Artigo Técnico

Revista de Informática Teórica e Aplicada

Transações de Bancos de Dados:Novos Requisitos de Processamento de Aplicações Não Convencionais

Maria Aparecida M. Souto
Cirano Iochpe
II/CPGCC-Universidade Federal do Rio Grande do Sul
Caixa Postal 15064
CEP 91501-970-Porto Alegre-RS
E-mail:souto,iochpe@inf.ufrgs.br

Resumo

Motivados pelo sucesso da tecnologia de banco de dados para sistemas comerciais, pesquisadores da área de banco de dados têm voltado sua atenção para as aplicações ditas não convencionais. Como resultado deste esforço, diversos novos modelos de transação têm sido propostos na literatura, com o objetivo de atender aos novos requisitos de processamento desta nova classe de aplicações.

Este trabalho apresenta uma análise sistemática dos novos requisitos de processamento das aplicações não convencionais. Cinco novos modelos de transação, propostos na literatura, são analisados quanto a aspectos tais como tipos de transações por ele suportados, estrutura das transações, controle de concorrência e recuperação em caso de falhas. Finalmente, os modelos analisados são comparados quanto à sua capacidade para suportar os requisitos listados.

Abstract

Motivated the by the success achieved by database systems tecnology in comercial applications, database researchers show ever growing

interest in non-standard applications. Some results of this new research effort are new database transaction models to support the requirements of this new application class.

This work presents a sistematic analysis of processing requirements posed by non-standard applications. Five new transaction models are investigated concerning important aspects like transaction types, transaction structure, concurrency control and recovery. Finally, the new transaction models are compared in order to identify wich of them can better cope with the new requirements.

Palavras-chave: Sistemas de banco de dados, Aplicações não convencionais de banco de dados, modelos não convencionais de transação.

1. Introdução

Nas duas últimas décadas, pesquisadores da área de Sistemas de Banco de Dados (SBDs) têm demonstrado um crescente interesse em adaptar a tecnologia convencional de banco de dados aos novos requisitos de processamento das aplicações não convencionais. Para exemplificar, aplicações tais como CAD, Automação de Escritório, Sistemas de Manufatura, entre outras, são consideradas não convencionais.

O resultado deste esforço expressa-se na proposta de novos modelos de dados (ex.: modelos orientados a objetos), novas arquiteturas de sistemas (ex.: arquiteturas do tipo cliente-servidor) e extensões ao paradigma da transação para suportar novos modelos de processamento (ex.: transações de grupo).

As principais extensões ao paradigma da transação, também chamadas de novos modelos de transação, foram propostas a partir de pesquisas em banco de dados para CAD e Automação de Escritório. De uma forma geral, estes modelos baseiam-se no modelo de transação convencional, devido aos resultados altamente favoráveis obtidos a partir de seu uso, por parte de aplicações convencionais, especialmente no que diz respeito aos aspectos de controle de concorrência, métodos de recuperação em caso de falhas e critérios de correção do banco de dados (BD).

Para adequar o paradigma da transação às necessidades desta nova classe de aplicações, o conceito tradicional de transação tem sido estendido no sentido de possibilitar a representação de unidades de trabalho de longa duração, com uma estrutura complexa de execução (atividades que desenvolvem subatividades), e que precisam cooperar umas com as outras, ao invés de compe-

tir pela informação que é compartilhada no BD. A manipulação de transações de longa duração (TLDs) como unidades *atômicas* de trabalho pode provocar situações indesejáveis diante de falhas, como desfazer, no BD, todos os efeitos causados pela transação, independente da quantidade de trabalho já executada e do tempo gasto para executá-la. Já a modelagem de transações longas como unidades de trabalho *isoladas* pode inibir o desenvolvimento do trabalho cooperativo entre elas.

Este trabalho enfoca as principais características dinâmicas das aplicações não convencionais em ambientes de CAD e Automação de Escritório e deriva, a partir destas, um conjunto de requisitos de processamento básicos que um modelo de transação deve satisfazer para suportar o modelo de processamento destas aplicações. São analisados cinco novos modelos de transação, propostos na literatura, ressaltando os importantes aspectos para um gerente de transações tais como tipos de transação suportadas pelo modelo, estrutura das transações, controle de concorrência e tolerância à falhas.

O trabalho está organizado como segue. A seção 2 apresenta as características dinâmicas das aplicações de banco de dados convencionais e suas necessidades de processamento. O modelo convencional de transação é apresentado na seção 3. A seção 4 apresenta as características dinâmicas das aplicações não convencionais e seus requisitos de processamento. A seção 5 descreve cinco modelos de transação não convencionais propostos na literatura. A seção 6 apresenta uma comparação entre os modelos analisados e os requisitos de processamento das aplicações não convencionais. Finalmente, na seção 7, são apresentadas algumas conclusões e considerações finais.

2. Aplicações de banco de dados convencionais

A necessidade de uso dos Sistemas de Banco de Dados foi primeiramente sentida pela classe dos implementadores de aplicações comerciais. Estas aplicações possuem um comportamento bastante similar no que diz respeito ao processamento de suas informações. Elas caracterizam-se por desenvolverem unidades de trabalho de curta duração (alguns segundos ou frações destes), que executam de forma independente, com acesso concorrente a dados comuns. De um modo geral, compartilham estas características os sistemas bancários, comerciais e de reserva de passagens aéreas, que hoje são

consideradas aplicações do tipo convencional.

A importância econômica das aplicações convencionais pode explicar, em parte, porque a tecnologia dos sistemas de BD evoluiu em direção aos requisitos de processamento destas. Assim, aliado ao fato de que os SBDs que suportam as aplicações convencionais representam, hoje, a maioria dos sistemas de banco de dados existentes, eles passaram a ser referenciados como *sistemas de banco de dados convencionais* ou *sistemas de banco de dados tradicionais.*

2.1 O modelo de processamento das aplicações convencionais

O modelo de processamento de uma aplicação descreve as abstrações que representam o seu comportamento, ou seja, suas características dinâmicas, ressaltando os aspectos relevantes para um gerenciador de transações. Estes aspectos incluem:

- como o trabalho do usuário e aplicação evolui
- como o usuário e aplicação interagem com o sistema de gerencia de banco de dados (SGBD)
- como as unidades de processamento do usuário e aplicação (i.e., programas de aplicação) se comunicam um com o outro.

Nas aplicações convencionais, o trabalho do usuário e aplicação são representados através de transações (unidades de trabalho) de curta duração, e que interagem com o SGBD de duas maneiras distintas: 1) através de processamento *batch* e 2) através de *diálogos curtos.* O processamento *batch* é usado para processar grandes volumes de dados, por períodos de tempo relativamente longos. Como exemplo, podemos citar um programa que executa, durante à noite, a movimentação diária das contas correntes dos clientes de uma agência bancária. Os *diálogos curtos* ocorrem entre o usuário e o SGBD, em sistemas de teleprocessamento. Neste caso, o usuário normalmente tem acesso a pequenas porções de dados no banco de dados. Uma situação típica é aquela em que um funcionário de uma companhia aérea efetua uma reserva em um vôo, em atenção à solicitação de um cliente.

Cada unidade de processamento é independente uma da outra e executa uma tarefa completa sob o ponto de vista da aplicação. Isto significa que, quando executada sozinha no sistema, o trabalho realizado por uma unidade de processamento leva o BD de um estado consistente para outro igualmente consistente.

Além disso, estas unidades de processamento independentes podem fazer acesso concorrente a dados compartilhados (p. ex.: duas ou mais reservas sendo solicitadas ao mesmo tempo em um mesmo vôo).

2.2 Requisitos de processamento das aplicações convencionais

Com base no modelo de processamento das aplicações convencionais, é derivado um conjunto de requisitos para que um modelo de transação suporte tais aplicações. Este modelo deve:

- suportar a execução de unidades de trabalho de curta duração;
- garantir que cada unidade de trabalho da aplicação mantenha a consistência do BD quando executada completamente;
- garantir que a consistência do BD não seja violada, mesmo quando unidades de trabalho da aplicação executarem concorrentemente sobre dados que são compartilhados no BD;
- garantir que a consistência do BD possa ser recuperada em situações de falha de *software*, *hardware* ou mesmo em situações de falha da própria aplicação.

3. O modelo convencional de transação

As necessidades de processamento das aplicações convencionais são suportadas através do *paradigma da transação* [HÄR 83]. Com base neste paradigma, os SBDs podem garantir a correção do BD em um ambiente multiusuário, onde as transações são tratadas como unidades de processamento *auto-contidas*, e que têm um significado *lógico* sob o ponto de vista da aplicação. A execução de uma transação pode deixar o BD no mesmo estado que este se

encontrava antes do início desta (ex.: transações apenas de leitura) ou pode levá-lo a um outro estado consistente da aplicação.

O conceito de transação assegura o isolamento do trabalho concorrente feito sobre o banco de dados, assim como a reconstrução da base de dados em caso de falhas. Uma transação é tipicamente formada por um conjunto de instruções de manipulação de dados. A Figura 1 apresenta o exemplo de uma transação que, para melhor entendimento, está escrita em uma pseudo-linguagem de programação de alto nível. As instruções "$BOT" e "$EOT" delimitam o trabalho a ser desenvolvido no âmbito do banco de dados para realizar uma tarefa específica da aplicação. A transação da figura realiza uma tranferência de fundos de uma conta bancária para outra. Esta transação pode ser interpretada como a manipulação de dados necessária para compensar um cheque.

```
Procedure Transferência-Fundos (conta_1, conta_2, valor)
Begin_transaction
$BOT (tr_id);
$FIND CONTAS Where nro_conta = "conta_1";
If NOTFOUND ou credito_conta < valor
Then $ABORT(tr_id)
Else begin
                credito_conta:= credito_conta - valor;
                $Overwrite CONTAS Where nro_conta = "conta_1";
                $FIND CONTAS Where nro_conta = "conta_2";
                If NOTFOUND Then ABORT(Tr_id);
                credito_conta := credito_conta + valor;
                $Overwrite CONTAS Where nro_conta = "conta_2";
                $COMMIT(tr_id);
                end (block);
$EOT (tr_id);
End procedure Transferência_Fundos;
```

Figura 1: Exemplo de uma transação de compensação de um cheque.

Assumindo que as transações são construídas de modo a manter a con-

sistência do banco de dados, o SBD assegura quatro propriedades, inerentes ao conceito de transação [HÄR 83]. As transações são *atômicas*, isto é, o sistema garante que se algum erro ocorrer durante a execução de uma transação, esta será abortada e nenhum de seus efeitos permanecerá no banco de dados. As transações são *consistentes* no sentido de que, quando executadas por completo, o SBD assume que nehuma restrição de integridade foi violada e que o banco de dados foi levado a um (novo) estado consistente. As transações executam em *isolamento*, ou seja, o SBD garante que cada transação só pode ter acesso e manipular informações criadas ou atualizadas por transações que já encerraram com sucesso. Com isso, as transações ficam impedidas de ter acesso a resultados temporários, gerados por outras transações, que ainda estão em andamento. Por último, o SBD deve garantir a *durabilidade*, no banco de dados, dos resultados gerados por transações que terminam corretamente (*commited transactions*), mesmo na presença de falhas do sistema.

Estas quatro propriedades da transação: *Atomicidade, Consistência, Isolamento* e *Durabilidade* são comumente referenciadas pelo termo *ACID*. Elas representam a base do modelo de transação convencional, e têm influenciado diversos aspectos do desenvolvimento e implementação dos sistemas de banco de dados.

A propriedade de *isolamento* forma a base para o compartilhamento de dados em sistemas de banco de dados. Pelo fato de as transações terem acesso somente aqueles dados que foram incluídos ou atualizados por transações que encerraram corretamente e, de cada transação, por definição, ser sempre correta, é garantido que as transações somente tem acesso a dados consistentes, mesmo quando executam em paralelo com outras transações.

A grande maioria dos SBDs convencionais implementam o isolamento de transações com base na teoria da serializabilidade [BER 87]. A idéia básica desta teoria é resolver conflitos de acesso aos dados entre as transações concorrentes, através do escalonamento serializável de um conjunto de transações concorrentes. Um escalonamento é serializável se, e somente se, os resultados produzidos pelas transações participantes deste são equivalentes aos resultados obtidos através de alguma execução serial destas mesmas transações. Considerando que cada transação produz um estado consistente do banco de dados, quando executa sozinha no sistema, um escalonamento serial de transações sempre produzirá um estado consistente do banco de dados. Assim, se cada escalonamento serializável é equivalente a, pelo menos, um esca-

lonamento serial, escalonamentos serializáveis sempre produzirão um estado consistente do banco de dados.

As propriedades de *atomicidade* e *durabilidade* das transações formam a base para a manutenção da integridade e da consistência do banco de dados em caso de falhas. Para projetar um mecanismo de recuperação de falhas para um sistema de gerência de banco de dados é necessário determinar, precisamente, os tipos de falhas a serem considerados. Os SBDs convencionais tratam três tipos importantes de falhas: 1) *falha da transação,* causada por uma divisão por zero, ocorrência de um *deadlock*, violação de restrição de integridade, etc.; 2) *falha do sistema,* causada por uma falha de código do SGBD ou do sistema operacional ou de hardware; e 3) *falha do meio de armazenamento,* causada pela perda de parte ou toda a memória secundária que mantém o BD.

Considerando os tipos de falhas acima mencionados, o modelo convencional de transação apresenta dois protocolos distintos para a recuperação do banco de dados. O primeiro deles garante que, para cada transação que é cancelada, o componente de recuperação desfaz, no banco de dados, os efeitos por ela causados. O outro, garante a permanência dos resultados de transações que encerraram com sucesso. Estes protocolos são implementados através das ações de: 1) *desfazer* (*UNDO*) as operações de atualização já incorporadas ao BD, correspondentes às transações que cancelaram; e 2) *refazer* (*REDO*) as operações de atualização ainda não incorporadas ao BD, correspondentes às transações que já haviam concluído com sucesso.

4. Aplicações não convencionais de banco de dados

Atualmente, banco de dados para aplicações ditas não convencionais tem se tornado uma área de pesquisa muito ativa. Para exemplificar, aplicações tais como CAD (*Computer-Aided Design*), CASE (*Computer-Aided Software Enginnering*), sistemas de informação de escritórios, inteligência artificial e processos industriais, entre outras, são consideradas não convencionais. Estas aplicações caracterizam-se por seu alto grau de complexidade, uma vez que devem suportar tipos de dados complexos, manipular diferentes níveis de abstração e suportar a execução de atividades de curta e longa durações (dias, semanas, etc.), as quais podem constituir uma estrutura complexa de

dependências de execução (ex.: atividades que desenvolvem sub-atividades) e estarem associadas a efeitos externos no mundo real.

Além destas características, certas aplicações não convencionais (ex.: aplicações de CAD) caracterizam-se, muitas vezes, pelo trabalho cooperativo desenvolvido entre suas atividades no sentido de que, ao invés de competirem por informação, unidades de trabalho distintas (transações) cooperam para atingir objetivos comuns.

Com o objetivo de superar as limitações dos SBDs tradicionais frente às características das aplicações não convencionais, novos modelos de transação têm sido propostos na literatura ([HAS 81], [MOS 82], [KLA 85], [GAR 87], [PUC 88], [KOR 87], [KÄF 90], [DAY 91],[NOD 92], [WÄC 92], etc.). De uma forma geral, estes modelos representam extensões do modelo de transação convencional, devido aos resultados altamente favoráveis obtidos por este último, no suporte as aplicações convencionais, especialmente no que diz respeito aos aspectos de controle de concorrência, recuperação em caso de falhas e critérios de correção do BD (serializabilidade [BER 87]).

4.1 Algumas características básicas das aplicações não convencionais

As aplicações de CAD, por exemplo, se dedicam aos projeto e especificação de sistemas técnicos. Tais aplicações estão normalmente associadas às atividades da área de engenharia. Exemplos de aplicações de CAD são o projeto de sistemas de engenharia de software, de peças mecânicas e o de circuitos integrados. Devido a complexidade destas aplicações, sistemas de computação vêm sendo construídos para apoiá-las. Na sua maioria estes sistemas são compostos de servidores de arquivos e ferramentas de software como editores gráficos e compiladores de máscaras, que ajudam os engenheiros nos projeto, desenvolvimento e teste de seus produtos. No que diz respeito à distribuição de tarefas e sua alocação aos projetistas, as aplicações de CAD geralmente apresentam uma estrutura hierárquica. Normalmente, projetos mais amplos são sucessivamente subdivididos em conjuntos de subprojetos até que cada subprojeto alcance os graus de complexidade e independência esperados. Cada uma das tarefas resultantes desta subdivisão é, então, executada por um pequeno grupo de projetistas (ex.: cinco a dez pessoas). Nestes grupos, cada projetista desenvolve uma parte do trabalho. Refletindo a organização hierárquica do projeto, projetistas alocados a gru-

pos diferentes, normalmente, manipulam objetos de dado também diferentes. Na maioria dos casos, projetistas de grupos diferentes somente tem acesso a dados comuns para fins de leitura (ex.: acesso a normas gerais de documentação de projeto). O mesmo tende a ocorrer quando projetistas de um grupo têm acesso aos dados criados por outro grupo (ex.: engenheiros projetando uma unidade de fita magnética necessitam de informações relativas às especificações de entrada e saída da mémória principal do computador sendo projetado por outro grupo). Projetistas de um mesmo grupo, normalmente, manipulam objetos de dado comuns. Porém, ao contrário de usuários de aplicações convencionais que fazem acesso aos dados de forma concorrente, projetistas de um mesmo grupo tendem a cooperar uns com os outros (ex.: construir partes distintas de uma mesma peça mecânica, usando uma interface comum).

Geralmente, projetistas de um mesmo grupo trabalham em contigüidade física, isto é, geograficamente próximos uns dos outros (ex.: na mesma sala, no mesmo prédio). A distribuição física de grupos de projetistas, assim como a estrutura hierárquica das aplicações de CAD influenciam a configuração dos sistemas computacionais que as suportam e, conseqüentemente, a arquitetura dos sistemas de banco de dados que suportam ambientes de desenvolvimento de projetos (ex.: arquiteturas do tipo cliente-servidor). A Figura 2 exemplifica um ambiente de CAD-VLSI.

Os Sistemas de Automação de Escritório igualmente apresentam um grande potencial de complexidade, o quê tem motivado os pesquisadores da área de banco de dados a investigá-los como aplicações do tipo não convencional. Neste contexto, um exemplo que tem sido comumente mencionado, refere-se à atividade de planejar uma viagem de negócios de um executivo [GRA 81, KLE 88, WÄC 92]. De uma forma simplificada, vamos considerar que a atividade de planejar uma viagem de negócios inclui fazer as reservas de vôo, hotel e carro de aluguel. Este conjunto de sub-atividades pode levar um longo período de tempo e implicar em várias tentativas junto às companhias aéreas, aos hotéis ou agências de locação de carros. É praticamente impossível efetuar todas as reservas dentro de uma única transação curta.

Para simplificar o exemplo, vamos supor que sejam consultadas apenas três companhias aéreas e somente dois hotéis, em combinação com duas companhias de aluguel de carro, como por exemplo, as combinações *Cathedral Hill Hotel* e *Avis* ou *Holiday Inn* e *Hertz*. É suposto que esta aplicação executa em um terminal de uma agência de viagem, conectado com uma rede mun-

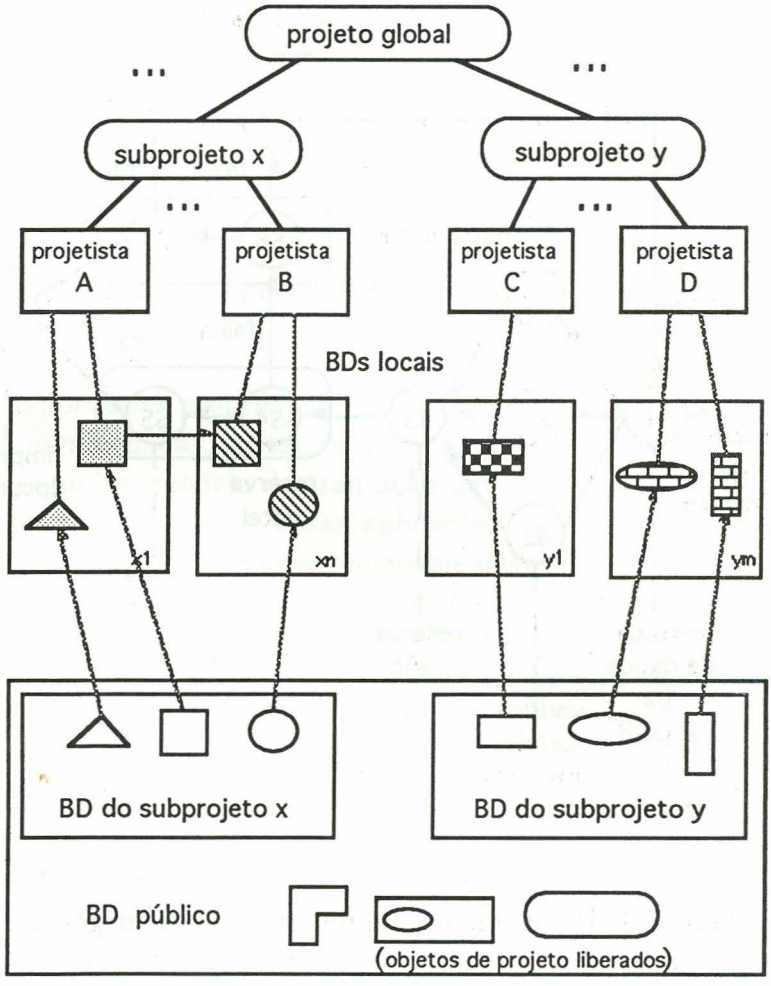

Figura 2: Exemplo de um ambiente de CAD-VLSI

dial de computadores heterogêneos que sediam vários servidores de banco de dados. Como o objetivo é enfocar os aspectos lógicos do processamento de transações, considerações sobre os aspectos físicos de comunicação e outros são omitidos.

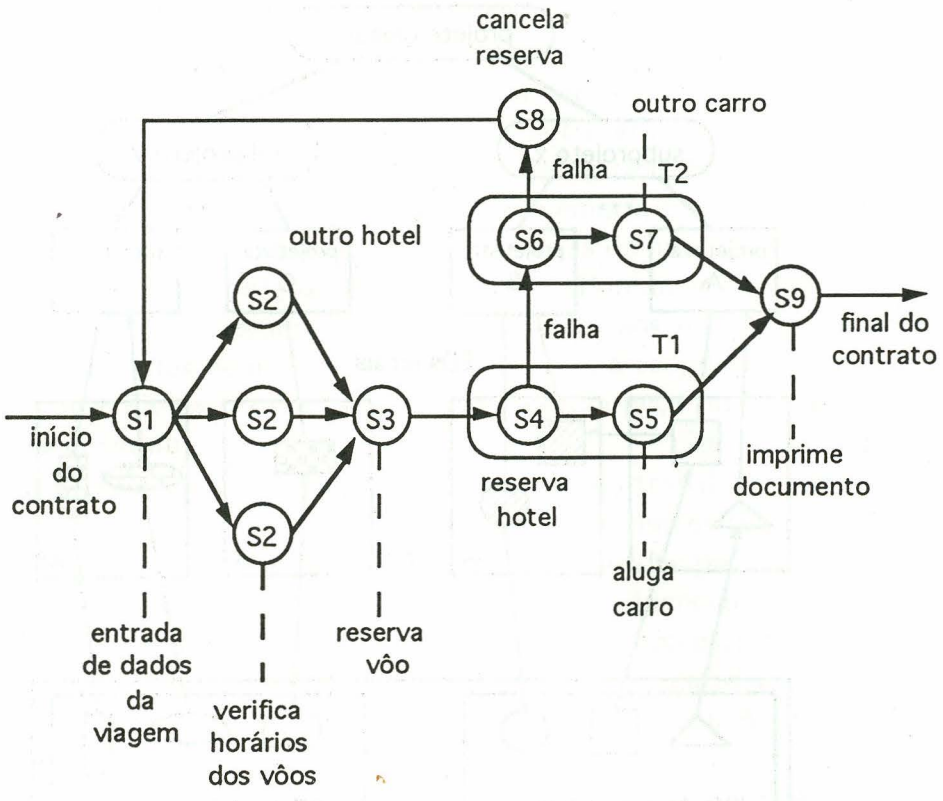

Figura 3: Planejamento de uma viagem de negócios [WÄC 92]

A figura 3 ilustra este exemplo segundo a concepção de Wächter e Reuter [WÄC 92]. Nele, os autores exploram interessantes aspectos da execução de uma atividade em um ambiente de automação de escritório. A atividade como um todo é subdividida em uma coleção de passos de processamento (passos S1, S2, S3, etc.). Através da figura podemos visualisar os diferentes níveis de dependência entre estes passos. As ações correspondentes às consultas de disponibilidade de vôos (passos S2) são independentes e executam em paralelo. A execução de uma reserva em algum vôo (passo S3) depende do término das consultas às companhias aéreas. O conjunto de ações correspondentes à combinação de reserva de hotel e carro de aluguel (passos

S4, S5 e S6, S7) deve executar como uma unidade atômica de trabalho (na figura, representados por T1 e T2). Se esta unidade composta falhar, então o sistema tenta outra alternativa, ou seja, se T1 falhar, então o sistema deve executar T2, e assim por diante. No exemplo proposto, é razoável imaginar que, diante de uma falha, o usuário da aplicação não permitiria que todas as reservas já afetuadas fossem canceladas. O cancelamento só deveria ocorrer sob demanda do mesmo.

4.2 Requisitos de processamento de aplicações não convencionais

A partir de importantes propriedades dinâmicas das aplicações não convencionais, acima mencionadas, passamos a enumerar os principais requisitos de processamento que um modelo de transação deve suportar para atender às necessidades de processamento destas aplicações. Isto não significa que todos os modelos de transação não convencionais devam suportar todos estes requisitos. A necesssidade de suportar um ou outro requisito depende do ambiente e de suas características dinâmicas. Para cada um dos requisitos apresentados, analisamos a sua adequação no contexto do modelo de transação convencional uma vez que, o suporte às novas necessidades de processamento destas aplicações fundamenta-se em extensões aos conceitos introduzidos por aquele modelo.

4.2.1 Processar atividades de longa duração

As atividades desenvolvidas pelo usuário em ambientes de CAD para engenharia, CASE, Escritório, Chão de fábrica, etc., caracterizam-se por demandarem uma quantidade de tempo substancial a sua execução (horas, dias, semanas). Dentre estas atividades, podemos citar as atividades de desenvolvimento de um projeto de CAD-VLSI, o projeto de software em ambientes CASE, processos de compra e venda de produtos em uma rede de supermercados (típico ambiente de escritório), execução de um processo industrial, correspondente à fabricação de um automóvel, etc.

A modelagem de uma unidade de trabalho (atividade) de longa duração como uma transação convencional tem motivado os pesquisadores da área a analisar, cuidadosamente, importantes aspectos desta abordagem. Em primeiro lugar, uma atividade de longa duração tem mais chances de ser in-

terrompida por falhas do sistema, em função de seu tempo de duração. Em ambientes ACID, todos os efeitos causados pela transação seriam desfeitos no banco de dados em situação de falha, independente da quantidade de trabalho já executada e do tempo gasto para executá-la.

Em segundo lugar, uma atividade de longa duração pode ter acesso a muitos itens de dados ao longo de sua execução. Entretanto, caso a transação executasse de forma *isolada*, estes itens de dados não poderiam ser liberados até o *commit* da transação. Assim, se uma outra transação curta desejasse fazer acesso a um item de dado mantido por esta transação que modela uma atividade de longa duração, ela seria obrigada a esperar, talvez por horas ou dias, até que a transação "longa" liberasse os dados através do seu *commit*. Conforme salientou Gray [GRA 81], a freqüência de *deadlocks* aumenta em função do tamanho da transação, em sistemas onde o controle de concorrência baseia-se em bloqueios (*locks*).

Conseqüentemente, uma atividade de longa duração quando modelada como uma transação convencional torna-se incompatível com as propriedades de *atomicidade* e *isolamento* das transações ACID. Enquanto representando uma unidade *lógica* de trabalho da aplicação, a transação que modela uma atividade de longa duração é considerada como uma unidade de trabalho *atômica*. Por outro lado, como uma unidade de *recuperação* da aplicação em situações de falha, a transação que modela a atividade de longa duração precisa relaxar a sua atomicidade. Desta forma, a ocorrência de falhas durante a execução da transação não deve desfazer no BD os efeitos por ela causados. Ao invés disso, o sistema deveria poder iniciar a execução de procedimentos de recuperação utilizando outras técnicas (ex.: *forward recovery*).

Por ser incompatível com a propriedade de *isolamento* das transações ACID, as TLDs precisam ter também tal propriedade relaxada, permitindo assim que outras transações possam ter acesso aos itens de dado, sendo por uma TLD manipulados, sem necessitar esperar que a TLD execute *commit*. Na seqüência do trabalho (seção 4.2.6) são discutidas as implicações desta característica das transações de longa duração, especificamente no que diz respeito aos critérios de correção do banco de dados quando do acesso concorrente a dados compartilhados.

4.2.2 Suporte à execução de atividades estruturadas (complexas)

No contexto das aplicações convencionais, cada atividade executa de

forma totalmente independente uma da outra.(ex.: compensação de um cheque, reserva de uma passagem aérea, etc.). Se observarmos como o trabalho do usuário se desenvolve em um ambiente de CAD (ex.: projetos que englobam subprojetos) ou em um ambiente de desenvolvimento de software (ex.: um projeto de software que é composto de diversos módulos os quais, por sua vez, são constituídos por diversas *procedures*, e assim por diante) ou em um ambiente de acompanhamento de processos industriais (ex.: o processo de fabricação de um automóvel sendo decomposto em sub-processos de fabricação dos componentes mecânicos, elétricos e de acessórios, etc.), podemos constatar que estas atividades organizam-se, naturalmente, em diversos níveis de complexidade (atividades que desenvolvem sub-atividades), como em uma estrutura hierárquica de atividades. Para que um modelo de transações possa suportar esta característica de algumas aplicações não convencionais, é necessário que ele disponha de propriedades que permitam a representação de uma estrutura hierárquica de atividades como uma estrutura hierárquica de transações.

Se o modelo de transação dispõe de propriedades através das quais a aplicação pode representar uma atividade complexa como uma estrutura hierárquica de transações, então este modelo também deve permitir à aplicação especificar as dependências de *commit* e *abort* entre transações pais e filhas e entre transações que pertencem a um mesmo nível hierárquico ou , até mesmo, especificar a ordem de execução das (sub)transações.

Por outro lado, devemos considerar que uma estrutura complexa de execução de transações não necessariamente significa uma hierarquia de transações. Por exemplo, em um ambiente de automação de escritório, o planejamento de uma viagem de negócios de um executivo poderia ser modelado como uma atividade de longa duração, composta por um conjunto de pequenas ações, onde a execução das quais deve respeitar uma ordem parcial a ser especificada pelo programador da aplicação. A Figura 3, acima apresentada, ilustra este exemplo.

Portanto, quer sob a forma de uma estrutura hierárquica, quer sob a forma de uma ordem seqüencial ou parcial, para representar uma estrutura complexa de execução de uma transação, é preciso que o modelo de transação apresente alguma facilidade através da qual a aplicação possa indicar ao sistema de banco de dados como executá-la. Com o controle de execução sob a responsabilidade do SBD, é possível gerenciar a execução da aplicação de forma segura, com suporte adequado ao controle de concorrência e à recu-

peração em caso de falhas.

4.2.3 Suporte à reestruturação dinâmica de atividades não determinísticas

Os ambientes de projeto, escritório, manufatura e outros apresentam uma importante característica de processamento, que diz respeito a natureza *não determinística* de certas atividades, ou seja, atividades com tempo de duração incerto, e cujas ações a serem executadas não são totalmente previsíveis no momento em que ela inicia.

Uma situação típica em um ambiente de programação interativa surge quando um programador decide efetuar a correção de um erro em um programa. Pu et al. [PUC 88] apresentam a situação da seguinte forma: normalmente, o programador executa um ciclo onde ele examina a execução do programa com o auxílio de um depurador, lê alguns arquivos fontes que ele imagina estarem relacionados com o erro, modifica um subconjunto destes fontes, compila, liga estes fontes, executa o teste para alguns casos, constata que a modificação não funcionou, abandona a versão anterior de alguns ou de todos os fontes, lê outros fontes, modifica, etc., até considerar que o problema foi solucionado e todas as modificações são liberadas.

Como se pode ver neste caso, o conjunto de operações realizadas pelo programador **não** são totalmente previsíveis no momento que ele inicia a sua atividade. Esta característica de processamento de algumas aplicações tem motivado os pesquisadores da área na busca de uma alternativa de solução. Para suportar atividades que são controladas pelo usuário, de forma interativa, o gerente de transações precisa fornecer a este usuário a capacidade de iniciar uma transação, executar operações sob o controle desta transação de forma interativa, reestruturá-la de forma dinâmica, se necessário, e executar *commit* ou abortá-la em algum momento desejado. A natureza não determinística de uma transação implica na impossibilidade do gerente de transação determinar, *a priori*, se a execução da transação poderá violar a consistência do BD ao longo de sua execução. Esta constatação é apresentada por Kaiser e Pu em [KAI 92]. Segundo estes autores, os mecanismos de controle de concorrência baseados em análises estáticas não são aplicáveis. Neste caso, a recuperação da consistência deveria passar a ser controlada pelo próprio usuário.

4.2.4 Suporte ao trabalho cooperativo entre atividades

Outra grande limitação do modelo convencional de transação é não permitir que, através do acesso compartilhado a itens comuns, transações troquem informações enquanto executam. No modelo convencional de transação, este tipo de cooperação é expressamente proibido, caso os itens de dado em questão estejam sendo modificados e consultados concorrentemente. Enquanto executa, uma transação ACID mantém seus resultados isolados, impedindo que outras transações possam vê-los. Entretanto, para algumas aplicações não convencionais (ex.: CAD para engenharia), esta necessidade de processamento é fundamental.

Para ilustrar o quê foi dito acima, pode-se imaginar um grupo de três engenheiros que deve projetar um novo carro, o qual é composto pela *carroceria*, *motor* e *interior*. Devido a complexidade do projeto, este é subdividido em três tarefas: (sub)projeto da carroceria, (sub)projeto do motor e (sub)projeto do interior. Imagine que o (sub)projeto do motor, dadas as restrições impostas pelo resto do carro, é novamente subdividido nos (sub)projetos do *bloco do motor* e *carburador*. Naturalmente, subdivisões também ocorrem em relação aos projetos da carroceria e interior. A Figura 4 apresenta uma possível subdivisão da tarefa de projetar um novo carro. Imagine que cada uma destas tarefas é alocada a um engenheiro distinto. Além disso, pode-se também imaginar que cada tarefa será executada, no sistema de CAD, como uma transação ACID.

O engenheiro que desenvolve o projeto da carroceria deve especificar o *tamanho* e a *forma* desta. Fazendo isso, ele está, indiretamente, impondo restrições de tamanho e forma ao motor, sendo projetado por outro engenheiro (através de uma segunda transação ACID). Pelo fato de o SBD manter isolamento entre as duas transações, o projeto do motor ficará, então, suspenso à espera das informações sobre os requisitos de tamanho e forma da carroceria, até que o projeto da carroceria seje dado por terminado (*committed*) e aquelas informações tenham sido liberadas para acesso por parte de outras transações.

Por outro lado, a terceira transação, aquela do engenheiro que desenvolve o projeto do interior, igualmente ficará bloqueada, à espera das informações de tamanho e forma da carroceria. Tais informações devem ser geradas através da transação que especifica a carroceria do carro.

Devido à propriedade de isolamento das transações ACID, as execuções

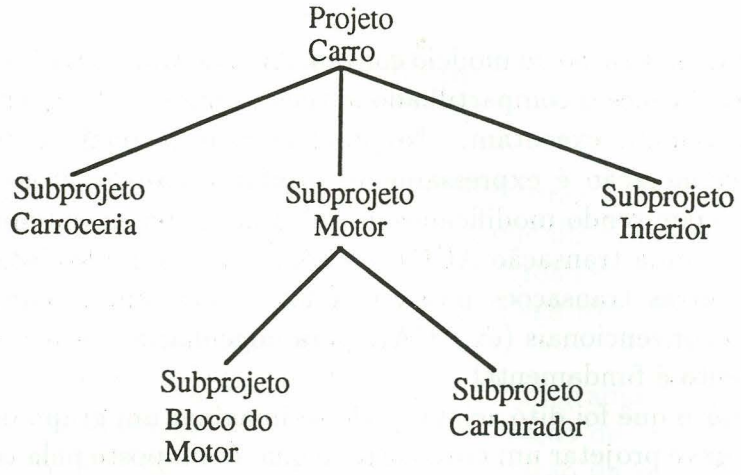

Figura 4: Subdivisão da tarefa de projetar um novo carro

das três transações de nosso exemplo terminam sendo executadas em série, o que pode aumentar significativamente o tempo de desenvolvimento do projeto do novo carro. Isto eleva o custo do projeto (projetistas e equipamentos alocados por mais tempo a ele), podendo torná-lo economicamente inviável.

Para modelar a cooperação entre tarefas de processamento de algumas aplicações é necessário que o modelo de transação disponha de meios através dos quais a aplicação possa especificar os protocolos que vão definir como as transações que cooperam devem interagir umas com as outras, de forma a garantir a integridade dos dados no BD.

4.2.5 Suporte à recuperação orientada à aplicação

A questão da recuperação do BD em situações de falha de transações de longa duração tem sido questionada sob dois aspectos básicos: 1) transações de longa duração precisam relaxar a atomicidade em relação à falhas (*failure atomicity*); e 2) transações de longa duração precisam relaxar a propriedade de isolamento das transações convencionais.

Quanto à primeira questão, consideramos o seguinte. O princípio do tudo-ou-nada das transações atômicas convencionais torna-se inviável às TLDs,

pois pode ser inviável desfazer os efeitos causados pela transação, após um período de tempo substancial de execução da mesma (dias ou meses). Devemos considerar, também, as situações nas quais uma transação não pode abortar. Por exemplo, poderíamos considerar uma transação controlando a operação de usinagem de uma peça em um torno, em um ambiente de chão de fábrica. Seria totalmente absurdo tentar desfazer no BD, os efeitos causados pela execução da transação uma vez que, neste caso, tanto o sistema físico (a peça) quanto o sistema de informação devem alcançar um estado compatível e consistente sob o ponto de vista da aplicação.

No contexto destes novos ambientes investigados, o objetivo maior é gerenciar transacões de longa duração que sempre executam *commit*, à exceção dos casos em que são forçadas a abortar (no exemplo da atividade de planejamento de uma viagem de negócios, acima mencionado, poderíamos considerar a situação na qual o executivo decide cancelar sua viagem de negócios). O fato de existir a exigência de uma TLD somente poder terminar por *commit,* gera a necessidade do SBD realizar uma recuperação, utilizando técnicas de avanço (*forward recovery*), ou seja, quando da ocorrência de uma falha, o sistema deve prover meios para que o processamento da TLD recupere um estado consistente para a aplicação e, a partir daí, alcançe o seu término normal, executando as (sub)transações pendentes.

Se o modelo de transação permite a definição da estrutura de execução de uma transação complexa (vide seção 4.2.2), então o SBD passa a ter controle sobre a execução da mesma. Neste caso, os procedimentos para a recuperação do tipo *forward recovery* podem ser automatizados, deixando de ser responsabilidade única da aplicação. Na verdade, uma recuperação de avanço implica no sistema manter a descrição do estado atual da execução em um meio seguro de armazenamento, de modo que tal estado possa ser recuperado, rapidamente, em caso de falha do sistema ou da transação.

Em relação à segunda questão, uma TLD relaxa a propriedade de isolamento liberando resultados parciais através do *commit* de suas subtransações. Neste caso, se considerarmos que uma (sub)transação do conjunto representa uma unidade ACID de trabalho, então, para cada (sub)transação que executa *commit*, seus bloqueios são liberados às outras transações e seus resultados tornam-se duráveis no BD. Assim, se após o término de execução de algumas subtransações, a TLD decide abortar, então tanto suas subtransações *ativas*, quanto as que já haviam executado *commit* também devem abortar.

As subtransações que estavam ativas abortam como no modelo de tran-

sação convencional (i. e., através da execução de uma operação de *undo*). O problema crucial é representado por aquelas subtransações que já haviam executado *commit*, cujos resultados devem agora ser desfeitos no BD. Para estas situações, o modelo de transações deve dispor de meios através dos quais uma transação que já executou *commit*, possa ter seus efeitos compensados no BD, sem ameaçar a integridade dos dados no contexto da aplicação.

No exemplo da viagem de negócios, poderíamos imaginar a seguinte situação. Após a TLD correspondente ter efetivado uma reserva em algum vôo, o executivo decide cancelar a viagem. Neste caso, mesmo que a TLD tenha liberado os dados correpondentes aquele vôo e que outras transações tenham feito suas reservas neste mesmo vôo, a TLD que agora está cancelando deve liberar a sua reserva. Segundo Gray [GRA 81], isto define uma ação de *compensação* que, por sua vez, exprime a noção de "*undo* semântico", ou seja, ao invés do sistema restaurar a versão anterior à transação do objeto modificado (*before images*), como no modelo de transação convencional, os efeitos sobre os objetos manipulados pela transação são anulados do ponto de vista semântico e o BD não necessáriamente retorna ao estado anterior à execução da TLD.

4.2.6 Suporte às definição e manutenção de critérios de correção do BD orientados à aplicação

O acesso concorrente a dados compartilhados impõe a definição de critérios de sincronismo entre as transações, de modo a evitar que o BD atinja um estado inconsistente em decorrência disto. A maioria dos SBDs convencionais implementam a *serializabilidade* [BER 87] como o critério básico de correção do BD em ambientes multiusuário. Sob este aspecto, o problema de consistência nos sistemas de banco de dados convencionais resume-se em testar se o escalonamento de um conjunto de transações que executam de modo concorrente é *serializável*, ou seja é equivalente a uma execução serial destas transações [BAR 91].

Nos SBDs para aplicações não convencionais, Barghouti e Kaiser [BAR 91] analizam esta questão em função de três importantes características de processamento destas aplicações: 1) processamento de transações de longa duração; 2) natureza não-determinística das transações e 3) cooperação sinergística entre transações.

O processamento de transações de longa duração implica em o SBD ter

que relaxar a propriedade de isolamento das transações. Conseqüentemente, o critério baseado na *serializabilidade* precisa ser revisado ou substituído. A liberação de bloqueios por uma TLD em execução, sem que haja qualquer controle no escopo desta, pode levar a sérios problemas de inconsistência. Por exemplo, imagine duas transações de longa duração TLD_1 e TLD_2 representando, respectivamente, o planejamento de duas viagens de negócio, como no exemplo da Figura 2. Vamos considerar que TLD_2 efetue suas consultas sobre as possibilidades de vôo e decida por um determinado vôo de uma companhia aérea. Antes que TLD_2 efetue a sua reserva, TLD_1 o faça para a mesma companhia, no mesmo vôo, tornando assim inválida a reserva de TLD_2. Isto causaria um sério problema de consistência para a aplicação (duas reservas do mesmo assento).

Em um ambiente de CAD, por exemplo, o trabalho do usuário começa a partir do momento em que ele inicia uma transação, executa operações dentro de seu escopo de forma interativa, reestrutura dinâmicamente uma transação e executa *commit* ou a aborta em algum tempo indeterminado. Esta característica de processamento define a natureza *não-determinística* de tal transação. Isto significa que o mecanismo de controle de concorrência desta transação não poderá determinar, *a priori*, se a execução da transação poderá violar a consistência do banco de dados, exceto se a transação for executada completamente e seus resultados avaliados no banco de dados. Então, o usuário precisa ter mais controle sobre as suas transações.

Em ambientes de projeto (CAD para engenharia, CASE), é sempre possível a troca de informação através de dados compartilhados. Estas trocas certamente não irão produzir um escalonamento serializável das operações que as realizam. Além disto, também é possível que dois usuários modifiquem diferentes partes de um mesmo item de dados (ex.: objeto estruturado), de modo concorrente, com a intenção de integrar estas partes para criar uma nova versão de objeto [BAR 91]. Yeh [Yeh 87] denominou este tipo de compartilhamento e troca de conhecimento como *interações sinergísticas.* Neste caso, a utilização do critério de *serializabilidade* tornaria inviável esta necessidade de processamento de algumas aplicações não convencionais, uma vez que está beseada na propriedade de isolamento das transações.

Ainda sobre a questão do controle de concorrência, Barghouti e Kaiser [BAR 91] chamam a atenção para o fato de que, no contexto dos SBDs convencionais, é aceito que, pelo menos parte das restrições de consistência sejam desconhecidas do sistema. Entretanto, se uma quantidade maior de

informação semântica sobre as transações e suas operações fosse transmitida ao SBD, a consistência do BD poderia ser garantida por outros meios que não a manutenção estrita da serializabilidade.

5. Modelos não convencionais de transação

Esta seção descreve cinco diferentes modelos de transação não convencionais: o modelo de *Transações Aninhadas* [MOS 82], *SAGAS* [GAR 87], *Split Transactions* [PUC 88, KAI 92], *ConTracts* [WÄC 92] e o modelo de Transações de Cooperação [NOD 90,NOD 92]. Estes modelos enfocam importantes características de processamento em ambientes de Projeto e Automação de Escritório. O modelo de *Transações Aninhadas* representa uma extenção ao modelo ACID que suporta distribuição e aninhamento de transações. Os modelos *SAGAS* e *ConTracts* exploram os aspectos referentes ao conceito de transações de longa duração em ambientes de Automação de Escritório. Os modelos *Split Transaction* e o modelo de *Transações de Cooperação* enfocam as necessidades de processamento em ambientes de projeto, destacando os aspectos de cooperação entre atividades e de atividades cujo início e término não são totalmente previsíveis.

Os modelos descritos neste trabalho são clássicos da literatura. Os modelos de Transações Aninhadas, SAGAS, *Split Transactions* e Transações de Cooperação de Nodine, por exemplo, influenciaram, de uma forma ou outra, os modelos que surgiram posteriormente. Já o modelo *ConTracts* constitui-se em uma das mais recentes propostas, e talvez a mais completa, quando consideramos os requisitos de processamento de grandes atividades estruturadas e de longa duração, modeladas como unidades lógicas de trabalho da aplicação. Além de apresentar um modelo de transação bem definido, o modelo *ConTracts* também apresenta um *modelo de programação* e um *modelo de execução*. Segundo os autores, grandes aplicações são definidas através da combinação de passos de processamento elementares. Através do modelo de programação, o Contrato permite a definição de tais passos e de suas possíveis combinações (relações de dependência). O modelo de execução representa o gerenciamento do fluxo de controle, em tempo de execução, através do componente denominado *gerente de contrato*. O gerente de contrato também é responsável por executar as ações de recuperação apropriadas em situações de falha, mantendo-as ocultas da aplicação.

Em Elmagarmid [ELM 92] o leitor encontra um ponto de referência importante sobre o estado da arte em relação às pesquisas sobre modelos de transação para aplicações não convencionais.

5.1 Descrição dos modelos

5.1.1 O modelo de Transações Aninhadas

O modelo de Transações Aninhadas de Moss foi proposto com o objetivo de superar algumas limitações do modelo convencional quando modelando aplicações complexas, que desenvolvem atividades estruturadas em ambientes distribuídos.

Em um sistema de Transações Aninhadas, qualquer transação pode apresentar subtransações. Uma transação e suas subtransações (filhas e descendentes indiretas) compõem uma árvore de transações. A nomenclatura utilizada é a padrão para árvores e aquela usada para relações familiares tais como mãe, filha, ancestral, descendente. A transação de mais alto nível é chamada de raiz. Na árvore de transações mostrada na Figura 5, a raiz é representada pela transação A. As transações filhas de C são D, F e G; a transação pai de C é B. As inferiores de C são D, E, F e G e as superiores são B e A. Desta forma, os conjuntos das transações descendentes e ancestrais de C contém C. Transações não descendentes de C são A, B, H, I e, também, K e L, que pertencem a outra transação aninhada. Como indica a Figura 5, a esfera de C é o conjunto das descendentes de C.

A transação raiz define uma unidade ACID de trabalho. Portanto, transações raiz executam de forma isolada e, em caso de falha, devem ter seus efeitos desfeitos no BD. Às subtransações, o sistema garante as propriedades de atomicidade e isolamento. A durabilidade de subtransações fica condicionada à durabilidade de suas superiores.

O *commit* de uma transação depende do término de suas filhas, assim como o *commit* de subtransações é condicionado ao *commit* de suas superiores. A transação pai herda os bloqueios de transações filhas que encerraram. O aborto de uma transação pai sempre descarta o trabalho executado por esta transação, e de qualquer subtransação existente na sub-árvore da qual ela é a raiz. Sendo assim, o encerramento com sucesso de uma transação não raiz não se constitui em uma operação definitiva. Esta operação de encerramento simplesmente indica que os resultados da transação que encerrou

tornam-se disponíveis no escopo da sua transação pai, que por sua vez, ainda poderá abortar. O único encerramento que é durável é o da transação raiz. Por outro lado, transações filhas poderão eventualmente falhar, sem que a raiz aborte, ou seja, a falha de uma ou mais transações filhas não implica, necessáriamente, em uma transação raiz abortada. Sobre esta questão, o modelo introduz as idéias de transação de *contingência* e transação *não-vital*. As transações de *contingência* têm a função de substituir uma outra transação que abortou. Isto quer dizer que a transação de *contingência* somente será executada no caso da transação à qual ela está associada abortar. Uma transação é considerada *não-vital* quando o seu aborto não impede que sua transação pai continue executando.

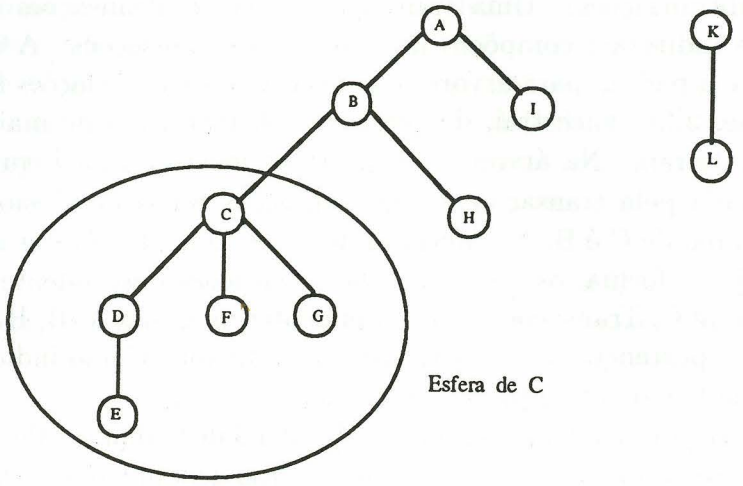

Figura 5: Exemplo de uma estrutura aninhada de transações [HÄR 88]

No contexto deste modelo, transações do tipo raiz são serializáveis como no modelo convencional de transação, quando do acesso concorrente a dados compartilhados. Dentro do universo de uma transação raiz, subtransações de um mesmo nível são igualmente sincronizadas entre si como se cada uma delas fosse a raiz. Em conseqüência disso, o acesso concorrente aos dados compartilhados é realizado com segurança, em qualquer nível da hierarquia. Além disso, se uma transação pai detém um bloqueio, ela não interfere com

suas filhas, apenas com as transações externas ao seu universo (qualquer transação que não seja inferior a uma transação pai é externa ao universo da mesma). Isto significa que uma subtransação tem acesso a todos os objetos que suas ancestrais têm.

A aplicação do modelo de Transações Aninhadas em sistemas distribuídos constitui-se em uma das principais contribuições da proposta de Moss. Este modelo permite que uma transação gere filhas em outro nodo da rede de computadores, sob o controle do SGBD. Para garantir a atomicidade da transação raiz quando do seu encerrramento, o seu *commit* é controlado por um protocolo do tipo *two-phase commit protocol* (2PC) [BER 87]. Conforme salientou Moss, esse protocolo de *commit* resolve a maioria dos problemas que surgem a partir de falhas em nodos ou falhas da própria rede. Quando uma transação conclui ou aborta, se o pai está em nodo diverso, a transação pai é informada do encerramento ou aborto. Igualmente, os nodos que executam descendentes da transação concluída são informados sobre o seu resultado. Assume-se, portanto, que ao concluir-se uma transação, sua transação pai é informada da identidade dos nodos que executaram inferiores da transação concluída.

Após todos os nodos que executaram inferiores terem informado à raiz sobre o seu encerramento, o nodo onde executa a raiz, o qual desempenha o papel de coordenador no protocolo 2PC, inicia a primeira fase. Nesta fase, o coordenador solicita a cada participante para que se prepare para encerrar. Esta preparação implica em o participante registrar em algum meio seguro de armazenamento as informações que ele necessita para executar *commit* ou abortar, baseado nas futuras instruções do coordenador. A preparação também implica em checar se o participante pode, ou não, dar o *commit* da transação naquele momento. Na segunda fase, o coordenador recebe a resposta dos participantes e, se todos concordam que podem dar *commit* e estão preparados, o coordenador os instrui para que efetivamente executem *commit*. Caso contrário, o coordenador os instrui para que abortem.

Quando a transação raiz solicita a cada nodo participante que se prepare para executar *commit*, ela envia uma lista das transações que ela acredita terem executado *commit* naquele nodo. O nodo participante então compara a lista recebida com a base de dados de suas transações. Se a lista tiver transações que não estão na base de dados, uma falha deve ter ocorrido em algum momento anterior, removendo a transação que imaginava-se tendo executado *commit*. Neste caso, a raiz não pode executar *commit*, e este par-

ticipante desfará tudo e forçará a transação raiz a abortar. Por outro lado, se o participante tiver transações inferiores que não estiverem na lista, estes inferiores são simplesmente abortados. Um de seus ancestrais deve ter abortado, então eles devem ser desfeitos. O *commit* ou aborto da raiz não é afetado por tais inferiores.

5.1.2 SAGAS

O modelo SAGAS, proposto por Garcia-Molina e Salem, fundamenta-se no conceito de *transações longas* (*long-lived transactions*). Segundo estes autores, uma *transação longa* representa uma unidade de trabalho cuja execução demanda uma quantidade de tempo substancial (horas, dias, semanas), mesmo sem interferência de outras transações.

Modeladas como transações ACID, transações longas apresentam sérios problemas com relação ao desempenho do sistema gerenciador de banco de dados que as executa. Estes problemas são uma conseqüência direta do fato destas transações serem tratadas como unidades de trabalho atômicas e isoladas. Com o objetivo de solucionar estes problemas, o modelo SAGAS apresenta um mecanismo de controle menos rígido que o modelo convencional de transação. No contexto de uma SAGA, o SBD passa a tratar uma transação longa como uma coleção de subtransações independentes, que devem executar como uma unidade. Objetos bloqueados a cada subtransação são liberados no final destas, antes que a transação longa complete sua execução, permitindo que outras transações prossigam sem esperar tanto tempo. Qualquer execução parcial de uma SAGA é indesejável e, se ela ocorrer, as subtransações já executadas devem ser compensadas. Com isto, o modelo introduz o conceito de *transações de compensação.* Uma transação de compensação desfaz, sob o ponto de vista semântico, todas as ações executadas por aquelas subtransações que já haviam executado quando a falha ocorreu. O *undo semântico* significa que o sistema não necessariamente retorna o BD ao estado em que este se encontrava quando a transação que está sendo compensada iniciou.

A partir da execução de uma SAGA, o sistema garante um dos seguintes resultados:

1) Caso normal: T_1, T_2, ..., T_n ou

2) Caso anormal: T_1, T_2, ..., T_k, T_{k+1}, C_k, ..., C_2, C_1 (assumindo que uma falha ou violação de consistência em T_{k+1}, para algum K, tal que k+1

< n).

Na situação em que T_{k+1} (em andamento) falha, o sistema retorna o BD ao estado anterior à sua execução. Para desfazer os efeitos de subtransações que já haviam encerrado quando a falha ocorreu, as transações de compensação C_k, ..., C_2 e C_1 são acionadas.

A definição de uma SAGA determina algumas dependências fundamentais de *commit* entre esta e suas subtransações. O *commit* de uma SAGA depende de *commit* de todas as suas subtransações, na ordem em que foram especificadas. No caso das subtransações, os *commits* podem ocorrer sem que estas tenham que esperar pelo *commit* de outras subtransações ou pelo *commit* da própria SAGA. Por outro lado, uma transação compensatória pode executar *commit* somente se sua transação correspondente executa *commit*, mas a SAGA que ela pertence deve abortar.

Além do modelo relaxar a propriedade de *isolamento* das transações ACID, a *atomicidade de falhas* (*failure atomicity*) também é relaxada. Quando uma falha interrompe uma SAGA, o trabalho já executado não é necessariamente desfeito. Neste caso, existem duas possibilidades: 1) o sistema executa uma recuperação do tipo *backward recovery*; ou 2) o sistema executa uma recuperação do tipo *forward recovery*. A primeira opção garante a *atomicidade semântica* no sentido de que, se o sistema por algum motivo não conseguir completar a execução da SAGA após a ocorrência de uma falha, então as subtransações que já executaram *commit* devem ser *compensadas*. A opção *forward recovery*, por sua vez, favorece o relaxamento da atomicidade de falhas. Neste caso, diante de uma falha, após o sistema executar as operações de *undo* e *redo* das subtransações ativas, ele passa a executar as transações que faltavam para completar a SAGA. Assim, o modelo suporta recuperação do tipo *forward* a nível da SAGA e não ao nível de suas subtransações curtas.

5.1.3 *Split Transactions*

O modelo *Split Transactions*, apresentado em [PUC 88, KAI 92], fundamenta-se no conceito de atividades com término em aberto (*open-ended activities*), ou seja, atividades que caracterizam-se por ser de longa duração (com tempo de duração incerto), envolver ações não previsíveis, *a priori*, e interagir com outras atividades concorrentes, no sentido de reunir esforços entre os co-usuários da aplicação na execução de alguma tarefa. Estas atividades são típicas em ambientes de Projeto.

Para atender as necessidades de processamento destas atividades, quando modeladas como uma transação, os autores introduzem as operações *transação de divisão* (*split-transaction*) e*transação de agrupamento* (*join-transaction*). A chamada de uma *transação de divisão* dentro de uma transação T divide esta transação em duas transações A e B, serializáveis uma em relação à outra, e em relação a todas as outras transações, e dissolve a transação original T (Figura 6a). Os recursos lidos ou atualizados por T são divididos entre as transações resultantes. Cada uma das novas transações A e B executa *commit,* ou aborta, de modo independente. Por outro lado, a chamada de uma *transação de agrupamento* dentro de uma transação T dissolve T e agrupa os seus resultados à transação S a qual ela está se agrupando (Figura 6b)). Esta operação permite que duas transações serializáveis sejam agrupadas em uma única transação, como se elas sempre tivessem feito parte da mesma transação. Na transação S, os resultados agrupados são liberados ou descartados atomicamente, quando S executa *commit* ou aborta, respectivamente. Desta forma, as operações de divisão e agrupamento de transações realizam a reestruturação de transações em tempo de execução. A Figura 6c ilustra uma possível combinação das operações de divisão e agrupamento.

Com o objetivo de superar as restrições do modelo ACID quando modelando transações longas, as operações de reestruturação de transações podem permitir que resultados parciais sejam liberados através do *commit* da transação A (resultante da divisão de uma T, por exemplo), enquanto a porção incompleta do trabalho continua através da transação B (também resultante da divisão de T). Nesta abordagem, a restrição relativa a atomicidade de falhas é de certa forma contornada uma vez que, a partir do momento em que a transação A executa *commit*, a recuperação do BD em situação de falhas fica restrita a recuperação da transação B. Por outro lado, o modelo nada garante sobre a atomicidade semântica da transação longa como uma únidade lógica de trabalho. Por exemplo, no caso do usuário decidir cancelar o seu trabalho, o modelo nada prevê em relação aos resultados já liberados, o que poderia ser catastrófico para a aplicação.

Os autores apresentam as operações de reestruturação dinâmica de transações como uma solução à necessidade de cooperação entre as transações em andamento. A cooperação pode ocorrer de duas formas: 1) através de uma *transação de divisão*: quando uma transação se divide para liberar uma porção dos seus resultados a um outro usuário da aplicação; e 2) através de de uma *transação de agrupamento*: quando transações originalmente separadas

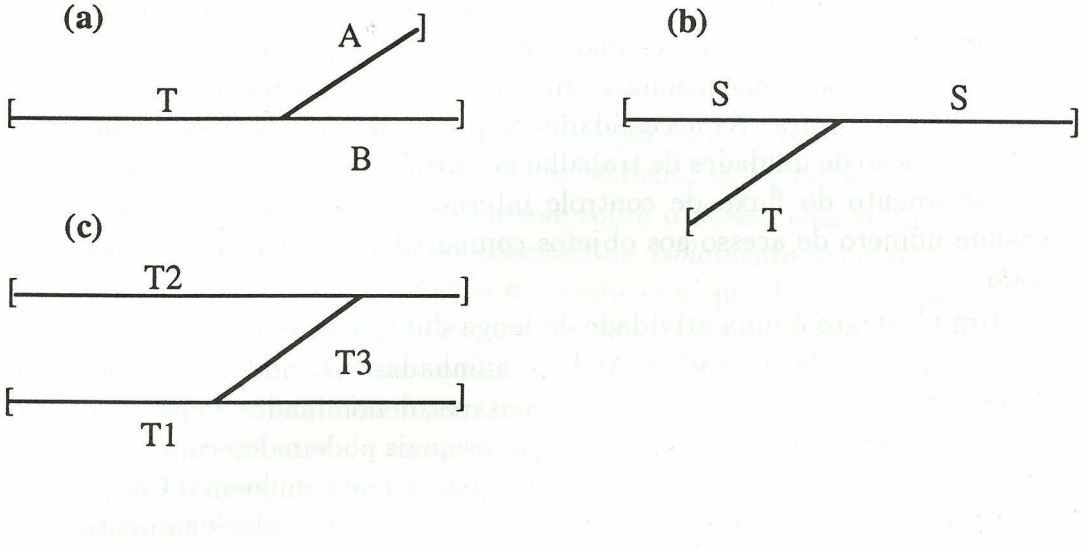

Figura 6: (a) operação de divisão (*split transaction*); (b) operação de agrupamento (*join transaction*); (c)combinação das operações de divisão e agrupamento [MOH 94].

se unem para produzir um efeito desejável. Por exemplo, a versão *DEMO* de um software pode necessitar que um conjunto de módulos interrelacionados sejam agrupados, de forma inesperada, para uma demonstração a um possível cliente.

Apesar do modelo permitir o relaxamento da atomicidade de falhas, ele realiza recuperação totalmente orientada à transação, como no modelo convencional. Quanto aos critérios de correção do BD, o modelo realiza o controle igualmente orientado à transação. Entre as transações que são divididas ou agrupadas, o critério *serializabilidade* garante a consistência do BD no que diz respeito ao acesso concorrente aos dados que são compartilhados por estas ou por outras transações em andamento. Entretanto, o modelo não prevê o controle sobre a atividade de longa duração como uma unidade lógica de trabalho. Conseqüentemente, não há controle sobre os possíveis resultados obtidos a partir de reestruturações de transações em tempo de execução.

5.1.4 *ConTracts*

A grande motivação do modelo proposto por Wächter e Reuter é prover um suporte adequado às necessidades de processamento de grandes aplicações distribuídas e não-convencionais, em ambientes de Escritório, Projeto e Controle de Manufatura. As necessidades de processamento básicas consideradas são: execução de unidades de trabalho de curta e longa duração, distribuição, gerenciamento do fluxo de controle interno de uma atividade complexa e grande número de acesso aos objetos compartilhados pelo sistema como um todo.

Um Contrato é uma atividade de longa duração que pode ser organizada como um grafo de transações ACID e aninhadas. Os nodos do grafo representam passos de processamento elementares, denominados *steps*. Seus arcos definem as dependências entre os passos, os quais podem determinar qualquer ordem de execução entre o conjunto dos passos que compõem o Contrato. O conceito de *script* representa a definição explícita do relacionamento entre os diversos passos de processamento. Ele constitui-se no mecanismo central para estender o controle além dos limites de uma transação.

O ambiente onde executa um Contrato inclui três diferentes modelos: o modelo de transação; o modelo de programação e o modelo de execução.

O Modelo de transação

No modelo de transação, duas unidades lógicas de trabalho da aplicação são consideradas: a do Contrato propriamente dita e a de um *step*. O Contrato como um todo não representa uma unidade ACID, uma vez que, como uma unidade de longa duração, ele relaxa as propriedades de atomicidade e isolamento das transações. Um Contrato pode ser interrompido e continuar sua execução após um período de tempo arbitrário. Além disso, uma falha de sistema, ou transação, não dispara um procedimento de *roll-back*. Ao invés disso, o sistema inicia uma recuperação do tipo progressiva, no sentido de executar os passos pendentes e finalizar a execução do Contrato. Conseqüentemente, o modelo suporta estratégias de recuperação do tipo *forward recovery* em caso de falha. Para realizar o esquema de recuperação progressiva e executar os passos que ainda não tinham iniciado quando a falha ocorreu, o modelo prevê que toda a informação de estado que o próximo passo necessita para ser executado seja recuperável. No âmbito deste modelo, o conjunto destas informações é referenciado como *contexto*. O *contexto* deve incluir todas aquelas informações que não são representáveis no modelo de

dados da aplicação de BD, mas que são essenciais para garantir a continuação do Contrato interrompido após uma falha do sistema.

O modelo igualmente permite que a própria aplicação defina seus procedimentos de recuperação em caso de falha de uma transação (transações de contingência). Por exemplo, na Figura 3 (seção 4.1), T2 irá executar somente se T1 falhar, assim como S8 irá executar somente se T2 falhar.

Ao nível de um Contrato, o controle sobre o acesso concorrente aos dados compartilhados baseia-se no conceito de *isolamento semântico* o qual, por sua vez, baseia-se na avaliação de predicados que descrevem os estados compartilhados. Os critérios de correção são expressos através de *predicados invariantes*, os quais são avaliados antes e depois da execução de um passo. Se estes predicados forem definidos sobre uma estrutura de dados que descreve os estados de execução dos passos de diferentes Contratos, estas invariantes podem ser utilizadas para sincronizar a execução de outros Contratos independentes.

Cada *step* é modelado como uma transação ACID convencional. Ele representa a menor unidade lógica de trabalho da aplicação, como por exemplo, a atividade de efetuar a reserva de uma passagem aérea em algum vôo. A um *step*, somente procedimentos de recuperação do tipo *undo*, *redo* e transações de compensação são permitidos. O modelo permite que a aplicação especifique, para cada *step* do *script*, uma ação de compensação que desfaça, sob o ponto de vista semântico, as modificações sobre os objetos do BD manipulados no respectivo *step*.

O modelo também permite à aplicação especificar unidades de trabalho atômicas, resultantes da composição ou aninhamento de outras transação ou *steps*. No exemplo da Figura 3 (seção 4.1), as transações T1 e T2 são formadas pelo agrupamento dos passos S4, S5 e S6, S7, respectivamente. Além do agrupamento de *steps*, a aplicação poderia aninhar transações, que resultassem do agrupamento de outras transações, como em uma estrutura hierárquica de transações. Por exemplo, a partir de T1 e T2, poderíamos compor T3 da seguinte forma:

T3 (T1, T2),

ou seja, T3 seria pai de T1 e T2. Para estes casos, os autores não explicitam quais seriam os protocolos de *commit* ou *abort* entre transações pais e suas filhas. Poderíamos imaginar uma implementação com os mesmos protocolos enunciados por Moss para Transações Aninhadas.

O modelo de programação

O modelo de programação de um Contrato define dois níveis distintos de programação: o nível dos *steps* e o nível dos *scripts*. Isto significa programadores diferentes para cada nível, que executam o seu trabalho em, possivelmente, locais físicos distintos.

Cada *step* implementa o algoritmo propriamente dito de um passo de processamento elementar. Ele pode ser codificado em uma linguagem de programação seqüencial arbitrária, não havendo nenhum paralelismo interno a ele associado. O programador de *steps* codifica o seu algoritmo sem considerar a sua possibilidade de ser combinado com outro *steps* para formarem, juntos, novas unidades de trabalho maiores, e sem considerar, também, os aspectos de sincronismo, paralelismo, comunicação, distribuição de recursos (localização) e recuperação em caso de falhas. Estas questões são especificadas a nível do *script*.

Por outro lado, o programador de *script* deve especificar o fluxo de controle do Contrato, assim como outras estratégias de execução deste. A computação que representa o *script* é um programa com fluxo de controle como qualquer ambiente de programação paralela, que dispõe de variáveis locais persistentes, acesso a objetos compartilhados, mecanismos de sincronização orientado à aplicação e uma semântica de erros precisa.

O modelo de execução

Para um Contrato, todos os aspectos referentes ao controle de execução, em tempo de execução, são controlados pelo *gerente de Contrato*. Internamente, o *gerente de Contrato* implementa o gerenciamento do fluxo de controle orientado à eventos. Para realizar isto, ele utiliza uma rede de transição de predicados, a qual especifica as condições de ativação e término de um passo de processamento. A execução de um passo é iniciada se o conjunto de predicados associados a sua ativação tornarem-se verdadeiros e os recursos necessários a sua execução estiverem disponíveis. Por exemplo, o *step* S3 (Fig. 3) é disparado quando todos os três passos paralelos de passo S2 tiverem concluído. O passo interativo S1 necessita que o usuário responsável esteja pronto para fornecer os dados de entrada, etc.

O disparo de eventos após o término de um passo pode ser controlado por um conjunto de condições. Cada condição verdadeira que é avaliada dispara um ou mais eventos. Isto, por sua vez, dispara os passos subseqüentes.

5.1.5 Transações de Cooperação

O Modelo de Transação de Cooperação, proposto em [NOD 90, NOD 92], enfoca os principais requisitos de processamento das aplicações de Projeto. Estes requisitos incluem o suporte à execução de atividades de longa duração, possivelmente não atômicas, que estruturam-se de forma hierárquica, possivelmente em tempo de execução, e que precisam interagir umas com as outras na execução de uma tarefa.

Com o objetivo de atender a estes requisitos, os autores introduzem o conceito das *hierarquias de transação de cooperação.* Uma hierarquia de transação de cooperação é um conjunto estruturado de transações de cooperação, cujas interações são estruturadas de modo a refletir a decomposição hierárquica básica da tarefa que estas transações modelam [NOD 92].

Neste modelo, cada tarefa da aplicação (ex.: projeto de engenharia ou de software) é modelada como uma *transação de grupo* (TG). As TGs são representadas pelos nodos internos da estrutura. Elas são formadas por um conjunto de transações membro, que cooperam entre si na execução correta da tarefa. Tarefas podem ser subdivididas em subtarefas. Assim, TGs podem ter como membros outras TGs, bem como conjuntos de *transações de cooperação* (TC). Uma TC é uma transação de longa duração, do tipo *open-ended*, que modela o trabalho individual de um projetista. As TCs são representadas pelos nodos folha da estrutura e são compostas por uma coleção de *operações*, associadas a alguma tarefa específica. Uma *operação* define uma ação atômica executada por um membro sobre um objeto. A transação raiz sempre existe e representa uma *transação de grupo.* A Figura 7 ilustra a organização do projeto de um novo carro segundo a concepção de Nodine, Ramaswamy e Zdonik [NOD 92]. Nesta caso, o projeto do *carro*, *carroceria*, *motor* e *interior* são modelados como TGs. O *bloco do motor*, *carburador* e *verificação* são modelados como CTs. O procedimento de verificação é executado para garantir que o motor esteja de ocordo com as normas técnicas obrigatórias.

O processo de cooperação ocorre entre as TCs de um mesmo grupo. As interações entre elas se dão através de operações sobre objetos do BD, associados a TG da qual estas TCs fazem parte. Os co-usuários da aplicação (normalmente projetistas) constróem e modificam a hierarquia de transações dinâmicamente à media que o esforço do projeto avança. A partir da TG raiz, que diretamente mantém o BD ao longo de toda a tarefa, novas TGs e TCs são agregadas à hierarquia sempre que necessário, assim como outras podem ser agrupadas em uma única.

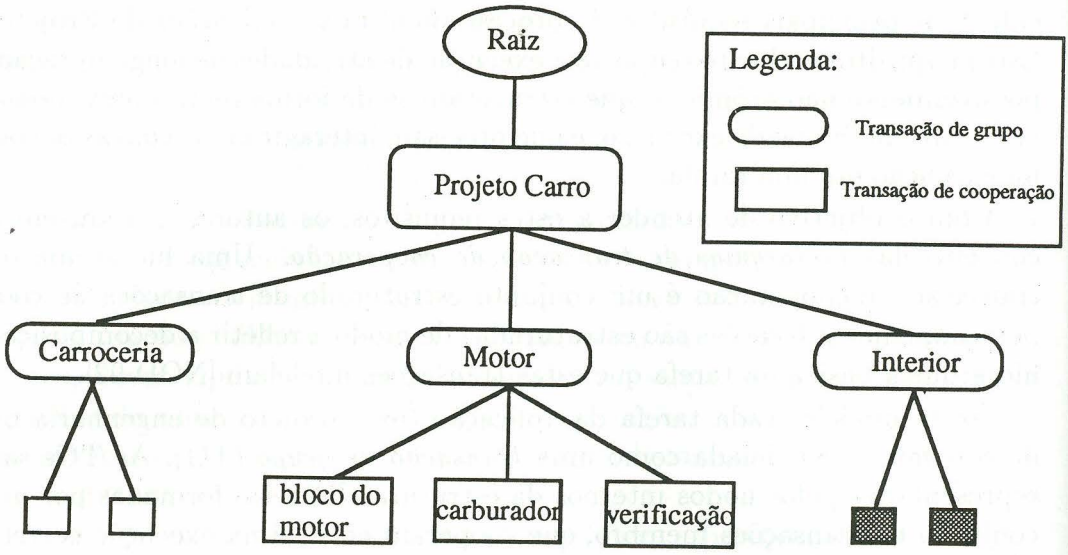

Figura 7: O projeto do carro como uma hierarquia de transações [NOD 92]

Segundo os autores, no domínio das aplicações onde existe cooperação, a noção do que é correto pode variar de aplicação para aplicação e de tarefa para tarefa. Isto impõe a necessidade de que o próprio usuário especifique seus critérios de correção. Para atender este requisito destas aplicações, o modelo prevê que cada TG tenha a ela associada sua *especificação de correção*, que considere a semântica da aplicação. A especificação de correção de uma TG, por sua vez, está associada às noções de especificação de *padrão* (*patterns*) e *conflitos* (*conflicts*), primeiramente referenciadas em [SKA 89, SKA 91]. Estes conceitos definem correção com base nas histórias que a TG deve ou pode produzir. História, neste contexto, é uma seqüência de chamadas de operações efetuadas pelos membros da TG. As especificações de *Padrão* descrevem seqüências obrigatórias de chamadas de operações que precisam ocorrer em uma história válida. As especificações de *Conflitos* são definidos no contexto de *padrões* e descrevem seqüências proibidas de chamadas de operações, que não podem ocorrer em uma história válida. O mecanismo de controle de acesso a dados compartilhados consiste em determinar se uma dada operação que está sendo chamada pode ser aceita ou deverá ser rejei-

tada. Esta decisão depende da avaliação dos *padrões* e *conflitos* associados a TG da qual a operação faz parte.

Para suportar o requisito de recuperação orientada à aplicação quando da ocorrência de falhas, o modelo prevê que não somente a história de um grupo de transações seja mantida pelo SBD, mas também as dependências entre as operações nesta história. É garantido que apenas os efeitos das operações direta ou indiretamente afetadas pelo aborto sejam eliminadas do BD, ao invés de abortar a transação como um todo.

Três tipos de dependências são consideradas: 1) dependências de *padrão*: dependências entre as operações que participam daquela especificação de *padrão*; 2) dependências de leitura: uma operação de leitura sobre uma versão de objeto por um membro M somente está correta se a operação que gravou aquela versão também está correta. Se a operação de gravação for invalidada, a operação de leitura também o será; e 3) dependências entre pais e filhas: quando um membro M lê a versão de um objeto, sua TG pai deve ter uma cópia do objeto para passar a M. Se posteriormente a versão de TG é invalidada, a versão de M também o será. As dependências para cada operação são mantidas no *log*.

Quando ocorre uma falha, os procedimentos de recuperação do BD incluem duas fases distintas: 1) *invalidação*: nesta fase, todos os efeitos das operações abortadas são eliminados do BD; e 2) *recuperação*: nesta fase, os membros que tiveram operações invalidadas determinam sobre as operações de compensação a serem executadas, as executam e disparam um procedimento do tipo *redo* para as operações que ainda não haviam executado *checkpoint*.

5.2 Resumo dos modelos apresentados

A Figura 8 resume as principais características dos modelos apresentados. Os modelos são listados na ordem em que aparecem no texto.

Na coluna Tipos de Transação são listados os tipos de transação suportados pelo modelo em função do seu tempo de duração. Na coluna Estrutura das Transações, *hierarquia de subtransações* significa uma árvore arbitrária de subtransações e, *subtransações* representa um conjunto de transações curtas. A coluna Critérios de Correção descreve o nível de correção das transações que o modelo oferece. Por fim, as colunas Protocolos de Recuperação e Resolução de Conflitos apresentam, respectivamente, as ações de recuperação

suportadas pelo modelo em situações de falha e os critérios de correção considerados quando da ocorrência de conflitos, a partir do acesso concorrente a dados compartilhados.

Modelo	Tipos de Transações	Estrutura das Transações	Critérios de Correção	Protocolos de Recuperação	Resolução de Conflitos	Comentários
Convencional	Curtas	Plana	ACID	*undo, redo*	Automática, baseada em Serializabilidade	
Transações Aninhadas	Curtas	Hierarquia de subtransações	ACID p/ raiz; ACI p/ subtransações	*undo/redo* transações de contingência e não-vitais	Automática, baseada em Serializabilidade	Sistemas distribuídos
SAGAS	Curtas (subtransações) longas (SAGA)	Subtransações	ACID p/ subtransações	*undo/redo* e compensatórias p/ subtransações, *backward/ forward* p/ SAGAS	Automática, baseada em Serializabilidade p/ Subtransações	Sistemas de Automação Escritório
Split Transactions	Curtas/ Longas	Plana	ACID	*undo/redo*	Automática, baseada em Serializabilidade	Sistemas Cooperativos c/ reestrutur. dinâmica de transações
Contracts	Curtas (*steps*)/ Longas (Contrato)	Subtransações c/ possibilidade de agrupamento e aninhamento	ACID p/ *steps* ; bloqueio semântico p/ o Contrato	*undo/redo* e compensatórias p/ *steps*; *backward/ forward* e trans. contingência p/ o Contrato	critérios definidos pelo usuário	Sistemas de Automação Escritório e Projeto Assistido por Computador
Transações de Cooperação	Curtas (operações sobre objetos); Longas (TGs e TCs)	Hierarquia de subtransações	ACID p/ operações; especificações de *padrões* e *conflitos* para as TGs	Aplicação controla a execução dos procedimentos de *undo/redo*	critérios definidos pelo usuário	Sistemas Cooperativos c/ reestrutur. dinâmica de transações

Figura 8: Características dos modelos de transação não convencionais

6. Comparação dos modelos de transação não convencionais analisados em relação aos novos requisitos

Os modelos de transação analisados neste trabalho permitem a modelagem de algumas das importantes características das aplicações não convencionais. A Figura 9 apresenta o quadro comparativo dos modelos analisados em relação aos requisitos discutidos na seção 4.2. Para cada modelo é indicado quão bem (ou mal) ele suporta cada um dos requisitos.

Conforme pode-se observar na Figura 9, o modelo ACID não apresenta suporte adequado a nenhum dos requisitos discutidos neste trabalho. No entanto, todos os modelos analisados representam, de uma forma ou outra, extensões ao modelo convencional de transação.

Apesar de não enfocar o conceito de transação de longa duração, o modelo de Transação Aninhada proposto por Moss atende ao requisito de organizar uma atividade complexa como uma hierarquia de subtransações. Esta solução é especialmente adequada em ambientes onde existem atividades que desenvolvem sub-atividades, tais como as atividades normalmente desenvolvidas em ambientes de Projeto. A vantagem disto é aumentar o paralelismo interno às transações diminuindo, possivelmente, o tempo e o custo de desenvolvimento de um projeto.

No modelo original de Transações Aninhadas, a estrutura hierárquica das transações deve estar completamente definida antes que a transação raiz inicie a sua execução. É suposto que transações aninhadas competem pelo acesso à informação. A decisão sobre conflitos baseia-se no critério da *serializabilidade*. Esta característica supõe transações executando de forma isolada, o quê é por demais restritivo às transações que precisam cooperar durante a execução de alguma tarefa. Portanto, não há possibilidade de o modelo suportar o requisito de cooperação.

Assim, conclui-se que este modelo não considera os aspectos relativos à modelagem de atividades de longa duração, cuja estrutura de execução precisa reestruturar-se dinâmicamente e cujas atividades precisam cooperar para atinjir os seus objetivos. Por suportar a especificação de transações de *contingência então-vitais*, considera-se que o modelo suporta, de modo insuficiente, o requisito de recuperação orientada à aplicação. Por outro lado, o modelo não suporta o requisito referente a definição e manutenção de

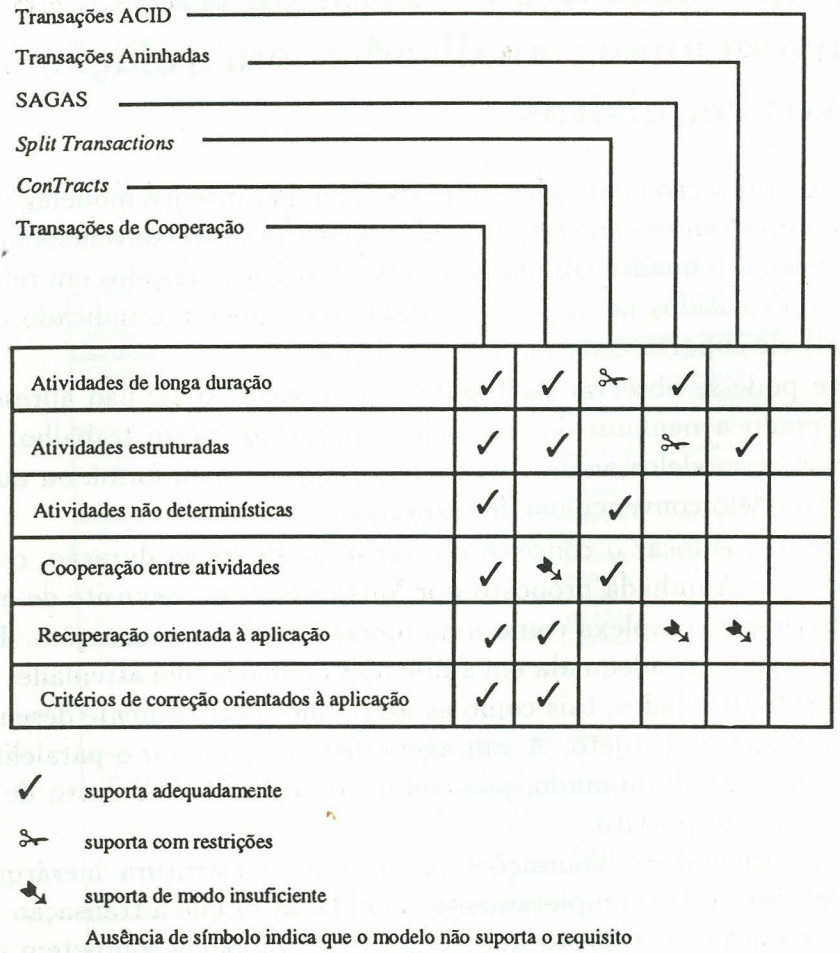

Figura 9: Como cada modelo suporta os requisitos das aplicações não convencionais

critérios de correção do BD orientados à aplicação.

O modelo SAGAS, por sua vez, suporta adequadamente a modelagem de atividades de longa duração, cujas subtransações executam seqüencialmente. Entretanto, o modelo não suporta aninhamentos arbitrários de subtransações. Nesta análise, foi investigado o modelo SAGAS original e não suas extensões. O modelo SAGAS original suporta apenas dois níveis de aninhamento de atividades: o nível da SAGA e de suas subtransações. Todas as subtransações

de uma SAGA precisam estar definidas antes que ela inicie a sua execução, motivo pelo qual o modelo não suporta a reestruturação dinâmica das subtransações.

O modelo SAGAS não suporta cooperação entre suas subtransações, assim como não permite ao usuário definir seus próprios procedimentos de recuperação ou protocolos para resolução de conflitos.

O modelo de reestruturação dinâmica de transações, ou *Split Transactions,* foi proposto para atender as necessidades de processamento em ambientes cujas atividades são de longa duração, não determinísticas, *a priori*, e que precisam interagir umas com as outras para a execução de alguma tarefa. O suporte que o modelo oferece à estas necessidades de processamento permite ao usuário controlar, de forma interativa, o tempo de duração de uma transação e o momento a partir do qual uma transação deve ser subdividida (operação *split-transaction*) ou agrupada (operação *join-transaction*), com o objetivo de liberar ou delegar responsabilidades, ou reunir recursos para alguma finalidade prática.

Split-Transactions realiza recuperação orientada à transação como no modelo convencional. O critério básico de correção quando do acesso concorrente a dados compartilhados é o da *serializabilidade.* Quanto ao critério de correção do BD, Elmagarmid et al. [ELM 92] fazem a seguinte consideração. No modelo *Split-Transactions*, é possível que o conjunto de transações que executam *commit* seja diferente do conjunto de transações submetido inicialmente, em função da reestruturação dinâmica das transações. Entretanto, considerando que as transações originais executam como unidades ACID e que as trasnsações que executam *commit* também executam como unidades ACID, a semântica original das transações é preservada.

O modelo não suporta a execução de uma atividade, definida como uma estrutura complexa de execução.

O modelo *ConTracts* pode ser visto como uma extensão ao modelo SAGAS, no sentido de que ele modela atividades de longa duração, subdivididas em passos de processamento elementares (*steps*). A vantagem do modelo *ConTracts* é suportar, de forma adequada, qualquer ordem de execução entre o conjunto dos passos que o compõem, através do *script* do Contrato. Assim como em uma SAGA, o *script* de um Contrato deve ser especificado, *a priori*, pelo programador de *script.* Isto significa que o modelo não suporta reestruturação dinâmica de suas transações.

Por permitir que resultados parciais tornem-se visíveis antes que o Con-

trato execute *commit*, o modelo possibilita cooperação entre seus passos de processamento (*steps*), muito embora não suporte, de forma explícita, este requisito de processamento de algumas aplicações não convencionais [ELM 92].

Um Contrato sempre deve executar *commit*. Portanto, o modelo suporta estratégias de recuperação do tipo *forward recovery*, permitindo ao usuário especificar, através de transações de contingência, os procedimentos a serem executados quando da ocorrência de uma falha do sistema ou transação. A nível de um *step*, valem somente os procedimentos de *undo*, *redo* e transações de compensação. Transações de contingência, assim como transações de compensação são especificadas no *script* do Contrato.

O modelo de Contrato suporta que a própria aplicação defina seus critérios de correção do BD quando do acesso concorrente a dados compartilhados.

O modelo das *hierarquias de transação de cooperação*, por sua vez, pode ser visto como uma extensão ao modelo de Transações Aninhadas de Moss. Os nodos internos são representados pelas transações de grupo (TGs), enquanto que os nodos folha são representados pelas transações de cooperação (TCs). TCs de um mesmo grupo cooperam através da execução de suas operações sobre os objetos associados a TG pai. TGs e TCs representam unidades de trabalho de longa duração, que podem ser reestruturadas enquanto executam.

Uma TG nunca aborta. Quando da ocorrência de uma falha, o modelo prevê que apenas os efeitos de operações diretamente afetadas pela falha sejam eliminados do BD, ao invés de abortar toda a transação. Através de uma especificação de dependências, o usuário informa ao SBD que operações devem abortar. Os critérios de correção do BD são totalmente especificados pela aplicação. Cada TG tem a ela associada um conjunto de especificações que indicam o quê é correto (especificações de *padrão*) e o quê é probibido (especificações de *conflitos*).

7. Considerações finais

Este trabalho abordou sobre as principais características dinâmicas das aplicações não convencionais em ambientes de escritório e projeto. Foram derivados seis requisitos de processamento das novas aplicações de BD: 1) suporte à execução de atividades de longa duração; 2) suporte à execução de

atividades estruturadas; 3) suporte à reestruturação dinâmica de atividades não determinísticas; 4) suporte ao trabalho cooperativo entre atividades; 5) suporte à recuperação orientada à aplicação; 6) suporte às definição e manutenção de critérios de correção do BD orientados à aplicação. Não consideramos que todos os modelos de transação não convencionais devam suportar todos estes requisitos. A necessidade de suportar um ou outro requisito depende do ambiente e de suas características dinâmicas.

Os modelos de Transações Aninhadas, SAGAS, *Split-Transactions*, Transações de Cooperação e *ConTracts*, analisados neste trabalho, representam importantes marcos nesta área do conhecimento. Eles implementam pelo menos um dos requisitos acima mencionados.

De um modo geral, todos os modelos de transação para aplicações não convencionais, que têm sido propostos na literatura, fundamentam-se no *paradigma das transações ACID*. A partir da análise e comparação dos modelos descritos neste trabalho, observa-se a preocupação de representar unidades de trabalho de longa duração, que organizam-se de forma estruturada, e que precisam trocar informações enquanto executam. A modelagem destas características das aplicações não convencionais impõe um grande desafio aos pesquisadores da área.

Para representar unidades de trabalho de longa duração, é inaceitável que o SBD as trate como unidades *atômicas*, que executam de forma *isolada*. A questão da atomicidade das transações longas tem sido abordada considerando que, como uma unidade lógica de trabalho da aplicação, uma atividade de longa duração define uma unidade de trabalho *atômica*. Entretanto, como uma unidade de recuperação em situações de falha, a transação que modela a atividade de longa duração precisa relaxar a sua atomicidade, no sentido de que, pode ser absurdo desfazer, no BD, todo o trabalho já executado pela transação. A questão do SBD tratar uma transação de longa duração como uma unidade que executa de forma *isolada*, acarreta sérios problemas de desempenho do sistema de banco de dados. A solução que tem sido adotada, consiste em fazer com que a transação longa libere resultados parciais enquanto executa. Uma vez que ocorra a liberação de resultados antes da execução de *commit* da transação longa, a oportunidade para que ela troque informações com outras transações longas surge naturalmente. A liberação de resultados parciais, bem como uma possível troca de informações entre transações em andamento impõem a especificação de novos critérios de correção do BD, em substituição ao critério da *serializabilidade*, apresentado

em [BER 87], para transações convencionais.

Em um ambiente onde executam transações longas, é lógico imaginar que o conjunto de suas ações seja estruturado em diferentes níveis de abstração, possibilitando uma melhor compreensão do problema, bem como um melhor desempenho do sistema de banco de dados no sentido de possibilitar a execução de ações em paralelo e cancelamento de determinadas ações sem implicar no cancelamento da atividade como um todo. Para representar unidades de trabalho estruturadas (complexas), o conceito de *script*, apresentado no modelo ConTracts, surge como uma alternativa de solução. Neste modelo, o controle sobre os diferentes níveis de abstração (representados no *script*), em tempo de execução, é responsabilidade de um novo componente situado entre o SBD e a aplicação, denomindado *gerente do contrato*. Este controle inclui a execução da aplicação tanto em situação normal de processamento, quanto na presença de falhas do sistema ou transação.

Até o presente momento, a exceção do modelo de Transações Aninhadas de Moss, os SBDs que implementam os novos modelos de transações apresentados neste trabalho existem apenas a nível experimental (protótipos), ou nem isso. Ainda existe uma série de questões em aberto que, imagina-se, dificultando tais implementações. Conforme observa Mohan [MOH 94], atualmente, encontram-se na literatura diversos modelos de transação para aplicações não convencionais, para os quais ainda não existe nenhuma implementação de fato.

Como um dos modelos de transação não convencional mais antigos, o modelo de Transações Aninhadas é o único já implementado em sistemas comerciais ou projetos de pesquisa em operação, tais como o sistema Camelot [EPP 91], o sistema Locus da Universidade da Califórnia (UCLA), Encina [EPP 92], Argus do Instituto MIT (Massachussetts Institute of Technology). Dentre os demais modelos apresentados neste trabalho, somente o ConTracts tem um protótipo implementado, denominado APRICOTS (A PRototype Implementation of a COnTract System) [SCH 93], desenvolvido na Universidade de Sttutgart na Alemanha.

Uma das maiores dificuldades por que passam estes modelos para serem integrados a produtos comerciais é o iato existente entre a visão que a aplicação tem das transações do modelo e as interdependências existentes entre elas ao nível do controle de concorrência, recuperação em caso de falhas, distribuição e terminação, entre outros. Para solucionar este problema, novas características devem ser introduzidas às linguagens de alto nível para

que, na definição das transações e dos *scripts*, ao nível da aplicação, possa-se tirar proveito das alternativas de controle de concorrência, distribuição, compensação, terminação e recuperação oferecidas pelos novos modelos.

Outro aspecto importante que merece maior estudo diz respeito à possibilidade que os modelos devem oferecer de que *scripts* previamente definidos possam ser modificados dinamicamente, durante a execução das transações longas. Em cada um dos modelos apresentados aqui, modificações no fluxo de tarefas, em tempo de execução das transações, implicam em um controle de vários tipos de interdependências entre subtransações.

Para aqueles modelos que suportam recuperação do estado da transação longa por meio de (sub)transações compensatórias, seria desejável, para efeitos de performance e flexibilidade, que fosse dada, à aplicação, a possibilidade de definir conjuntos de estados passíveis de serem reconstruídos, a partir de combinações de transações compensatórias a serem executadas. A partir dessas definições, a aplicação teria liberdade, a cada momento, de decidir qual o caminho a ser seguido pelo mecanismo de *recovery* do sistema e qual o estado anterior da transação a ser restituído.

Uma última observação que consideramos importante fazer no contexto deste estudo é a de que a pesquisa de modelos transacionais para aplicações das áreas de engenharia, indústria e automação de escritórios deveria convergir, em algum futuro próximo, para a investigação da interoperabilidade de transações de modelos diferentes. Em outras palavras, os pesquisadores da área devem, agora, investigar modelos transacionais que generalizam os modelos propostos até agora. Sem isso, será sempre muito difícil integrar aplicações construídas com base nos modelos apresentados aqui em sistemas de banco de dados heterogêneos.

Referências

[BAR 91] BARGHOUTI, Naser S.; KAISER, Gail E. Concurrency control in advanced database applications.

[BER 87] BERNSTEIN, P.A.; HADZILACOS, V.; GOODMAN, N. **Concurrency control and recovery in database systems**. Reading: Addison-Wesley, 1987. 370p.

[DAY 91] DAYAL, U.; HSU, M. LADIN, R. A Transaction model for long-running activities. In: INTERNATIONAL CONFERENCE on VERY LARGE DATABASE SYSTEMS, 17, 3-6 Sept. 1991, Barcelona. **Proceedings ...** New York: IEEE, 1991. p. 113-122.

[ELM 92] ELMAGARMID A. K. et al. Introduction to advanced transaction models. In: ELMAGARMID, Ahmed k. (Ed.). **Database Transaction Models for Advanced Applications**. San Mateo, California: Morgan, 1992. 611p. p. 33-52.

[EPP 91] EPPINGER, Jeffrey L.; MUMMERT, Lily B.; SPECTOR, A. **Camelot and Avalon: A Distributed Transaction Facility**. San Mateo: Morgan Kaufmann Publishers, INC, 1991. 505p.

[EPP 92] EPPINGER, Jeffrey L.; DIETZEN, Scott. Encina: Modular Transaction Processing. In: COMPCON SPRING, San Francisco, feb 24-28, 1992. **Digest of papers**. Los Alamitos, IEEE Computer Society Press, 1992. p. 378-382.

[GAR 87] GARCIA-MOLINA, H.; SALEM, K. SAGAS. In: ACM-SIGMOD on MANAGEMENT of DATA, 1987, San Francisco. **Proceedings ...** New York: ACM, 1987. p. 249-259.

[GRA 81] GRAY, Jim. The transaction concept: virtues and limitations. In: INTERNATIONAL CONFERENCE on VERY LARGE DATABASE SYSTEMS, 7., 1981, Cannes, France. **Proceedings ...** New York: IEEE, 1981. p. 144-154.

[HÄR 83] HÄRDER, T.; REUTER, A. Principles of transaction-oriented database recovery. **ACM Computing Surveys**, New York, v. 15, n. 4, p. 287-317, Dec. 1983.

[HÄR 88] HÄRDER, T.; ROTHERMEL, K. **Concurrency control issues in nested transactions**. San Jose: California, [1988?]. 25p. (IBM Research Report).

[HAS 81] HASKIN, R.L.; LORIE, R.A. **On extending the functions of a relational database system**. San Jose, California: IBM Research Laboratory, 1981. 13p. (Research Report RJ3182).

[IOC 91] IOCHPE, Cirano. Modelos de processamento em sistemas de banco de dados para aplicações de CAD. **Revista de Informática Teórica e Aplicada**, Porto Alegre, v.1, n.3, p. 41-62, mar. 1991.

[KÄF 90] KÄFER, W. **A Framework for version-based cooperation control**. Kaiserslautern: Universität Kaiserslautern, 1990. 20p. (Research Report)

[KAI 92] KAISER, G. E.; PU, Calton. Dynamic reestructuring of transactions. In: ELMAGARMID, Ahmed k. (Ed.). **Database Transaction Models for Advanced Applications**. San Mateo, California: Morgan, 1992. 611p. p. 265-295.

[KLA 85] KLAHOLD, P. et al. A Transaction model supporting complex applications in integrated information systems. In: ACM-SIGMOD INTERNATIONAL CONFERENCE on MANAGEMENT of DATA, 1985, Austin, Texas. **Proceedings ...** New York: ACM, 1985. p. 388-401.

[KLE 88] KLEIN, J.; REUTER, A. Migrating transactions. In: WORKSHOP on FUTURE TRENDS of DISTRIBUTED COMPUTER SYSTEMS IN THE 90s, 1988, Hong Kong.**Proceedings ...** New York: IEEE, 1988.

[KOR 87] KORTH, H.F.; KIM, W.; BANCILHON, F. On Long-duration CAD transactions. **Information Sciences**, New York, v. 46, p. 73-107. 1987.

[LIV 91] LIVI, Maria Aparecida C. **Estudo de requisitos e proposta de um modelo de trabalho cooperativo para o ambiente de projeto AMPLO**. Porto Alegre, CPGCC da UFRGS, 1991. 33p. (Trabalho Individual).

[MOH 94] MOHAN, C. Tutorial: Advanced Transaction Models - Survey and Critique. In: ACM-SIGMOD INTERNATIONAL CONFERENCE on MANAGEMENT of DATA, Minneapolis, 1994. 56p.

[MOS 82] MOSS, J. Nested Transactions and reliable distributed computing. In: IEE SYMPOSIUM on RELIABILITY in DISTRIBUTED SOFTWARE and DATABASE SYSTEMS, 2., Jul. 1982, Pittsburgh. **Proceedings ...** New York: IEE, 1982. p. 33-39.

[NOD 90] NODINE, Marian; ZDONIK, S. Cooperative transaction hierarquies: a transaction model to support design applications. In: INTERNATIONAL CONFERENCE on VERY LARGE DATABASE, 16., Brisbane, 1990. **Proceedings ....**

[NOD 92] NODINE, Marian H.; RAMASWAMY, S.; ZDONIK, S. A Cooperative transaction model for design databases. In: ELMAGARMID, Ahmed k. (Ed.). **Database Transaction Models for Advanced Applications**. San Mateo, California: Morgan, 1992. 611p. p. 53-85.

[PUC 88] PU, Calton; Kaiser, G.; HUTCHINSON, N. Split transactions for open-ended activities. In: VLDB CONFERENCE, 14., 1988, Los Angeles: California. **Proceedings ...** [s.l.: s.n.], 1988.

[SCH 93] SCHWENKREIS, F. APRICOTS - A Prototype Implementation of a ConTract System: Management of the Control Flow and the Communications System. In: SYMPOSIUM on RELIABLE DISTRIBUTED SYSTEMS, 12, 1993.

[SKA 89] SKARRA, Andrea H. Concurrency control for cooperating transactions in an object-oriented database. **SIGPLAN Notices**, 24(4), April 1989.

[SKA 91] SKARRA, Andrea H. Localized correctness specifications for cooperating transactions in an object-oriented database. **Office Knowledge Engineering**, 4(1):79-106, 1991.

[SOU 92a] SOUTO, Maria Aparecida M. **Modelos de transação para suportar sistemas de controle de produção**. Porto Alegre, CPGCC da UFRGS, 1992. 135p. (Dissertação de Mestrado).

[SOU 92b] SOUTO, Maria Aparecida M.; IOCHPE, Cirano. Suporte à Execução de Planos de Processo Industriais com Base em Transações de Banco de Dados. In: CONGRESSO DA SOCIEDADE BRASILEIRA DE COMPUTAÇÃO, 12., 1992, Rio de Janeiro. **Anais ...** Rio de Janeiro: SBC, 1992. 390p. p. 123-136.

[SOU 93] SOUTO, Maria A.M.; IOCHPE, Cirano. Transaction-Based Support for Production Plan Execution: A Position Paper. In: SIMPÓSIO

BRASILEIRO DE AUTOMAÇÃO INTELIGENTE, 1., 1993, Rio Claro, SP, BRASIL. **Anais ...** Rio Claro: UNESP, 1993. 542p.

[WÄC 92] WÄCHTER, H.; REUTER, A. The ConTract Model. In: ELMAGARMID, Ahmed k. (Ed.). **Database Transaction Models for Advanced Applications**. San Mateo, California: Morgan, 1992. 611p. p.219-263.

[YEH 87] YEH, S.; ELLIS, C.; EGE, A.; KORTH, H. **Performance analysis of two concurrency control schemas for design environments**. Tech. Rep. STP-036-87, MCC, Austin, Texas.

Autores:

Maria Aparecida M. Souto: Professora Adjunta no Departamento de Informática da UFRGS, Mestre em Ciência da Computação no CPGCC- UFRGS. Áreas de interesse: Bancos de Dados para Aplicação Não Convencionais, modelos de transação em SBDs não convencionais.

Cirano Iochpe: Professor Adjunto no Departamento de Informática Aplicada da UFRGS, Doutor em Informática pela Universidade de Karlsruhe, Alemanha. Área de interesse: implementação de sistemas de banco de dados (SBDs), modelos de transação em SBDs (convencionais ou não), SBDs para aplicações de engenharia e indústria, investigação sobre cooperação em atividades de projeto e produção.