Um Modelo de Transações Cooperativas Integrado a um Modelo de Versões

George Marconi de Araújo Lima

Laboratório de Sistemas Distribuídos - LaSiD
Centro de Processamento de Dados - CPD
Universidade Federal da Bahia - UFBA
Av. Ademar de Barros, SN
40.000-000 - Salvador - BA

E-mail:gmlima@lasid.ufba.br

Maria Beatriz F. de Toledo

Instituto de Computação - IC
Universidade Estadual de Campinas - UNICAMP
Cidade Universitária Zeferino Vaz
13081-970 - Campinas - SP

E-mail:beatriz@dcc.unicamp.br

Resumo

Transações têm por objetivo manter de forma transparente a consistência dos dados. Tradicionalmente isso têm sido feito através de mecanismos que preservam propriedades como atomicidade e seriabilidade. Entretanto, para aplicações avançadas, o gerenciamento de transações deve ser mais flexível, permitindo a cooperação entre usuários e fornecendo suporte à longa duração. Além disso, essas aplicações lidam com objetos complexos para os quais é necessário manter várias versões. Esse trabalho discute mecanismos de suporte para aplicações avançadas baseados em transações aninhadas e versionamento.

1. Introdução

Associados ao aumento da capacidade de armazenamento de dados, foram sendo incorporadas aos sistemas de gerenciamento de banco de dados ferramentas integradas para o desenvolvimento de aplicações, tais como *automação de escritório*, *projeto assistido por computador*, entre outras. Chamamos esse tipo de aplicação de *avançada* que difere das *tradicionais* nos seguintes aspectos:

- **quanto ao tipo de dados** - os dados manipulados pelas aplicações tradicionais são, geralmente, alfanuméricos, possuem tamanho fixo e estrutura plana. Em contrapartida, as aplicações avançadas manipulam dados de estrutura complexa. Podem ser constituídos por vários atributos que, por sua vez, podem ser estruturados recursivamente. Determinadas aplicações avançadas, principalmente aquelas ligadas ao desenvolvimento de projetos, tais como CAD/CAM, CASE, necessitam manipular vários estados do mesmo dado, representando diferentes alternativas e evoluções do projeto. Além disso, o mesmo item de dados pode possuir várias representações equivalentes entre si, como por exemplo, o código fonte e objeto do mesmo módulo de programa;
- **quanto à duração** - as aplicações tradicionais têm curta duração, enquanto que as aplicações avançadas, geralmente, são longas;
- **quanto à interação entre os usuários** - aplicações tradicionais requerem isolamento entre si, enquanto que aplicações avançadas são baseadas no trabalho de grupo, requerendo, portanto, interação entre os usuários.

A fim de manter a consistência dos dados manipulados por tais aplicações e suportar cooperação[1] é necessário investigar outros mecanismos, pois os mecanismos convencionais não se adaptam a tais características. Isso envolve o gerenciamento de transações para fornecer suporte às operações das aplicações cooperativas e o gerenciamento de versões para permitir a manipulação dos dados, considerando sua estrutura complexa.

Os gerenciamentos de versões e de transações estão fortemente ligados, pois para garantir a consistência dos dados deve-se considerar os diferentes estados desses dados [PK84] e suas configurações [Kat90,CJ94]. Portanto, a integração entre ambos os gerenciadores pode fornecer um suporte adequado às aplicações e em particular às aplicações cooperativas.

Neste artigo apresentamos um modelo de gerenciamento de transações cooperativas integrado a um modelo de versões. O modelo de transações é baseado em aninhamento de transações e áreas de trabalho. Algumas operações cooperativas (de cópia e transferência de dados entre diferentes áreas de trabalho) foram elaboradas. O gerenciamento de versões fornece suporte a dados complexos e implementa o controle de concorrência utilizando um conjunto simples de *trancas*[2], as quais permitem que usuários interajam através das operações cooperativas.

Esse artigo estrutura-se da seguinte forma. A seção 2 analisa os requisitos das aplicações a fim de traçar o perfil do modelo de transações mais adequado para suportá-las. A seção 3 apresenta o modelo proposto que envolve o gerenciamento de versões (seção 3.1) e de transações (seção 3.2). E finalmente, a seção 4 conclui esse trabalho.

2. Análise dos Requisitos das Aplicações

Nesta seção analisaremos os principais requisitos das aplicações tradicionais e avançadas. Dentre as aplicações avançadas, estamos nos concentraremos apenas nas cooperativas assíncronas.

2.1 Transações para Aplicações Tradicionais

As aplicações tradicionais caracterizam-se por possuir curta duração e não necessitar de cooperação. Essas características permitem que o gerenciamento de transações possua um critério de correção baseado em *seriabilidade*[3] [BHG87]. Tal critério é restritivo, pois penaliza alguns intercalamentos corretos [BK91]. No entanto, essa restrição é aceitável para esse tipo de aplicação. O gerenciamento de transações baseado em seriabilidade agrupa um conjunto de operações em unidades, as quais possuem as seguintes propriedades, conhecidas como propriedades ACID:

- **<u>A</u>tomicidade** - todas as operações pertencentes a uma transação são executadas com sucesso ou nenhuma delas é executada.
- **<u>C</u>onsistência** - o conjunto de itens de dados do sistema obedece a algumas assertivas que permanecem válidas após a execução da transação.

[1] Somente consideramos nesse trabalho cooperação assíncrona, onde os usuários trabalham isoladamente em espaços privados e interagem através de espaços compartilhados.
[2] Tradução do inglês *locks*.
[3] Tradução do inglês *seriazability*.

- **Isolamento** - estados intermediários de uma transação não devem ser observados por outras transações.
- **Durabilidade** - uma vez confirmadas as atualizações de uma transação, estas devem ser mantidas, mesmo se falhas ocorrerem.

Versões dos itens de dados manipulados pelas transações podem eventualmente ser consideradas [BGH87,PK84]. Entretanto, a finalidade é aumentar o nível de concorrência do sistema.

2.2 Transações para Aplicações Cooperativas

Existem vários modelos de transações que levam em conta os requisitos das aplicações avançadas, e em particular as cooperativas [BK91,Elm94]. Esses modelos revisaram os mecanismos de controle de concorrência e recuperação tradicionais para um ambiente em que as aplicações têm longa duração e usuários desenvolvem tarefas cooperativamente. Os novos mecanismos propostos consideram os seguintes aspectos:

- a longa duração aumenta o número de conflitos entre transações. Resolver conflitos através de bloqueios ou cancelamentos de transações (métodos convencionais) degrada o desempenho do sistema;
- o critério de correção baseado em seriabilidade impossibilita a interação entre usuários trabalhando cooperativamente;
- desfazer totalmente uma transação interrompida por uma falha pode ser inaceitável quando essa transação possui longa duração.

Esses aspectos impõem novos requisitos, os quais devem ser observados pelo modelo de gerenciamento de transações:

- **Suporte a longa duração** - existem aplicações, como por exemplo, aplicações para desenvolvimento de projeto que podem durar horas ou semanas. Os métodos de controle de concorrência convencionais são baseados em bloqueio ou cancelamentos de transações para resolver conflitos. Os métodos de recuperação convencionais também são baseados em cancelamentos de transações a fim de garantir a propriedade de atomicidade. Quando transações têm longa duração é inaceitável usar mecanismos baseados em bloqueio ou desfazer totalmente uma transação que já dura semanas.
- **Suporte ao trabalho de grupo** - aplicações cooperativas ligadas ao desenvolvimento de projetos complexos envolvem vários projetistas. As tarefas são divididas em subtarefas e atribuídas a grupos de projetistas. Cada projetista pode interagir mais intensivamente com um subconjunto do banco de dados, mas pode também precisar interagir com outros projetistas e seus subconjuntos de dados. O trabalho desenvolvido por um projetista pode exigir consultas ou atualizações no trabalho em andamento de outros projetistas. Nesses ambientes o modelo de transações não deve, portanto, impor barreiras de visibilidade. Métodos de controle de concorrência baseados no critério de seriabilidade impossibilitam a cooperação.
- **Suporte a critérios flexíveis de correção** - aplicações cooperativas requerem critérios de correção mais flexíveis que levam em conta sua duração e a necessidade de interações entre membros do mesmo grupo. Novos critérios baseados em semântica dos dados foram

propostos [SZ89,Ska89,NRZ94]. Contudo, a dificuldade de especificar restrições de consistência e de detectar automaticamente as violações dessas restrições em tal ambiente provocou o desenvolvimento de novas propostas, tal como em [NG92], onde o grupo de usuários define dinamicamente as alterações que podem ser aplicadas sobre o banco de dados.

- **Suporte a dados complexos** - algumas aplicações avançadas necessitam modelar os objetos do mundo real que são compostos de outros objetos. É necessário, portanto, o suporte a representação de dados complexos [Kat90].
- **Suporte a evolução dos dados** - em ambientes de desenvolvimento de projetos, refinamentos são acrescentados gradualmente gerando novas versões para objetos de projeto já existentes. Desta forma, uma cadeia de versões pode ser criada para um objeto se cada versão tiver somente uma versão ancestral e gerar somente uma versão descendente (versões evolutivas), ou ainda, uma árvore pode representar as diversas alternativas da evolução do objeto. No caso de dados complexos o sistema deve fornecer suporte à ligação entre a versão de um objeto complexo e a versão de cada um de seus componentes, ou seja, deve fornecer suporte a *configuração*.
- **Suporte a múltiplas representações** - um projeto pode ser especificado em várias representações diferentes, porém equivalentes. Cada uma dessas representações expressa um ponto de vista distinto, mas deve haver consistência entre elas. Ambientes para desenvolvimento de projeto devem ser capazes de organizar as diversas representações de um projeto e as dependências entre elas.

3. Gerenciamento de Transações e Versões

Usamos, para apresentar tanto o gerenciamento de versões quanto de transações, uma modelagem orientada a objetos e a notação OMT [RBP+94]. Desta forma, acreditamos fornecer melhor clareza devido à semântica de tal notação.

Baseados em [KBC+87] apresentamos um modelo de gerenciamento de versões. Entretanto, diferentemente do modelo original, modificamos a hierarquia de composição e propomos um controle de concorrência mais flexível, fornecendo suporte à cooperação. Algumas alterações foram também necessárias para fazer a integração entre o gerenciamento de versões e transações, a fim de contemplar a hierarquia de transações e operações para criação/remoção de objetos considerando áreas de trabalho. As seções seguintes descrevem os modelos de versões (seção 3.1) e transações (seção 3.2).

3.1 Gerenciamento de Versões

Apresentamos a seguir um modelo de gerenciamento de versões que fornece suporte às operações de derivação e leitura. As operações sobre os objetos são realizadas em áreas de trabalho privadas e posteriormente transferidas para áreas de trabalho compartilhadas. O controle de concorrência baseia-se em trancas, as quais implementam suporte à cooperação.

Duas classes implementam o modelo: Objeto Genérico e Objeto Versão. Um objeto de aplicação para adquirir as propriedades e operações relacionadas com versões deve herdá-las da classe Objeto Versão. A Classe Objeto Genérico organiza a *hierarquia de derivação*[4] para um dado objeto de aplicação. Além disso, um objeto da aplicação pode ser composto de

[4] Utilizamos o termo *hierarquia de derivação* para denominar a história de versões.

outros objetos, ser equivalente a outros objetos e participar de várias transações, como ilustra a figura 1.

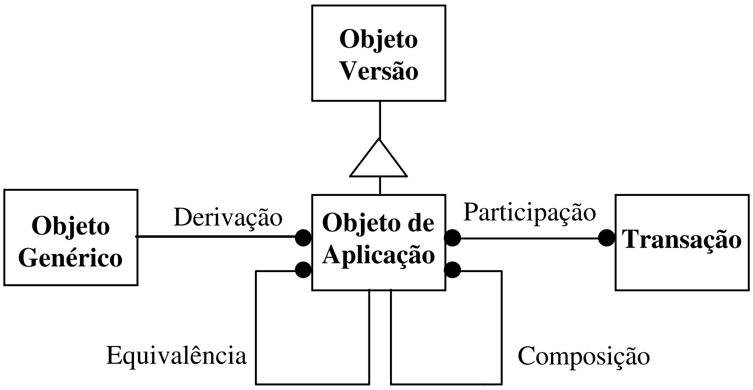


Figura 1: Estrutura de classes do modelo de versões.

Essas duas classes fornecem às aplicações as funcionalidades descritas abaixo. As operações do modelo de versões foram brevemente descritas. Maiores detalhes podem ser vistos em [Lim96].

Identificação dos objetos e suas versões

Um objeto é composto de uma instância da classe Objeto Genérico e várias versões, representando estágios distintos na história do objeto.

O ambiente cooperativo é estruturado numa hierarquia de áreas de trabalho, para onde os objetos são transferidos quando alocados por transações. Esses protocolos são comumente conhecidos na literatura como protocolos de *checkin/chekout* [BK91]. Quando a transação termina, as atualizações feitas no objeto são transferidas para a área de trabalho superior, só então as atualizações são efetivadas. O mecanismo de identificação deve, portanto, identificar unicamente cada cópia das diversas versões dos objetos em diferentes áreas de trabalho.

Uma instância de Objeto Genérico recebe um identificador no momento da criação, consistindo de duas partes: o identificador da área de trabalho concatenado com um contador que identifica unicamente o objeto na área de trabalho. Esse contador é incrementado toda vez que um objeto genérico é criado.

Quando uma nova versão de um objeto é criada, essa recebe um identificador que consiste do identificador do objeto genérico concatenado a um contador de versão. Esse contador de versões é mantido pelo objeto genérico e incrementado toda vez que uma nova versão do objeto é criada.

Quando um objeto é copiado ou transferido de uma área para outra, seu identificador é atualizado substituindo-se a área de trabalho original pela área destino. As outras partes do identificador do objeto continuam as mesmas.

Resolução de referências dinâmicas

Aplicações podem usar referências estáticas ou dinâmicas para objetos. Uma referência estática aponta para uma versão específica de um objeto enquanto que um referência dinâmica aponta para uma instância do objeto genérico. O objeto genérico armazena a versão *atual* do objeto para a qual a referência dinâmica é resolvida. Essa abordagem segue a mesma idéia de [KBC+87].

Criação e manipulação do objeto genérico

A classe Objeto Genérico fornece operações para criação/remoção de instâncias e cópia/transferência de instâncias de/para áreas de trabalho de transações (áreas de trabalho serão vistas em seções posteriores).

Criação e manipulação de versões

A classe Objeto Versão fornece operações para criação/remoção de instâncias da classe e cópia/transferência de instâncias de/para áreas de trabalho de transações.

Atualização da hierarquia de derivação

O relacionamento de derivação entre as versões de um objeto representa a história evolutiva deste objeto. No modelo apresentado, a história de versões é linear. A versão atual é sempre a última versão efetiva da qual uma nova versão pode ser derivada. A classe Objeto Versão tem operações para inserir e remover versões da história de versões. A classe Objeto Genérico possui como atributo a história de versões. Quando uma versão é inserida/removida, um nodo da hierarquia de derivação é inserido/removido na instância do objeto genérico correspondente.

Atualização da hierarquia de composição

Cada versão de um objeto pode ser composta de outras versões. Tanto referências estáticas como dinâmicas podem ser usadas. A classe Objeto Versão fornece operações para atualizar a hierarquia de composição.

O relacionamento de composição é representado por dois relacionamentos: *é-composto-por* e *é-componente-de*. Desta forma, os objetos são estruturados numa hierarquia de composição duplamente ligada e foi concebida de tal forma para facilitar o controle de concorrência.

Atualização de equivalências

Os objetos possuem várias facetas por causa de suas várias representações. Por exemplo, o código fonte e objeto de um mesmo módulo de programa são objetos distintos, porém são equivalentes entre si. Assim, se o código fonte for atualizado e não for gerado outro código objeto referente a essa atualização, eles não estarão consistentes entre si. O relacionamento de equivalência faz as ligações entre versões equivalentes, facilitando o gerenciamento da consistência. No modelo proposto, a manutenção da consistência entre versões deve ser realizada pela própria aplicação, por esse relacionamento possuir caráter puramente semântico. A classe Objeto Versão fornece operações para atualizar equivalências.

Controle de Concorrência

O controle de concorrência é baseado em trancas e implementado na classe Objeto Versão que contém operações para trancar e destrancar versões. Os dois modos de trancas adotados são: tranca para leitura de versão (**l**) e tranca para derivação de versão (**d**). Além disso, para a remoção de um objeto, é necessário uma tranca <u>exclusiva temporária</u> sobre todos os componentes na hierarquia de composição.

Para trancar um objeto composto utilizamos trancas de intenção [KBC+87], **il** e **id**. Se for requisitada uma tranca **l** sobre um objeto, todos os seus ancestrais na hierarquia de composição são trancados em modo **il**. Da mesma forma, se for requisitada uma tranca **d** sobre um objeto, todos seus ancestrais são trancados em modo **id**. Já seus descendentes na hierarquia de composição são implicitamente trancados em **l** ou **d**, respectivamente. Somente depois que todas as trancas de intenção sobre todos os ancestrais são conseguidas a versão do objeto é trancada.

Para alocar qualquer objeto é preciso verificar a compatibilidade entre as diferentes trancas. A tabela a seguir define essa compatibilidade:

	il	**l**	**id**	**d**
il	+	+	+	-
l	+	+	-	-
id	+	-	+	-
d	-	-	-	-

Tabela 1: Compatibilidade entre trancas. (+) Compatível; (-) Incompatível.

A fim de fornecer suporte às operações cooperativas, as quais transferem e copiam objetos entre áreas de trabalho (descritas na seção 3.2), anexamos uma segunda dimensão às trancas, que indica a permissão ou não de cooperação. São três os tipos de permissão: de cópia (**c**), de empréstimo (**e**) e de concessão (**o**). Por exemplo, se uma transação possui uma tranca sobre uma determinada versão do tipo **d/ceo**, então ela está permitindo que outras transações requisitem a cópia, empréstimo ou concessão da versão derivada (essas operações serão descritas na seção 3.2). As seguintes combinações são possíveis: permissões para cópia, empréstimo ou concessão associadas à tranca do tipo **d** e; permissões para cópia associadas com a tranca do tipo **l**.

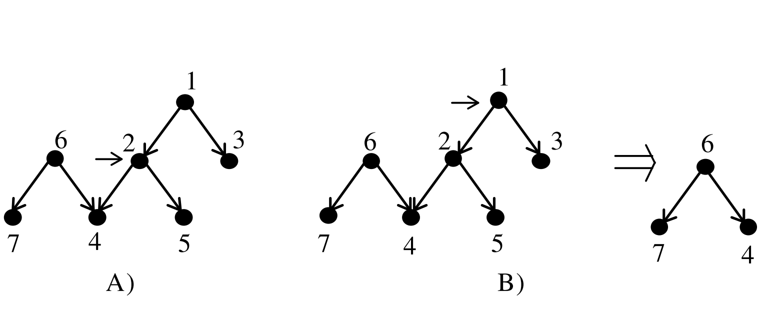


Figura 2: Ilustração de remoção de versões. Em A) falha na tentativa da versão 2. Em B) remoção da versão 1 e todas as suas componentes.

Uma versão só pode ser removida se a mesma não é componente de nenhuma outra. Se tais referências não existirem, a versão é removida, juntamente com as versões componentes. A figura 2 resume o processo de remoção. Cada vértice representa a versão de um objeto. O vértice 2, em 2A, não pode ser removido, pois é componente do vértice 1. Em 2B, a remoção do vértice 1 implica na remoção de todos os seus componentes, exceto o vértice 4, pois faz parte de outra configuração. Neste caso, apenas a referência de composição é removida.

A seguir descreveremos sucintamente as operações mais importantes das classes Objeto Genérico e Objeto Versão. Uma descrição mais completa destas e de outras operações são encontradas em [Lim96].

3.1.2 A Classe Objeto Genérico

Essa classe possui os seguintes atributos e sua funcionalidade pode ser ilustrada na figura 3 abaixo:

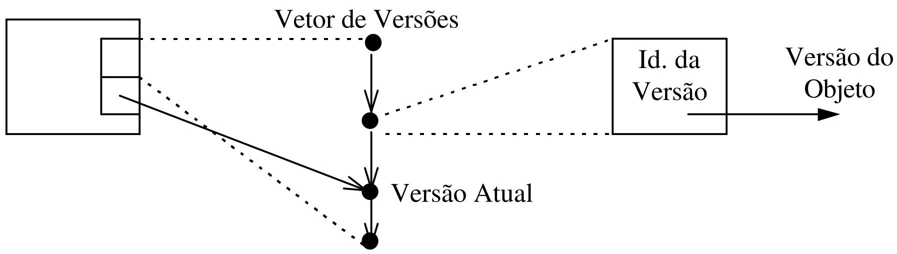


Figura 3: Ilustração do Objeto Genérico.

Os atributos da classe são:

- **Identificador do Objeto** - identifica unicamente o Objeto Genérico.
- **Versão Atual** - identifica a versão atual do objeto. É responsável pelas referências dinâmicas. Por simplicidade, assumimos que a versão atual é sempre a última versão derivada. Contudo, esse conceito pode ser estendido, tal como ilustra a figura 3.
- **Vetor de Versões** - mantém uma entrada contendo o identificador da versão para cada uma das versões na história do objeto. A história é linear e as posições no vetor indicam os relacionamentos ancestral/descendente entre as versões.
- **Contador de Versões** - usado na geração do identificador único de cada versão.

As seguintes operações são implementadas nessa classe:

1. **CriarNovoObjetoGenérico** - cria o objeto genérico retornando seu identificador único.
2. **CriarObjetoGenérico** - cria um objeto genérico na área de trabalho passada como parâmetro e inicializa o estado do objeto genérico com o estado passado como parâmetro. Cria o identificador único do objeto genérico, substituindo o identificador da área de trabalho original pela área de trabalho especificada como parâmetro.
3. **RemoverObjetoGenérico** - remove o objeto genérico se não existirem versões a ele associadas. Caso contrário, retorna falha.

4. **CopiarObjetoGenérico** - cria o objeto genérico na área de trabalho destino, inicializando o estado do novo objeto com o estado do objeto original, através da operação CriarObjetoGenérico.
5. **TransferirObjetoGenérico** - transfere o objeto genérico para a área de trabalho destino e remove o objeto genérico da área de trabalho atual.
6. **InserirVetorDerivação/RetirarVetorDerivação** - insere/retira o identificador passado como parâmetro no vetor de versões. Se o identificador corresponder à versão atual e a operação for retirar, atualiza a versão atual para a versão ancestral.

3.1.3 Operações da Classe Objeto Versão

Essa classe possui os seguintes atributos:

- **Identificador da Versão -** Identifica unicamente a versão e é formado de acordo com as regras já descritas anteriormente.
- **Vetor de Compostos** - contém uma entrada para cada versão que usa a versão em questão como componente.
- **Vetor de Componentes** - contém uma entrada para cada identificador das versões componentes.
- **Estado da Versão** - uma versão pode estar em três estados: (i) *temporário,* o qual indica que a versão foi criada por uma transação que ainda está em execução e portanto, está disponível somente à transação que a criou; (ii) *efetivo*, indica que a transação que criou a versão já foi validada e, portanto, a versão está disponível para outras transações e; (iii) *liberado para leitura*, indica que uma transação liberou a versão antes do término através da operação LiberarObjeto, disponibilizando a versão apenas para leitura. Em caso de cancelamento dessa transação a versão poderá ser removida.

As operações da classe Objeto Versão são as seguintes:

1. **CriarVersão** - cria uma nova versão associada a um objeto genérico. Um identificador é criado e a versão criada possui o estado temporário.
2. **DerivarVersão -** essa operação é responsável por derivar uma nova versão a partir da versão atual, requisitando a criação de uma nova versão através da operação CriarVersão. A nova versão é inicializada com o conteúdo da versão corrente. Requisita a atualização do vetor de derivação no Objeto Genérico.
3. **RemoverVersão** - responsável por remover uma versão, respeitando as restrições explicadas anteriormente (figura 2).
4. **CopiarVersão** - copia a versão para a área de trabalho destino, implementando a operação *checkout*. Para tanto, lineariza o estado da versão e requisita a operação CriarVersão. Se existirem versões componentes, estas também são copiadas.
5. **TransferirVersão** - transfere a cópia da versão e de suas versões componentes, se existirem, para a área de trabalho destino, implementando a operação *checkin*. Se a versão foi trancada para leitura, apenas a libera.
6. **TrancarVersão** - essa operação é responsável por atender a requisição de uma transação para trancar uma versão. Se houver compatibilidade com as trancas já existentes sobre a versão (considerar a hierarquia de composição) a operação retorna sucesso. Caso

contrário, pode falhar ou não (a depender do parâmetro passado pela transação bloqueante/Não-bloqueante). Se a requisição for bloqueante insere a transação requisitante na lista de transações que esperam por trancas da versão em questão.

7. **DestrancarVersão** - destranca a versão corrente e todas as suas componentes, se existirem. Se existirem transações a espera da versão liberada, concede a versão para a próxima transação, desbloqueando-a.
8. **InserirVetorComposição/RetirarVetorComposição** - atualiza o vetor de composição da versão inserindo/retirando o identificador da versão passada como parâmetro. Insere/retira no vetor de compostos (hierarquia duplamente ligada) a versão corrente.
9. **InserirVetorEquivalência/RetirarVetorEquivalência** - mantém as relações de equivalência entre a versão em questão e a versão passada como parâmetro, inserindo/retirando a entrada correspondente no vetor de equivalência.
10. **CongelarReferência** - substitui as referências dinâmicas pelos identificadores das versões atuais. Para tanto, percorre a hierarquia de composição no sentido descendente, resolvendo as referências dinâmicas de todos os seus componentes. Para cada referência indireta, consulta o Objeto Genérico a fim de obter o identificador da versão atual.

3.2 Gerenciamento de Transações

Nesta seção apresentamos o modelo de gerenciamento *orientado à cooperação*, possibilitando a criação de grupos de usuários através da hierarquias de transações e áreas de trabalho. Para facilitar a cooperação, objetos podem ser *copiados* entre áreas de trabalho, *emprestados* de uma transação e posteriormente retornados, ou ainda, *concedidos* de uma transação para uma outra que adquire todos os direitos da primeira.

O diagrama ilustrado na figura 4 apresenta a modelagem da hierarquia de transações. Uma aplicação deve criar instâncias das classes Transação de Usuário e Transação de Grupo de forma a construir o ambiente adequado às suas necessidades. Cada uma das classes do modelo é descrita a seguir.

Classe Transação Cooperativa

A Classe Transação Cooperativa é uma classe abstrata e define as operações comuns às classes Transação de Grupo e Transação de Usuário. Uma aplicação cooperativa pode ser estruturada como uma hierarquia de transações cujos nós internos são instâncias de Transação de Grupo e folhas são instâncias de Transação de Usuário.

Cada transação cooperativa está associada a uma área de trabalho de onde os objetos e suas versões podem ser copiados pelas transações descendentes e posteriormente transferidos de volta quando uma transação descendente terminar o trabalho sobre os objetos. Essas áreas de trabalho são criadas quando uma transação cooperativa é iniciada e são removidas quando terminada.

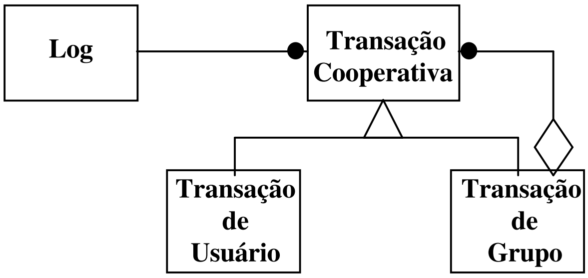


Figura 4: Estrutura de classes do modelo de transações.

A figura 5 ilustra a hierarquia de transações e a alocação de objetos. T_1 é uma transação de grupo constituída pelas subtransações T_2 (transação de grupo), T_4 (transação de grupo) e T_3 (transação de usuário). T_1 alocou e requisitou cópias de cinco versões de objetos das quais três foram alocadas pelas suas transações descendentes.

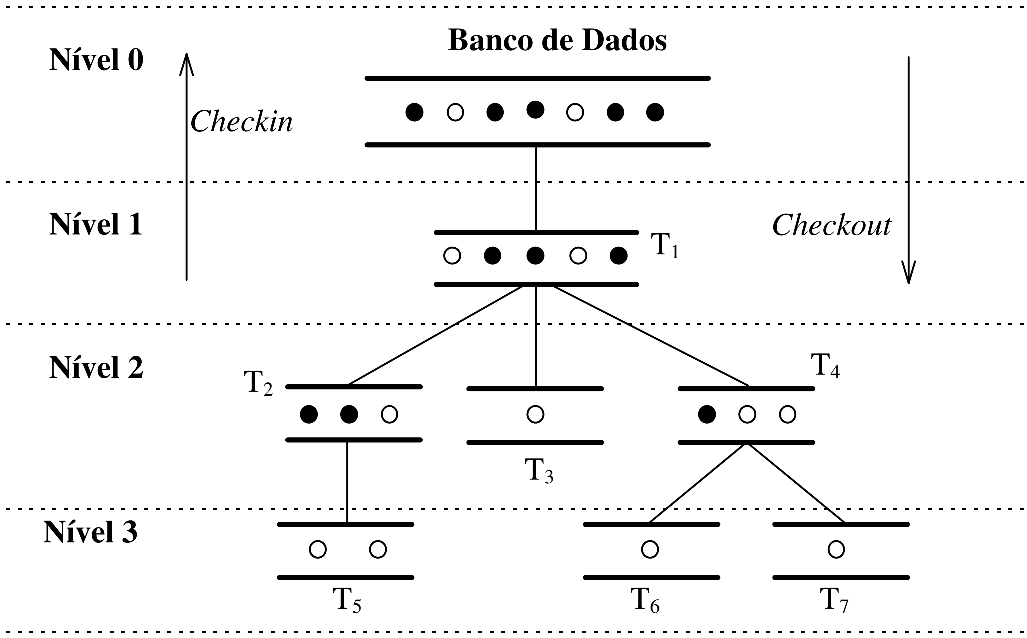
As operações definidas na classe Transação Cooperativa são as seguintes:

1. **Iniciar** - cria a área de trabalho e um identificador; inclui a transação na lista de transações da transação ancestral; tranca atomicamente os objetos que participarão da transação; e armazena um registro no Log indicando início da transação.

   Se a tranca tem sucesso, insere os objetos e suas versões na lista de objetos da transação e requisita uma cópia dessas versões da área da transação pai para a área de trabalho da transação cooperativa. Se a versão não for encontrada na área da transação pai, a busca continua na hierarquia de áreas de trabalho no sentido folha-raiz. Quando a versão é encontrada, uma tranca no modo especificado é solicitada e se for concedida, uma cópia é realizada da área ancestral para a área descendente. Esse processo se repete até a área da transação cooperativa requisitante. É necessário que, para cada versão a ser copiada, haja uma cópia do objeto genérico correspondente na mesma área de trabalho destino. Por exemplo, a transação T_6 (figura 5) requisita uma cópia da área de trabalho de T_4. Se T_4 não possui a versão, ela solicita a T_1 e assim sucessivamente até chegar ao nível 0. Assim que a versão é trancada, a cópia é realizada nas áreas de T_6 ,T_4 e T_1. Processo semelhante ocorre para o objeto genérico, porém, não há necessidade de mantê-lo trancado.

2. **Terminar** - verifica o estado das transações descendentes, se existirem. A transação cooperativa pode ser validada dependendo do estado de suas transações descendentes e do tipo de terminação requisitado pela aplicação. Se o tipo de terminação for "cancelamento", a transação cooperativa será abortada. Os outros tipos de terminação possíveis são: "validação se todas as transações filhas terminarem com sucesso", "validação se a maioria das transações filhas terminarem com sucesso" e "validação das transações filhas que terminaram com sucesso".

   Esta operação também verifica o estado dos objetos participantes da transação. A transação cooperativa só poderá ser terminada se todas as versões emprestadas para outras transações tiverem sido retornadas à transação original. Se o tipo de terminação for "validação", armazena registro no log indicando que a transação será validada. O registro deve conter o identificador da transação e dos objetos participantes e suas versões. Versões emprestadas são transferidas para as áreas das transações originais. Versões que foram copiadas através da operação RequisitarCópia são descartadas. Versões derivadas são transferidas para a área da transação pai ou para a área pública (dependendo do nível da transação cooperativa), juntamente com os respectivos objetos genéricos. As versões que originaram as versões derivadas são destrancadas. Versões trancadas para leitura são removidas da área da transação e destrancadas. Se o tipo de terminação for "cancelamento", descarta as versões e os objetos genéricos correspondentes na área de trabalho da transação e libera todas as trancas requisitadas pela transação. Para versões emprestadas, retorna os privilégios para as transações originais sem contudo propagar as mudanças. Armazena registro no log indicando que a transação foi terminada e o tipo de terminação. Indica à transação pai que a transação cooperativa terminou e o tipo de terminação.



● Versão trancada por transação descendente e *checked out* para sua área de trabalho

○ Versão não trancada por transação descendente

Figura 5: Estrutura hierárquica das transações.

3. **RequisitarObj** - requisita a tranca sobre a versão do objeto no modo especificado (pode bloquear ou não a transação requisitante conforme o parâmetro da operação e a compatibilidade entre o modo de acesso solicitado e as trancas já existentes sobre a referida versão). Se a tranca tem sucesso, insere o identificador do objeto e sua versão na lista de objetos da transação, copia a versão e o objeto genérico correspondente da área da transação pai para a área da transação cooperativa. Busca em níveis superiores e cópias em vários níveis podem ser necessárias como explicado anteriormente.
4. **LiberarObj** - se a versão do objeto foi derivada, transfere a versão para a área da transação pai. A versão liberada fica disponível somente para leitura. Se a versão do objeto foi trancada para leitura, destranca a versão e descarta sua cópia da área da transação cooperativa.

Classe Transação de Usuário

Essa classe herda as operações da classe Transação Cooperativa adicionando operações que relaxam a seriabilidade e melhoram a visibilidade entre usuários trabalhando cooperativamente. Tais operações baseiam-se nas trancas cooperativas discutidas na seção 4.1. As operações são as seguintes:

1. **RequisitarCópia** - busca a versão mais atual do objeto, copiando-a para a área de trabalho da transação requisitante. A transação original que derivou a versão mantém sua cópia e seus privilégios. A cópia tem sucesso somente se a versão ancestral que originou a versão tiver sido trancada em modo que permita a cópia por outras transações. Após a cópia, a transação de usuário pode realizar qualquer operação sobre a versão do objeto, mas essa cópia será descartada no final da transação.
2. **RequisitarEmpréstimo** - Essa operação busca a cópia mais atual da versão, copiando-a para a área da transação requisitante. O empréstimo tem sucesso se o modo de acesso permitir empréstimo. A transação original mantém uma cópia e pode continuar aplicando atualizações sobre ela. Contudo, essa cópia será substituída pela segunda transação quando invocar a operação RetornarEmpréstimo. Se um *timeout*, especificado pela aplicação, esgotar sem que a segunda transação tenha retornado a versão emprestada, a transação original volta a ter os privilégios iniciais e a transação que tomou emprestado perde o direito de alterar o objeto na área origem. Não é permitido o empréstimo de versões que estejam emprestadas a outras transações e nem que tenham sido adquiridas por empréstimo ou cópia.
3. **RequisitarConcessão** - A versão mais atual do objeto é transferida para a área da transação requisitante. A transação original concede todos seus direitos sobre o objeto para a transação requisitante. A concessão é obtida se os modos de acesso desde a transação raiz até a transação descendente que possui a cópia mais recente do objeto permitirem. Cópias do objeto são realizadas na hierarquia de áreas de trabalho da mesma forma descrita para a operação RequisitarObj. O objeto genérico associado também é transferido. Não é permitida a concessão de versões que estejam emprestadas a outras transações e nem que tenham sido adquiridas por empréstimo ou cópia.

Classe Transação de Grupo

Esta classe herda as operações da Classe Transação Cooperativa adicionando duas operações que permitem construir a hierarquia de transações cooperativas. As instâncias dessa classe representam os vértices não-folhas da árvore de transações (figura 5). As operações da classe são as seguintes:

1. **IncluirTransação** - inclui o identificador da transação filha na lista de transações da transação pai.
2. **ExcluirTransação** - muda o estado da transação filha na lista da transações da transação pai indicando o tipo de terminação.

4. Conclusões

Existem poucos trabalhos na literatura que tratam o gerenciamento de versões e transações de forma integrada. Em [CJ94] é apresentado um modelo de versões, baseado no conceito de constelações [CVJ94]. São definidas três tipos de transações, as quais operam em níveis diferentes do banco de dados. No entanto, o modelo não apresenta suporte à cooperação. O modelo de transações apresentado em [KSUW85] é baseado numa hierarquia de dois níveis de áreas de trabalho, não permitindo formação de grupos hierárquicos. Provê operações de empréstimo e concessão, no entanto, como não há trancas cooperativas, é necessário uma comunicação prévia entre os usuários para executá-las. Em [Unl94] é descrito um modelo de transações baseado num complexo conjunto de trancas e na hierarquia multinível de áreas de trabalho. Nesse modelo, trancas de derivação fornecem suporte a versões, entretanto não é descrito nenhum modelo de versões subjacente.

Apresentamos, após analisar os requisitos das aplicações cooperativas, de forma integrada, dois modelos de gerenciamento: de transações e de versões. Juntos constituem um ambiente adequado à cooperação, fornecendo suporte às aplicações cooperativas. Tal ambiente está baseado no aninhamento de transações associado a áreas de trabalho, favorecendo a interação entre os usuários e possibilitando a formação de grupos hierárquicos. O suporte à cooperação é reforçado através das operações cooperativas que relaxam a seriabilidade, permitindo que transações tenham acesso a estados intermediários dos dados manipulados por outras transações. A implementação dessas operações foi simplificada através da dimensão cooperativa das trancas.

O reduzido número de trancas simplifica o controle de concorrência, facilitando o suporte a dados complexos e a versões. Contudo, isso não impede que se estenda o número de trancas para suportar outras operações, como por exemplo, atualizações ou derivações alternativas de objetos.

A modelagem orientada a objetos facilita, através do mecanismo de herança, que ambos os modelos sejam estendidos. Uma aplicação pode, por exemplo, adicionar outras funcionalidades a uma classe existente a fim de que o modelo se adapte melhor a suas necessidades.

Referências

[BHG87] P. A. Bernstein, Vassos Hadzilacos, e Nathan Goodman. *Concurrency Control and Recovery in Databases Systems*. Addison-Whesley Publishing Company, 1987.

[BK91] N. S. Barghouti e G. E. Kaiser. Concurrency Control in Advanced Database Applications. *ACM Computing Surveys*, pp. 169-297, março de 1991.

[CJ94] W. Cellary e G. Jomier. Apparent Versioning and Concurrency Control in Object-Oriented Databases. *In Proc. of the International Conference on Computing and Information*, pp. 1-19, 1994.

[CVJ94] W. Cellary, G. Vossen, e G. Jomier. Multiversion Object Constellations: A New Approach to Support a Designer's Database Work. *Engineering with Computer*. Spring Verlab, pp. 230-244, 1994.

[Elm94] Ahmed K. Elmagarmid. *Database Transaction Models for Advanced Applications*. 1994.

[Kat90] Randy H. Katz. Toward a Unified Framework for Version Modeling in Engineering Databases. *ACM Computing Surveys*, 22(4):375-408, dezembro de 1990.

[KBC+87] W. Kim, J. Banerjee, Hong-Tai Chou, J. F. Garza, e D. Woelk. Composite Object Support in an Object-Oriented Database System. In *Proc. of the 2nd OOPSLA.,* Orlando, outubro de 1987.

[KSUW85] P. Klahold, G. Schlageter, R. Unland, e W. Wilkes. A Transaction Model Supporting Complex Applications in Integrated Information Systems. *In ACM SIGMOD*, pp. 388-401, maio de 1985.

[Lim96] G. M. de A. Lima. Gerenciamento de Transações: Um Estudo e Uma Proposta. Dissertação de Mestrado, UNICAMP-IC, setembro de 1996.

[Mos81] J. E. B. Moss. *Nested Transactions: An Approach to Reliable Distributed Computing*. PHD thesis, Dept. of Electrical Engineering and Computer Science, MIT, abril de 1981.

[NG92] K. Narayanaswamy e N. Goldman. "Lazy" Consistency: A Basis for Cooperative Software Development. In *CSCW Proceedings*, pp. 257-264, novembro de 1992.

[NRZ94] M. H. Nodine, S. Rasmaswamy, e S. Zdonik. A Cooperative Transaction Model for Design Databases. In Ahmed K. Elmagarmid, editor, *Database Transaction Models for Advanced Applications,* pp. 53-85.

[PK84] C. H. Papadimitriou e P. Kanellakis. On Concurrency Control by Multiple Versions. *ACM Transactions on Database Systems*. 9(1):89-99, março de 1984.

[RBP+94] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, e W. Lorensen. *Modelagem e Projetos Baseados em Objetos.* Editora Campus, 1994.

[Unl94] Rainer Unland. TOPAZ: A Tool Kit for the Assembly of Transactions for Non-standard Applications. Technical report, University of Münster, Germany, Institute of Business Informatics, novembro de 1994.

[Ska89] A. H. Skarra. Concurrency Control for Cooperating Transactions in an Object-Oriented Database. *SIGPLAN Notices*, 24(4):145-147, abril de 1989.

[SZ89] A. H. Skarra e S. B. Zdonik. Concurrency Control and Object Oriented Databases. In *W. Kim and F. H. Lochovsky, Object-Oriented Concepts, Databases and Applications.* pp. 395-421, 1989.