uma transação de longa duração que não contém nenhuma transação micro;
- uma transação de longa duração que aninha diversas transações micro.

Em ambos os casos, o usuário define explicitamente os pontos de atuação, conforme a figura 3.1.

Este conceito complementa o modelo de Gray, que não é apropriado ao caso em que a duração da transação é longa, por envolver, por exemplo, decisões de projetistas.

É portanto altamente indesejável forçar a espera de transações até que outras transações com conflitos de acesso aos dados terminem ou eliminar todas as atualizações do sistema sobre a base de dados na presença de uma falha. Quando uma transação meta-micro falha, a base de dados é levada ao estado em que se encontrava antes do começo da transação micro corrente e não ao estado anterior da base de dados no início de execução da transação meta-micro. Esta peculiaridade das transações meta-micro faz com que de alguma maneira sejam implicitamente definidos pontos de retorno ("*checkpoints*") associados ao começo das transações micro que compõem a transação meta-micro. Este conceito será abordado a um nível de detalhamento maior em seções posteriores.

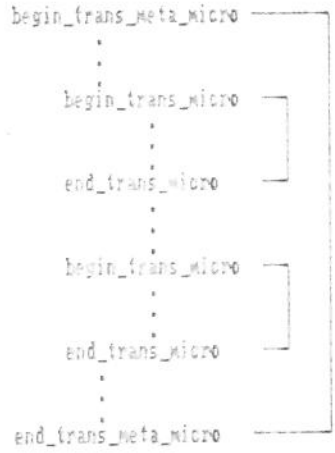


Figura 3.1. Estrutura de uma transacao meta-micro

3.1.3 - Transações macro

O nome de transação, aqui utilizado, buscou explicitar a analogia de uma falha semântica, com uma falha de sistema (conceito de transação).

Assim, transações macro estão associadas à questão de consistência, ou seja, elas verificam se restrições de consistência a nível de objetos de projeto são observadas.

Uma transação macro não permite, contudo, que dados globalmente inválidos sejam perdidos pois isto é inadmissível em um processo de projeto, onde os dados só alcançam consistência no final do processo.

Transações macro serão suportadas no GERPAC/UniCOSMOS por um sistema de tratamento de restrições, que analogamente ao sistema de tratamento de transações tratado no presente trabalho, é implementado de forma independente, comunicando-se com o GERPAC/UniCOSMOS através de mensagens. Atualmente, uma tese de mestrado do DCA/FEE/UNICAMP trata este tema e /BANC 85/ e /NEUM 82/ referenciam também esta temática.

3.2 - Especificação da solução proposta

Nesta seção serão abordados os principais aspectos do sistema de gerência de transações do GERPAC/UniCOSMOS. Assim, questões relativas ao gerenciamento de dois tipos de transações definidas dentro de nosso contexto serão consideradas, a saber: transações micro e transações meta-micro. Serão definidos também os comandos de controle associados a cada tipo de transação e comentadas as características do arquivo de "*log*".

Finalmente será feita uma comparação entre o modelo de transação aqui adotado com o modelo proposto por Jim Gray em /GRAY 78/ e /GRAY 80/.

3.2.1 - Transações micro

Como mencionado anteriormente, uma transação pode ser vista como uma sequência atômica de operações de leitura e escrita que é executada sobre uma base de dados. Em outras palavras, quando uma transação finaliza com sucesso todas as atualizações são tornadas permanentes na base de dados e, quando a transação falha, o efeito das atualizações até o ponto da falha é completamente eliminado da base de dados. Esta sequência de operações deve ser delimitada de alguma maneira para, desta forma, saber qual é o começo e qual o fim da transação. Para tais efeitos consideraremos no nosso modelo os comandos de controle de transação propostos por Gray, i.e., *begin_transaction*, *commit_transaction* e *abort_transaction*[1].

No nosso modelo as propriedades das transações como definidas por /GRAY 80/ e /HAER 83/ mantêm sua validade, já que transações micro são análogas àquelas ditas convencionais.

A operação de "*checkpoint*" está implicitamente definida na execução do comando *commit_transaction*, ou seja, toda vez que o comando *commit_transaction* é executado, as atualizações realizadas são feitas permanentes na base de dados estável (i.e., na memória secundária). Desta forma podem-se identificar

---

[1] Para este último utilizaremos a denominação dada por DATE em /DATE 83/ de *rollback_transaction*.

algumas diferenças entre o presente modelo e o apresentado por Gray em /GRAY 78/ e /GRAY 80/ as quais serão mencionadas na seção 3.2.4.

A semântica dos comandos de controle *begin_transaction*, *commit_transaction* e *rollback_transaction* para transações micro é idêntica à considerada por Gray, não necessitando maiores explicações.

Este modelo de transações tem funcionado efetivamente em aplicações de processamento de dados chamadas convencionais (ou seja, sistemas bancários, aplicações comerciais, etc.) porém, não se tem mostrado apropriado para ambientes de projeto sendo necessário definir transações com outro tipo de características que considerem as necessidades dos ambientes de projeto. Assim, é definido no nosso modelo um outro tipo de transação que considera as características próprias dos ambientes de projeto, chamadas de transações meta-micro, as quais serão discutidas na seção 3.2.2.

3.2.1.1 - Descrição dos comandos de controle de transação micro

Como mencionado anteriormente, os comandos de controle de transação são utilizados entre outros para indicar em que ponto começa uma transação, quando termina, etc.

Os comandos de controle de transação são descritos na continuação com o intuito de definir quais operações estão associadas a eles. Questões de implementação de cada um dos comandos de controle serão discutidas no apêndice desta monografia.

- *begin_transaction micro*

Implica na criação de um registro, no arquivo de "*log*", correspondente à transação que está sendo iniciada.

A informação que é preciso gerar é o identificador do registro, isto é:

- nome do arquivo,
- nome do programa,
- identificador da transação,

- data e hora (AA-MM-DD-HHHH).

- *commit_transaction micro*

Implica na execução das seguintes operações:

- gravar a informação obtida no arquivo "*log*" do disco (memória secundária). Observações: neste momento considera-se que a execução de todo commit faz com que a informação do "*log*" seja estável (i.e., seja gravada na memória secundária) e sejam efetivadas as alterações na base de dados . Dito de outra maneira, a execução de um *commit_transaction* tem associado um "*checkpoint*" implícito;
- gravar páginas atualizadas em memória secundária (base de dados);
- marcar as páginas alteradas como gravadas na tabela do registro de "*log*" (lista de páginas atualizadas);
- gravar conteúdo do "*log*" em memória secundária.

- *rollback_transaction micro*

Implica eliminar as alterações feitas pela transação até o momento do rollback. A execução desta operação independe do "*checkpoint*" já que sempre são eliminadas todas as ações da transação.

- deve ser executada a eliminação do registro, do arquivo de "*log*", pertencente à transação eliminada e gravar as páginas iniciais na base de dados voltando, desta forma, ao estado inicial.

3.2.2 - Transações meta-micro

Com o intuito de atender às características de ambientes de projeto, o SGBD GERPAC/UniCOSMOS incorpora o conceito de transação meta-micro. Esta complementa o conceito clássico de transação (i.e., modelo de Gray) podendo ser vista como uma transação longa que contém a) nenhuma, b) diversas transações micro aninhadas. Tal conceito exibe alguma similaridade com o de

transações aninhadas definidas em /MOSS 81/.

O sistema de gerência de transações do GERPAC/UniCOSMOS define comandos de controle de transação associados às transações meta-micro para diferenciá-los dos comandos de transação micro. Assim, foram definidos os comandos de controle *begin_transaction meta-micro*, *commit_transaction meta-micro*, *rollback_transaction meta-micro*, similares àqueles das transações micro, e os comandos de controle de transação *hang_transaction meta-micro* e *continue_transaction meta-micro* próprios do ambiente de projeto.

A semântica dos comandos *commit_transaction meta-micro* e *rollback_transaction meta-micro* necessita novas definições operacionais. Contrariamente, o comando de controle de transação *begin_transaction meta-micro* se mantém igual do ponto de vista operativo, sendo que a diferença reside nos dados com os quais se trabalha.

- *commit_transaction meta-micro*

Indica a finalização normal de uma transação meta-micro. Resulta na imediata efetivação de todas as atualizações feitas na base de dados, caso não existam transações micro em andamento neste instante. Caso contrário, a finalização é postergada até que todas as transações micro componentes daquela transação meta-micro finalizem.

- *rollback_transaction meta-micro*

Indica o cancelamento de uma transação meta-micro. Implica no cancelamento de todas as transações micro correntes que a constituem [/BANC 85/, /KETA 87/ e /KORT 88/].

- *hang_transaction meta-micro*

A consequência da invocação do comando *hang_transaction meta-micro* é a de provocar que o atendimento da transação meta-micro por parte do gerenciador de transações seja interrompido até que uma indicação explícita, no sentido de continuar a transação, seja feita pelo usuário que ativou a mesma. Existem três maneiras diferentes de continuar a execução de uma transação, a saber:

1) executando operações pertencentes à transação (comando

continue_transaction meta-micro);

2) executando o procedimento *commit_transaction meta-micro*;

3) executando o procedimento *rollback_transaction meta-micro* (veja Figura 3.2, seção 3.2.2.4).

No primeiro caso o que acontece é que o sistema continua executando a transação que se encontrava interrompida, a partir do ponto em que se ativou o comando *hang_transaction meta-micro*. Neste caso o estado da transação volta a ser de execução. No segundo, o que acontece é que o sistema passa a executar o comando *commit_transaction* associado, efetivando todas as alterações realizadas sobre a base de dados. Isto implica na atualização no arquivo de "*log*" e na base de dados estável das atualizações realizadas pela transação corrente. Finalmente, no terceiro caso, o que acontece é que o sistema passa a executar o comando de controle de transação *rollback_transaction* o que implica em voltar ao estado consistente da base de dados existente antes do começo de execução da transação que está sendo desfeita.

3.2.2.1 - Algumas considerações de processamento

As operações executadas no contexto de uma transação meta-micro que efetuam algum tipo de alteração de página da base de dados provocam a realização do mesmo tipo de operações que são executadas no ambiente micro (i.e., a atualização das listas de páginas iniciais e finais no registro de "*log*"). A terminação de uma transação meta-micro implica na efetivação destas alterações na base de dados (commit).

O tratamento de uma transação micro, englobada em uma transação meta-micro, mantém-se inalterado no que diz respeito aos procedimentos definidos em seções anteriores com a única exceção das operações ativadas pela finalização com sucesso da transação micro (i.e., *commit_transaction micro*) já que, neste ponto, além dos procedimentos definidos se deve executar o que denominaremos aqui de "*transferência de páginas atualizadas*".

O procedimento que entendemos como "*transferência de páginas atualizadas*" consiste em gravar na lista de páginas iniciais do registro de "*log*" da transação meta-micro o conteúdo das páginas iniciais que se encontram na lista

de páginas iniciais da transação micro, considerando a incorporação das páginas que neste momento não se encontram na lista do registro da transação meta-micro.

3.2.2.2 - Alguns comentários sobre a recuperação de uma falha

Os tipos de falhas que devem ser considerados são os mesmos que foram apresentados quando da abordagem do aspecto micro. Sendo que, toda vez que se apresenta uma falha dentro do sistema deve-se voltar ao último estado inicial consistente do banco de dados, a saber:

* falha ocorrida entre um comando *begin_transaction meta-micro* e um *begin_transaction micro*
  - não existe *hang_transaction meta-micro*:
    voltar ao estado anterior ao *begin_transaction meta-micro* (equivalente a *rollback_transaction meta-micro*).
  - existe *hang_transaction meta-micro*:
    voltar ao estado do *hang_transaction meta-micro*.

* falha ocorrida entre um comando *begin_transaction micro* e um *commit_transaction micro*
  - voltar ao estado anterior ao *begin_transaction micro*.

* falha ocorrida depois de um comando *commit_transaction micro*
  - não existe *hang_transaction meta-micro*:
    voltar ao estado do *commit_transaction micro*.
  - existe *hang_transaction meta-micro*:
    voltar ao estado no momento do *hang_transaction meta-micro*.

3.2.2.3 - Comentários sobre ações de recuperação e os comandos associados

Os comandos de controle de transação para os dois tipos de transações

considerados diferem no que diz respeito às operações que efetuam. Por outro lado, as operações realizadas por um comando de controle de transação podem variar dependendo do contexto em que este se encontra. Um exemplo desta situação é o comando de controle *commit_transaction micro*. Assim sendo, apresenta-se na continuação uma breve descrição de cada um dos comandos de controle que podem se apresentar durante execução no contexto meta-micro.

- *begin_transaction micro*

No contexto de uma transação meta-micro implica na efetivação tanto do "*log*" quanto da base de dados. Isto possibilita poder voltar ao ponto inicial da transação micro na presença de uma falha.

- *commit_transaction micro*

No contexto de uma transação meta-micro implica:

- atualizar a lista de páginas iniciais meta-micro (LPImMi) com as páginas iniciais da lista de páginas iniciais micro (LPIMi) que não figuram na LPImMi;
- atualizar a lista de páginas atualizadas meta-micro (LPAmMi) com as páginas atualizadas da lista de páginas atualizadas micro LPAMi;
- "*checkpoint*" micro e meta-micro.

- *begin_transaction meta-micro*

Inicia o controle de uma transação meta-micro e gera um registro no arquivo de "*log*" (do tipo apresentado na figura 3.3).

- *commit_transaction meta-micro*

Indica o fim com sucesso de uma transação meta-micro. Efetiva todas as atualizações realizadas por uma transação meta-micro (i.e., grava as páginas atualizadas dentro da transação meta-micro na base de dados estável).

- *rollback_transaction meta-micro*

Elimina todas as atualizações realizadas por uma transação meta-micro

(i.e., restabelece as páginas atualizadas durante o processamento da transação meta-micro ao estado inicial das mesmas). A ação de rollback deve saber em que estado se encontravam as transações micro que formam a transação meta-micro atual (i.e., executando ou não) para determinar o momento de execução da ação de rollback. Em outras palavras a execução do rollback deverá ser postergada, caso existam transações micro executando, ou efetivada, caso contrário.

- *hang_transaction meta-micro*

A função deste comando é a de indicar um "*ponto de espera*" na execução da transação meta-micro (i.e., deixá-la em um estado pendente) depois do qual pode ser tomada alguma das seguintes decisões:

- determinar a finalização com sucesso da transação meta-micro (*commit_transaction meta-micro*);
- determinar a finalização com erro (*rollback_transaction meta-micro*);
- determinar a continuidade da execução da transação meta-micro (*continue_transaction meta-micro*).

- *continue_transaction meta-micro*

A função deste comando é a de indicar a continuação na execução de uma transação meta-micro que se encontrava em um estado pendente devido à ativação anterior de um comando de controle *hang_transaction meta-micro*.

3.2.2.4 - Diagrama de estados de uma transação meta-micro

No diagrama de estados da figura 3.2 pode-se observar a dinâmica de uma transação meta-micro. O comando *begin_transaction meta-micro* produz a ativação de uma transação provocando a passagem da mesma ao estado de execução das ações que a constituem. O comando *hang_transaction meta-micro* provoca a passagem da transação meta-micro do estado de execução ao estado de latência (i.e., espera) depois do qual podem ser recebidos três tipos de estímulos, a saber: o comando *commit_transaction meta-micro*, o comando *rollback_transaction*

meta-micro ou o comando *continue_transaction meta-micro*. O comando *commit_transaction meta-micro* altera o estado da transação para o estado commit, o comando *rollback_transaction meta-micro* produz a passagem para o estado rollback e o comando *continue_transaction meta-micro* faz com que a execução volte ao ponto em que se encontrava o processo no momento da invocação do comando *hang_transaction meta-micro*.

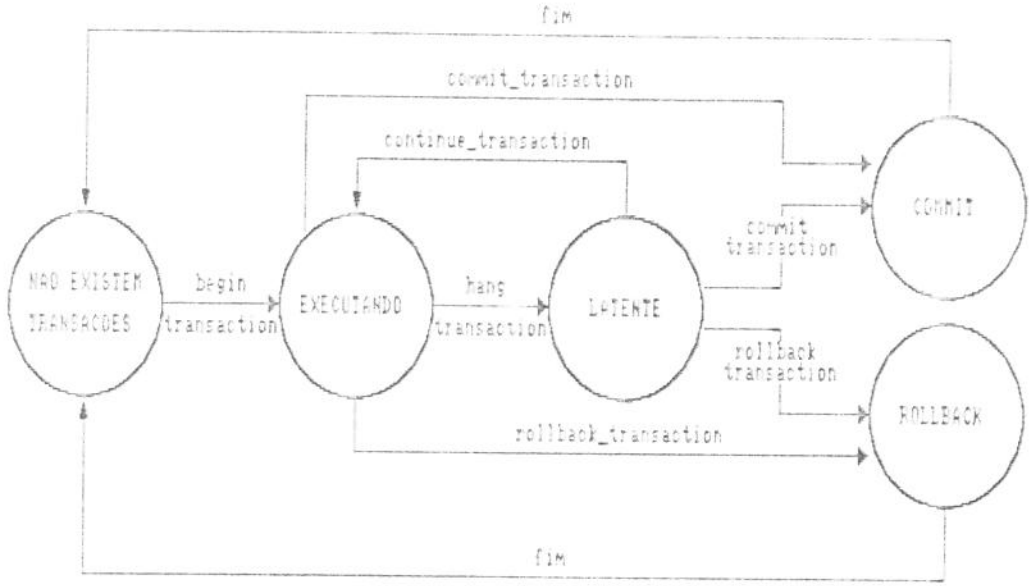


FIGURA 3.2. Diagrama de estados de uma transacao meta-micro

3.2.3 - O arquivo de log (histórico)

Cada registro do arquivo de "*log*" é composto por um identificador, um campo "*status*" (indica em que estado a transação associada se encontrava no momento da falha), uma lista (dinâmica) de páginas iniciais e uma lista (também dinâmica) de páginas atualizadas (veja figura 3.3).

ID - identificador do registro.

O identificador do registro é formado por campos que permitem identificar a base de dados que estava sendo atualizada (i.e., versão), qual o programa que ativou a transação, qual foi o número de transação atribuído pelo sistema[2] e o momento da ativação (considerando o ano, mês, dia e hora - AA/MM/DD/HHHH).

ST - status da transação.

O campo status é utilizado para, como dito anteriormente, saber em que estado se encontrava a transação no momento da falha. Isto possibilita reconhecer, dependendo do tipo de falha, quais transações devem ser desfeitas/refeitas para restabelecer o último estado consistente da base de dados.

FIGURA 3.3. Estrutura do registro do arquivo de "log"

[2] O identificador da transação é único, i.e., não existem duas transações que possuam o mesmo identificador

Valores admitidos:

" " - transação em andamento e

"c" - commited.

LPI - lista de páginas iniciais.

A lista de páginas iniciais é dinâmica, ou seja, o espaço é alocado à medida em que se incorporam elementos. O conteúdo de cada elemento da lista é uma cópia da página da base de dados como se encontrava no momento de ser chamada (pela primeira vez durante a execução da transação) para processamento (leitura/escrita).

A quantidade de elementos da lista pode variar de 1 a N [3] onde N é a quantidade de páginas que foram trazidas da memória principal para o pool (i.e., uma área de trabalho em memória principal na qual são armazenadas um número fixo de páginas distintas associadas a um determinado arquivo da base de dados UniCOSMOS) durante o processamento da transação.

LPA - lista de páginas atualizadas.

A lista de páginas atualizadas possui em cada elemento o conteúdo de uma página após ter sofrido algum tipo de alteração. Da mesma forma que a lista de páginas iniciais esta lista também é dinâmica. Os elementos das listas devem conter informação adicional que permita saber o "*endereço físico*" da página no arquivo associado.

A quantidade de elementos da lista varia de 0 a M [4] onde M é a quantidade de páginas que foram atualizadas durante o processamento da transação.

Dito de outra maneira, o conteúdo das listas de páginas iniciais e finais é o estado inicial e final, respectivamente, das páginas que foram manipuladas

---

3 A quantidade de elementos vai de 1 a N pois assume-se que uma transação vai, no mínimo, ler uma página durante seu processamento

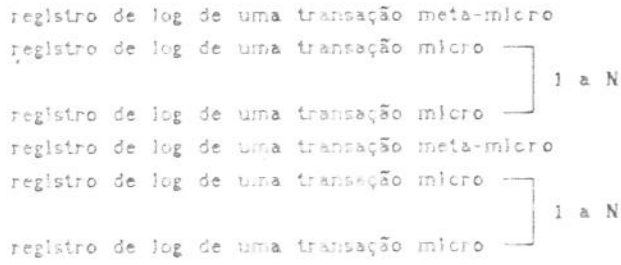
4 A quantidade de elementos pode ser zero pois é possível que uma transação não atualize nenhuma página durante seu processamento

durante a execução da transação associada.

Como uma observação final é bom dizer que no presente esquema de trabalho é assumido o uso do protocolo "*write ahead log*" (i.e., toda atualização realizada sobre a base de dados deve ser previamente gravada no arquivo de "*log*") [/ESWA 76/, /DATE 83/ e /BERN 87/], assegurando desta maneira a recuperação do estado da base de dados.

3.2.3.1 - Considerações sobre o processamento do arquivo de log

a. inicialmente não existem páginas nas tabelas (elas são carregadas à medida que são necessárias);
b. o mecanismo de "*log*" faz uso de uma estrutura de dados em memória principal (mostrada anteriormente), mais uma estrutura similar em memória secundária cuja finalidade é manter informação "*persistente*" (ou seja, estados iniciais e finais das páginas) para posterior recuperação, do próprio arquivo de "*log*", de uma falha;
c. toda vez que uma página é levada da memória secundária para a memória principal, o conteúdo da página é gravado no registro de "*log*" (caso ainda não exista) da transação que a chamou;
d. toda vez que uma página sofre uma alteração (i.e., atualização ou remoção) a mesma é incorporada à lista de páginas atualizadas (LPA), refletindo-se desta forma a alteração;
e. a estrutura em memória secundária do arquivo LOG, em princípio, é igual à estrutura em memória principal sendo que, as páginas informadas são exatamente as mesmas que se encontram na memória principal, ou seja, cria-se um "*back-up*";
f. o tipo de registro de "*log*" de uma transação meta-micro é similar ao tipo de registro de uma transação micro. A ordem na qual os registros aparecem dentro do arquivo de "*log*" pode ser vista da seguinte maneira:



observações: deve existir uma identificação no registro associado a uma transação micro que indique se esta transação pertence a uma transação meta-micro anterior ou é independente, i.e., está associada a uma ação fora do contexto da transação meta-micro anterior.

3.2.3.2 - *Exemplo de execução*

Vamos imaginar que durante a execução do sistema se apresenta a seguinte sequência. Em um determinado momento começa a ser executada uma transação de tipo meta-micro. O estado das listas de páginas iniciais e finais associadas à transação meta-micro (i.e., I1 e A1) é o que se mostra na figura 3.4a onde pode-se ver que, inicialmente, as mesmas se encontram vazias. Suponhamos que durante a execução da transação são chamadas para manipulação as páginas P1 e P2 sendo que a página P1 é atualizada gerando desta maneira uma nova versão da mesma denominada P11. O estado das listas de páginas iniciais e atualizadas agora é o que se apresenta na figura 3.4b.

Imaginemos agora que na sequência do processamento da transação meta-micro é ativada uma transação micro (i.e., é encontrado um *begin_transaction micro*). Como consequência disto o que acontece é que, por um lado, são criadas duas listas, uma de páginas iniciais (I2) e outra de páginas atualizadas (A2), as quais estão associadas à transação micro que acabou de começar e por outro, as listas associadas à transação meta-micro (I1 e A1) são gravadas na memória secundária realizando desta maneira, um *checkpoint*. Suponhamos que durante a execução da transação micro a mesma chama as páginas P1 e P3. A página P1 é agora a nova instância criada pela

transação meta-micro (i.e., página P11). O estado das listas associadas à transação micro é o que se mostra na figura 3.4c. Imaginemos que a transação micro atualizou as duas páginas chamadas criando desta forma as páginas P12 e P31 (veja figura 3.4d).

Vamos nos deter um momento aqui e analisar o que é que acontece caso se apresente uma falha neste instante. Dado que o comando de controle de transação *begin_transaction micro* provocou um *"checkpoint"* das listas associadas à transação meta-micro, podemos voltar ao estado anterior ao começo da transação micro e recomeçar a execução a partir do comando de controle *begin_transaction micro*. Continuemos mais um pouco e imaginemos que a transação micro chega ao fim (i.e., executa um *commit_transaction micro*); o que acontece aqui é que se executa um novo *"checkpoint"* e é realizado o processo que denominamos de transferência de páginas mencionado na seção 3.2.2.1 provocando, por um lado, a gravação das páginas manipuladas pela transação micro na memória secundária (devido ao *"checkpoint"*) e por outro, a atualização das listas associadas à transação meta-micro (devido ao processo de transferência) sendo que o estado das listas é o que se mostra na figura 3.4e. O porquê da necessidade desta transferência de páginas se deve à possibilidade da eliminação das ações realizadas pela transação meta-micro (i.e., comando *rollback_transaction meta-micro*) voltando, desta maneira, ao estado da base de dados anterior ao começo da execução da transação meta-micro (ou seja, páginas P1, P2 e P3) e ao fato que pode ser que a falha se apresente depois da execução do comando de controle *commit_transaction micro* sendo que, neste caso, é preciso voltar ao estado em que se encontrava a base de dados no momento da finalização da transação micro.

3.2.4 - Comparação entre o modelo GERPAC/UniCOSMOS e o modelo de Gray para transação

Como mencionado anteriormente, o modelo aqui apresentado não possui diferenças fundamentais com o de Gray quando consideramos os comandos de controle de transação micro. A semântica dos comandos é basicamente a mesma. Uma das diferenças entre os modelos está associada à operação de

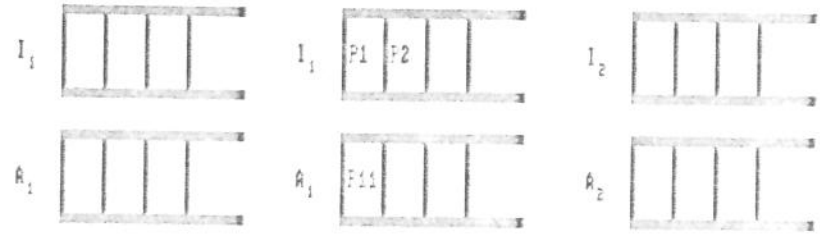


Figura 3.4a Figura 3.4b Figura 3.4c

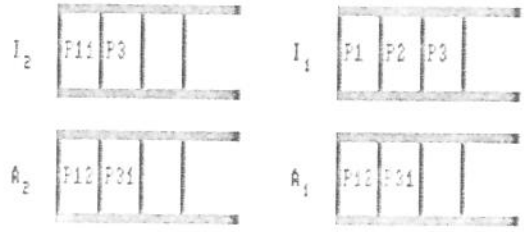


Figura 3.4d Figura 3.4e

FIGURA 3.4. Exemplo de execucao

"*checkpoint*", já que em nossa abordagem, a mesma é definida implicitamente associada à execução do comando *commit_transaction*. A utilização dos comandos desta maneira faz com que no presente modelo os tipos de transações que podem ser reconhecidos, em relação às ações de recuperação na presença de uma falha, seja reduzido. Segundo nossa perspectiva, transações do tipo T1, T2 e T4 no modelo de Gray [/GRAY 81a/] (figura 3.5) são consideradas do mesmo tipo (digamos "*concluídas*") sendo que o mesmo acontece com as transações de tipo T3 e T5 da figura[5] (consideradas como "*não concluídas*").

Um outro aspecto diferenciador entre os dois modelos e que influenciou a decisão de definir a operação de "*checkpoint*" implicitamente no comando *commit_transaction*, é o tipo de informação que é gravada no arquivo "*log*" para uma possível recuperação de um estado consistente da base de dados na presença de uma falha.

5 Da análise da figura não se deve interpretar que existe necessariamente execução paralela entre as transações.

Enquanto nos modelos tradicionais a informação armazenada no arquivo de "*log*" está associada com as operações executadas na base de dados (i.e., leitura e escrita) no nosso modelo consideram-se os estados iniciais e finais das páginas, as quais são o elemento de manipulação do sistema [/MAGA 89/ e /RICA 87]. O motivo pelo qual esta situação tem condicionado o esquema de "*checkpoint*" tem a ver com a questão da persistência dos dados gerados, facilitando o processo de recuperação já que só transações ativas (i.e., não concluídas) devem ser consideradas, dado que não existe a possibilidade de uma transação concluída não ter sido gravada na base de dados estável.

A outra diferença que existe entre o modelo de Gray para transações e o modelo aqui apresentado reside na incorporação do conceito de transação meta-micro. Tal conceito está baseado no conceito de transação aninhada

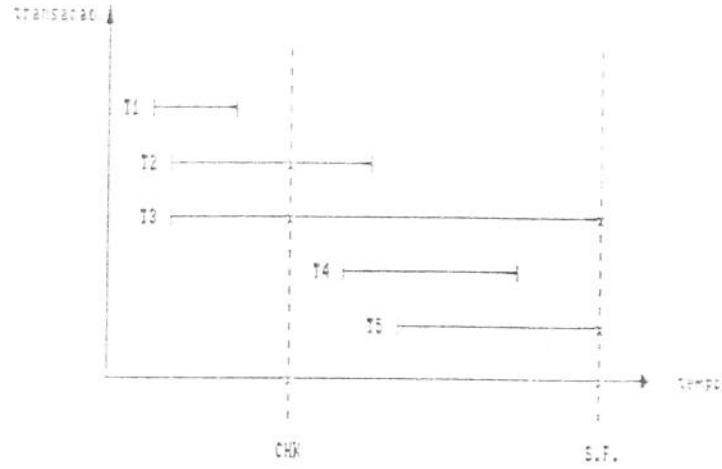


Figura 3.5a. Tipos de transacções no modelo de Gray

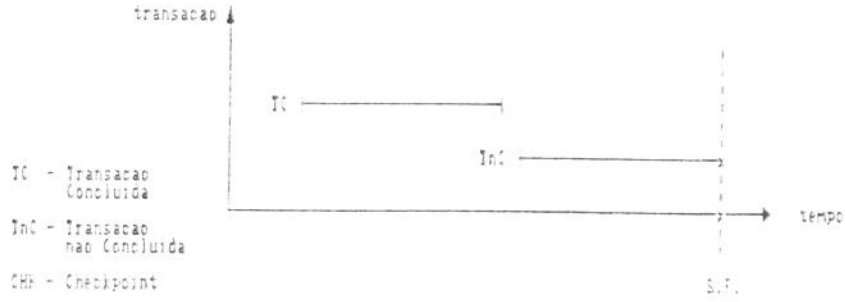


Figura 3.5b. Tipos de transacções no modelo [illegible]

FIGURA 3.5. Comparacao entre os modelos

definido em /MOSS 81/ já que se entendeu que era interessante utilizar o modelo de Moss para conseguir modelar certas características que se apresentam em ambientes de projeto como por exemplo:

- subdivisão de tarefas devido ao tamanho do projeto considerado;
- subdivisão de tarefas devido ao envolvimento de equipes de projetistas trabalhando em paralelo nas distintas partes do projeto considerado;
- características próprias dos sistemas atualmente disponíveis (i.e., sistemas de janelas).

3.3 - Características do gerenciador de transações e integração com o GERPAC/UniCOSMOS

O gerenciador de transações, implementado em linguagem C, é composto de duas partes, a saber, o *roteador* e o *gerenciador* propriamente dito. O *roteador* é a parte do sistema através da qual se estabelece a comunicação entre os usuários (i.e., GERPAC, programas de aplicação, etc.) e o gerenciador, sendo sua função reconhecer os pedidos de execução dos usuários, e habilitar ou não (dependendo do estado do sistema), a execução dos mesmos invocando as rotinas associadas aos pedidos. O *gerenciador* efetivamente executa as funções associadas a cada um dos comandos de controle da transação.

O esquema de comunicação pode ser visto na figura 3.6 e na figura 3.8 onde o mesmo é apresentado a um nível de detalhamento maior. Uma observação interessante é que o tipo de mensagens que é empregado para estabelecer comunicação entre processos é *assíncrono* sendo que, a tarefa de prover sincronismo é deixada ao processo interessado (i.e., o processo deve prever algum mecanismo de espera).

O esquema de execução dos processos que se encontram ativos em um dado momento no sistema estabelece uma modalidade de execução paralela. O *roteador* deve considerar esta situação e verificar a consistência dos pedidos de execução recebidos (i.e., solicitação da execução de um *begin_transaction* depois de outro *begin_transaction*, etc.) motivo pelo qual existe uma tabela onde é mantido o estado de execução de cada um dos processos ativos.

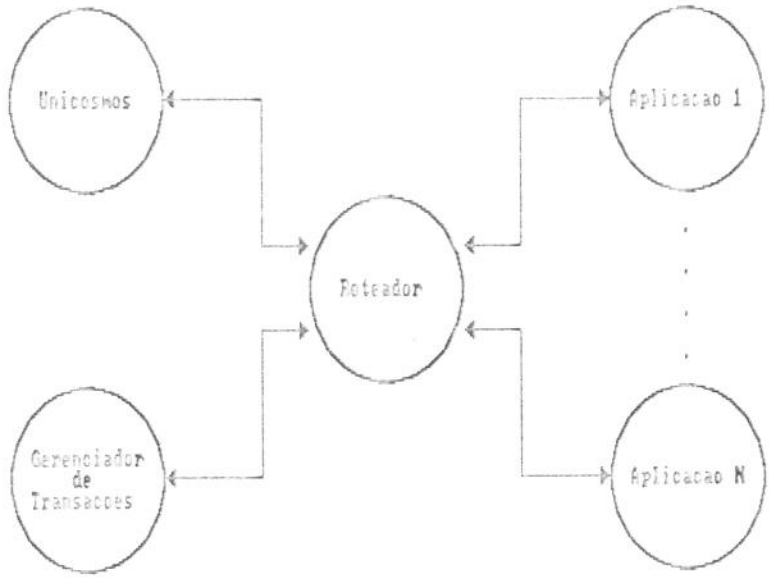


FIGURA 3.6. Esquema de comunicacao do sistema

A ativação do gerenciador de transações do sistema é realizada pelas rotinas de inicialização do sistema de base de dados, sendo que, uma vez feito isto o estado do sistema é "*latente*", ou seja, fica esperando um pedido de execução de algum comando de controle por parte de alguma aplicação.

Os estados possíveis do gerenciador de transações são básicamente dois:

inativo e

ativo.

No primeiro caso fica estabelecido que nenhuma operação é atendida pelo gerenciador até que ele seja "*acordado*" pela operação correspondente do GERPAC. No segundo caso, o gerenciador atende todas as operações ativadas pelas aplicações ou pelo nível primário do UniCOSMOS, que estão associadas ao próprio gerenciador. As operações associadas ao gerenciador são:

begin_transaction,

commit_transaction,

rollback_transaction e

página.

As operações *begin_transaction*, *commit_transaction* e *rollback_transaction* são disparadas pelas aplicações e tem como função indicar o começo e fim (normal ou não), respectivamente, do conjunto de ações que se quer controlar pelo gerenciador de transações. A operação página é ativada pelo nível primário do UniCOSMOS, particularmente pelo algoritmo de paginação do sistema, a partir do tratamento das páginas envolvidas na execução das atualizações que estão sendo realizadas sobre a base de dados.

Dentro do estado de ativação do gerenciador de transações podem ser definidos "*sub-estados*" associados aos processos indicando em que situação se encontra um processo em um momento dado. Os sub-estados definidos são os seguintes:

bgt - begin_transaction,

cmt - commit_transaction,

rbt - rollback_transaction,

ent - end_transaction e

prt - processing_transaction.

A dinâmica do sistema pode ser vista no diagrama de estados apresentado na figura 3.7 o qual não distingue entre as operações executadas pelo processo encarregado do roteamento das operações e pelo gerenciador de transações propriamente dito.

Como dito anteriormente, o gerenciador de transações pode se encontrar em um dos seguintes estados: inativo ou ativo. Inicialmente, o estado do gerenciador de transações é inativo. Neste estado o gerenciador ignora qualquer comando de controle que recebe (i.e., o comando de controle não é atendido). A ativação do gerenciador de transações do sistema é feita no momento da inicialização do sistema GERPAC/UniCOSMOS mediante a execução do comando correspondente. Uma vez ativado, o sistema de gerência de transações fica em um estado de espera (i.e., latente) e só sai deste estado mediante: 1) a chegada de uma indicação de desativação do sistema ou 2) a chegada de um comando de controle *begin_transaction*.

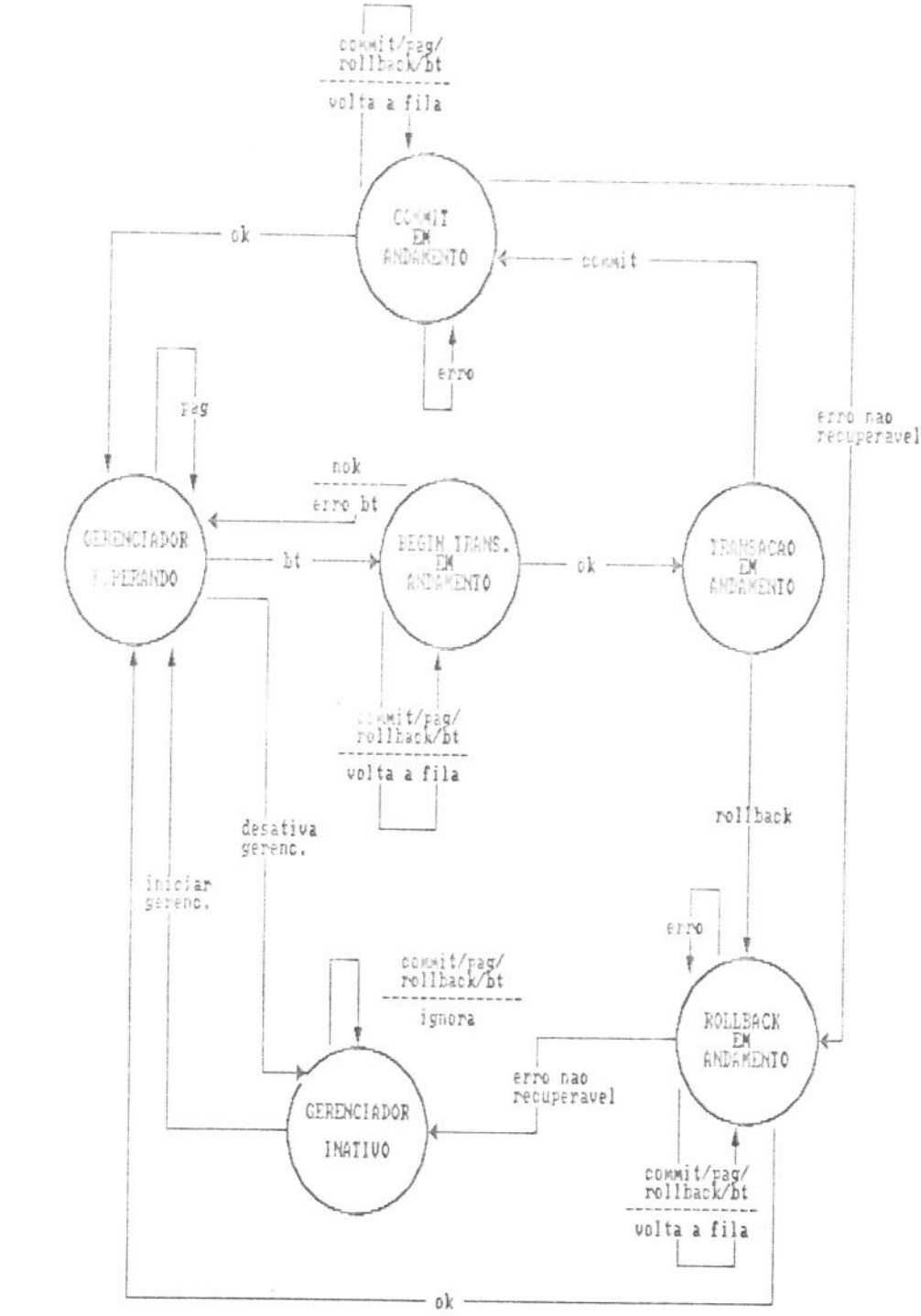
No primeiro caso, o sistema de gerência de transações volta ao estado inativo inicial. No segundo caso, o sistema passa a executar as operações associadas à execução do comando de controle de transação *begin_transaction*. Estando o sistema no estado de begin_transaction é assumido então que se estão desenvolvendo as operações próprias do comando de controle *begin_transaction* como por exemplo gerar um identificador para o registro de "*log*". Neste estado o sistema ignora qualquer requerimento de execução que chegue e só sai dele através da finalização, com sucesso ou não, do comando de controle.

A execução de um comando *begin_transaction* pode finalizar de forma errônea devido à presença de uma falha; neste caso o sistema avisa tal situação ao processo que solicitou a execução do *begin_transaction*, elimina todas as ações que foram realizadas até o momento da falha e o estado volta a ser de espera. A outra possibilidade é que a execução do comando finalize com sucesso; se isto acontecer o sistema muda do estado *begin_transaction* para um estado no qual se encontra efetivamente executando a transação (i.e., se leva um controle da execução da transação através da registração de todas as atualizações realizadas sobre a base de dados mediante a gravação no arquivo de "*log*" dos estados iniciais e finais das páginas que estão sendo utilizadas). Este estado, chamado aqui de "*transação em andamento*", implica no tratamento de páginas até que aconteça uma das seguintes possibilidades:

- é recebido o comando de controle de transação *commit_transaction*,
- é recebido o comando de controle de transação *rollback_transaction*.

No primeiro caso, o estado do sistema passa de "*transação em andamento*" para o estado de "*commit em andamento*". No segundo caso, o estado do sistema passa para o estado de "*rollback em andamento*".

Quando o estado do sistema é o de execução do comando de controle de transação *commit_transaction*, as operações que estão sendo realizadas estão associadas à efetivação das atualizações feitas sobre a base de dados, i.e., gravação das páginas que se encontram na lista de páginas atualizadas do registro de "*log*" associado, na base de dados estável e gravação do registro de "*log*" na memória secundária. Qualquer outro pedido de execução que chegue



pag - pagina
bt - begin_transaction

FIGURA 3.7. Diagrama de estados do sistema

enquanto o sistema se encontrar processando estas operações é adiado até a finalização do processamento atual. Existem duas maneiras possíveis de finalização: com sucesso e sem sucesso.

No primeiro caso, pode-se assegurar que o estado atual da base de dados é consistente e o estado do sistema volta a ser de espera. No segundo caso, o que se faz é reexecutar o comando de controle para, efetivamente, realizar as operações associadas. Esta reexecução se repete até que uma das seguintes possibilidades acontece:

- a execução do comando de controle finaliza com sucesso;
- o comando de controle continua sendo executado e o tempo previsto para a execução dele é ultrapassado.

No primeiro caso a situação que se apresenta é tal que novamente se pode assegurar um estado consistente da base de dados e o estado do sistema passa a ser de espera. No segundo caso, estamos na presença da seguinte situação: as ações que fazem parte da transação podem ter sido executadas mas as alterações feitas não foram efetivadas na base de dados ou foram parcialmente efetivadas. Isto significa que, embora todas as ações da transação possam ter sido executadas, não é possível assegurar que o estado da base de dados seja consistente, sendo portanto, necessário desfazer tudo o que foi feito pela transação mediante a execução do comando de controle *rollback_transaction*.

Se o estado do sistema for o de "*rollback em andamento*" as operações que estão sendo realizadas estão associadas à operação de desfazer todas as ações que foram realizadas pela transação sobre a base de dados, voltando ao estado consistente da base de dados antes do começo da execução da transação i.e., gravação das páginas, que se encontram na lista de páginas iniciais do registro de "*log*" associado, na base de dados estável e eliminação do registro de "*log*" correspondente. Novamente, as considerações que foram apresentadas para o comando de controle de transação *commit_transaction* são válidas aqui com algumas alterações, isto é, qualquer pedido de execução que chegar enquanto o sistema se encontra processando o comando de controle *rollback_transaction* é adiado até a finalização do processamento atual.

Novamente, existem duas maneiras possíveis de finalização: com sucesso e sem sucesso. No primeiro caso, pode-se assegurar que o estado atual da base

de dados é o último estado consistente que existia antes do começo da execução da transação e o estado do sistema volta a ser de espera. No segundo caso, o que se faz é reexecutar o comando de controle para, efetivamente, realizar as operações associadas. Esta reexecução se repete até que uma das seguintes possibilidades acontece:

- a execução do comando de controle finaliza com sucesso;
- o comando de controle continua sendo executado e o tempo previsto para a execução dele é ultrapassado.

No primeiro caso a situação que se apresenta é tal que novamente pode-se assegurar um estado consistente da base de dados e o estado do sistema passa a ser de espera. No segundo caso o que acontece é que, por algum motivo, não foi possível recuperar o último estado consistente da base de dados provocando-se, desta maneira, uma situação de erro fatal que não pode ser corrigida pelo sistema. Como consequência disto, o sistema de geréncia de transações é desativado, o usuário é notificado desta situação ficando por conta dele, com o uso dos "*back-ups*" de longo termo, a recuperação de um estado consistente da base de dados.

3.3.1 - Esquema de comunicação entre processos

O esquema de comunicação entre processos no sistema de transações do GERPAC é o que se mostra na figura 3.8 onde os desenhos indicados com o nome de "fifo u", "fifo a", "fifo r" e "fifo g" correspondem às filas de comunicação associadas a cada tipo de processo. Sendo que "u" é igual a UniCOSMOS, "a" é igual a aplicação, "r" é igual a roteador e "g" é igual a gerenciador de transações.

Podemos identificar dois tipos básicos de comunicações dentro do sistema. Por um lado, temos comunicação do processo roteador com os demais processos envolvidos no sistema e, por outro, a comunicação inversa (i.e., comunicação dos processos com o roteador). A comunicação entre processos é feita utilizando-se uma fila associada a cada processo e o envio de sinais entre os processos para indicar a existência de informação na fila associada

[/ROCH 85/].

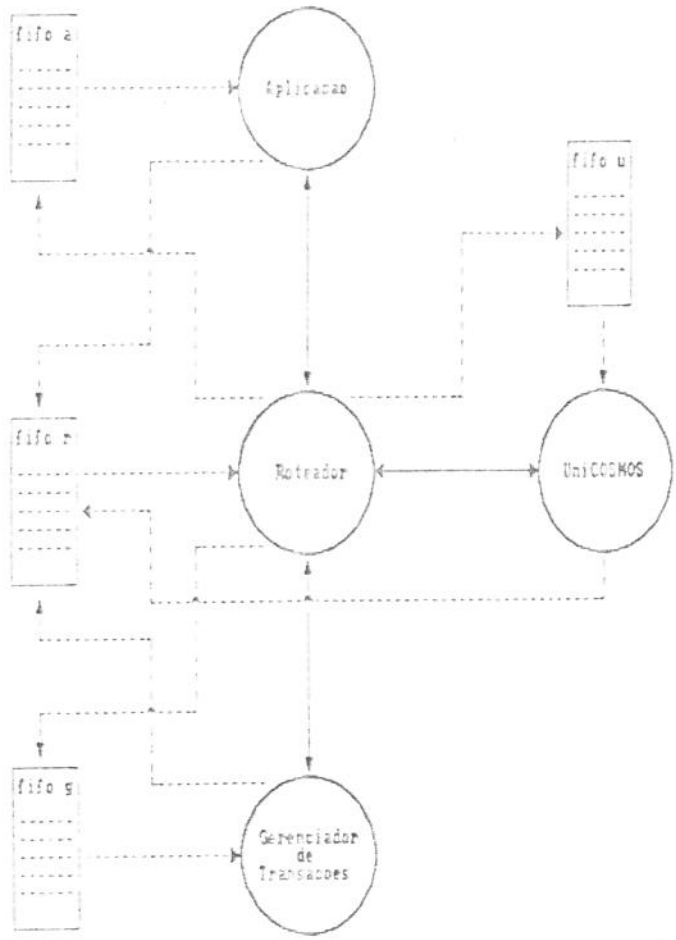


FIGURA 3.8. Esquema de comunicacao detalhado

- Comunicação r/g, r/a ou r/u
  - enviar um KILL(17,x)[6] para indicar que alguma informação deve ser lida da fifo (onde x corresponde ao identificador do processo dentro do sistema (PID));
  - o roteador deve, se preciso, repassar a informação lida da fifo para a

6 O KILL é uma chamada do sistema que é usada para enviar um sinal de um processo para outro.

fifo do "*receiver*" correspondente.

- Comunicação a/r, u/r ou g/r
  - cada processo que quer se comunicar com um outro deve fazê-lo através do roteador;
  - o "*sender*" deve enviar um KILL(16,r) para avisar a intenção de se comunicar com o roteador;
  - existe uma fifo associada a cada um dos processos (utilizada para enviar informação entre processos);
  - toda vez que o roteador recebe um KILL lê o conteúdo da fifo, analisa-o e executa as ações correspondentes;
  - um processo pode "*ignorar*" um KILL. Entende-se por ignorar o consumo do elemento da fifo.

Observações: como não existe forma de saber quem enviou o sinal, uma informação a ser gravada na fifo do roteador é o número de processo de forma a poder mandar o retorno.

3.3.2 - *Funções do roteador*

- estabelecer comunicação entre os diversos processos do sistema

- verificar a validade dos pedidos

  Permite reconhecer estruturas válidas ou inválidas de uma transação dentro de um processo para eliminar a possibilidade de aninhamento de transações micro, o que não é permitido no nosso modelo.

  Exemplo:

  - estrutura válida

    bgt, cmt/rbt

    BGT, bgt, cmt/rbt, CMT/RBT

  - estrutura inválida

    bgt, bgt

    cmt, cmt/rbt

Observações: bgt - begin_transaction micro
cmt - commit_transaction micro
rbt - rollback_transaction micro
BGT - begin_transaction meta-micro
CMT - commit_transaction meta-micro
RBT - rollback_transaction meta-micro

O reconhecimento é possível através de:

1) a manutenção de uma tabela onde se mantém o status de cada processo, i.e., saber em que estado se encontra um dado processo, se o processo se encontra executando ou não, que comando de controle o processo está executando, etc., e
2) uma lógica, embutida no roteador, onde se estabelece quais seqüências de aparição dos comandos de controle são válidas ou não.

- controlar o tempo de execução das rotinas que assim o precisarem

Os comandos de controle de transação commit e rollback devem poder ser recuperados caso se apresente uma falha durante sua execução, isto significa que se pode apresentar uma situação de execução repetitiva no caso em que a falha permaneça constante durante cada tentativa de recuperação.

Para solucionar este problema foi definida uma rotina de "*time out*" (controle de tempo expirado) cuja função é controlar o tempo que uma função executa. Esta rotina tem como parâmetros de entrada a rotina que se quer controlar e o tempo máximo de execução permitido para a rotina.

- reconhecer pedidos de execução

Esta tarefa é feita para poder saber, a partir dos pedidos de execução que chegam e o estado de execução dos processos dentro do sistema, quais as rotinas do gerenciador de transações que devem ser invocadas para execução.

O reconhecimento dos pedidos de execução por parte do roteador é realizado através do uso de duas ferramentas do sistema operacional UNIX

System V que são o LEX (Lexical Analyzer Generator) - que permite reconhecer elementos, dentro de uma frase, que pertencem a uma determinada linguagem, e o YACC (Yet Another Compiler Compiler) - que realiza a análise sintática [/CHRI 83/ e /PCS 84/].

3.4 - Conclusões

O presente trabalho enquadra-se no contexto da implementação do Sistema GERPAC/UniCOSMOS, definindo o sistema de tratamento de transações que se acoplará ao GERPAC/UniCOSMOS.

Foram apresentadas as características envolvidas no processo de gerenciamento de transações para sistemas de gerenciamento de banco de dados orientados a projeto e, em particular, para o GERPAC/UniCOSMOS.

A explicitação dos conceitos de transação micro (envolve um objeto de projeto), transação meta-micro (envolve reflexos que uma modificação em um objeto de projeto produz em objetos associados) e transação macro diferencia o modelo aqui adotado daquele proposto por Gray em /GRAY 78/ e /GRAY 80/ fazendo com que a nossa abordagem seja mais simples do ponto de vista conceitual.

Outros autores como /NEUM 82/, /BANC 85/, /KETA 87/, /HAER 87/ e /KORT 88/ consideram igualmente transações longas, porém, a nosso ver, separar o controle semântico (transação macro) do sintático (transação meta-micro) é mais apropriado, permitindo associar mecanismos próprios a cada um deles deixando o controle explícito dos mesmos ao usuário.

Adicionalmente, o fato de transações meta-micro poderem aninhar transações micro é apropriado em certos casos, mas não deve ser obrigatório. A coexistência de ambas as soluções parece-nos adequada.

A semântica dos comandos de controle de transação tanto para transações de tipo micro quanto meta-micro exigiram o estabelecimento de novas definições operacionais para os comandos de controle de transações de tipo meta-micro, como também a incorporação dos comandos *hang_transaction meta-micro* e *continue_tranaction meta-micro*, apresentados anteriormente.

Finalmente, para ilustrar a implementação em execução, questões relativas ao arquivo de "log", ao gerenciador de transações micro e os aspectos

associados à integração dos mesmos ao GERPAC/UniCOSMOS foram abordadas.

3.5 - Referências

/BANC 85/ BANCILHON F., KIM W. e KORTH H.F.
"A Model of CAD Transactions"
Proceedings of International Conference on Very Large Data Bases - 1985

/BERN 87/ BERNSTEIN P. A., HADZILACOS V. e GOODMAN N.
"Concurrency control and recovery in database systems"
Addison-Wesley Co. - 1987

/CHRI 83/ CHRISTIAN K.
"The UNIX Operating System"
John Willey & Sons - 1983

/DATE 83/ DATE C.J.
"An introduction to database systems - Volume II"
Addison-Wesley Co. - 1983

/DELG 86/ DELGADO A.L.N., MAGALHÃES L.P. e RICARTE I.L.
"Sistemas de Gerenciamento de Banco de Dados para P.A.C."
Anais do I Simpósio Brasileiro de Banco de Dados (SBBD) - Abr.1986

/DELG 87/ DELGADO A.L.N.
"Bancos de dados no contexto de projeto auxiliado por computador"
Tese de mestrado DCA-FEE-UNICAMP - Jan.1987

/DELG 88/ DELGADO A.L.N., MAGALHÃES L.P. e RICARTE I.L.
"Supporting design environments through the Entity-Relationship Model"
12th IMACS World Conference on Scientific Computation - Paris, France - Jul.1988

/ESWA 76/ ESWARAN K.P., GRAY J.N., LORIE R.A. e TRAIGER I.L.
"The notions of consistency and predicate locks in a database system"
Communications of the ACM - Nov.1976

/FIPE 88/ *"Sistemas de Banco de Dados no Contexto de Projeto Auxiliado por Computador"*
Relatório Final - FIPEC, processo 1.1731-0
LEA/DCA/FEE/UNICAMP - Janeiro 1988

/GRAY 78/ GRAY J.N.
"Notes on database operating systems"
Lecture notes in computer sciences, Vol.60
Springer-Verlag - 1978

/GRAY 80/ GRAY J.N.
"A transaction model"
Lecture notes in computer sciences, Vol.85
Springer-Verlag - 1980

/GRAY 81/ GRAY J.N.
"The Transaction Concept: Virtues and Limitations"
Proceedings of International Conference on Very Large Data Bases - 1981

/GRAY 81a/ GRAY J.N., McJONES P., BLASGEN M., LINDSAY B., LORIE R., PRICE T., PUTZOLU F. e TRAIGER I.
"The Recovery Manager of the System R Database Manager"
ACM Computing Surveys - Jun.1981

/HAER 83/ HAERDER T. e REUTER A.
"Principles of transaction-oriented database recovery"
ACM Computing Surveys - Dez.1983

/HAER 87/ HAERDER T. e ROTHERMEL K.
"Concepts for Transaction Recovery in Nested Transactions"
ACM SIGMOD - Maio 1987

/KATZ 83/ KATZ R.H
"Managing the Chip Design Database"
ACM SIGMOD/IEEE - 83

/KETA 87/ KETABCHI M.A. e BERZINS V.
"Modelling and Managing CAD Databases"
IEEE Software - Fev.1987

/KORT 88/ KORTH H.F., KIM W. e BANCILHON F.
"On Long-Duration CAD Transactions"
Information Sciences - An International Journal, Vol.46, Nros.1/2; North-Holland - Out./Nov.1988

/LORI 83/ LORIE R.A. e PLOUFFE W.
"Complex Objects and Their Use in Design Transactions"
ACM SIGMOD/IEEE - 1983

/MAGA 89/ MAGALHÃES L.P., DELGADO A.L.N., RICARTE I.L., RUSCHEL R.C. e OLGUIN C.J.M.
"Implementação de um Banco de Dados Não Convencional: A experiência GERPAC/UniCOSMOS"
Anais do IV Simpósio Brasileiro de Banco de Dados (SBBD) - Abr.1989 e
Anais das 18 Jornadas Argentinas de Informática e Investigación Operativa (18 JAIIO), Buenos Aires, Argentina - Ago.1989

/McLE 83/ McLEOD D., NARAYANASWAMY K. e BAPA RAO K.V.
"An Approach to Information Management for CAD/VLSI Applications"
ACM SIGMOD/IEEE - 1983

/MOSS 81/ MOSS J.E.B.
"Nested Transactions: An Approach to Reliable Computing"
M.I.T., Laboratory of Computer Science - 1981

/NEUM 82/ NEUMANN T. e HORNUNG C.
"Consistency and Transactions in CAD Databases"
Proceedings of International Conference on Very Large Data Bases - 1982

/OLGU 89/ OLGUIN C.J.M. e MAGALHÃES L.P.
"Alguns Aspectos Associados ao Gerenciamento de Transações no contexto do GERPAC/UniCOSMOS"
I Seminário de Atividades de Pós-Graduação do DCA DCA/FEE/UNICAMP; Nov.1989

/OLGU 90/ OLGUIN C.J.M. e MAGALHÃES L.P.
"Um Gerenciador de Transações no Contexto do GERPAC/UniCOSMOS"
Anais do V Simpósio Brasileiro de Banco de Dados (SBBD) - Abr.1990

/PCS 84/ PERIPHERE COMPUTER SYSTEME GmbH
"MUNIX V.2/03 - Tutorials - Volume IIa"
PCS GmbH - 1984

/RICA 87/ RICARTE I.L. e DELGADO A.L.N.
"GERPAC - Um SGBD para P.A.C."
Anais do II Simpósio Brasileiro de Banco de Dados (SBBD) - Mai.1987

/RICA 87a/ RICARTE I.L.
"Sistemas de gerência de dados para projeto auxiliado por computador"
Tese de mestrado DCA-FEE-UNICAMP - Jun.1987

/ROCH 85/ ROCHKIND M.J.
"Advanced UNIX Programming"
Prentice-Hall Inc. - 1985

/STON 83/ STONEBRAKER M., RUBENSTEIN B. e GUTTMAN A.
"Application of Abstract Data Types and Abstract Indices to CAD Databases"
ACM SIGMOD/IEEE - 1983

"E AGORA, José?
A festa acabou,
a luz apagou,
o povo sumiu,
a noite esfriou,
e agora, José?
e agora, você?..."

"José"
Carlos Drummond de Andrade

CAPÍTULO 4

Capítulo 4 - Conclusões

Conteúdo

4.1 - Introdução

Em ambientes PAC são manipulados objetos de projeto os quais são normalmente construídos a partir de outros objetos que por sua vez, podem ser compostos de objetos menores; qualquer modificação de uma parte pode levar à alteração dos objetos de projeto que possuem aquele objeto como uma componente. Assim, pode-se ver que neste ambiente é importante considerar uma atualização não só como uma operação envolvendo um objeto de projeto (*nível micro de projeto*) mas também de que maneira isto se reflete nos objetos de projeto associados (*nível macro de projeto*). Esta peculiaridade das aplicações de projeto exige uma abordagem de transação diferente da tradicional. Como visto no capítulo 3, aqui foram definidos três tipos de transação - micro (Mi), meta-micro (mMi) e macro (Ma) - correspondendo à recuperação de falhas (dois primeiros) e ao controle de consistência (o terceiro).

Sendo o tratamento da recuperação de falhas o aspecto central deste trabalho, consideramos em maior detalhe os conceitos de transação micro e meta-micro, abordando-se em menor profundidade as transações macro, que tratam do controle de consistência.

Dois são os aspectos que devem ser considerados quando se fala em transações em um ambiente de projeto:

- recuperação, e
- integridade.

Neste trabalho questões associadas à integridade dos dados não foram consideradas; porém, foi definido um tipo de transação que considera aspectos semânticos chamada de *transação macro*. Este tipo de transação considera a validade dos dados que foram gerados durante o processamento, ou seja, para que se possa dizer que uma transação finalizou com sucesso ("*commit*") devem ser respeitadas todas as restrições definidas para os itens de dados (páginas) que foram atualizados. Assim, quando uma transação finaliza devem ser verificadas as relações entre os dados que foram armazenados podendo ser necessario desfazer a transação que terminou aparentemente com sucesso devido à violação de alguma restrição definida.

No que se refere a questões de recuperação foram definidos dois tipos de transação, os quais consideram os aspectos de recuperação associados a um objeto isolado e questões relativas à recuperação dos dados quando os objetos que estão sendo considerados são definidos como formando parte de uma hierarquia à qual aquele objeto pertence. Estes dois tipos de transações receberam o nome de *transação micro* e *transação meta-micro* respectivamente.

4.2 - Transações micro

Este tipo de transação não tem nenhuma diferença com o modelo de transação considerado por, entre outros, /GRAY 78/ e /DATE 83/. O tratamento é praticamente o mesmo que o convencional. Dito de outra maneira, do ponto de vista conceitual, as transações micro são análogas às transações convencionais sendo que a diferença entre elas reside unicamente no aspecto operacional do comando "*checkpoint*" já que a execução do mesmo está associada à execução do comando de controle de transação *commit_transaction micro*; ou seja, toda vez que uma transação finaliza com sucesso as alterações por ela realizadas são efetivadas na base de dados estável , assegurando desta maneira a durabilidade

das mudanças. Foram definidos dentro deste contexto os comandos de transação *begin_transaction micro*, *commit_transaction micro* e *rollback_transaction micro*. Estes comandos mantêm as características que se apresentam em ambientes ditos convencionais (bancários, comerciais, etc.).

4.3 - Transações meta-micro

O porquê da existência deste tipo de transação se deve ao fato de que o conceito de transação, como definido por Gray e outros pesquisadores, não é suficiente para atender as características de operações que envolvam mais de um objeto.

Desta forma, o GERPAC/UniCOSMOS incorporou o conceito de transação meta-micro com o intuito de considerar as peculiaridades de um ambiente de projeto, complementando o modelo de transação convencional. Assim, uma transação meta-micro pode ser vista como uma transação de longa duração que contém várias ou nenhuma transação micro.

Do dito até aqui pode-se ver que existe uma certa similaridade entre este tipo de transação e transações aninhadas (nested transactions) definidas em /MOSS 81/ já que, de fato, uma transação meta-micro pode conter transações micro (veja figura 3.1).

Esta característica é muito interessante dado que permite considerar aspectos de projeto tais como:

- subdivisão de tarefas decorrente do tamanho do projeto considerado,
- subdivisão de tarefas devido ao envolvimento de equipes de projetistas,
- aspectos associados às características das estações de trabalho atuais (i.e., sistemas de janelas).

Por outro lado, os comandos de controle de transação *hang_transaction meta-micro* e *continue_transaction meta-micro* foram definidos neste contexto para permitir ao usuário a interrupção e continuação, respectivamente, da transação em qualquer ponto da mesma. Através da utilização destes comandos de controle de transação é possível tratar algumas das características que se apresentam durante a atividade de projeto.

4.4 - Transações macro

O termo transação, aqui utilizado, buscou explicitar a analogia de uma falha semântica, com uma falha de sistema (conceito de transação).

Assim, transações macro estão associadas à questão de consistência, ou seja, elas verificam se restrições de consistência a nível de objetos de projeto são observadas.

Uma transação macro não permite, contudo, que dados globalmente inválidos sejam perdidos pois isto é inadmissível em um processo de projeto, onde os dados só alcançam consistência no final do processo.

Transações macro serão suportadas no GERPAC/UniCOSMOS por um sistema de tratamento de restrições, que analogamente ao sistema de tratamento de transações tratado no presente trabalho, será implementado de forma independente, comunicando-se com o GERPAC/UniCOSMOS através de mensagens. Atualmente, uma tese de mestrado do DCA/FEE/UNICAMP trata este tema e /BANC 85/ e /NEUM 82/ referenciam também esta temática.

4.5 - Comentários finais

Como devidamente exposto nas conclusões do capítulo 3, o sistema de transações proposto, considerando o contexto do GERPAC/UniCOSMOS, definiu os conceitos de transação micro (Mi), meta-micro (mMi) e macro (Ma), utilizando uma abordagem mais simples que Gray [/GRAY 78/, /GRAY 80/ e /GRAY 81/] do ponto de vista conceitual e possibilitando uma modelagem mais adequada dos ambientes de projeto.

Outros autores [/NEUM 82/, /BANC 85/, /KETA 87/, /HAER 87/ e /KORT 88/] têm estudado questões relativas a transações longas; entretanto, a nosso entender, a separação do controle semântico do sintático parece-nos melhor já que permite associar mecanismos próprios a cada um deles deixando o controle explícito dos mesmos ao usuário.

Um trabalho que merece atenção é o feito por W. Kim e outros [/KIM 84/]

onde o modelo de transação por eles apresentado combina três conceitos chave: a noção de transações aninhadas, a noção de bases de dados semi-públicas associada com transações aninhadas e a visão de que uma transação em engenharia é uma sequência de transações convencionais de curta duração.

Uma das diferenças entre a abordagem do Kim e a abordagem aqui apresentada é que em /KIM 84/ a visão que se tem de uma transação longa é a de uma sequência de transações curtas e na nossa abordagem, isto não é necessariamente assim, já que consideramos aqui o fato de que transações meta-micro possam aninhar transações micro é apropriado em certos casos, mas não deve ser obrigatório. A coexistência de ambas as soluções parece-nos apropriada.

Para finalizar, situamos os principais resultados deste trabalho:

* do lado teórico:
  - apresentação de um modelo para o tratamento integrado de todas as classes de transações existentes em sistemas PAC. Em particular, o modelo proposto inclue, porém não exige, a utilização do aninhamento de transações para cobrir algumas das necessidades que se apresentam em ambientes de projeto. É ainda proposta a incorporação dos comandos *hang_transaction meta-micro* e *continue_transaction meta-micro* os quais permitem modelar algumas características que se apresentam em ambientes de projeto.

Em resumo, o modelo proposto fornece mecanismos que possibilitam a separação dos controles semântico e sintático, o que nos parece melhor, já que permite que o usuário defina explicitamente os controles que ele desejar.

* do lado prático:
  - implementação de um protótipo, no contexto do GERPAC/UniCOSMOS, que permitiu obter subsídios para os conceitos apresentados.

  A implementação pode ser caracterizada como a seguir:

  a. o modelo oferece um módulo de funções independente que permite estabelecer comunicação com as outras componentes do sistema GERPAC/UniCOSMOS e implementa os comandos que permite definir uma transação. Isto é possível através da utilização de recursos do sistema UNIX para comunicação entre processos;

b. tira-se proveito do esquema hierárquico do sistema para, por exemplo, possibilitar a implementação de uma rotina de controle de tempo de execução (*timeout*) a qual é necessária para detetar possíveis anomalias durante a execução do sistema;

c. a utilização do *roteador* como elemento centralizador do processamento permite a fácil incorporação de novas capacidades, i.e., tratamento de novos comandos de controle.

Consideramos que o uso do sistema de transações no contexto do GERPAC/UniCOSMOS contribuirá no teste de aplicabilidade do modelo e do sistema de transações em sistemas reais.

A nível especulativo sugerimos os seguintes tópicos para reflexão e próximos trabalhos:

- implementar transações de tipo macro, definidas aqui para considerar aqueles aspectos associados com o controle de integridade dos dados gerados; - desenvolver um sistema de recuperação que consiga, a partir da informação gravada no arquivo de "*log*" pelo sistema de transações, restabelecer o último estado consistente da base de dados anterior à presença da falha;
- testar o comportamento do sistema em um ambiente distribuído (i.e., multi-processamento);
- estudar a possibilidade de incorporar mecanismos que permitam fazer um controle de concorrência. Consideramos que deva ser testada a factibilidade de associar às transações esquemas de bloqueio dos itens de dados por ela utilizados.

4.6 - Referências

/BANC 85/ BANCILHON F., KIM W. e KORTH H.F.
"A Model of CAD Transactions"
Proceedings of International Conference on Very Large Data Bases - 1985

/DATE 83/ DATE C.J.
"An Introduction to Database Systems - II"
Addison-Wesley - 1983

/GRAY 78/ GRAY J.N.
"Notes on database operating systems"
Lecture notes in computer sciences, Vol.60
Springer-Verlag - 1978

/GRAY 80/ GRAY J.N.
"A transaction model"
Lecture notes in computer sciences, Vol.85
Springer-Verlag - 1980

/GRAY 81/ GRAY J.N.
"The Transaction Concept: Virtues and Limitations"
Proceedings of International Conference on Very Large Data Bases - 1981

/HAER 83/ HAERDER T. e REUTER A.
"Principles of transaction-oriented database recovery"
ACM Computing Surveys - Dez.1983

/HAER 87/ HAERDER T. e ROTHERMEL K.
"Concepts for Transaction Recovery in Nested Transactions"
ACM SIGMOD - Maio 1987

/KETA 87/ KETABCHI M.A. e BERZINS V.
"Modelling and Managing CAD Databases"
IEEE Software - Fev.1987

/KIM 84/ KIM W., LORIE R., McNABB D. e PLOUFFE W.
"A Transaction Mechanism for Engineering Design Databases"
Proceedings of International Conference on Very Large Data Bases - 1984

/KORT 88/ KORTH H.F., KIM W. e BANCILHON F.
"On Long-Duration CAD Transactions"
Information Sciences - An International Journal, Vol.46, Nros 1/2, North-Holland - Out./Nov.1988

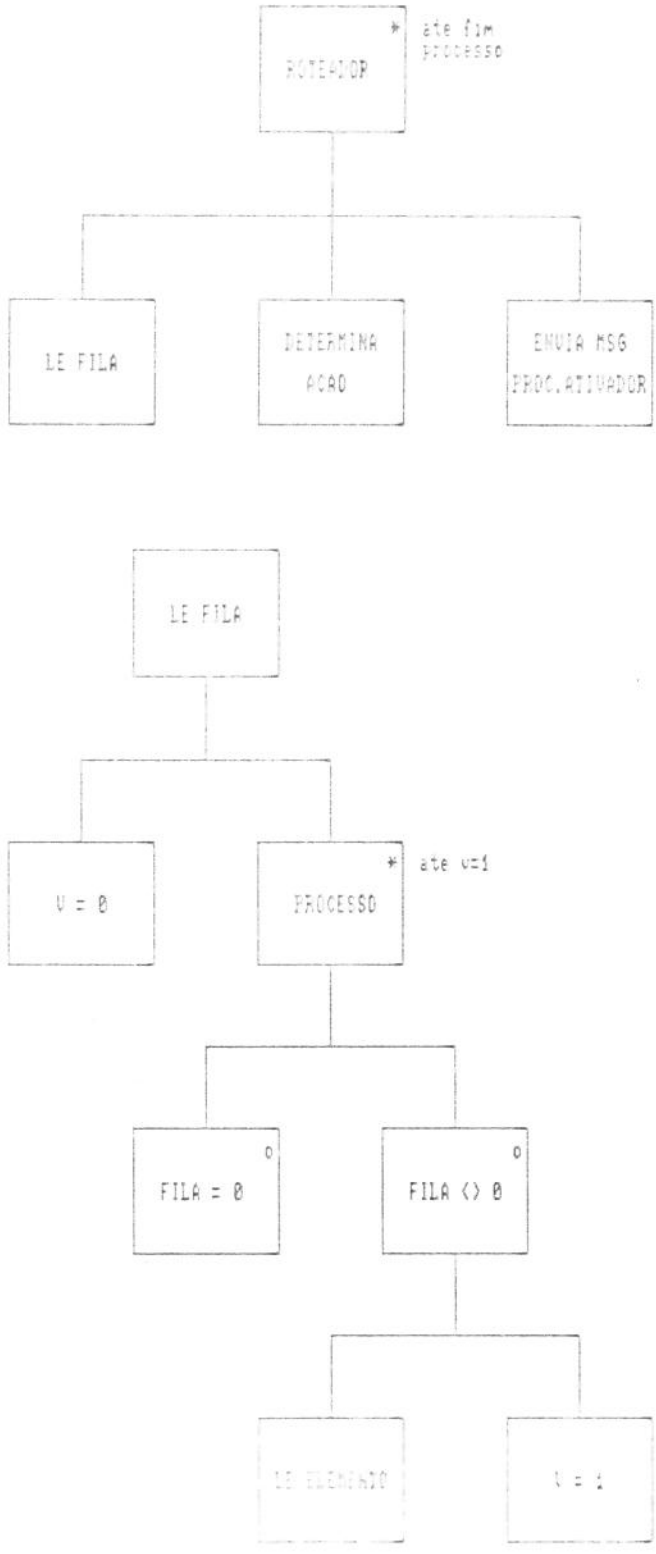
/MOSS 81/ MOSS J.E.B.
"Nested Transactions: An Approach to Reliable Computing"
M.I.T., Laboratory of Computer Science - 1981

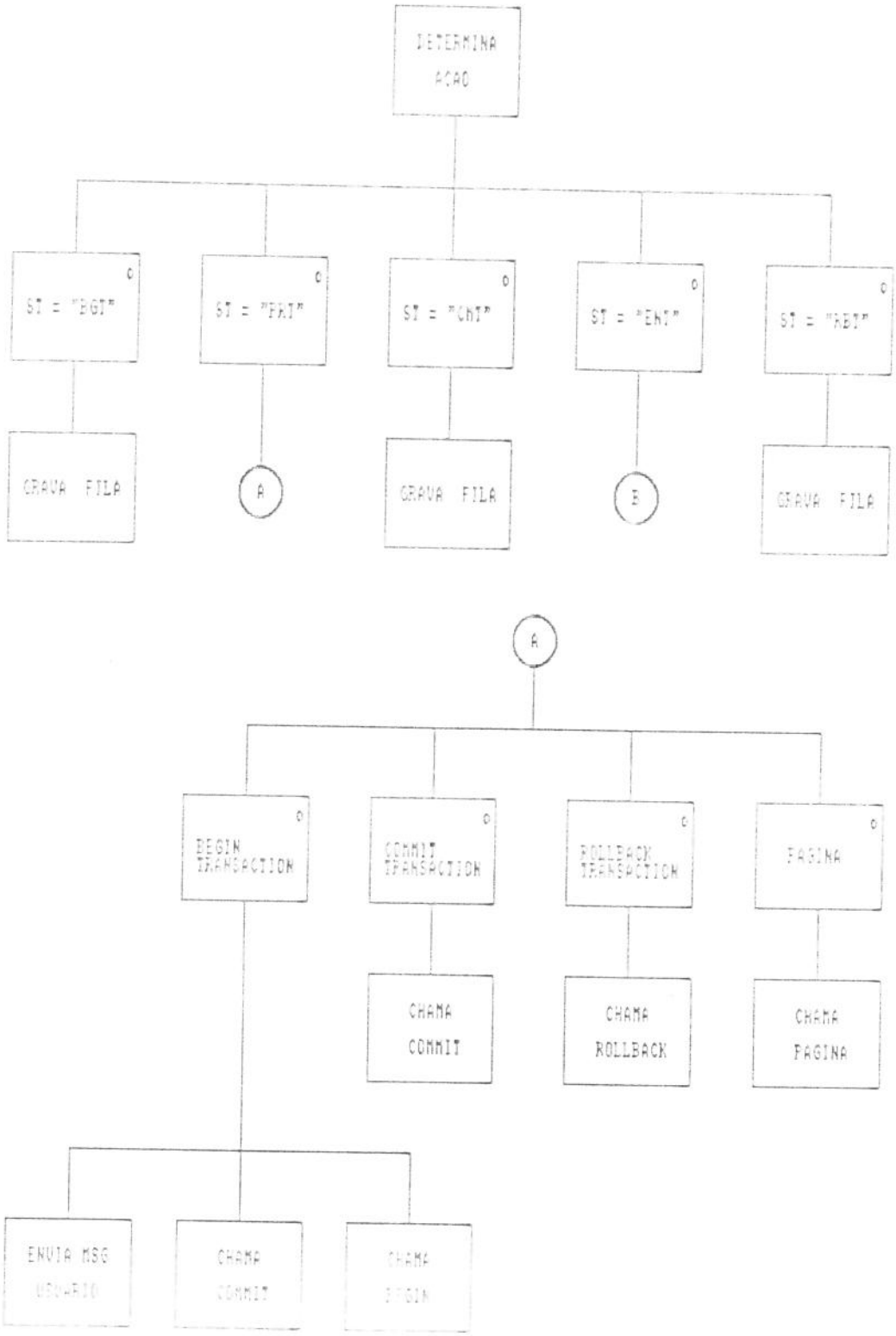
/NEUM 82/ NEUMANN T. e HORNUNG C.
"Consistency and Transactions in CAD Databases"
Proceedings of International Conference on Very Large Data Bases - 1982

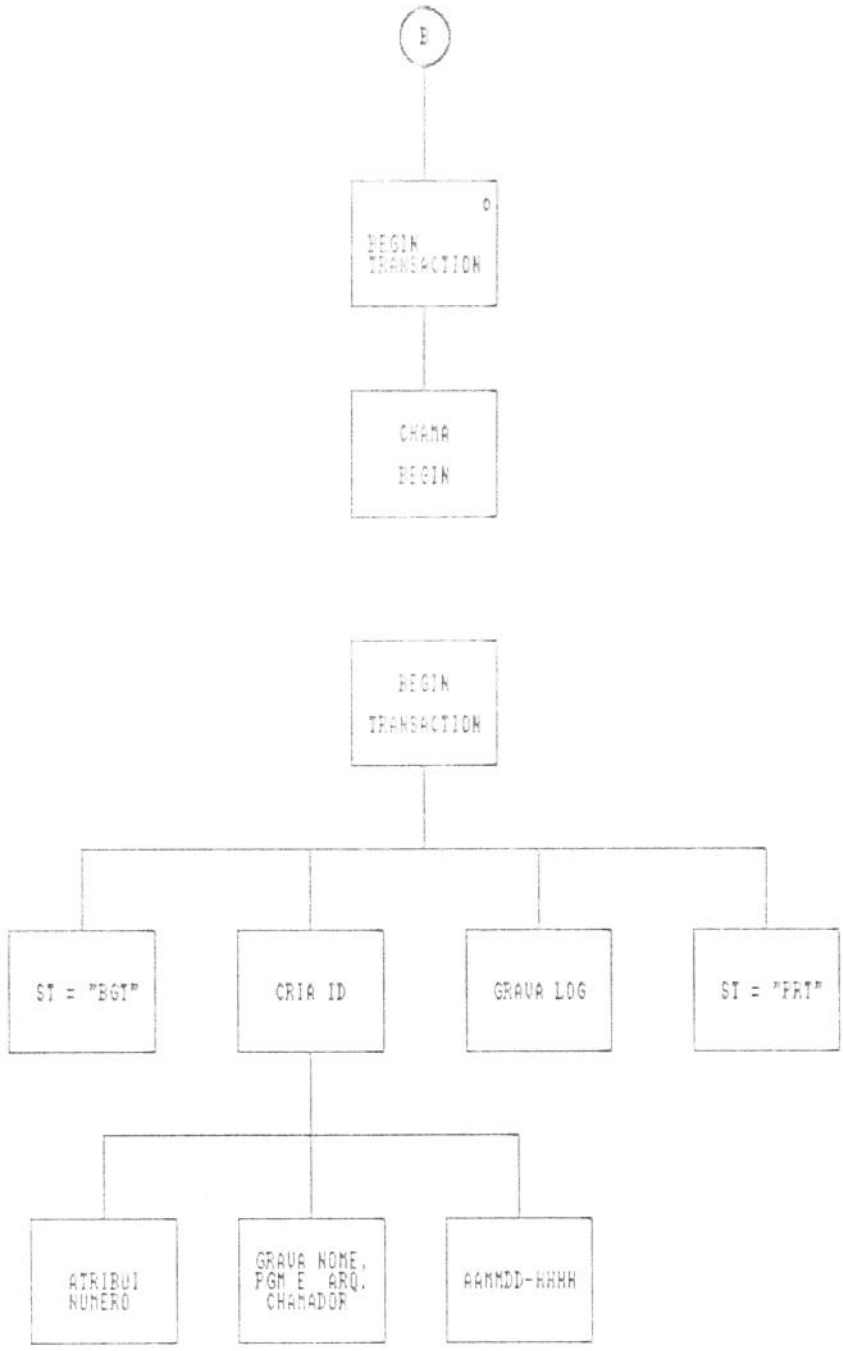
ANEXOS

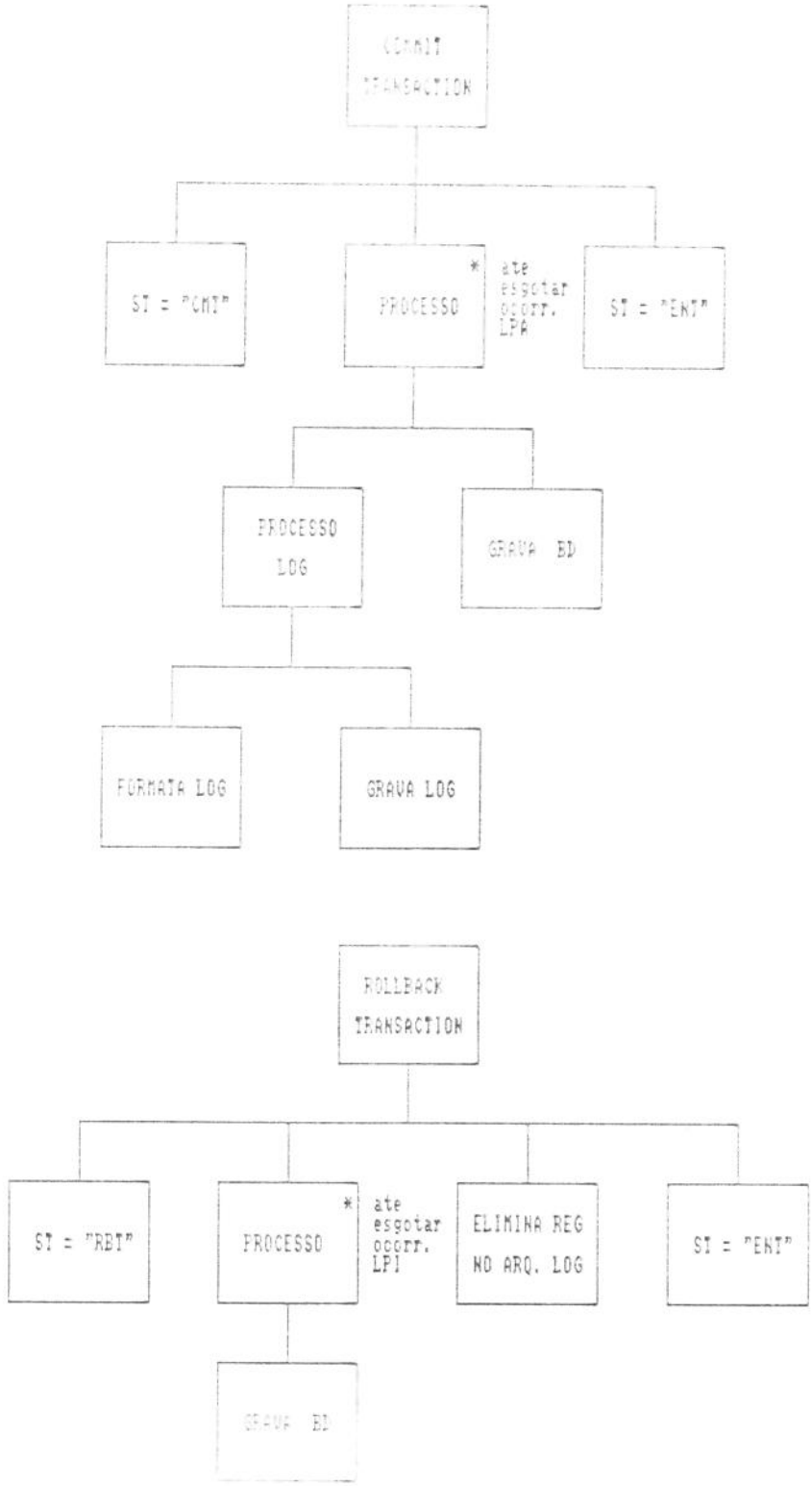
ANEXO A

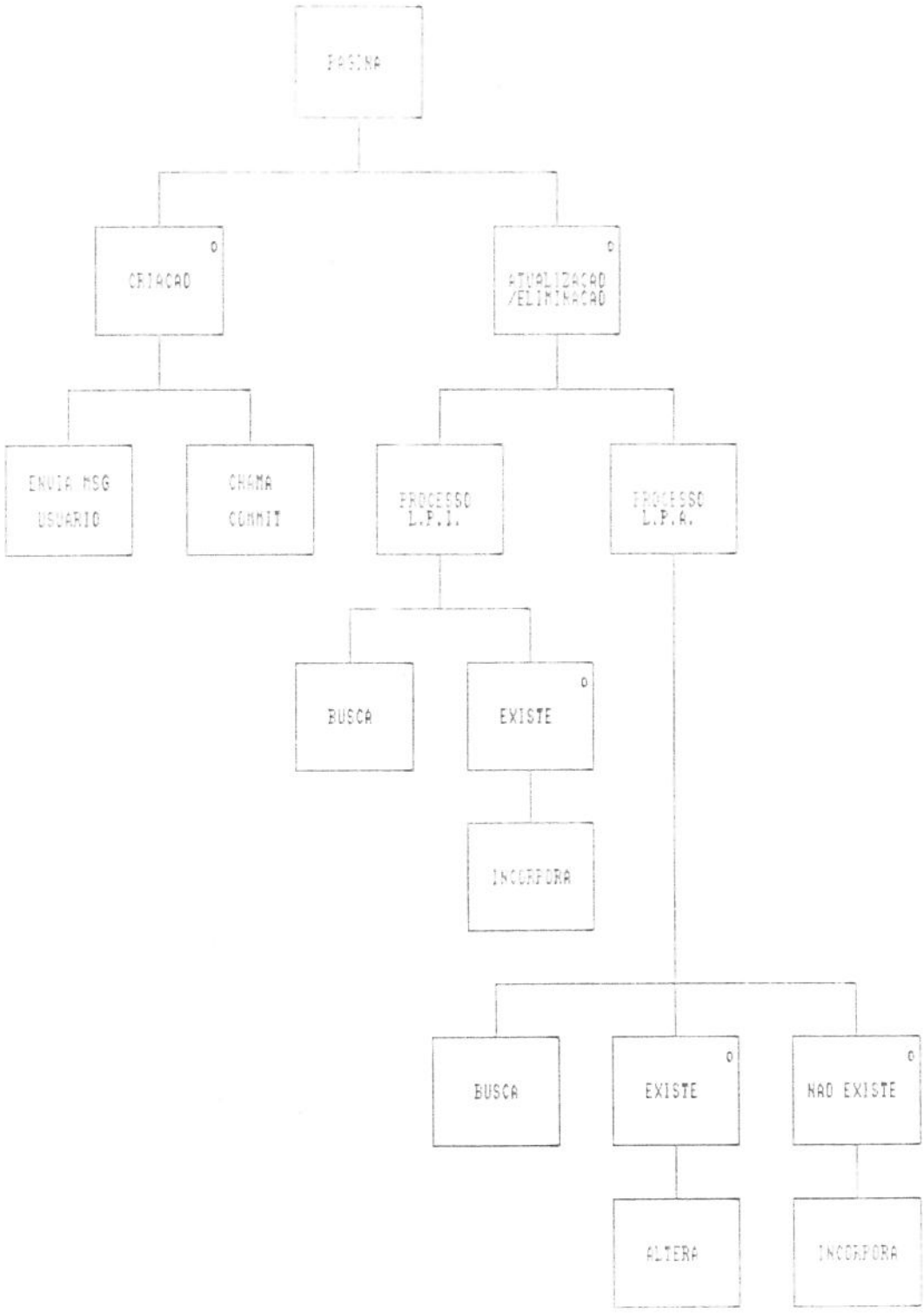
Sistema de Transações: Diagramas de Jackson


ROTEADOR
*
até fim processo
LE FILA
DETERMINA ACAO
ENVIA MSG PROC.ATIVADOR
LE FILA
V = 0
PROCESSO
*
até v=1
FILA = 0
o
FILA <> 0
o


DETERMINA
ACAO
ST = "BGT"
ST = "FAT"
ST = "CMT"
ST = "ENT"
ST = "RBT"
GRAVA FILA
A
GRAVA FILA
B
GRAVA FILA
A
BEGIN
TRANSACTION
COMMIT
TRANSACTION
ROLLBACK
TRANSACTION
PAGINA
CHAMA
COMMIT
CHAMA
ROLLBACK
CHAMA
PAGINA
ENVIA MSG
USUARIO
CHAMA
COMMIT
CHAMA
BEGIN


E
BEGIN TRANSACTION
CHAMA BEGIN
BEGIN TRANSACTION
ST = "BGT"
CRIA ID
GRAVA LOG
ST = "PRT"
ATRIBUI NUMERO
GRAVA NOME, PGM E ARQ. CHAMADOR
AAMMDD-HHHH


COMMIT TRANSACTION
ST = "CMT"
PROCESSO
*
até esgotar ocorr. LPA
ST = "ENT"
PROCESSO LOG
GRAVA BD
FORMATA LOG
GRAVA LOG
ROLLBACK TRANSACTION
ST = "RBT"
PROCESSO
*
até esgotar ocorr. LPI
ELIMINA REG NO ARQ. LOG
ST = "ENT"
GRAVA BD


PASINA
CRIACAO
ATUALIZACAO /ELIMINACAO
ENVIA MSG USUARIO
CHAMA COMMIT
PROCESSO L.P.I.
PROCESSO L.P.A.
BUSCA
EXISTE
INCORPORA
BUSCA
EXISTE
NAO EXISTE
ALTERA
INCORPORA

ANEXO B

Sistema de Transações: Rotinas Desenvolvidas

```
/* ************************************************************************ */
/* Estas rotinas estão associadas ao tratamento de filas. As operações      */
/* definidas são:                                                           */
/*        - definição de filas                                              */
/*        - tratamento de filas                                             */
/*        - incorporação de elementos                                       */
/*        - recuperação de elementos                                        */
/* sendo as duas últimas associadas aos processos: roteador, aplicação,     */
/* gerenciador e UniCOSMOS.                                                 */
/* ************************************************************************ */
/* Veja no anexo A                                                          */
/*        - processo LE FILA                                                */
/*        - processo GRAVA FILA                                             */
/* ************************************************************************ */

/*****************/
/** define fila **/
/*****************/

struct  fifo  *fila_ap[15]
struct  fifo  *fila_rot[15]
struct  fifo  *fila_gt[15]
struct  fifo  *fila_uni[15]
int ocorr_ap = 0;
int ocorr_rot = 0;
int ocorr_gt = 0;
int ocorr_uni = 0;



/*************************/
/** tratamento de filas **/
/*************************/

unsigned int trata_fifo (proc, op, elemento)
char proc,
     op;
struct fifo elemento;

{
int  ret;

  switch (proc)
  {
     case APLIC:
   if (op == "l")
      ret = recfifo_ap ();
   if (op == "g")
      ret = incfifo_ap (elemento);
```

```
    break;
      case ROT:
    if (op == "l")
       ret = recfifo_rot ();
    if (op == "g")
       ret = incfifo_rot (elemento);
    break;
      case GT:
    if (op == "l")
       ret = recfifo_gt ();
    if (op == "g")
       ret = incfifo_gt (elemento);
    break;
      case UNI:
    if (op == "l")
       ret = recfifo_uni ();
    if (op == "g")
       ret = incfifo_uni (elemento);
    break;
  }
}



/*************************************************/
/** incorpora elemento em fila aplicacao **/
/*************************************************/

unsigned int incfifo_ap (elem)
struct fifo elem;

{
int  retorno;

  if (ocorr_ap > 15)
  {
     retorno = 0;
  }
  else
  {
     fila_ap[ocorr_ap] = elem;
     ocorr_ap = ocorr_ap + 1;
     retorno = 1;
  }
  return(retorno);
}
```

```
/*****************************************/
/** recupera elemento em fila aplicacao **/
/*****************************************/

unsigned int recfifo_ap ()
struct fifo elem;

{
int  retorno;

  if (ocorr_ap > 0)
  {
     ocorr_ap = ocorr_ap - 1;
     elem = fila_ap[ocorr_ap];
     retorno = 1;
  }
  else
  {
     retorno = 0;
  }
  return(retorno);
}



/*****************************************/
/** incorpora elemento em fila roteador **/
/*****************************************/

unsigned int incfifo_rot (elem)
struct fifo elem;

{
int  retorno;

  if (ocorr_rot > 15)
  {
     retorno = 0;
  }
  else
  {
     fila_rot[ocorr_rot] = elem;
     ocorr_rot = ocorr_rot + 1;
     retorno = 1;
  }
  return(retorno);
}
```

```
/*****************************************/
/** recupera elemento em fila roteador **/
/*****************************************/

unsigned int recfifo_rot ()
struct fifo elem;

{
int  retorno;

  if (ocorr_rot > 0)
  {
     ocorr_rot = ocorr_rot - 1;
     elem  =  fila_rot[ocorr_rot];
     retorno = 1;
  }
  else
  {
     retorno = 0;
  }
  return(retorno);
}



/*****************************************/
/** incorpora elemento em fila gerenciador **/
/*****************************************/

unsigned int incfifo_gt (elem)
struct fifo elem;

{
int  retorno;

  if (ocorr_gt > 15)
  {
     retorno = 0;
  }
  else
  {
     fila_gt[ocorr_gt] = elem;
     ocorr_gt = ocorr_gt + 1;
     retorno = 1;
  }
  return(retorno);
}
```

```
/*****************************************/
/** recupera elemento em fila gerenciador **/
/*****************************************/

unsigned int recfifo_gt ()
struct fifo elem;

{
int  retorno;

  if (ocorr_gt > 0)
  {
     ocorr_gt = ocorr_gt - 1;
     elem  =  fila_gt[ocorr_gt];
     retorno = 1;
  }
  else
  {
     retorno = 0;
  }
  return(retorno);
}



/*****************************************/
/** incorpora elemento em fila UniCOSMOS **/
/*****************************************/

unsigned int incfifo_uni (elem)
struct fifo elem;

{
int  retorno;

  if (ocorr_uni > 15)
  {
     retorno = 0;
  }
  else
  {
     fila_uni[ocorr_uni]  =  elem;
     ocorr_uni = ocorr_uni + 1;
     retorno = 1;
  }
  return(retorno);
}
```

```
/*****************************************/
/** recupera elemento em fila UniCOSMOS **/
/*****************************************/

unsigned int recfifo_uni ()
struct fifo elem;

{
int  retorno;

  if (ocorr_uni > 0)
  {
     ocorr_uni = ocorr_uni - 1;
     elem  =  fila_uni[ocorr_uni];
     retorno = 1;
  }
  else
  {
     retorno = 0;
  }
  return(retorno);
}
```

```
/* ************************************************************************ */
/* Estas rotinas implementam os comandos de controle de transação:          */
/*        - BEGIN_TRANSACTION                                               */
/*        - COMMIT_TRANSACTION                                              */
/*        - ROLLBACK_TRANSACTION                                            */
/*        - PAGINA                                                          */
/* ************************************************************************ */
/* Veja no anexo A                                                          */
/*        - processo BEGIN TRANSACTION                                      */
/*        - processo COMMIT TRANSACTION                                     */
/*        - processo ROLLBACK TRANSACTION                                   */
/*        - processo PAGINA                                                 */
/* ************************************************************************ */

#include   "stdio.h"
#include   "time.h"
#include   arqlog.h

extern char *status;
extern int   *nro;




/*************************************************/
/**  Esta rotina executa BEGIN-TRANSACTION  **/
/*************************************************/

unsigned int beg-trans (pgm, arquivo)
char  *arquivo,
      *pgm;

{
int    retorno;
struct  arqlog  registro;
struct  identificador  a1;
struct  idtrans  b1;
struct  data  c1;
struct tm tempo;

  status  =  "bgt";

/***********************************    cria id **/
  strcpy  (a1.arquivo,  arquivo);
/*.........*/
  if (nro < LIMITRANS)
     nro ++;
  else
     nro = 1;
```

```
  bl.numero = nro;
/*.....*/
  strcpy (bl.nomepgm, pgm);
/*.....*/
  cl.aa = tempo.tm_year%100;
  cl.mm = tempo.tm_mon;
  cl.dd = tempo.tm_mday;
  cl.hora = tempo.tm_hour;
  cl.min = tempo.tm_min;
  al.aammddhhhh = cl;

/*************************************   grava disco **/
#ifdef BSD41
  ;
#else
  registro.arqlog_id.id_t  =  bl;
#
  registro.status  =  "prt";
  retorno = grava-disco ();
  if  (retorno)
     status = "prt";
  return(retorno);
}
```

```
/*****************************************************/
/**  Esta rotina executa COMMIT-TRANSACTION  **/
/*****************************************************/

void cmt-trans (action)
char *action;

{
  status = "cmt";
  printf (" status= %s ",status);
  printf (" Executando %s ",action);

/*********************************  processo **/

/*************************************  grava bd **/
  retorno = grava-bd ();

/*************************************  processo log **/

/*****************************************  formata log **/
  retorno = formata-log ();

/*****************************************  grava log **/
  retorno = grava-log ();
  status = "ent";
  printf (" status= %s ",status);
}
```

```
/*****************************************************/
/**  Esta rotina executa ROLLBACK-TRANSACTION  **/
/*****************************************************/

void  rbt-trans  (action)
char  *action;

{
int   retorno;

  status  =  "rbt";
  printf  ("  status=  %s  ",status);
  printf  ("  Executando  %s  ",action);

/***********************************    processo  **/

/*****************************************    grava  bd  **/
  retorno  =  grava-bd  ();

/***********************************    elimina  reg.arq.log  **/
  retorno  =  elim-reg-log  ();
  status  =  "ent";
  printf  ("  status=  %s  ",status);
}
```

```
/*****************************************************/
/**  Esta rotina executa a atualizacao das paginas  **/
/*****************************************************/

void  pag-trans  (action)
char  *action;

{
int    retorno;

  printf  (" Executando %s ",action);

/*********************************  atualizacao pagina **/

/*********************************  criacao pagina **/
  retorno = cria-pag ();

/*************************************  atualiza lista **/
                                       pag.iniciais   **/
  retorno = atualiza-pag-inic ();

/*************************************  atualiza lista **/
                                       pag.atualiz.   **/
  retorno = atualiza-pag-atual ();

/*********************************  eliminacao pagina **/
  retorno = elim-pag ();

/*************************************  atualiza lista **/
                                       pag.iniciais   **/
  retorno = atualiza-pag-inic ();

/*************************************  atualiza lista **/
                                       pag.atualiz.   **/
  retorno = atualiza-pag-atual ();
}
```

```
/*****************************************/
/****  estrutura do arquivo de log  ****/
/*****************************************/

#define LIMITRANS       32000
#define ARQNOME          12
#define PGMNOME          12

struct   identificador
  {
  char arquivo[ARQNOME+1];
  struct  idtrans  id_t;
  struct data aammddhhhh;
  } ;

struct  idtrans
  {
  unsigned  int  numero;
  char nomepgm[PGMNOME+1];
  } ;

struct  arqlog
  {
  struct  identificador  arqlog_id;
  char  status[3];
  unsigned  int  qtdin;
  union descpag *infpagin[QTDPAG];
  unsigned  int  qtdat;
  union descpag *infpagat[QTDPAG];
  } ;

struct  data
  {
  int  aa;
  int mm;
  int  dd;
  int  hora;
  int  min;
  } ;

union  descpag
  {
  Stn_Pag;
  Std_Pagobj;
  struct  frmtrd;
  Sto_Pagaoc;
  Stv_Pagavl;
  Sta_pag;
  } ;
```

```
/* ************************************************************************ */
/* Estas rotinas implementam o roteador do sistema de transações. São      */
/* apresentados os módulos que definen a sintaxe da linguagem utilizada e  */
/* a semântica das mensagens                                                */
/* ************************************************************************ */
/* Veja no anexo A                                                          */
/*         - processo DETERMINA AÇÃO                                        */
/* ************************************************************************ */

%{
  /*
   * Especificação Analizador Lexico
     Linguagem  Gerenciador  de transações
   */

#include "y.tab.h"

%}

%%

begin_transaction|begin_trans|bgt|begin          return (BGT);
commit_transaction|commit_trans|cmt|commit       return (CMT);
rollback_transaction|rollback_trans|rbt|rollback return (RBT);
pagina|pag|page                                  return (PAG);
[a-z|A-Z|^|0-9]+                                 return (STR);
[ ]+                                             ;
                                                 return (-1);
["!"|"@"|"#"|"$"|"%"|"&"|"*"|"("|")"|"_"]+      return (UNKNOWN);
["+"|"{"|"}"|":"|"~"|"'"|"<"|">"|"?"|"="|";"]+  return (UNKNOWN);
["'"|"`"|","|"."|"/"]+                           return (UNKNOWN);
.                                                return (UNKNOWN);
%%
yywrap()
{
return (1);
}
```

```
%token  BGT CMT RBT PAG STR
%token  UNKNOWN
%start  comando

%{

int command;
char *status, *stringl, *action;

%}
%%

comando  :      comval
         |      cominv
                        { command = UNKNOWN;
                          treatunk (); }
         ;

com      :      BGT
                        { command = BGT; }
         |
                CMT
                        { command = CMT; }
         |
                RBT
                        { command = RBT; }
         |
                PAG
                        { command = PAG; }
         ;

comval   :      com STR
                        { treat (); }
         ;

cominv   :      com | STR | UNKNOWN
         |
                com cominv
         |
                STR cominv
         |
                UNKNOWN cominv
         ;

%%

#include   <stdio.h>
#include   <malloc.h>
#include   <string.h>
```

```
extern int   yyleng;
extern char  yytext[];

yyerror (s)
char *s;

{
  if (yychar) fprintf (stderr, "%s - TOKEN = %d", s, yychar);
}

treat ()

{
  switch (command)
  {
    case BGT: action = "Begin_Transaction";
/*
      if (!strcmp (status, "ent"))
          beg_trans (action);   * (parametros); *
      else
          * volta a fila * ;
*/
      break;
    case RBT: action = "Rollback_Transaction";
/*
      if (!strcmp (status, "prt"))
          rbt_trans (action);   * (parametros); *
      else
          * volta a fila * ;
*/
      break;
    case CMT: action = "Commit_Transaction";
/*
      if (!strcmp (status, "prt"))
          cmt_trans (action);   * (parametros); *
      else
          * volta a fila * ;
*/
      break;
    case PAG: action = "Pagina";
/*
      if (!strcmp (status, "prt"))
          pag_trans (action);   * (parametros); *
      else
          * volta a fila * ;
*/
      break;
    }
  printf (" Executando %s", action);
}
```

```
treatunk  ()

{
  if (command == UNKNOWN) action = "desconhecido";

  printf (" Pedido de execucao %s", action);
}
```

```
/* ********************************************************************** */
/* Esta rotina faz o controle de timeout para uma rotina qualquer da qual */
/* se queira verificar que o processamento nao excede um tempo            */
/* determinado.                                                           */
/* input : nome da rotina que se quer controlar e tempo a controlar       */
/* output: -1 = erro, 0 = finalizacao ok e 1 = tempo expirado             */
/* ********************************************************************** */
/* Rotina complementar                                                    */
/* ********************************************************************** */

#include <stdio.h>

void exit ();
unsigned sleep ();

void processo1 ()          /* processo sobre o qual se quer */
                           /* fazer controle de tempo       */
{
  printf ("Proc1: exec.parte 1");
  sleep (1);
  printf ("Proc1: exec.parte 2");
  sleep (1);
  printf ("Proc1: exec.parte 3");
  sleep (1);
}

void processo2 ()          /* processo sobre o qual se quer */
                           /* fazer controle de tempo       */
{
  printf ("Proc2: exec.parte 1");
  sleep (1);
  printf ("Proc2: exec.parte 2");
  sleep (1);
  printf ("Proc2: exec.parte 3");
  sleep (1);
}

main ()

{
  int ret;

  ret = timeout (processo1, 1);      /* forma de chamar timeout */
  switch (ret) {                   /* acoes a serem realizadas */
    case -1:                       /* dependendo do retorno da */
      printf ("Erro");             /* rotina de controle de tempo */
      break;
    case 0:
      printf ("OK");
      break;
```

```
    case 1:
      printf ("Tempo expirado");
      break;
    default:
      break;
  }
  ret = timeout (processo2, 6);
  switch (ret) {
    case -1:
      printf  ("Erro");
      break;
    case 0:
      printf  ("Ok");
      break;
    case 1:
      printf ("Tempo expirado");
      break;
    default:
      break;
  }
}

int timeout (funcao, tempo)
void (*funcao) ();
unsigned tempo;

{
  int x = 0;
  int retorno, proc, slep;

  switch (proc = fork ()) {
    case -1:
      retorno = -1;              /* erro no fork */
      break;
    case 0:
      (*funcao) ();
      exit (0);
    default:
      break;
  }
  switch (slep = fork ()) {
    case -1:
      retorno = -1;              /* erro no fork */
      break;
    case 0:
      sleep (tempo);
      exit (0);
    default:
      break;
  }
```

```
  for ( ; x != proc && x != slep; ) x = wait ((int *) 0);
  retorno = (x == slep) ?
    (kill (proc, 9), 1) :          /* tempo expirado */
    (kill (slep, 9), 0) ;          /* finalizacao ok */
  return  (retorno);
}
```

```
/* ********************************************************************** */
/* Recebe requerimentos e envia respostas                                 */
/* ********************************************************************** */
/*  Rotina complementar                                                   */
/* ********************************************************************** */

#include  <errno.h>
#include  <stdio.h>
#include  <string.h>
#include  "dbms.h"
#include  "fn1.h"
#include  "fn2.h"

extern  int  errno;

main ()

{
  MESSAGE m;

  char *dbmsval,
 name[30],
       *getenv();

  long dbmskey,
       atol();

  if ((dbmsval = getenv ("DBMSKEY")) == NULL)
    fatal ("falta a variavel de ambiente DBMSKEY");
  dbmskey = atol (dbmsval);
  while (receive (dbmskey, &m, sizeof (m))) {
    switch (m.cmd) {
      case  'o':
m.status = Dopen (m.file);
break;
      case  'c':
m.status = Dcreate (m.file);
brkpt  (8);
break;
      case  'q':
m.status = Dclose ();
rmqueue (dbmskey);
break;
      case  'g':
strcpy (name, m.rcd.name);
m.status = Dget (name, &m.rcd);
break;
      case  'n':
m.status = Dgetnext (&m.rcd);
break;
```

```
        case  'p':
  m.status = Dput (&m.rcd);
  break;
        case  'd':
  m.status = Ddelete (m.rcd.name);
  break;
        case  't':
  m.status = Dtop ();
  break;
        default:
  errno = EINVAL;
  m.status = ERROR;
        }
        m.errno = errno;
        if ( !send (m.clientkey, &m, sizeof (m)))
  printf ("Nao pode enviar a %ld; errno = %d", m.clientkey, errno);
        if ( m.cmd == 'q')
  exit (0);
     }
     syserr    ("receive");
}
```

```
/* ************************************************************************ */
/* Rotinas para testar o send e receive entre processos                     */
/* ************************************************************************ */
/*  Rotina complementar                                                     */
/* ************************************************************************ */

#include "message.h"

main()
{
  MESSAGE m;

  setbuf(stdout, NULL);
  while (receive(1000L, &m, sizeof(m)))
    printf("Received %d from %d", m.number, m.pid);
  syserr    ("receive");
}



#include "message.h"

main ()
{
  MESSAGE m;

  m.pid = getpid();
  for (m.number = 1; m.number <= 2; m.number++) {
    sleep (1);
    if (!send(1000L, &m, sizeof(m)))
      syserr("send");
  }
  exit(0);
}
```

ANEXO C

O UNICOSMOS

ANEXO C - O UNICOSMOS

Conteúdo

c.1 - Introdução

O UniCOSMOS - UNICAMP Object Storage Management O System - [/RICA 86/] é um sistema de software utilizável como núcleo de SGBD, independente de modelos de dados, que evoluiu a partir da proposta de CORAS [/ENCA 83/, /WU 83/]. No CORAS, todos os elementos do mundo real de interesse são tratados como objetos, no presente contexto entendidos como seus elementos primitivos, seja qual for a interpretação associada a eles (entidades, atributos ou valores associados aos atributos). Atributos podem ser representados através de uma classe especial de objetos denominada

objetos-relação; a associação entre entidades e valores de atributos pode ser representada através de uma associação binária (tríade) que liga um valor a um objeto através de um atributo. Um outro tipo de objetos, os objetos-conjunto, permite definir classes de elementos que também podem ser associados através de objetos-relação. Embora esta representação seja muito adequada do ponto de vista do alto nível de abstração alcançado, sob o ponto de vista da implementação há diversas deficiências que devem ser levadas em conta, entre elas:

acesso: o acesso associativo simulado através de "hashing" não é flexível quanto a quantidade de elementos que pode ser manipulada. Nesta abordagem, é difícil prever esta quantidade, pois cada elemento representado - entidades, atributos, valores, conjuntos - ocupa uma entrada nesta tabela "hash";

representação dos dados: todos os elementos do mundo CORAS são representados através de seu nome - uma sequência de caracteres com dimensão pré-definida. Não há forma de tratar diretamente outros tipos de dados ou casos em que se desejasse representar elementos com maior dimensão que a estabelecida pelo Sistema;

velocidade: todas as associações, sejam relacionamentos entre conjuntos ou ocorrências de atributos, são representadas através das tríades. Para agilizar o acesso a estas informações, são armazenadas as três permutações de cada tríade, ou seja, as três possíveis representações para uma associação binária. Esta redundância acarreta uma sobrecarga de controle para a implementação, além da questão da ocupação de espaço;

interpretação: não há distinção entre entidades e valores, devendo qualquer interpretação ser suportada pela aplicação.

Para tornar este núcleo mais eficiente, embora reduzindo o nível de abstração suportado, o UniCOSMOS foi proposto. O Sistema será descrito nos próximos itens, englobando seus elementos primitivos, arquitetura interna e uma visão funcional. Finalmente, alguns aspectos particulares à implementação serão abordados.

c.2 - Elementos Primitivos

Na representação de informações, estas podem em geral ser classificadas em uma das seguintes formas [/COST 84/]:

- unidades de informação, com propriedades associadas;
- classes ou tipos, constituindo um conjunto de unidades de informação com características semelhantes; e
- relações, que permitem a conexão entre unidades de informação ou entre classes.

Os elementos primitivos do UniCOSMOS são objetos. Estes objetos têm por função representar classes, que internamente descreverão os elementos primitivos do modelo suportado pelo SGBD implementado sobre o UniCOSMOS. Desta forma, estes objetos podem representar relações, tipos de registros, tipos de conjuntos, tipos de relacionamentos, tipos de entidades, dependendo do modelo utilizado. Objetos podem ser de três tipos:

SIMPLES: representa uma classe, à qual podem estar associados atributos (ao contrário do que ocorria no CORAS);

CONJUNTO: define agrupamentos de outros objetos, que podem ser simples, conjuntos ou mesmo relações;

RELAÇÃO: permite estabelecer relacionamentos binários entre outros objetos que não sejam do tipo relação (simples/simples, conjunto/conjunto ou simples/conjunto).

Os objetos UniCOSMOS são identificados por seu nome. Este nome constitue a chave de acesso a todas as informações sobre o objeto. Para garantir a flexibilidade do Sistema, este nome não é necessariamente único para cada objeto, podendo ser definidos sinônimos que definem igualmente este acesso. No entanto, nomes não podem ser repetidos.

As associações entre objetos UniCOSMOS são definidas na forma OBJETO RELAÇÃO (OBJETO NÃO RELAÇÃO 1, OBJETO NÃO RELAÇÃO 2), onde o objeto não-relação pode ser do tipo simples ou conjunto. Somente são permitidos relacionamentos binários (tríades). A representação de outras formas de associação deve ser suprida pelas aplicações suportadas.

Cada objeto pode ser caracterizado por seus atributos. No UniCOSMOS estes atributos são armazenados em máscaras associadas ao objeto, que descrevem cada agregação de propriedades. Estas máscaras contêm informações tais como nome e tipo de dado associado a cada atributo, definindo as características comuns entre os elementos que constituem as instâncias do objeto (suas ocorrências). Um mesmo atributo pode ser associado a diferentes objetos. Internamente o Sistema isola estes elementos, facilitando o controle de acesso a informações por parte de diferentes usuários (concorrência e segurança).

A caracterização de um objeto pode não ser única. Em alguns casos pode haver mais de um conjunto de atributos associado a um objeto, representando diferentes tipos de informação além da descrição de propriedades, como o momento de criação do objeto, características que instâncias do objeto devem obedecer, descrições sobre os atributos e outras informações. A fim de atender estes casos o UniCOSMOS permite a declaração de mais de uma máscara de atributos associada a cada objeto, sendo cada máscara associada a um rótulo que a identifica. A forma de utilização das diferentes máscaras que podem ser associadas ao objeto depende da aplicação, de forma a tornar este Sistema flexível às diversas necessidades.

Ocorrências são instâncias do objeto definidas através de valores associados aos seus atributos. Estas instâncias - que podem ser visualizadas como registros lógicos na base de dados (embora não exista um "registro físico" associado) - constituem a unidade de manipulação dentro do objeto. O conceito de ocorrência pode ser utilizado para mapear tuplas, registros, entidades, relacionamentos, de acordo com o modelo suportado.

Uma das características do UniCOSMOS é a 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 nível global e a nível local simultaneamente (controle de macro-aspecto e micro-aspecto). Outra vantagem de distribuir a base de dados em vários Sistemas de Arquivos (em relação à proposta de haver uma única base de dados para todas as aplicações, como ocorre em grande parte dos SGBD existentes) é principalmente a independência que se cria entre as

aplicações. Existem duas formas de se definir um novo Sistema de Arquivos:

- iniciando uma nova base de dados;
- definindo uma nova base de dados a partir de outra existente, mantendo a base de dados original inalterada (criar uma nova versão da base de dados).

Os dados operacionais do Sistema estão armazenados na forma de ocorrências, definidas através de ligações entre valores de atributos, enquanto que as informações por ele mantidas sobre os objetos permitem a gerência a um nível mais alto (como uma espécie de Dicionário de Dados local a cada aplicação). O acesso a estes dados operacionais pode ser agilizado pelo UniCOSMOS através de índices mantidos automaticamente para atributos definidos pela aplicação.

O suporte ao controle dos Sistemas de Arquivos e Tipos de Dados suportados é provido no UniCOSMOS através de outra base de dados com as mesmas características das demais. Nesta base de dados, o Dicionário/Diretório integrado ao Sistema, Sistemas de Arquivos e Tipos de Dados são objetos com atributos pré-definidos.

Tipos de dados não são pré-definidos no UniCOSMOS como forma de manter a flexibilidade desejada. Todos os elementos internos são armazenados na forma de caracteres, estando a conversão a outros tipos subordinada ao SGBD. No entanto, o UniCOSMOS permite associar informações à definição de um tipo de dado, o que permite verificar por exemplo se o valor do elemento que está sendo armazenado é válido ou quais operações são permitidas. Este tipo de informação pode ser utilizado para especificar Tipos Abstratos de Dados, tais como datas ou elementos geométricos, definidos a partir dos tipos elementares (como inteiro, real e caráter). Especificações de consistência a nível de interrelacionamento dos valores dos atributos de um objeto ou entre objetos distintos devem ser controladas pelo SGBD, assim como o suporte a definições e tratamento de tipos de dados.

Cada Sistema de Arquivos também tem suas características armazenadas no Diretório, tais como versão a que corresponde, seu autor e data de criação. Além destas informações, particulares de cada Sistema, existe um outro tipo de informação que especifica o relacionamento entre as versões

existentes, que permite controlar consistência entre versões.

c.3 - Tratamento de Objetos

Cada objeto UniCOSMOS contém basicamente três tipos de informação:

nome, que permite sua identificação;

atributos, que caracterizam as propriedades da classe por ele representada;

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

A definição de um objeto passa por duas fases: a criação de seu nome e a caracterização da classe por ele representada. Dois objetos distintos podem ter um mesmo conjunto de propriedades sob o ponto de vista do usuário UniCOSMOS (por exemplo, a propriedade "endereço" pode estar associada tanto à classe dos "professores" como à classe dos "alunos"), mas internamente são associados conjuntos de valores distintos para cada atributo.

Os elementos dos conjuntos de valores de cada atributo (os valores propriamente ditos) são definidos à medida que são definidas as ocorrências da classe. Cada ocorrência é especificada por um conjunto de associações atributo-valor. Internamente, se o valor especificado para um atributo já foi definido através de outra ocorrência, então apenas a associação deste valor com a nova ocorrência é estabelecida; caso contrário, o valor é inicialmente registrado no conjunto de valores correspondente para então ser definida a associação.

A associação de diferentes valores para um atributo em uma mesma instância é permitida, caracterizando um atributo multivalorado. Para permitir a representação de atributos que não devem ser multivalorados (por exemplo, "data de nascimento"), deve ser feita uma distinção a nível de definição dos atributos, que pode posteriormente ser alterada desde que se mantenha a consistência em relação aos dados já armazenados.

As informações sobre objetos podem ser atualizadas sem impacto para o Sistema. Um mesmo objeto pode ter diversos nomes associados através da

definição de sinônimos. A remoção de nomes não traz impactos ao Sistema, desde que pelo menos um nome permaneça associado ao objeto. Atributos podem ser acrescidos às propriedades do objeto, nomes e funcionalidade (monovaloração ou multivaloração) de atributos podem ser alterados. A remoção de um atributo acarreta a remoção do conjunto de valores associados e, consequentemente, das associações com suas ocorrências.

c.4 - Tratamento de Ocorrências

Ocorrências representam as instâncias dos objetos UniCOSMOS.

Dependendo da classificação do objeto (simples, conjunto ou relação), o tratamento de suas ocorrências pode diferir. No entanto, um conceito comum às instâncias de objetos é a identificação interna da ocorrência, uma chave interna de acesso às informações sobre as propriedades da instância. O usuário do Sistema não tem acesso a esta chave, que é definida e manipulada internamente, de forma que a única maneira de acessar instâncias é através de cláusulas especificando atributos e valores (na verdade, existe outra forma de acesso que será discutida no Item C.5.6). Esta chave é independente dos valores de atributos da instância, de forma que alterações destes valores não acarretam inconsistências sobre as informações já existentes (como será visto adiante no caso de instâncias de objetos relação).

O tipo de ocorrência mais elementar é a que está associada a objetos simples. As ocorrências são totalmente definidas através dos valores associados aos atributos, sem depender das associações do objeto com outros.

Para objetos conjuntos, há dois conceitos de instâncias associados: ocorrências como definidas para objetos simples e/ou objetos pertencentes ao conjunto. Um tratamento especial destas duas formas de ocorrências ou mesmo a interligação entre elas (por exemplo, definindo um atributo "nome do elemento" para o objeto conjunto) é de responsabilidade da aplicação; sob o ponto de vista de UniCOSMOS são duas informações distintas.

A remoção de uma destas ocorrências implica na remoção de todas as associações atributos-valores de uma determinada instância. Se algum dos valores associados fica, devido a esta operação, sem ocorrências associadas,

então este valor é também removido da base de dados. Caso ainda existam associações do valor com outras instâncias, o valor permanece armazenado.

A atualização de um valor de atributo em uma ocorrência significa mudar a associação entre a instância e o elemento do conjunto de valores (o valor). Se não existir o novo valor que foi definido para a instância, este é inicialmente criado para então ser definida a associação.

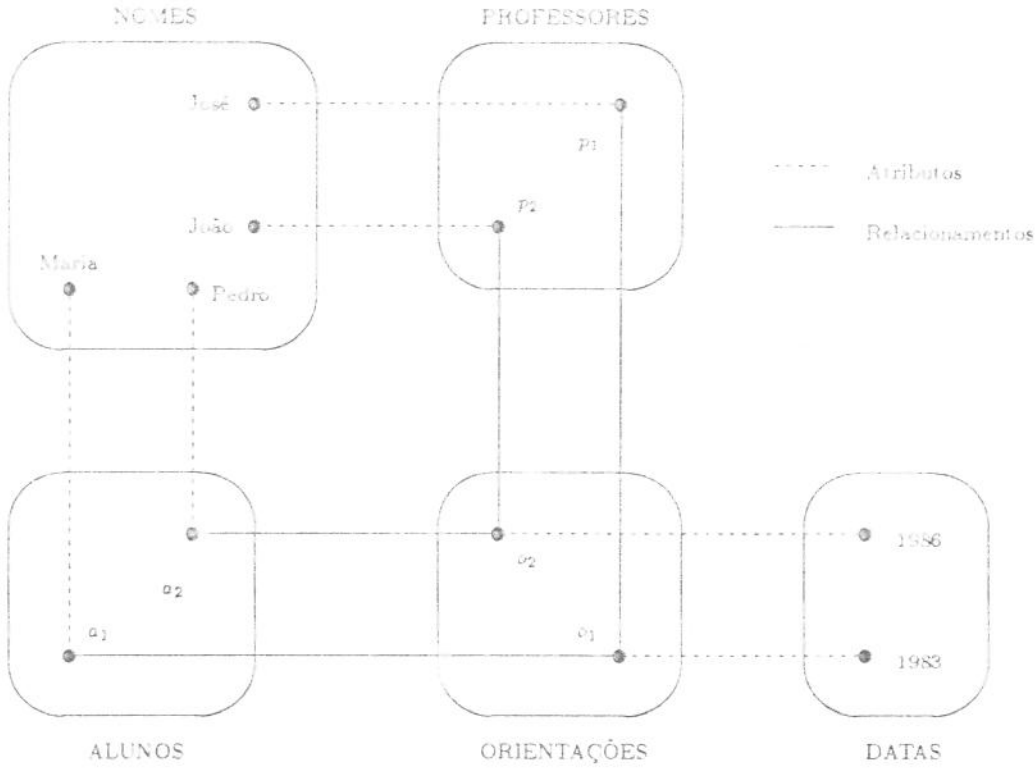


Figura C.1: Associações entre conjuntos.

O conceito de ocorrência de um objeto relação exige uma análise mais detalhada. Inicialmente, é interessante relembrar os conceitos de associação

a um nível mais abstrato para depois particularizar para o caso de UniCOSMOS. Pode-se dizer que existem basicamente dois tipos de associações entre conjuntos, como mostrado na Figura C.1: as que definem as propriedades de um elemento (por exemplo, p_1 e "José") e as que definem relacionamentos entre elementos (por exemplo, p_1 e o_1). Note-se que especificar "orientação" como uma classe distinta, e não apenas como arcos de associação, é a forma de permitir caracterizar cada elemento de orientação, permitindo associar propriedades a ela.

No UniCOSMOS, as associações também são definidas a nível de conjuntos. No primeiro caso, na caracterização de propriedades de elementos, esta associação é definida através dos atributos,sendo no entanto que cada objeto tem associado um conjunto de valores particular a cada atributo. Haveria internamente, neste exemplo, dois conjuntos de valores "nomes de professores" e "nomes de alunos", mesmo que o nome dos atributos fosse o mesmo ("nome") para os dois objetos.

O segundo tipo de associação é estabelecido através de objetos relações e tríades. Desta forma, para se permitir uma associação entre um elemento da classe dos "professores" a um elemento da classe "orientação", seria necessário criar uma tríade envolvendo os dois objetos - no exemplo, através de um objeto relação "orientador". Uma ocorrência de "orientador" não tem, neste caso, propriedades; apenas define uma associação entre pares de elementos (por exemplo, [p_1, o_1] e [p2, o2]). É interessante observar que, como estas associações são definidas em termos de identificadores internos, são independentes de valores de atributos dos elementos envolvidos, não sendo afetadas por atualizações nos objetos envolvidos (ao contrário do que ocorre em sistemas que suportam tabelas).

A remoção de ocorrências de objetos relações envolve outros cuidados. Algumas vezes é interessante que a remoção de um relacionamento provoque a remoção de ocorrências associadas (conceito de dependência), enquanto outras vezes isto não é desejado. Para enquadrar estes dois casos, é definida uma subclassificação do objeto relação, que especifica "propagação" ou "não-propagação". Em uma tríade R(A,B), por exemplo, a remoção de uma ocorrência do objeto A implicará na remoção das ocorrências de B associadas

através de R. No entanto, a remoção de uma ocorrência de B acarretará a remoção das ocorrências de A associadas via R se e somente se sua subclassificação especificar a "propagação". Esta subclassificação constitui uma propriedade do objeto como um todo e não apenas de seus elementos, como é o caso de atributos.

c.5 - Arquitetura Interna

O UniCOSMOS armazena as informações distribuídas entre diversos domínios internos. São eles:

- Domínio de Nomes;
- Domínio de Objetos;
- Domínio de Tríades;
- Domínio de Valores;
- Domínio de Ocorrências;
- Domínio de Índices;
- Domínio de Acesso.

Destes domínios, os três primeiros são derivados diretamente da proposta do CORAS. A seguir, cada um destes domínios é descrito.

c.5.1 - Domínio de Nomes

O Domínio de Nomes, a partir do qual os demais domínios são acessados, constitui a interface de acesso entre o usuário e as informações armazenadas sobre o objeto. Este acesso é estabelecido através do nome do objeto. Nomes não devem ser repetidos, mesmo para objetos de diferentes tipos, mas não necessitam ser únicos para um objeto. Nomes alternativos podem ser criados, constituindo assim os diversos sinônimos do objeto.

Além dos sinônimos de cada objeto, o Domínio de Nomes também mantém informação sobre a classe do objeto (se simples, conjunto ou relação) e seu identificador interno. Este identificador (ID) corresponde ao endereço onde

se armazenam as informações no Domínio de Objetos referentes a cada objeto individual. No Domínio de Nomes traduz-se o nome do objeto para seu ID, de forma que os demais domínios referenciam o objeto apenas por este identificador e não por seu nome. Sinônimos são dois ou mais nomes distintos com o mesmo ID associado. A troca de nome de um objeto não tem, desta forma, qualquer impacto sobre os dados já armazenados para ele.

c.5.2 - Domínio de Tríades

O Domínio de Tríades mantém informações sobre os relacionamentos binários (as tríades) entre os objetos. Da mesma forma que no CORAS, as três permutações de uma tríade são armazenadas de forma que a associação possa ser consultada a partir de qualquer um de seus componentes. Cada objeto que faça parte de pelo menos uma tríade tem associada uma entrada no Domínio de Tríades. Esta entrada associada a cada objeto é acessada através do identificador interno deste domínio (IR).

No Domínio de Tríades, os objetos associados são identificados por seus identificadores internos do Domínio de Objetos (ID). Desta forma, a partir do Domínio de Tríades é possível acessar o Domínio de Objetos, que centraliza a comunicação com os demais domínios.

c.5.3 - Domínio de Valores

O Domínio de Valores constitui o depósito dos conjuntos de valores definidos para os atributos dos objetos. Para cada atributo definido no Domínio de Objetos existe uma entrada no Domínio de Valores, acessada através de uma identificação interna (IC) que endereça uma "seção" do domínio que contém o conjunto de todos os valores associados ao atributo. Cada um dos elementos deste conjunto (ou seja, cada valor) tem associado um outro identificador interno deste domínio, o identificador do valor (IV).

Valores estão associados a ocorrências. A informação sobre quais valores estão associados a quais ocorrências também está armazenada neste domínio. A

cada ocorrência está associado um identificador interno do Domínio de Ocorrências (IO). A cada valor armazenado no Domínio de Valores está associada uma lista de IO, com pelo menos um elemento. Cada elemento desta lista corresponde a uma ocorrência que está associada a este valor.

É importante notar que o Domínio de Valores não suporta o conceito matemático de domínios, pois a dois atributos distintos são associadas listas de valores distintas, mesmo que os atributos tenham o mesmo tipo de dado associado. Entretanto, para cada atributo não há repetição de valores, mesmo que diversas ocorrências distintas atribuam um mesmo valor para o atributo, evitando desta forma a redundância que existe em sistemas que manipulam registros ou tabelas.

A representação interna dos valores armazenados está na forma de caracteres. Não há limitações para a dimensão de cada valor, bem a obrigatoriedade de dimensões pré-fixadas. Caso seja interessante fixar a dimensão que a representação de um atributo vai assumir em uma aplicação suportada pelo UniCOSMOS, esta tarefa deverá ser executada pela própria aplicação através do tratamento de regras, por exemplo.

c.5.4 - Domínio de Ocorrências

Ocorrências são instâncias associadas a objetos. A cada ocorrência está associada uma entrada no Domínio de Ocorrências, endereçada por um identificador interno (IO). Uma ocorrência de um objeto é definida através da associação de valores aos atributos deste objeto.

No Domínio de Ocorrências, estes valores são representados pelos identificadores internos do Domínio de Valores (IV). Assim, a informação completa sobre uma ocorrência é composta por sua identificação (IO) associada a uma lista de pares [identificação do atributo, IV do valor associado], onde cada elemento da lista define uma associação atributo-valor.

É interessante notar que não há restrições quanto à unicidade da identificação do atributo nesta lista, de forma que atributos multivalorados não representam nenhum impacto sobre a estrutura de armazenamento do Sistema (dois ou mais elementos da lista com a mesma identificação de atributo). Por

outro lado, atributos que não tenham valor associado em uma ocorrência (conceito de valor nulo) não ocupam espaço de armazenamento; simplesmente não há um elemento nesta lista associado a este atributo.

c.5.5 - Domínio de Índices

A comunicação entre o Domínio de Objetos e o Domínio de Valores pode ser realizada diretamente, através do identificador IC, associando cada atributo ao seu conjunto de valores. Uma vez acessada a entrada ao conjunto de valores, a busca de um valor específico é realizada de maneira sequencial, o que pode representar uma sobrecarga de processamento se o atributo correspondente tem muitos valores associados e é frequentemente utilizado como item de busca.

O Domínio de Índices foi proposto para permitir uma otimização no acesso para estes casos. Se um atributo for definido como indexado, o identificador associado a ele no Domínio de Objetos indica não uma entrada para o conjunto de valores no Domínio de Valores, como no caso normal, mas sim uma entrada a uma estrutura de índice definida no Domínio de Índices. Esta entrada é endereçada por um identificador interno para este domínio (II). A partir desta entrada, o valor correspondente ao desejado é diretamente acessado no Domínio de Valores através do identificador do valor IV.

A definição do tipo de estrutura índice utilizada neste domínio (e.g., árvore binária, árvore-B) corresponde a um detalhe de implementação não abordado nesta apresentação.

c.5.6 - Domínio de Acesso

A informação sobre a definição de cada instância de um objeto está definida no Domínio de Ocorrências. A cada objeto está associado um conjunto de instâncias, que constituem suas ocorrências. Desta forma, o Domínio de Ocorrências contém a definição de todas as ocorrências de todos os objetos definidos.

O Domínio de Acesso define quais ocorrências, entre todas as definidas no Domínio de Ocorrências, estão associadas a um objeto específico. Para cada objeto definido no Domínio de Objetos está associada uma entrada para o Domínio de Acesso endereçada por seu identificador interno IA. A cada IA está associado um conjunto de endereços internos IO para o Domínio de Ocorrências correspondente ao conjunto de ocorrências do objeto.

A especificação deste domínio permite uma forma alternativa de acesso às ocorrências (não-associativo), pois a partir dele é possível acessá-las sem conhecimento prévio de seus valores.

Entretanto, a sequência definida pelo acesso através deste domínio não necessita seguir necessariamente nenhum tipo de ordenação (como momento de armazenamento ou índices), devendo-se buscar a otimização das formas de acesso suportadas internamente.

c.5.7 - *Domínio de Objetos*

O Domínio de Objetos centraliza a comunicação com todos os demais domínios UniCOSMOS. Além desta função de comunicação, as informações sobre as estruturas de atributos de cada objeto e sobre a relação entre objetos do tipo conjunto e seus elementos também estão armazenadas neste Domínio.

A comunicação entre o Domínio de Objetos e o Domínio de Nomes (e, consequentemente, com os usuários do Sistema) é estabelecida através de uma lista de sinônimos associada ao identificador interno do objeto (ID). Assim, se for necessário por exemplo obter qual a classe (simples, conjunto, relação) correspondente a um objeto cujo ID é conhecido, esta informação é buscada no Domínio de Nomes a partir de um dos sinônimos desta lista.

A comunicação com o Domínio de Tríades é estabelecida através de uma lista de identificadores internos IR associada ao objeto.Esta lista contém cada entrada IR para o Domínio de Tríades definida para objetos que fazem parte de pelo menos uma tríade (como descrito no Item C.5.2).

Existem duas listas associadas a cada identificador ID para representar informações sobre conjuntos. Uma delas mantém as informações sobre quais objetos conjuntos contém o objeto emquestão. A outra descreve, para os

objetos conjuntos, quais são seus elementos. Nos dois casos os objetos são representados através de seus identificadores ID.

Outra informação diz respeito às estruturas de atributos de cada objeto. Esta estrutura foi definida sobre a antiga lista de caracteres associada a cada objeto definida no CORAS [/WU 83/].

Por questões de flexibilidade, foi definido que um objeto pode ter mais de uma estrutura de atributos associada, pois pode haver mais de um tipo de informação associada a cada objeto. Por exemplo, um tipo de informação poderia corresponder às propriedades do objeto, outro às regras associadas ao objeto (formação, consistência) ou outro tipo de informação semântica. Desta forma, uma lista de caracteres contém as diversas estruturas de atributos definidas para um mesmo objeto, sendo que cada estrutura é identificada por um rótulo. O controle destes rótulos (qual o tipo de informação associada a cada estrutura de atributos e como esta informação é utilizada) é exercido pela aplicação suportada por UniCOSMOS.

Para cada estrutura está associado um identificador IA correspondente à comunicação com o Domínio de Acesso, que por sua vez permite a comunicação com o Domínio de Ocorrências (Item C.5.6), permitindo assim agilizar buscas exaustivas (não associativas). Por sua vez, a comunicação com o Domínio de Valores e o Domínio de Índices é definida por indicadores associados a cada um dos atributos de cada estrutura; se o atributo é indexado, este indicador será um II (Item C.5.5); caso contrário, o indicador será um IC (Item C.5.3).

Uma visão geral dos Domínios UniCOSMOS é apresentada na Figura C.2 na continuação.

c.6 - Visão Funcional

O UniCOSMOS provê funções para a definição e manipulação de suas primitivas. O usuário UniCOSMOS não necessita ter acesso à estrutura interna de armazenamento, à definição do protocolo de comunicação entre a memória principal e secundária ou à definição do formato de arquivos em disco. Para atingir estes objetivos, são definidos internamente três níveis de funções

UniCOSMOS:

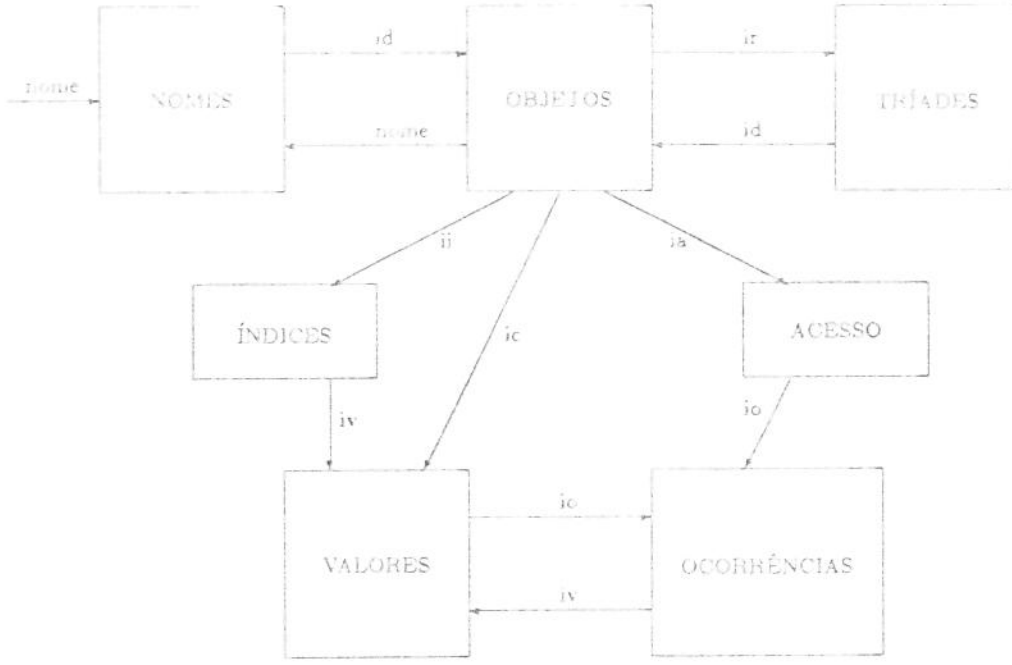


Figura C.2: Domínios UniCOSMOS.

Nível Terciário: contém rotinas disponíveis ao usuário UniCOSMOS (a aplicação), englobando as rotinas de definição de bases de dados ("sistemas de arquivos") e rotinas de definição e manipulação dos elementos primitivos;

Nível Secundário: é o nível de visão lógica dos dados, onde são definidas e manipuladas as estruturas internas de dados referentes aos diversos domínios;

Nível Primário: contém as rotinas que manipulam o nível físico. Estas rotinas definem a forma de acesso aos dados em disco, controlando também o Sistema de Paginação (manutenção em memória principal apenas de dados relevantes, mantendo os demais em memória secundária).

c.6.1 - Nível Primário

As informações referentes à comunicação entre o dispositivo de armazenamento e a memória principal podem ser divididas em duas categorias:

Temporárias: que só têm relevância enquanto o Sistema está ativo;

Permanentes: que devem perdurar entre ativações distintas do Sistema.

As informações permanentes incluem a descrição da alocação do espaço em disco referente aos arquivos armazenados de cada Sistema de Arquivos. Como estas informações devem ser mantidas entre sessões de trabalho distintas, constituem um arquivo de dados de apoio ao Sistema de Arquivos de dados operacionais (dados referentes aos domínios internos de UniCOSMOS) que também deve ser armazenado em disco.

As informações temporárias dizem respeito ao sistema interno de paginação, ou seja, constituem uma descrição sobre quais registros físicos estão ocupando a memória principal e se estes registros foram ou não modificados enquanto permaneceram na memória.

Existem basicamente quatro grupos de rotinas do nível primário:

Rotinas de Acesso a Disco: definem as funções de criação e utilização (abrir/fechar) dos arquivos de dados em disco. Definem o acesso direto a registros tanto para escrita como para leitura. Constituem a interface entre o Sistema Operacional e UniCOSMOS;

Rotinas de Gerência das Informações: manipulam as informações temporárias e permanentes do nível primário. Através destas rotinas, tornam-se transparentes ao Sistema de Paginação a estrutura interna destas informações;

Rotinas do Sistema de Paginação: acessam as rotinas do Sistema de Gerência de Informações e as rotinas de Acesso a Disco (escrita e leitura, acesso direto), implementando os algoritmos de paginação para os arqui vos de dados;

Rotinas de Interface com o Nível Secundário: as rotinas do nível primário tratam os arquivos de dados do sistema através de códigos internos. Como não é interessante que haja a utilização deste tipo de informação interna nos demais níveis, são definidas interfaces que as isolam. Estas rotinas

englobam todas as funções relativas ao Nível Primário.

c.6.2 - Nível Secundário

A cada um dos Domínios UniCOSMOS está associada uma estrutura de dados independente. Os arquivos de dados associados a estas estruturas são denominados Arquivos Internos. Desta forma, existe um Arquivo Interno de Nomes, um Arquivo Interno de Objetos e assim por diante. As funções responsáveis pela manipulação de cada um destes Arquivos Internos constituem as rotinas associadas ao Nível Secundário das funções UniCOSMOS.

Em seu nível mais alto, estas rotinas constituem a interface para as funções do Nível Terciário, que é o responsável pela interconexão das informações distribuídas pelos Arquivos Internos.

Internamente, as funções desempenhadas por estas rotinas dependem da estrutura de dados e das opções de implementação associadas a cada Arquivo Interno. Por exemplo, se as informações em um Arquivo Interno estão armazenadas na forma de listas, então deverão ser providas funções para a manipulação de listas e seus elementos.

As rotinas deste Nível utilizam rotinas do Nível Primário, de forma que alterações processadas no Nível Secundário (correspondente a uma visão lógica dos Arquivos Internos) têm reflexo sobre o nível físico, mantendo atualizadas informações tais como quantidade de páginas utilizadas pelo Arquivo Interno e o espaço disponível em cada página.

c.6.3 - Nível Terciário

As rotinas deste nível constituem a interface externa do UniCOSMOS. As funções desempenhadas por estas rotinas são basicamente:

- tornar transparente ao usuário a utilização das funções de níveis inferiores;
- realizar a comunicação entre os diversos Domínios UniCOSMOS;

- uniformizar o tratamento de códigos de erro.

O primeiro objetivo visa isolar o usuário UniCOSMOS da arquitetura interna do Sistema, de forma que para utilizá-lo não é necessário conhecer, por exemplo, quais arquivos serão acessados durante uma operação.

A comunicação entre domínios é realizada através de chamadas às rotinas de interface do Nível Secundário, de forma que a troca de informações entre domínios ocorre apenas neste nível.

O grupo de rotinas associado a um domínio apresenta, em seu nível mais alto, códigos de erro que são independentes dos códigos definidos para os demais domínios. A tradução destes códigos internos em erros do Sistema também é tarefa das rotinas deste nível.

A especificação das rotinas deste Nível pode ser encontrada em /RICA 86/.

c.7 - *Aspectos de Implementação*

Serão apresentados dois aspectos referentes à implementação dos conceitos até aqui expostos:

- estruturas de dados associadas aos Arquivos Internos; e
- estratégia de alocação e paginação.

c.7.1 - *Estruturas de Dados*

De uma forma geral, as estruturas de dados buscam atingir um objetivo comum, que é a flexibilidade. Para tanto, estas estruturas são definidas a partir de células, estruturas mais elementares associadas a cada Arquivo Interno. Sob o ponto de vista físico, células estão agrupadas em conjuntos de quantidades pré-fixadas que constituem as páginas dos arquivos. Páginas constituem a unidade de acesso a disco, ou seja, estão associadas aos registros físicos.

c.7.2 - Estratégias de Alocação e Paginação

A alocação da memória principal é pré-estabelecida em UniCOSMOS, sendo que existem áreas reservadas a cada um dos arquivos internos. A quantidade de páginas que cada área pode conter é definida na configuração do Sistema, assim como a dimensão de cada página. A designação da área em memória principal que pode conter uma página de um arquivo interno é um "frame"; o conjunto de "frames" associado a cada arquivo interno constitui um "pool".

A cada arquivo interno está associado um arquivo armazenado em disco. A alocação do espaço em disco era limitada no Sistema CORAS-UNICAMP, sendo definida no momento da iniciação do sistema de arquivos, podendo ser expandida durante sua operação. Havia nesse Sistema um limite superior de quantidade de páginas que poderiam ser manipuladas. No UniCOSMOS, tal limitação de gerência não existe, sendo que também não é necessário especificar a dimensão inicial dos arquivos internos. Esta alocação é dinâmica, ocorrendo à medida que o Sistema o exige.

Com relação ao Sistema de ocupação das páginas, as estratégias são diferenciadas de acordo com os diversos arquivos internos. Por exemplo, no Arquivo de Nomes a ocupação é definida através de "hashing", sendo que a alocação das páginas de colisão é definida sequencialmente. Já para o Arquivo de Objetos, a estratégia é buscar a primeira página que tenha espaço suficiente para a execução da operação especificada. Para o Arquivo de Tríades a estratégia é semelhante, com a diferença de que se busca reservar uma página para cada objeto que componha uma tríade, enquanto for possível. Para os demais Arquivos, buscou-se uma estratégia que permitisse uma modularização física dos dados, através da reserva de páginas diferentes para informações associadas a objetos distintos. Assim, uma página do Arquivo de Ocorrências contém informações sobre as ocorrências de um único objeto, e no Arquivo de Valores cada página está associada basicamente a um atributo distinto.

O Sistema de Paginação, que permite a troca de registros entre o disco e a memória principal, é controlado por rotinas do Nível Primário do UniCOSMOS. A estratégia adotada busca inicialmente ocupar "frames" livres quando se requisita a leitura de uma página para a memória principal. Caso

não haja "frame" disponível, desocupar-se-á aquele que contenha a página que tenha sofrido menos ativações (operações de armazenamento, remoção ou leitura) durante o período de ocupação no "frame". Se a página em questão não sofreu alterações, é simplesmente descartada; caso contrário, o arquivo armazenado é atualizado.

c.8 - Dicionário/Diretório UniCOSMOS

Uma das características do UniCOSMOS é permitir a manipulação simultânea de mais de um Sistema de Arquivos (base de dados), forma adotada para permitir a integração entre dados de distintas aplicações. Um dos Sistemas de Arquivos, que permanecerá ativo durante toda uma sessão de trabalho UniCOSMOS (período entre sua ativação e desativação), corresponde ao Dicionário/Diretório UniCOSMOS (DDU).

As funções do DDU são basicamente duas:

- controle de Sistemas de Arquivos existentes e suas versões, e
- registro de todos os tipos de dados disponíveis aos usuários UniCOSMOS.

Estas tarefas são suportadas através de funções UniCOSMOS e as informações são representadas em termos de seus elementos primitivos.

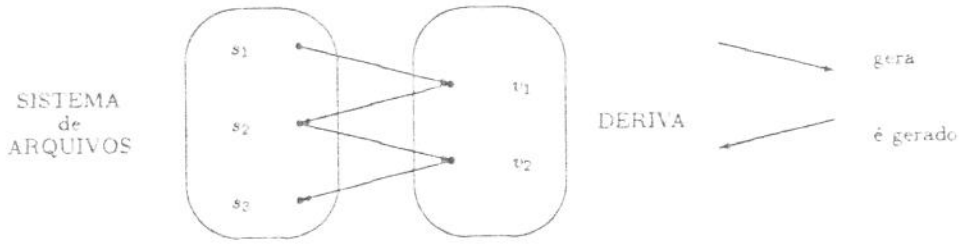


Figura C.3: Representação de Sistemas de Arquivos e Versões.

O DDU reúne sob seu controle um conjunto de Sistemas de Arquivos. A cada

um destes Sistemas podem estar associados diversas versões, cada uma constituindo uma base de dados distinta. A geração de uma versão a partir de outra é uma operação que deve estar registrada, pois algumas vezes uma modificação em alguma versão pode exigir, segundo o controle da aplicação, medidas em relação às versões subordinadas, sendo importante manter esta informação (Figura C.3).

Em termos de primitivas UniCOSMOS, cada Sistema de Arquivos é representado por um objeto simples, agrupado sob um objeto conjunto que congrega todos os Sistemas de Arquivos definidos para o DDU. Dois objetos relação são utilizados para controlar as associações existentes entre duas versões e a operação que gerou a versão subordinada; o primeiro objeto relação especifica qual base de dados desempenha o papel de "geradora" na operação, e o segundo especifica o papel de base de dados "gerada" (Figura C.4). Note-se que como uma triade não pode ser especificada sobre um único objeto (auto-associação), foi necessário acrescentar um outro objeto simples que indica apenas as ocorrências de operações que levam à geração de novas versões.

Os atributos associados a estes objetos definem as propriedades de cada base de dados, como "data de criação", "data de última atualização", "código de acesso", "espaço ocupado", "consistência" etc.

Tipos de Dados constituem uma classe, representada por um objeto simples. Os atributos dos objetos especificam a forma de conversão do tipo básico UniCOSMOS (representação em caracteres) para o tipo desejado, o que pode ser realizado através de um subsistema integrado a UniCOSMOS. Outros atributos podem especificar, por exemplo, operações permitidas sobre cada tipo de dado.

É necessário, como no caso de versões de Sistemas de Arquivos, especificar as associações entre tipos de dados como forma de documentar dependências entre suas definições. Por exemplo, um tipo de dado "Ponto 2D" poderia ser especificado a partir do tipo de dado "Real", que por sua vez pode ter sido especificado a partir do tipo básico "Caracter". Se houver modificações na especificação do tipo de dado "Real", pode ser necessário modificar também os tipos de dados subordinados, entre os quais estaria

"Ponto 2D".

Figura C.4: Representação do Diretório em termos SIGA.

c.9 - Comentários

O UNICAMP Object Storage Management O System - UniCOSMOS - constitui uma alternativa a outros sistemas associativos que manipulam objetos e conjuntos, como é o caso de CORAS. Apesar do alto nível de abstração apresentado por tais sistemas, o desempenho em geral deixa a desejar. Particularmente em relação ao CORAS, pode-se citar:

acesso: o UniCOSMOS distingue o acesso a objetos do acesso a valores. No CORAS, um valor deveria ser definido como um objeto para permitir sua representação e consequentemente seria acessado da mesma forma que os objetos, através de "hashing". No UniCOSMOS, valores independem de "hashing". Em termos de implementação, a quantidade de valores que pode ser acessada não tem limitação lógica, como ocorre com algoritmos de "hashing";

representação interna de valores: a vantagem com relação à implementação do CORAS se encontra no fato de que não há limitação lógica para a

dimensão da lista que contém a representação de um valor (no CORAS, sendo cada valor representado como um objeto havia a limitação correspondente à dimensão de um nome, que constitui a entrada para a Tabela Hash), sendo que em comum se manteve a representação na forma de caracteres. A restrição permanece para nomes de objetos;

velocidade: o fato de não se utilizar tríades para representar associações do tipo atributo (como era necessário em CORAS para representar a associação entre um objeto-instância e um objeto-valor através de um objeto-atributo) alivia o volume das informações relacionadas a este domínio. As associações armazenadas no Domínio de Tríades representam apenas associações do tipo elacionamento entre classes;

interpretação: ao pré-especificar algumas abstrações (o conceito de atributos representa uma agregação dos conjuntos correspondentes aos conjuntos de valores possíveis) permite-se agilizar o tratamento computacional. No entanto, é exigido que haja um conhecimento sobre os dados armazenados, o que não era exigido no CORAS, onde valores, conjuntos e atributos poderiam ser acessados da mesma forma (eram todos acessados através do nome dos objetos correspondentes). No UniCOSMOS, o acesso a um valor é definido através do acesso a um atributo que, por sua vez, é acessado através do conhecimento do objeto a que pertence.

Outra característica do UniCOSMOS é a existência do Dicionário e Diretório, uma base de dados manipulada paralelamente à base de dados da aplicação. Este princípio de manipulação simultânea pode ser expandido de duas para diversas bases de dados, onde várias aplicações podem ser interconectadas através do DDU, que controlaria também a alocação de áreas em memória para as bases de dados.

Os tipos de aplicações que podem utilizar o UniCOSMOS são diversos. 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. Por outro lado, é um sistema computacionalmente viável, principalmente levando em conta o desenvolvimento

de processadores dedicados (manipulação de listas, funções de acesso a dados, máquinas de inferências) e sistemas multiprocessadores.

Dentre as aplicações possíveis para o UniCOSMOS, uma foi analisada em maior detalhe [/RICA 86/]. Trata-se de sua utilização como núcleo de Sistemas de Gerência de Banco de Dados, em particular para o Sistema Gerenciador de Dados para PAC, o GERPAC. A implementação de um SGBD para PAC utilizando internamente um sistema voltado para a manipulação de objetos constitui uma alternativa em relação a outras propostas, que em geral partem de um SGBD baseado em modelos de dados convencionais, utilizados em aplicações comerciais, com camadas externas que visam adequá-los a estas aplicações. A especificação do UniCOSMOS o torna muito adequado a representações de dados através da abordagem entidade-relacionamento, permitindo que SGBD que sejam suportados por ele possam suportar diretamente modelos como o MER e o MER/PAC, que é o caso de GERPAC.

A integração do Sistema de Transações com o UniCOSMOS está discutida ao longo do capítulo 3 e no anexo b.

c.10 - Referências

/COST 84/ COSTA MARTINS M.A.
"Concepção duma Base de Dados"
Editora Rés (Portugal) - 1984

/ENCA 83/ ENCARNAÇÃO J. e SCHLECHTENDAHL E.G.
"Computer Aided Design - Fundamentals and System Architectures"
Springer-Verlag - 1983

/RICA 86/ RICARTE I.L.M.
"Definição e Implementação de um Modelo de Dados para o Núcleo CORAS-UNICAMP"
Relatório de Atividades 3, FAPESP Proc.84/2591-1, Ago.1986

/WU 83/ WU S.T.
"Implementação de um Núcleo de Banco de Dados baseado na Filosofia de CODAS"
Documentação Preliminar, DCA/FEE/UNICAMP - 1983

ANEXO D

Exemplo de Aplicação

ANEXO D - EXEMPLO DE APLICAÇÃO

Conteúdo

d.1 - Introdução

Com o objetivo de mostrar a aplicabilidade do MER/PAC para a estruturação de dados PAC, será apresentada neste anexo a modelagem de um problema de projeto a ser ambientado em um sistema PAC. Esta aplicação se refere ao projeto do "lay-out" de placas de circuito impresso /DELG 87/.

Uma placa de circuito impresso é um dos vários tipos de implementação física de um circuito eletro-eletrônico. Consta de uma placa de material isolante (fibra de vidro ou equivalente) sendo que em cada face da placa é colocada uma película de metal (geralmente cobre), formando assim uma placa rígida metalizada.

A partir de um gabarito denominado "lay-out" do circuito impresso e que é desenvolvido considerando-se alguma representação genérica do circuito eletro-eletrônico (esquema elétrico ou tabelas de ligações e de componentes), são gerados dados de projeto e de fabricação. Utilizando estes dados e através de processos químicos e industriais específicos, a película de metal é modificada de forma que o produto final se torne uma placa com trilhas de

metal que conectam furos e/ou áreas metalizadas. Nesta placa, os furos e áreas metalizadas correspondem, aos pontos de conexão elétrica dos componentes eletro-eletrônicos, constantes da representação do circuito elétrico. Da mesma forma, as trilhas de metal correspondem às ligações existentes entre os pontos de conexão destes componentes.

Além desta composição básica, uma placa de circuito impresso conta também com alguns elementos auxiliares. Entre outros, tem-se os elementos de representação gráfica dos componentes eletro-eletrônicos e elementos de referência para a gerência do processo de projeto e de fabricação da placa de circuito impresso.

Os elementos de representação gráfica têm o objetivo de estabelecer uma correspondência biunívoca entre os componentes físicos, que serão eventualmente instalados na placa e os componentes correspondentes da representação do circuito elétrico (esquema elétrico ou tabela de ligações). Estes elementos gráficos são geralmente "impressos" sobre a placa através de processos específicos, e são subdivididos em:

- *elementos gráficos*, representações simbólicas das diversas classes de componentes eletro-eletrônicos existentes;
- *elementos de texto*, identificam o componente físico e estabelecem a correspondência com os componentes da representação do circuito elétrico.

O objetivo do sistema-exemplo é o suporte ao projeto de "lay-out" de placas de circuito impresso, a partir de alguma representação do circuito elétrico. Como resultado final o sistema produz toda a informação referente ao "lay-out", necessária ao processo de fabricação e a gerência do projeto.

A modelagem do sistema-exemplo será feita da seguinte forma:

- *Modelagem conceitual:* será feita uma descrição geral do problema "projeto de lay-out de um circuito impresso", para em seguida ser feita uma descrição textual detalhada dos objetos e relacionamentos envolvidos no problema;
- *Desenvolvimento do modelo descritivo:* utilizando o MER/PAC, será desenvolvido o esquema correspondente aos dados de projeto, a partir do modelo conceitual desenvolvido na fase anterior.

d.2 - Projeto de circuito impresso - Modelagem conceitual

d.2.1 - Descrição geral

De modo geral, uma placa de circuito impresso é composta por uma placa com determinadas características mecânicas e que possui uma área útil, dentro da qual é permitido posicionar componentes e traçar rotas.

O componente é um conjunto de elementos, eletrônicos ou não eletrônicos, que é identificado de maneira única dentro da placa. Assim, diz-se que um componente é composto por vários componentes, sendo a placa também uma classe de componente.

Um componente contém pontos de acesso elétrico, os pinos, além de uma área de ocupação, geometria e furos de componente. A geometria é uma representação gráfica do componente. Os furos de componente representam os diversos tipos de furos que podem ocorrer em um componente. Correspondem fisicamente aos pontos de interconexão entre os elementos físicos, e/ou aos pontos usados para fixação de um elemento físico na placa.

Os pinos dos componentes são interligados através de rotas, de acordo com uma lista de sinais. As rotas representam as trilhas de cobre que farão parte da placa, após a fabricação. Também fazem parte da placa os furos de sinal, também chamados furos de rota.

Uma placa de circuito impresso é constituída fisicamente por diversos níveis "layers" ou camadas, sendo o número destes níveis sempre maior ou igual a 2. O conceito de níveis de uma placa é uma extensão à idéia de "faces de uma moeda", podendo ser dito informalmente que uma placa possui várias "faces". Como há a possibilidade de que uma determinada rota não possa ser localizada totalmente em um único nível, define-se o furo de sinal, que representa o ponto no qual uma rota continua em outro nível da placa.

Neste exemplo não consideraremos o conceito de níveis mas sim os conceitos de sinal e ligação. O sinal representa o relacionamento entre pinos dos diversos componentes de uma placa, sendo identificado por um nome único dado pelo projetista. A ligação é um elemento abstrato que define a conexão entre dois pinos, sendo a rota a realização física de uma ligação.

d.2.2 - Modelagem conceitual

Serão especificados os objetos da estrutura de dados, os relacionamentos entre estes objetos e restrições válidas nestes relacionamentos. Esta especificação, feita de maneira textual, constituirá o modelo conceitual do projeto, sendo que a descrição formal, o esquema descritivo, será feita nas próximas seções, a nível de modelo de dados.

d.2.2.1 - Grupo A - Regras referentes a componentes

1. Um componente deve ter uma área de ocupação associada.
2. Um componente deve ter um furo de componente associado.
3. Um componente pode ser formado por vários componentes.
4. Um componente deve ter uma geometria e um texto de descrição.
5. Um componente que não tem componente pai é classificado como placa.
6. Só existe 1 (uma) placa.
7. As posições dos furos de componente e áreas de ocupação, em relação ao componente a que estão associados, não podem ser modificadas.
8. Máscaras de ocupação não podem se sobrepor.
9. Máscaras de ocupação devem sobrepor-se à área útil da placa.

d.2.2.2 - Grupo B - Regras referentes a rotas

1. Uma rota é definida por uma sequência ordenada de pontos de rota, classificados em:
    1. ponto de geometria;
    2. ponto de furo.
2. Uma rota pode pertencer a uma das seguintes classes:
    1. rota de componente não posicionado;
    2. rota de componente posicionado.
3. Um furo de sinal pode pertencer a uma das seguintes classes:
    1. furo de componente não posicionado;

2. furo de componente posicionado.

4. Para criar um furo de sinal, deve-se associá-lo a um sinal.
5. Por outro lado, o furo de sinal deve ser associado também a uma rota.
6. Uma rota pode ter vários furos de sinal. Um furo de sinal pode ser associado a várias rotas, relativas a um mesmo sinal.
7. Rotas pertencentes a sinais diferentes não podem se sobrepor (leva-se em conta a espessura da rota).

d.2.2.3 - *Grupo C - Regras referentes a componentes, rotas, ligações e sinais*

1. Somente é possível conectar componentes (pinos) através de ligações e rotas.
2. Uma lista de sinais estabelece uma associação 1:n entre sinal (1) e pinos (n).
3. Para criar uma rota, deve-se associá-la a um sinal.
4. Uma rota pode ser criada somente a partir de uma ligação.
5. Para criar uma ligação entre 2 (dois) pinos, estes devem pertencer ao mesmo sinal e os componentes correspondentes aos pinos devem estar posicionados na placa.
6. Para criar uma ligação entre um pino e uma rota, estes elementos devem pertencer ao mesmo sinal e o componente correspondente ao pino deve estar posicionado na placa.
7. Seja uma rota R, tal que os pinos V1 e V2 sejam seus pontos de rota (i.e., a rota está resolvida). Então, se o componente correspondente a V1 é retirado da placa tem-se:
    1. elimina-se a rota R;
    2. segue-se a regra C.8.
8. A eliminação de uma rota R implica na eliminação das ligações e furos de rota associados a R e na eliminação de todos os pontos de rota de R.
9. Seja uma ligação L1 que relaciona dois pinos V1 e V2. Então, se for criada uma rota R associada a L1 e com origem em V1, tem-se:
    1. cria-se uma ligação L2 entre V2 e R;

2. o pino V1 se torna ponto de rota de R;
3. o ponto de ligação L2 se torna ponto de rota de R;
4. elimina-se a ligação L1.

10. Seja uma ligação L entre um pino V1 e uma rota R. Então, se a posição do ponto de ligação é igual à posição do pino, a ligação L deve ser eliminada.

d.2.2.4 - Estruturas de dados e regras de manipulação

Nas regras citadas anteriormente, deve-se notar que algumas não podem ser vistas como especificações da estrutura de dados, mas sim como regras de manipulação dos dados. Por regras de manipulação entende-se aquelas que dizem respeito a instâncias específicas dos objetos descritos. Tais regras envolvem as ocorrências (valores) dos objetos descritos e não o relacionamento estrutural entre eles. As regras que dizem respeito à manipulação são, a princípio: A.8, A.9, B.7, C.4, C.7 e C.9.

Uma vez estabelecidas as regras de estrutura de dados e as regras de manipulação, pode-se agora estabelecer o modelo descritivo através do MER/PAC, o que será feito na próxima seção.

d.3 - Modelo descritivo - MER/PAC

A partir do modelo conceitual estabelecido na seção anterior, serão utilizadas as primitivas MER/PAC (descritas em /MAGA 89/) para estabelecer o modelo descritivo ou esquema do projeto de "lay-out" de um circuito impresso. As figuras d.1 e d.3 apresentam os diagramas MER/PAC correspondentes a este esquema.

Na figura d.1, procura-se modelar um componente como uma entidade complexa, que é constituído pelos tipos de entidades fracas TEXTO, FURO e MASC_OCUP (máscara de ocupação). Isto quer dizer que, ao se criar um componente, devem ser criadas entidades texto, furo e máscara de ocupação. Por outro lado, como um componente não precisa ter necessariamente pinos, o tipo de entidade associado é fraco mas não subordinado à entidade

complexa. Assim, a entidade complexa é modelada através do agrupamento COMP_COMPLEX, que possui regras de formação que seguem a especificação citada anteriormente.

O relacionamento COMP_FILHO modela diretamente a regra A.3 do modelo conceitual.

A figura d.2 apresenta a parte do esquema correspondente às regras de definição de rotas (seção d.2.2.2). Note-se que o tipo de entidade FURO_SINAL é fraco em relação aos tipos de entidades SINAL e ROTA, sendo que tais dependências modelam as regras B.4 e B.5.

A figura d.3 refere-se às regras que envolvem a interação entre componentes, rotas, ligações e sinais (seção d.2.2.3). As ligações são modeladas através de relacionamentos que envolvem os agrupamentos R_ROTA (figura d.2) e P_PINO (figura d.3). Desta forma, as ligações envolvem apenas rotas e pinos que possuem sinais associados. As regras de integridade se encarregam de verificar se os elementos a serem envolvidos em uma ligação possuem o mesmo sinal.

Para ilustrarmos de que maneira o sistema de transações pode ser utilizado apresentaremos em seguida uma situação hipotética de manipulação do exemplo apresentado.

d.4 - Exemplo de utilização do sistema de transações

De forma a ilustrar de que maneira o sistema de transações pode ser utilizado apresenta-se aqui uma possível situação de execução envolvendo os três tipos de transação definidos.

Uma vez que o projetista realiza a modelagem conceitual do objeto que está projetando e faz o mapeamento do modelo conceitual para o modelo descritivo (i.e., em termos de primitivas MER/PAC) como visto em d.2 e d.3, estamos em condições de utilizando o sistema de gerência de banco de dados GERPAC/UniCOSMOS, armazenar as primitivas GERPAC, associadas a cada uma das primitivas MER/PAC, que descrevem o objeto de projeto em termos de elementos componentes do objeto de projeto e das associações que vinculam cada um destes

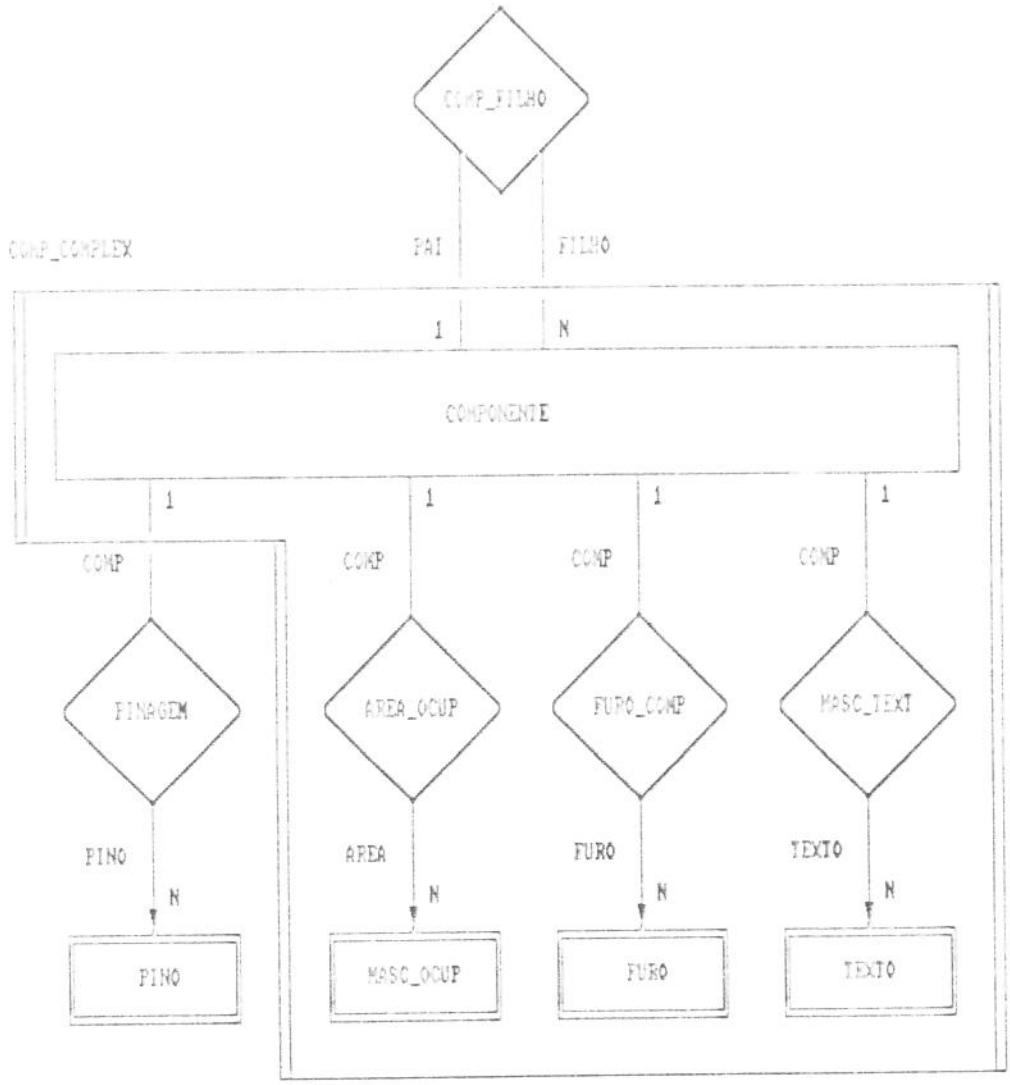


FIGURA D.1. Diagrama MER/PAC - Esquema de Projeto - Grupo A

elementos com outros elementos componentes do objeto de projeto.

Como visto no capítulo 3 desta monografia, é o usuário quem estabelece os pontos de ocorrência dos comandos de controle que delimitam a transação que se está definindo. Isto é, ele define onde ocorrem os comandos de controle *begin_transaction*, *commit_transaction*, *rollback_transaction*, etc.

Esta característica permite que o usuário tenha total liberdade para escolher aquelas ações que devem ser protegidas de uma possível falha do

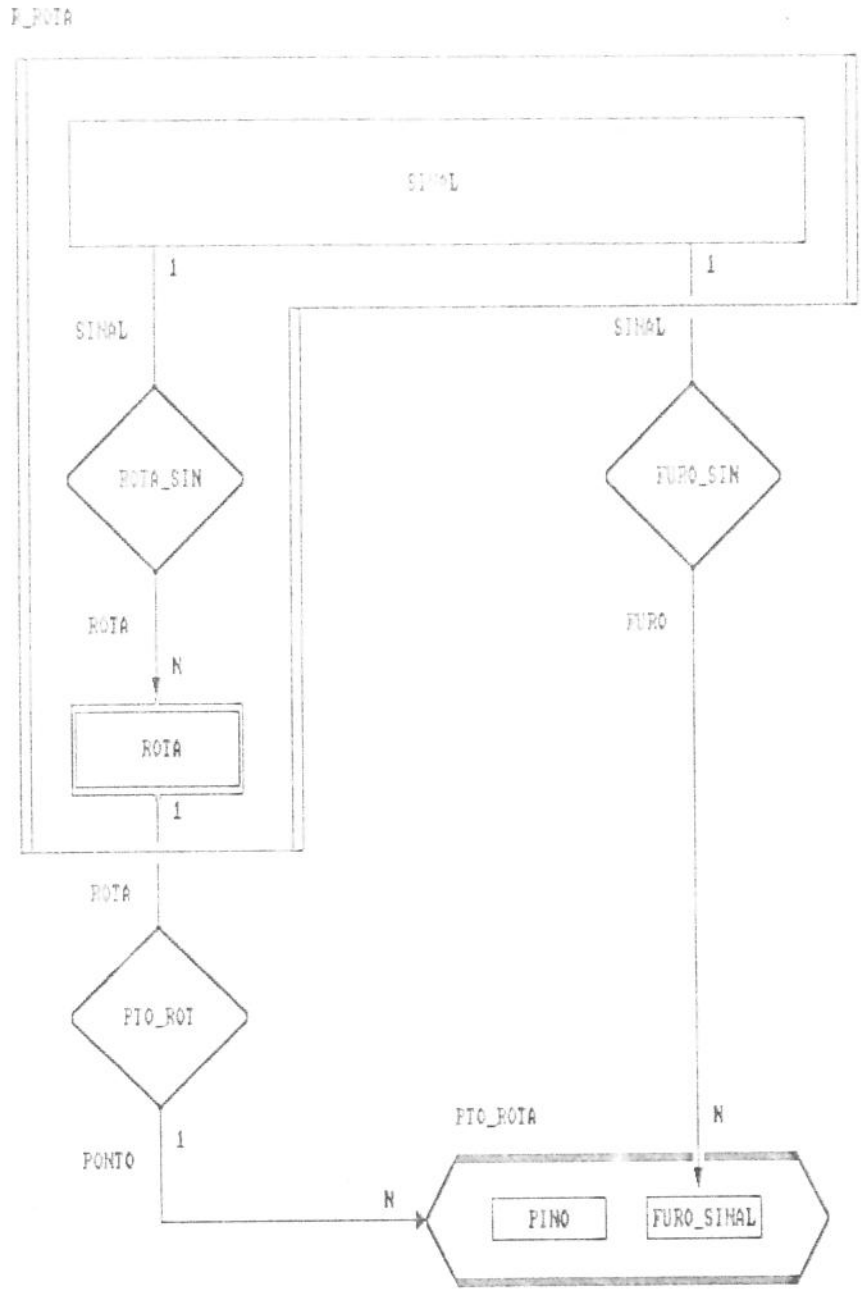


FIGURA D.2. Ilustração "SINAL" - Esquema de projeto - Grupo B

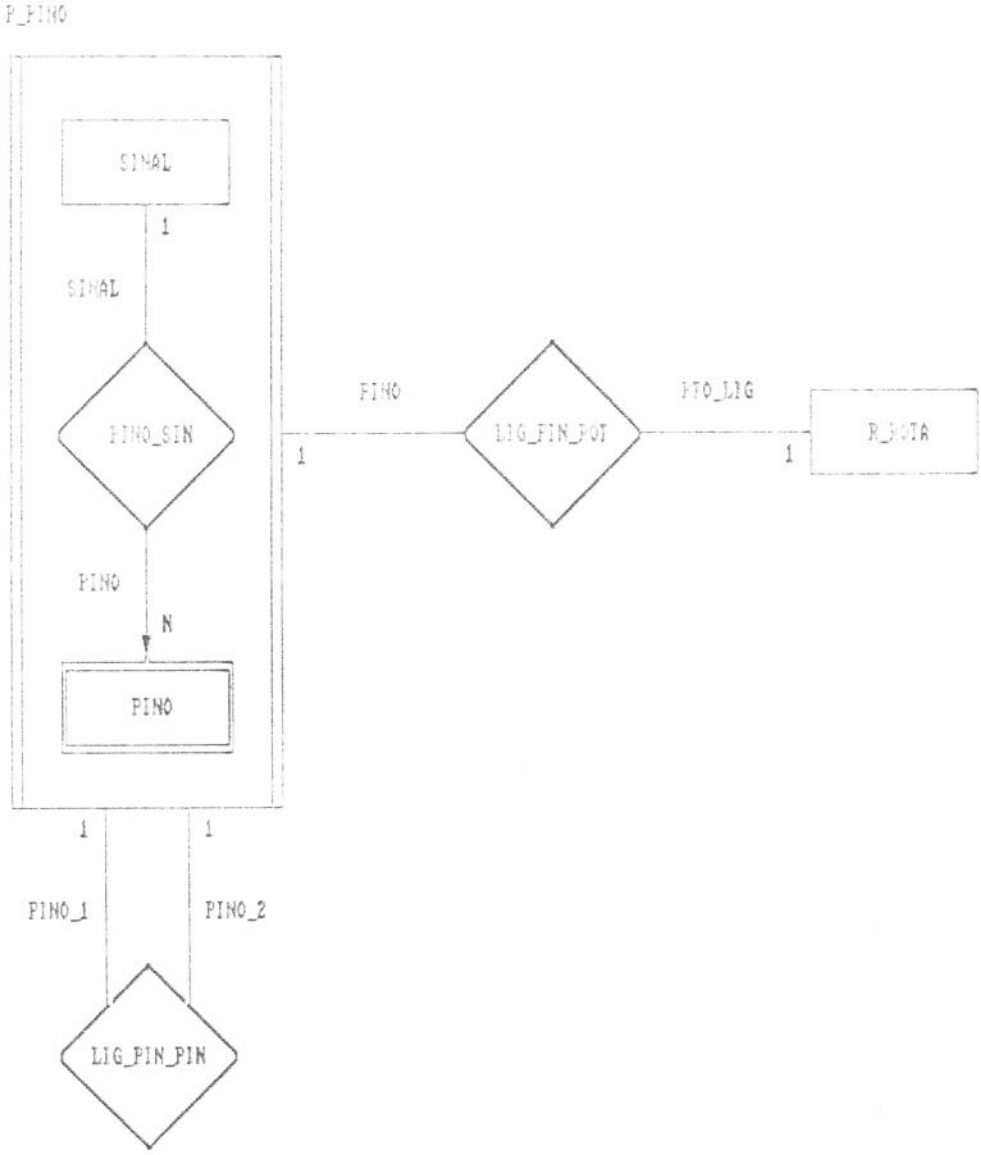


FIGURA D.3. Diagrama MER/PAC - Esquema de projeto - Grupo C

sistema.

Como mencionado anteriormente, no sistema-exemplo do projeto de circuito impresso foram definidos três grupos de regras diferentes (associados aos diversos elementos que conformam o circuito impresso), a saber:

- referentes a componentes,
- referentes a rotas e
- referentes a componentes, rotas, ligações e sinais,

onde algumas destas regras se referem à especificação da estrutura de dados e outras à manipulação dos dados.

A partir desta diferenciação podemos, por exemplo, relacionar cada tipo de transação considerada nesta proposta (i.e., transação micro, meta-micro e macro) com a atividade de instanciação de uma estrutura de dados pré-definida. Considerado o exemplo do circuito impresso podemos ter as seguintes associações:

transação micro	⟶	instanciação de cada um dos objetos simples (sinal, pino_sin, pino, etc. da figura d.3)
transação meta-micro	⟶	instanciação de objetos complexos (p_pino da figura d.3, r_rota da figura d.2, etc.)
transação macro	⟶	verificação da consistência através de regras

Figura D.4 - Correspondência transação/atividade

Deve-se observar que estes conceitos aplicar-se-iam igualmente à fase de construção da base de dados, visto que em uma atividade de projeto, a instanciação ocorre tanto a nível de estruturação dos dados como a nível de manipulação de valores.

Sugere-se, que toda vez que se esteja trabalhando, por exemplo na instanciação da estrutura de dados do objeto considerado, as ações envolvidas em tal atividade sejam agrupadas em uma transação meta-micro já que dita atividade pré-supõe uma duração longa.

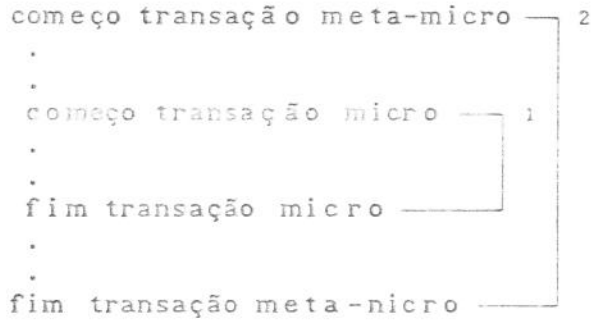
Por outro lado, quando se está trabalhando com uma atividade que requer manipular dados que dizem respeito a instâncias específicas dos objetos descritos, as ações envolvidas consideradas críticas, devem ser agrupadas em uma transação micro para, desta maneira, conseguir recuperar o último estado consistente na presença de uma falha.

Finalmente, uma transação macro encontra-se associada à manipulação dos dados no que diz respeito às regras que definem as associações entre objetos.

Exemplo

No nosso sistema-exemplo podemos considerar que a definição de um circuito impresso é uma tarefa de longa duração, sendo portanto aconselhável considerar as ações de definição do circuito pertencentes a uma transação meta-micro.

Se considerarmos críticos cada um dos elementos que conformam um circuito impresso (i.e., rotas, componentes, ligações, sinais, etc.), podemos pensar em associar uma transação micro a cada um deles, obtendo uma estrutura do tipo mostrado a seguir.



1- componentes, rotas, ligações, etc.
2- circuito impresso

Figura D.5 - Estrutura de uma transação

Temos apresentado desta maneira, um exemplo que tenta esclarecer os conceitos e idéias que foram utilizados na concepção do sistema de gerência de transações do PERFAC/CNUCE/CNR/IOS.

d.5 - Referências

/DELG 87/ DELGADO A.L.N.
"Bancos de dados no contexto de projeto auxiliado por computador"
Tese de mestrado DCA-FEE-UNICAMP - Jan 87

/MAGA 89/ MAGALHÃES L.P., DELGADO A.L.N., RICARTE I.L., RUSCHEL R.C. e OLGUIN C.J.M.
"Implementação de um Banco de Dados Não Convencional: A experiência GERPAC/UniCOSMOS"
Anais do IV Simpósio Brasileiro de Banco de Dados (SBBD) - Abr.1989 e
Anais das 18 Jornadas Argentinas de Informática e Investigación Operativa (18 JAIIO),
Buenos Aires, Argentina - Ago.1989

ANEXO E

Referências unificadas e extendidas

/BANC 85/ BANCILHON F., KIM W. e KORTH H.F.
"A Model of CAD Transactions"
Proceedings of International Conference on Very Large Data Bases - 1985

/BARO 82/ BARON N. et alii
"An Approach to the Integration of Geometrical Capabilities into a Database for CAD Applications"
em /ENCA 82/

/BERN 81/ BERNSTEIN P. A. e GOODMAN N.
"Concurrency control in Distributed Database Systems"
ACM Computing Surveys, Vol.13 Nro.2 - 1981

/BERN 87/ BERNSTEIN P. A., HADZILACOS V. e GOODMAN N.
"Concurrency control and recovery in database systems"
Addison-Wesley Co. - 1987

/BJOR 73/ BJORK L.A.
"Recovery scenario for a DB/DC system"
Proc.of the ACM 73 National Conference - Ago.73

/BORG 85/ BORGES M.R.
"Towards a Flexible Mechanism for Concurrency Control in Database Systems"
Proc. 4th. British National Conf. on Databases
Julho 1985

/CERI 84/ CERI S.
"Distributed databases, principles and systems"
Mc.Graw-Hill Co. - 1984

/CHRI 83/ CHRISTIAN K.
"The UNIX Operating System"
John Willey & Sons - 1983

/COST 84/ COSTA MARTINS M.A.
"Concepção duma Base de Dados"
Editora Rés (Portugal) - 1984

/DATE 83/ DATE C.J.
"An Introduction to Database Systems - II"
Addison-Wesley - 1983

/DAVI 73/ DAVIES C.T.
"Recovery semantics for a DB/DC system"
Proc.of the ACM 73 National Conference - Ago.73

/DELG 86/ DELGADO A.L.N., MAGALHÃES L.P. e RICARTE I.L.
"Sistemas de Gerenciamento de Banco de Dados para P.A.C."
Anais do I Simpósio Brasileiro de Banco de Dados (SBBD) - Abr.1986

/DELG 87/ DELGADO A.L.N.
"Bancos de dados no contexto de projeto auxiliado por computador"
Tese de mestrado DCA-FEE-UNICAMP - Jan 87

/DELG 88/ DELGADO A.L.N., MAGALHÃES L.P. e RICARTE I.L.
"Supporting design environments through the Entity-Relationship Model"
12th IMACS World Conference on Scientific Computation - Paris, France - Jul.1988

/EAST 81/ EASTMAN C.M.
"Database Facilities for Engineering Design"
Proceedings of the IEEE - 69 (10) - 1981

/ENCA 80/ ENCARNAÇÃO J.
"Computer Aided Design Modelling, Systems Engineering, CAD Systems"
Lecture Notes in Computer Sciences, 89
Springer-Verlag - 1980

/ENCA 82/ ENCARNAÇÃO J. e KRAUSE F. (Eds.)
"File structures and Data Bases for CAD"
North-Holland - 1982

/ENCA 83/ ENCARNAÇÃO J. e SCHLECHTENDAHL E.G.
"Computer Aided Design - Fundamentals and System Architectures"
Springer-Verlag - 1983

/ESWA 76/ ESWARAN K.P., GRAY J.N., LORIE R.A. e TRAIGER I.L.
"The notions of consistency and predicate locks in a database system"
Communications of the ACM - Nov.76

/FIPE 88/ *"Sistemas de Banco de Dados no Contexto de Projeto Auxiliado por Computador"*
Relatório Final - FIPEC, processo 1.1731-0
LEA/DCA/FEE/UNICAMP - Janeiro 1988

/FOIS 82/ FOISSEAU J. e VALETTE F.R.
"A Computer Aided Design Data Model: FLOREAL"
em /ENCA 82/

/GARZ 88/ GARZA J.F. e KIM W.
"Transaction Management in an Object-Oriented Database System"
ACM SIGMOD - 1988

/GRAY 78/ GRAY J.N.
"Notes on database operating systems"
Lecture notes in computer sciences, Vol.60
Springer-Verlag - 1978

/GRAY 80/ GRAY J.N.
"A transaction model"
Lecture notes in computer sciences, Vol.85
Springer-Verlag - 1980

/GRAY 81/ GRAY J.N.
"The Transaction Concept: Virtues and Limitations"
Proceedings of the International Conference on Very Large Data Bases - 1981

/GRAY 81a/ GRAY J.N., McJONES P., BLASGEN M., LINDSAY B., LORIE R.A., PRICE T., PUTZOLU F. e TRAIGER I.
"The Recovery Manager of the System R Database Manager"
ACM Computing Surveys, Vol.13 Nro.2 - 1981

/HAER 83/ HAERDER T. e REUTER A.
"Principles of transaction-oriented database recovery"
ACM Computing Surveys - Dez.1983

/HAER 87/ HAERDER T. e ROTHERMEL K.
"Concepts for Transaction Recovery in Nested Transactions"
ACM SIGMOD - Maio 1987

/HAER 87a/ HAERDER T., MEYER-WEGENER K., MITSCHANG B. e SIKELER A.
"PRIMA: A DBMS Prototype Supporting Engineering Applications"
Proceedings of International Conference on Very Large Data Bases - 1987

/HAER 88/ HAERDER T., HUBEL C., MEYER-WEGENER K. e MITSCHANG B.
"Processing and Transaction Concepts for Cooperation of Engineering Workstations and a Database Server"
Data & Knowledge Engineering, North-Holland Nro.3 1988

/HASK 81/ HASKIN R.L. e LORIE R.A.
"On Extending the Functions of a Relational Database System"
IBM Research Report RJ3182 - 1981

/JOCH 89/ JOCHPE C.
"Database Recovery in the Design Environment: Requirements Analysis and Performance Evaluation"
Ph.D. dissertation, Fakultat für Informatik, Universitat Karlsruhe - 1989

/KATZ 83/ KATZ R.H.
"Managing the Chip Design Database"
ACM SIGMOD/IEEE - 83

/KATZ 84/ KATZ R.H. e WEISS S.
"Design Transaction Management"
Proc. 19th. Design Automation Conference - 1984

/KETA 87/ KETABCHI M.A. e BERZINS V.
"Modelling and Managing CAD Databases"
IEEE Software - Fev.1987

/KIM 84/ KIM W., LORIE R., McNAMB D. e PLOUFFE W.
"A Transaction Mechanism for Engineering Design Databases"
Proceedings of International Conference on Very Large Data Bases - 1984

/KLAH 85/ KLAHOLD P., SCHLAGETER G., UNLAND R e WILKES W.
"A Transaction Model Supporting Complex Applications in Integrated Information Systems"
ACM SIGMOD - 1985

/KORT 88/ KORTH H.F., KIM W. e BANCILHON F.
"On Long-Duration CAD Transactions"
Information Sciences - An International Journal, Vol.46, Nros.1/2; North-Holland - Out./Nov.1988

/LORI 82/ LORIE R.A.
"Issues in Database for Design Applications"
em /ENCA 82/

/LORI 83/ LORIE R.A. e PLOUFFE W.
"Complex Objects and Their Use in Design Transactions"
ACM SIGMOD/IEEE - 1983

/MAGA 89/ MAGALHÃES L.P., DELGADO A.L.N., RICARTE I.L., RUSCHEL R.C. e OLGUIN C.J.M.
"Implementação de um Banco de Dados Não Convencional: A experiência GERPAC/UniCOSMOS"
Anais do IV Simpósio Brasileiro de Banco de Dados (SBBD) - Abr.1989 e
Anais das 18 Jornadas Argentinas de Informática e Investigación Operativa (18 JAIIO),
Buenos Aires, Argentina - Ago.1989

/McLE 83/ McLEOD D., NARAYANASWAMY K. e BAPA RAO K.V.
"An Approach to Information Management for CAD/VLSI Applications"
ACM SIGMOD/IEEE - 1983

/MOHA 86/ MOHAN C., LINDSAY B. e OBERMARCK R.
"Transaction Management in R* Distributed DBMS"
ACM TODS, Vol.11 Nro.4 - 1986

/MOHA 89/ MOHAN C. et alii
"ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollback Using Write-Ahead Logging"
IBM Research Report RJ6649 - 1989

/MOSS 81/ MOSS J.E.B.
"Nested Transactions: An Approach to Reliable Computing"
M.I.T., Laboratory of Computer Science - 1981

/MOSS 87/ MOSS J.E.B.
"Log-Based Recovery for Nested Transactions"
Proceedings of International Conference on Very Large Data Bases - 1987

/NEUM 80/ NEUMANN T.
"CAD Databases Requirements and Architectures"
Lecture Notes in Computer Science - 89
Springer-Verlag - 1980

/NEUM 82/ NEUMANN T. e HORNUNG C.
"Consistency and Transactions in CAD Databases"
Proceedings of International Conference on Very Large Data Bases - 1982

/OLGU 89/ OLGUIN C.J.M. e MAGALHÃES L.P.
"Alguns Aspectos Associados ao Gerenciamento de Transações no contexto do GERPAC/UniCOSMOS"
I Seminário de Atividades de Pós-Graduação do DCA
DCA/FEE/UNICAMP; Nov.1989

/OLGU 90/ OLGUIN C.J.M. e MAGALHÃES L.P.
"Um Gerenciador de Transações no Contexto do GERPAC/UniCOSMOS"
Anais do V Simpósio Brasileiro de Banco de Dados (SBBD) - Abr.1990

/PCS 84/ PERIPHERE COMPUTER SYSTEME GmbH
"MUNIX V.2/03 - Tutorials - Volume IIa"
PCS GmbH - 1984

/RICA 86/ RICARTE I.L.M.
"Definição e Implementação de um Modelo de Dados para o Núcleo CORAS-UNICAMP"
Relatório de Atividades 3, FAPESP Proc. 84/0291-1, Ago.1986

/RICA 87/ RICARTE I.L. e DELGADO A.L.N.
"GERPAC - Um SGBD para P.A.C."
Anais do II Simpósio Brasileiro de Banco de Dados (SBBD) - Mai.1987

/RICA 87a/ RICARTE I.L.M.
"Sistemas de gerência de dados para Projeto auxiliado por computador"
Tese de mestrado DCA-FEE-UNICAMP - Jun.1987

/ROCH 85/ ROCHKIND M.J.
"Advanced UNIX Programming"
Prentice-Hall Inc. - 1985

/ROTH 89/ ROTHERMEL K. e MOHAN C.
"ARIES/NT: A Recovery Method Based on Write-Ahead Logging for Nested Transactions"
IBM Research Report RJ6650 - 1989

/SION 83/ STONEBRAKER M., RUBENSTEIN B. e GUTTMAN A.
"Application of Abstract Data Types and Abstract Indices to CAD Databases"
ACM SIGMOD/IEEE - 1983

/VERN 84/ VERNADAT F.B.
"A Commented and Indexed Bibliography on Data Structuring and Data Management in CAD/CAM: 1970 to Mid-1983"
National Research Council of Canada No. 23373
Março 1984

/WU 83/ WU S.T.
"Implementação de um Núcleo de Banco de Dados baseado na Filosofia de CORAS"
Documentação Preliminar, DCA/FEE/UNICAMP - 1983