Controle de Acesso para Bancos de Dados Geográficos Multiversão

Este exemplar corresponde à redação da Dissertação apresentada para a Banca Examinadora antes da defesa da Dissertação.

Campinas, 11 de Dezembro de 2007.

Profa. Dra. Claudia M. Bauzer Medeiros
Instituto de Computação – UNICAMP
(Orientadora)

Dissertação apresentada ao Instituto de Computação, UNICAMP, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP
Bibliotecária: Maria Júlia Milani Rodrigues – CRB8a / 2116

Pierre, Mateus Silva
P614c Controle de acesso para bancos de dados geográficos multiversão / Mateus Silva Pierre-- Campinas, [S.P. :s.n.], 2007.

Orientador : Claudia Maria Bauzer Medeiros
Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1. Sistemas de informação geográfica. 2. Bancos de dados. I. Medeiros, Claudia Maria Bauzer. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título.

Título em inglês: Access control in multiversion geographic databases.

Palavras-chave em inglês (Keywords): 1. Geographic information systems. 2. Databases.

Área de concentração: Banco de Dados

Titulação: Mestre em Ciência da Computação

Banca examinadora: Prof. Dr. Luis Mariano Del Val Cura (Faculdade Campo Limpo Paulista - FACCAMP)
Profa. Dra. Maria Beatriz Felgar de Toledo (IC-UNICAMP)
Prof. Dr. Ricardo da Silva Torres (IC-UNICAMP)

Data da defesa: 11/12/2007

Programa de Pós-Graduação: Mestrado em Ciência da Computação

TERMO DE APROVAÇÃO

Dissertação Defendida e Aprovada em 11 de dezembro de 2007, pela Banca examinadora composta pelos Professores Doutores:

Prof. Dr. Luis Mariano Del Val Cura
Faculdade Campo Limpo Paulista (FACCAMP)

Profª. Drª. Maria Beatriz Felgar de Toledo
IC - UNICAMP

Profª. Drª. Claudia Maria Bauzer Medeiros
IC - UNICAMP

Instituto de Computação
Universidade Estadual de Campinas

Controle de Acesso para Bancos de Dados Geográficos Multiversão

Mateus Silva Pierre

Dezembro de 2007

Banca Examinadora

- Claudia M. Bauzer Medeiros
  Instituto de Computação – UNICAMP (Orientadora)
- Luis Mariano Del Val Cura
  Faculdade Campo Limpo Paulista – FACCAMP
- Maria Beatriz Felgar de Toledo
  Instituto de Computação – UNICAMP
- Ricardo da Silva Torres
  Instituto de Computação – UNICAMP

Resumo

Aplicações geográficas estão cada vez mais influenciando todas nossas atividades diárias. Seu desenvolvimento exige, via de regra, trabalho em equipe de múltiplos perfis de especialistas, com diferentes visões e direitos de acesso aos dados. Em conseqüência, vários mecanismos vêm sendo propostos para controlar autorização a bancos de dados geográficos ou disponibilizar o uso de versões. Estes mecanismos, no entanto, trabalham de forma isolada, priorizando apenas o direito de acesso ou o versionamento flexível.

A dissertação aborda esta questão, propondo um modelo unificado de autorização em bancos de dados que ataque os dois problemas em conjunto. O modelo trata da questão de controle de acesso em bancos de dados geográficos, levando-se em consideração a existência de mecanismos de versionamento dos dados armazenados. Este modelo pode, assim, servir como base para trabalho cooperativo e seguro em aplicações que usem Sistemas de Informação Geográficos (SIGs).

Abstract

Geographic applications are increasingly influencing our daily activities. Their development requires efforts from multiple teams of experts with different views and authorizations to access data. As a result, several mechanisms have been proposed to control authorization in geographic databases or to provide the use of versions. These mechanisms, however, work in isolation, prioritizing only either data access or versioning systems.

This dissertation addresses this issue, by proposing a unified authorization model for databases that faces both problems. The model deals with the access control issue in geographic databases, taking into account the existence of data versioning mechanisms. This model may serve as the basis for cooperative and secure work in applications that use Geographic Information Systems (GIS).

Agradecimentos

A professora Claudia Bauzer Medeiros, pela sua atenção, paciência, ensinamentos e críticas que tanto contribuíram para o meu crescimento pessoal, profissional e acadêmico. Foi um grande prazer e privilégio tê-la como orientadora.

A minha família, pelo incentivo, apoio e confiança durante a realização deste mestrado. Em especial aos meus queridos pais, exemplos de caráter, educação, força e perseverança.

A minha querida esposa, pelo apoio incondicional, pela compreensão e companheirismo que sempre me ajudaram a superar desafios e persistir em meus objetivos.

Ao professor Ricardo Torres e aos amigos do Laboratório de Sistemas de Informação – LIS, que contribuíram direta ou indiretamente na realização deste trabalho, através de idéias, criticas e sugestões.

Ao povo brasileiro por financiar e apoiar este trabalho.

Aos membros da banca, pelas sugestões e contribuições no trabalho.

Sumário

Lista de Figuras

Capítulo 1

Introdução e Motivação

Uma quantidade crescente de aplicações tem necessidade de gerenciar informações espaciais, desde o nível micro (controle de tráfego em um bairro) até o nível macro (aquecimento global). O desenvolvimento de projetos nesta área requer atenção para dois importantes aspectos, dentre outros: segurança da informação e trabalho cooperativo. A questão de segurança da informação está relacionada a métodos de proteção aplicados sobre um conjunto de dados. Já o trabalho cooperativo está relacionado ao fato de que pessoas participam simultaneamente no desenvolvimento de um projeto, podendo acessar concorrentemente os mesmos dados e criar suas próprias interpretações sobre eles.

Como o banco de dados é em geral uma das maiores e principais fontes de dados, ele vem recebendo muita atenção com relação à sua segurança. A segurança em bancos de dados consiste de um conjunto de medidas, políticas e mecanismos com o intuito de combinar disponibilidade à confidencialidade (*secrecy*) e integridade dos dados [BDPSN96]. Estes conceitos podem ser explicados conforme a seguir:

- Confidencialidade: propriedade que limita o acesso à informação tão somente às entidades legítimas, ou seja, àquelas autorizadas pelo proprietário da informação.
- Integridade: propriedade que garante que a informação manipulada mantenha todas as características originais estabelecidas pelo proprietário da informação, incluindo controle de mudanças e garantia do seu ciclo de vida (nascimento, manutenção e destruição).
- Disponibilidade: propriedade que garante que a informação esteja sempre disponível para o uso legítimo, ou seja, por aqueles usuários autorizados pelo proprietário da informação.

Denning [Den83] lista quatro tipos de controle a fim de se obter segurança em bancos de dados: controle de acesso, controle de fluxo de informações, controle criptográfico e controle de inferências. Um controle de acesso garante que todos os acessos diretos ao sistema são autorizados de acordo com regras da política de segurança. O mecanismo de controle de acesso define quem pode acessar os objetos, que tipo de acesso é permitido e seu papel termina uma vez que o acesso é concedido. O controle de fluxo de informações assegura que informações contidas em alguns objetos não fluem explicitamente (através de cópias) ou implicitamente para objetos menos protegidos que estes, e regula como a informação pode ser acessada. O controle criptográfico torna os dados incompreensíveis para quem (pessoa ou sistema) não

possua a chave para descriptografá-la. O controle de inferências protege as informações contra qualquer tipo de dedução de seu conteúdo. Esta dissertação irá ater-se somente ao controle de acesso.

Bancos de dados espaciais são utilizados atualmente em inúmeras áreas que necessitam de um mecanismo que possa prover uma maior segurança aos dados, como aplicações governamentais, ambientais, administração de recursos naturais e gerenciamento de serviços de utilidade pública.

Aplicações espaciais são em geral desenvolvidas usando Sistemas de Informação Geográfica (SIGs). Um SIG é um software que gerencia grandes volumes de dados geo-referenciados, isto é, referenciados em relação à sua posição na superfície terrestre [CCH96]. Ele permite a análise, gestão e representação do espaço e dos fenómenos que nele ocorrem. Dados geográficos são coletados a partir de diversas fontes e armazenados via de regra em um banco de dados geográfico. Os campos de aplicação dos SIGs são muito versáteis, podendo ser usados para tomada de decisão governamental, planejamento tático para fins militares, gerenciamento de impacto ambiental, cartografia, planejamento de rotas e outros fins.

Aplicações espaciais podem exigir diferentes níveis de autorização em função do perfil do usuário e do tipo de operação desejada. O que torna o controle de acesso em bancos de dados geográficos distinto é a natureza dos dados controlados [Sasaoka02]. Outra questão é que pode haver diferentes níveis de granularidade de objetos espaciais em termos de acesso. Seja, por exemplo, uma aplicação que irá gerenciar a manutenção de equipamentos e infraestrutura elétrica em uma área. Nesta aplicação, deve ser possível acessar apenas um objeto especial (uma torre elétrica), partes de uma região (um bairro) ou mesmo ainda a região inteira (uma cidade).

O trabalho de [Sasaoka02] está voltado à solução da questão, propondo um mecanismo de acesso a bancos de dados geográficos.

Além da questão da segurança da informação, outra questão inerente à maioria dos desenvolvimentos de projeto é o trabalho cooperativo. Neste contexto, pessoas podem acessar concorrentemente os mesmos dados, gerando múltiplas representações para eles. Um sistema de informação com possibilidades para versionamento é aquele no qual podem ser mantidos múltiplos estados ou variações das entidades modeladas. Por exemplo, em aplicações CAD com suporte a mecanismo de versionamento [Kat90, TOC93], o banco de dados pode armazenar diferentes alternativas para o mesmo objeto. Já em aplicações CASE [EV94, BCJ92], diferentes versões de um mesmo programa podem ser mantidas.

Um sistema de controle de versão (ou versionamento), VCS (do inglês *version control system*) ou ainda SCM (do inglês *source code management*), é um software com a finalidade de gerenciar diferentes versões no desenvolvimento de um documento qualquer. Esses sistemas são comumente utilizados no desenvolvimento de software para controlar as diferentes versões, histórico e desenvolvimento dos códigos fontes e também da documentação. Esse tipo de sistema é usado em empresas e instituições de tecnologia e desenvolvimento de software, sendo muito comum no desenvolvimento de software livre. É útil, em diversos aspectos, tanto para projetos pessoais pequenos e simples como também para grandes projetos comerciais.

No contexto de banco de dados, existem propostas para possibilitar múltiplas representações dos dados nele armazenados [AACSV03, CAASV04]. Isto permite que desenvolvedores diferentes de aplicações para bancos de dados trabalhem em versões

diferentes dos mesmos objetos do banco, sem afetar o trabalho de outros desenvolvedores.

Em SIGs há a necessidade de manipular múltiplos estados de uma mesma entidade do mundo real. Mecanismos de versionamento podem ajudar no projeto espacial (ligado à noção de planejamento urbano e ambiental, onde vários usuários podem trabalhar cooperativamente), múltiplas representações (onde diferentes usuários representam de maneira diferente uma mesma entidade) e modelagem espaço-temporal (que trata da evolução de objetos geográficos no tempo). Face a este cenário, [delVal97] propôs um mecanismo de versões para dar apoio à modelagem temporal, de múltiplas representações e de projeto em SIG.

Tendo estes aspectos em vista, não foi encontrado na literatura um modelo de controle de acesso para bancos de dados geográficos que, ao mesmo tempo, permita trabalho cooperativo em múltiplas versões. Todos os modelos estudados consideram apenas uma única versão de cada objeto do banco.

Assim, o objetivo da dissertação é propor um mecanismo de controle de acesso para bancos de dados geográficos multiversão. A solução aqui proposta é baseada na extensão do modelo de controle de acesso de [Sasaoka02], que considera um banco de dados geográfico monoversão, para considerar versões em bancos de dados geográficos. O modelo de versões a ser usado como referência foi proposto por [CJ90] e estendido para SIG em [delVal97]. Os modelos usados foram escolhidos porque consideram as questões relacionadas a trabalho cooperativo, o que é essencial para o desenvolvimento de projetos em equipe.

O processo de autorização foi definido a partir da seguinte seqüência de etapas:

1. Definição de regras de autorização.
2. Mapeamento destas regras no conjunto de estruturas de dados do bando de dados.
3. Gerenciamento da obediência às regras.
4. Definição de algoritmos de controle de acesso para as operações definidas no modelo de versões de referência.

As principais contribuições são, desta forma:

- Proposta de um modelo de controle de acesso em bancos de dados geográficos, que leve em consideração trabalho cooperativo e controle de versões.
- Especificação de algoritmos para verificação de acesso neste contexto, considerando restrições de integridade espaciais entre versões.

O restante do deste texto está organizado da seguinte forma. O Capítulo 2 apresenta uma revisão bibliográfica que introduz a terminologia e os conceitos necessários ao entendimento do texto e a literatura correlata. O capítulo 3 trata das etapas (1), (2) e (3) estudando como definir regras de autorização em um contexto multiversão e apresentando uma proposta de mecanismo de controle de acesso para

bancos de dados geográficos multiversão. O capítulo 4 aborda a etapa (4), apresentando os algoritmos de controle e acesso para as operações definidas no modelo de [delVal97]. Finalmente, o capítulo 5 apresenta as conclusões finais da dissertação e suas possíveis extensões.

Capítulo 2

Conceitos básicos e revisão bibliográfica

Este capítulo introduz os conceitos básicos ao entendimento desta dissertação e a revisão bibliográfica realizada. A parte inicial do capítulo apresenta os conceitos e trabalhos relacionados a controle de acesso (seção 2.1). A seção 2.2 apresenta as definições sobre Sistemas de Informação Geográfica (SIG). A seção 2.3 é centrada nos conceitos e trabalhos envolvendo versões. Finalmente, a seção 2.4 apresenta uma breve conclusão sobre o capítulo.

2.1 Controle de acesso

O controle de acesso, na segurança da informação, é composto dos processos de autenticação, autorização e auditoria (*accounting*). Neste contexto, o controle de acesso pode ser entendido como a habilidade de permitir ou negar a utilização de um objeto (uma entidade passiva, como um sistema ou arquivo) por um sujeito (uma entidade ativa, como um indivíduo ou um processo). A autenticação identifica quem acessa o objeto, a autorização determina o que um usuário autenticado pode fazer, e a auditoria diz o que o usuário fez.

A identificação e autenticação fazem parte de um processo de dois passos que determina quem pode acessar determinado objeto. Durante a identificação o usuário diz ao sistema quem ele é (normalmente através de um nome de usuário). Durante a autenticação a identidade é verificada através de uma credencial (uma senha, por exemplo) fornecida pelo usuário.

A autorização define quais direitos e permissões o usuário do sistema tem. Após o usuário ser autenticado, o processo de autorização determina o que ele pode fazer no sistema. A auditoria é uma referência à coleta da informação relacionada à utilização, pelos usuários, dos recursos de um sistema. Esta informação pode ser utilizada para gerenciamento, planejamento, cobrança etc. Esta dissertação tratará apenas do processo de autorização, não abordando os aspectos relacionados à autenticação e auditoria.

2.1.1 Modelo de autorização

Todo mecanismo de controle de acesso é baseado em algum modelo de autorização, definindo como o gerenciador do banco de dados deve implementar o controle de acesso aos dados [Sasaoka02]. Geralmente, um modelo especifica os seguintes componentes:

- Granularidade de acesso
- Estruturas para representar a autorização
- Conjunto de políticas para gerenciar e administrar autorizações
- Algoritmos para analisar pedidos de acesso baseando-se nas autorizações

A granularidade de acesso define a unidade de controle de acesso aos dados. Por exemplo, pode-se ter um controle de acesso em níveis de tuplas, tabelas e até mesmo bancos de dados. A granularidade pode variar de acordo com o tipo do banco de dados. Dessa forma, ela não será a mesma para bancos de dados geográficos e bancos de dados relacionais.

A estrutura de autorização mais comum é representada pela tupla $< s, o, m >$, onde:

- s: sujeito que recebe a autorização, podendo ser usuários, grupos ou processos que atuam em nome de algum usuário.
- *o*: objeto ao qual *s* terá autorização, como tabelas, tuplas ou mesmo elementos de uma tupla.
- *m*: é o modo de acesso (leitura, escrita, etc.).

Autorizações podem, ainda, ser refinadas em positivas ou negativas. As positivas concedem o privilégio de acessar os dados, enquanto que as negativas negam o acesso explicitamente.

O conjunto de políticas para administrar autorizações é composto por regras para definir: quem irá conceder e revogar permissões (proprietário, administrador, qualquer usuário), operações (leitura, escrita), e como estas serão executadas (de forma centralizada, descentralizada ou baseada em propriedade). Na forma centralizada, somente alguns usuários podem conceder ou revogar autorizações. Na forma descentralizada, usuários, diferentes do proprietário, podem conceder ou revogar autorizações. Por fim, na forma baseada em propriedade, somente o criador do objeto pode realizar tais atividades.

Finalmente, para que um modelo de autorização esteja completo, deve haver mecanismos ou algoritmos que permitam validar um pedido de acesso baseado nas autorizações definidas.

Do ponto de vista de autorização, sistemas abertos são aqueles onde tudo é acessível, a menos que seja negado. Já nos sistemas fechados todas as permissões são negadas e somente aquelas que possuem uma autorização positiva é que são permitidas.

2.1.2 Mecanismos de controle de acesso

Há três tipos principais de controle de acesso utilizados para prover segurança em bancos de dados [BDPSN96]: Controle de Acesso Seletivo (ou DAC, do inglês *Discretionary Access Control*), Controle de Acesso Mandatório (ou MAC, do inglês *Mandatory Access Control*) e o Controle de Acesso Baseado em Papéis (ou RBAC, do inglês *Role Based Access Control*).

2.1.2.1 Controle de Acesso Seletivo (DAC)

O DAC é uma política de controle de acesso determinada pelo proprietário (*owner*) do recurso (um objeto, por exemplo) e que se baseia em conceder e revogar privilégios [GW76]. O proprietário do recurso decide quem tem permissão de acesso a determinado recurso e qual privilégio ele tem. O acesso às informações é controlado de acordo com a identidade do usuário e com as regras que especificam os modos de acesso para cada objeto. A política é dita seletiva, porque os usuários podem conceder permissões de acesso aos objetos para outros usuários.

A maioria dos SGBDs comerciais utiliza este tipo de controle de acesso devido à sua flexibilidade, o que o torna apropriado para uma variedade de ambientes com diferentes requisitos de proteção. Entretanto, este modelo contém uma desvantagem: embora cada acesso seja controlado e permitido somente se autorizado, é possível burlar as restrições de acesso definidas pelas autorizações. Um sujeito que tem permissão de ler dados pode passá-los para outros sujeitos não autorizados sem o conhecimento do proprietário dos dados.

Um exemplo de controle de acesso seletivo é o modelo proposto por [BBCDN04]. Ele propõe um modelo de controle de acesso seletivo para mapas geográficos, compostos por objetos representados por uma camada geométrica e uma camada topológica. Na camada geométrica, a forma e localização dos objetos na superfície terrestre são representadas através de pontos, linhas e polígonos. Na camada topológica, as propriedades espaciais dos objetos são representadas por relações topológicas com outros objetos [RBKW91] (*Disjoint*, *Touch*, *In*, *Contains*, *Equal*, *Cross*, *Overlap*).

A estrutura adotada para representar uma autorização é baseada na tupla < *s*, *o*, *m* >, onde *s* é o sujeito que recebe a autorização, *o* o objeto ao qual se terá autorização e *m* o modo de acesso. Tanto autorizações positivas, atribuindo permissões, quanto autorizações negativas, negando acesso, são suportadas. O modelo considera que mapas são representados de acordo com o Modelo de Dados Espaciais em Camadas (LSDM) [BCB03], porém modificado para suportar apenas as camadas topológica e geométrica, chamando-o agora de TSDM. A figura 2.1, retirada de [BBCDN04], apresenta ambas as camadas.

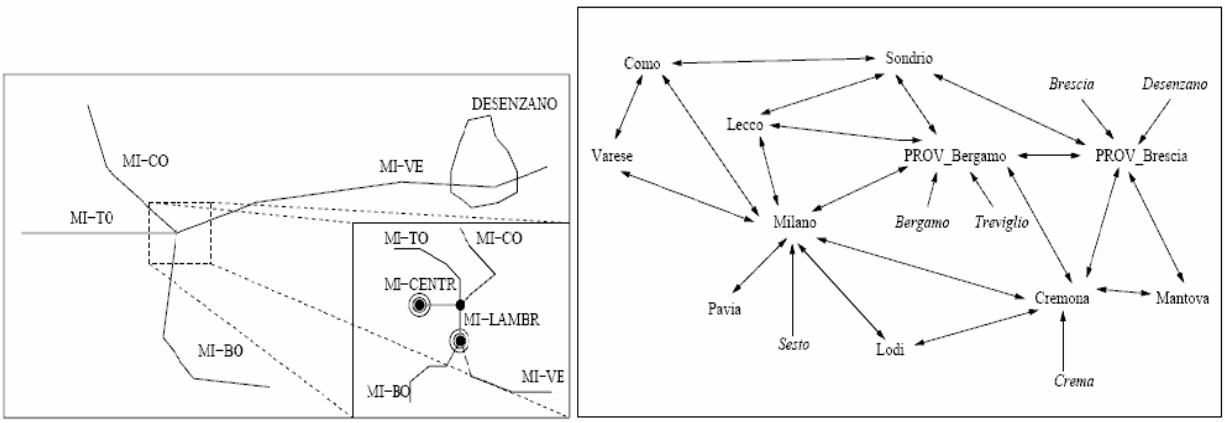


Figura 2.1: Modelo de dados espaciais em camadas

O lado esquerdo da figura 2.1 representa um exemplo de uma base de dados geográfica: rede ferroviária da Lombardia (Norte da Itália). O lado direito representa relações

topológicas entre os objetos. Setas bidirecionais representam uma relação do tipo "Touch", enquanto que as unidirecionais representam uma relação do tipo "In".

No TSDM, o esquema do banco de dados espacial é definido como um conjunto de tipos de objetos (chamados *feature type*) e um conjunto de grupos de objetos (chamados de *map types*). Um *map type* é um conjunto de *feature types*. Um *feature type* tem atributos descritivos e um atributo espacial.

Suponha que o esquema TSDM de um banco de dados contenha 4 *feature types* (Província, Cidade, Rede Ferroviária e Acidente) e um *map type* (Rede_M). Os *feature types* Província, Cidade, Rede Ferroviária e Acidente pertencem à Rede_M. Uma instância deste esquema, figura 2.1, contém uma representação geométrica completa da *feature* Rede Ferroviária e uma representação topológica das *features* Província e Cidade. Quando um acidente acontece, tanto uma representação geométrica (um ponto), quanto uma representação topológica, representada por uma relação *In* entre Acidente e a Cidade onde ele ocorreu, podem ser inseridos no banco de dados

O privilégio, no modelo, é definido de acordo com o tipo de objeto a que ele pode ser concedido. Por exemplo, para objetos do tipo *feature type*, existem privilégios do tipo seleção, remoção e atualização. Já para grupos de objetos, *map type*, existem privilégios do tipo inclusão ou exclusão de objetos. As autorizações são aplicadas aos objetos, ou grupo deles, assegurando aos sujeitos determinados tipos de acesso sobre eles.

2.1.2.2 Controle de Acesso Mandatário (MAC)

No Controle de Acesso Mandatório (MAC) a política de acesso é determinada pelo sistema e não pelo proprietário do recurso. Por classificar hierarquicamente sujeitos e objetos do sistema, o MAC satisfaz os requisitos de organizações militares, governamentais e comerciais, que são naturalmente hierárquicas. Típicos níveis de segurança são: altamente secreto (TS), secreto (S), confidencial (C) e não classificado (U), onde TS>S>C>U.

A organização hierárquica assegura que informações confidenciais não fluam para níveis inferiores da hierarquia, baseando-se em dois princípios formulados por Bell e LaPadula [BP76]. O primeiro define que um sujeito não tem acesso de leitura a objetos de um nível superior ao seu. O segundo define que um sujeito não tem permissão de escrita em objetos de um nível inferior ao seu. Isso assegura que informações não fluam de níveis superiores para inferiores, protegendo-as contra invasões através de meios sofisticados, como Cavalos de Tróia embutidos em programas [EBS95].

Um Cavalo de Tróia é um programa com uma função útil, que contém funções adicionais ocultas que exploram clandestinamente as autorizações legítimas do processo que o invocou. Considere o seguinte exemplo, retirado de [Sasaoka02], para entender como um Cavalo de Tróia pode passar informações a usuários não autorizados.

Suponha que Ana, uma gerente de alto nível, crie uma tabela Mercado contendo importantes informações sobre novos produtos que deveriam ser mantidos em segredo. Considere agora Tom, um dos subordinados de Ana, que secretamente trabalha para uma outra empresa concorrente. Tom cria uma tabela Auxiliar e autoriza Ana a acessar dados desta tabela com permissão de escrita. Note que Ana nem precisa saber da existência desta tabela. Tom, então, altera uma aplicação de planilhas para incluir duas operações ocultas, uma de leitura da tabela Mercado e outra de escrita na tabela

Auxiliar. Em seguida, Tom envia a nova aplicação à gerente. Suponha agora que Ana execute esta aplicação. Como resultado, informações sensíveis de Mercado são transferidas para a tabela Auxiliar e assim Tom poderá acessá-las e repassá-las ao concorrente.

No controle de acesso seletivo, este tipo de aplicação maliciosa funciona perfeitamente, uma vez que restrições de acesso definidas por autorizações podem ser burladas. No exemplo, Ana e Tom acessam as tabelas de acordo com as autorizações definidas, porém a informação sensível não foi protegida. Já no controle de acesso mandatório, se Tom não tem acesso de leitura à tabela Mercado, esta tabela terá um nível de segurança superior ao nível de Tom. Um sujeito ou processo com permissão de leitura à tabela Mercado não seria capaz de escrever em objetos de nível inferior. No exemplo, Ana não teria permissão de acesso para escrever dados na tabela Auxiliar. Desta forma, o controle de acesso mandatório evita ataques de Cavalos de Tróia.

Um exemplo de controle de acesso mandatório é o modelo proposto por [MBT89]. Ele propõe uma extensão ao modelo de dados orientado a objetos ORION [BANE87] para incorporar propriedades de segurança ao modelo, chamando-o agora de SORION. No ORION todas as entidades são modeladas como objetos (instâncias de classes), que possuem um identificador único (*Object-ID*). Dois tipos de hierarquias de classes são considerados: *IS-A* e *IS-PART-OF*. No primeiro, uma classe pode ter subclasses associadas a ela. No segundo, um objeto de uma classe pode ser formado por uma agregação de objetos pertencentes a outras classes (objeto composto).

O SORION define níveis de segurança para todas as entidades, além de propriedades de segurança que devem ser satisfeitas. As entidades consideradas são: classes, objetos, métodos e atributos. Por exemplo: se um objeto *o* possui um nível de segurança Secreto, todos os seus atributos também serão Secretos; o nível de segurança de uma subclasse *C1* deve ser igual ou superior ao nível de segurança da superclasse *C*; se um método *m* é definido em mais de uma subclasse de *C*, seu nível de segurança será o maior nível entre todas as subclasses derivadas de *C*.

As políticas de segurança definidas pelo modelo são as seguintes:

1. Níveis de segurança são atribuídos a sujeitos e entidades.
2. Um sujeito tem permissão de acesso (leitura) a uma entidade se seu nível de segurança for maior ou igual ao da entidade.
3. Um sujeito tem permissão de acesso (escrita) a uma entidade se seu nível de segurança for igual ao da entidade.
4. Um sujeito pode executar um método se seu nível de segurança for igual ou maior ao do método e ao do objeto ao qual o método pertence.
5. O método é executado no nível de segurança do sujeito que o iniciou.
6. Durante a execução do método *m1*, se outro método precisar ser executado, então o nível de segurança definido para *m1* deve ser igual ou maior ao do método *m2* e ao do objeto ao qual *m2* pertence.
7. Se um objeto for criado a partir da execução de um método, é atribuído a ele o nível de segurança do sujeito que o iniciou.

O modelo também propõe uma solução para o problema de poliinstanciação. Isto acontece quando dois sujeitos pertencentes a diferentes níveis possuem visões diferentes de uma mesma entidade. Por exemplo, suponha que um sujeito *S1*, com nível de

segurança Secreto, tenha criado uma classe *C1*. Um sujeito *S2*, com um nível de segurança Não-Classificado, não sabe da existência de *C1* uma vez que seu nível de segurança é menor que o nível de *C1*. Suponha agora que *S2* crie uma classe para representar outra entidade do mundo real, porém com a mesma identificação de *C1*. Neste caso, há duas classes com o mesmo identificador, porém com diferentes níveis de segurança. Na solução, o identificador do objeto é composto pelo nome atribuído pelo usuário e pelo nível de segurança ao qual ele pertence.

2.1.2.3 Controle de Acesso Baseado em Papéis (RBAC)

Existe uma vasta gama de políticas de segurança dependendo da área e das empresas onde estas são aplicadas. Em uma organização, cada área possui requisitos únicos de segurança e muitas delas são difíceis de satisfazer com apenas modelos tradicionais de controle de acesso seletivo e mandatório.

A política de controle de acesso baseada em papéis (RBAC) [FK92] tem suas decisões de acesso baseadas nas funções que um usuário pode executar dentro de uma organização. Por exemplo, os papéis que um indivíduo pode desempenhar em um hospital incluem cirurgião, enfermeira, anestesista. O RBAC baseia-se em autorizar ou não o acesso aos objetos de acordo com o papel do sujeito na organização. Neste modelo, os direitos de acesso do sujeito aos objetos são determinados pelos papéis a ele atribuídos e pelas permissões atribuídas a cada papel. Um papel é uma coleção de permissões determinadas pelo sistema, baseado no conjunto de responsabilidades e atividades do sujeito dentro de uma organização. A figura 2.2, retirada de [Sasaoka02], apresenta o relacionamento entre usuários, papéis, operações e objetos.

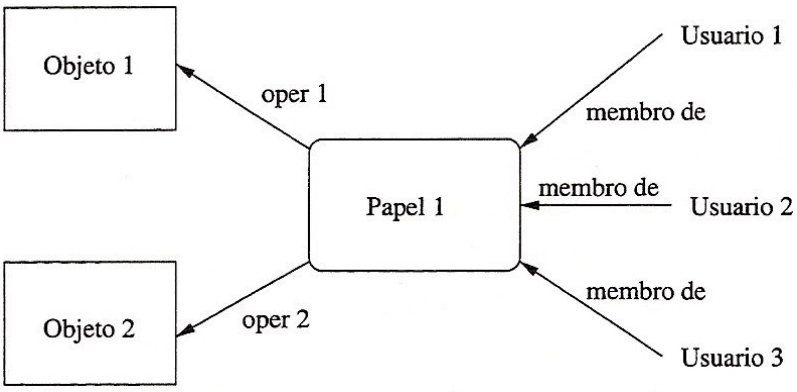


Figura 2.2: Relacionamento entre usuários, papéis, operações e objetos [Sasaoka02]

[BCDP05] baseia-se no modelo RBAC e o estende para lidar com informações espaciais e baseadas em localização, chamando-o de GEO_RBAC. Papéis são associados a um usuário e são ativados de acordo com a sua localização no espaço. Ou seja, de acordo com a localização de um usuário, ele poderá assumir um determinado papel que lhe permitirá executar operações específicas sobre determinados objetos do sistema.

Os papéis são definidos pelo par < *r*, *e* >, onde *r* representa o nome do papel e *e* a sua extensão, ou seja, os limites espaciais dentro dos quais o papel pode ser assumido por um usuário. Uma vez que um usuário tenha assumido um determinado papel dentro

do sistema, de acordo com sua localização, ele poderá executar todas as operações que as permissões daquele papel lhe autorizam a realizar. O modelo também suporta hierarquia de papéis, de modo que permissões atribuídas a níveis inferiores são automaticamente atribuídas às permissões de níveis superiores.

Outro exemplo de modelo de autorização baseado em papéis é o modelo desenvolvido para o protótipo Orion [RBKW91], usando o conceito de autorização implícita no domínio de entidades. Por exemplo, uma permissão em uma classe implica em permissão de leitura em todas as instâncias da classe. Regras para sujeitos dependem dos relacionamentos especificados entre papéis de usuários. Por exemplo, uma ligação entre secretária e empregada indica que todas as secretárias são também empregadas; portanto, todas as autorizações para empregadas são válidas também para secretárias.

Uma nova forma de controle de acesso é baseada no uso de serviços baseados em localização e aplicações móveis. Em particular, permissões atribuídas a um sujeito dependem da sua localização no espaço de referência. Uma vez que sujeitos geralmente pertencem a categorias bem definidas, e objetos, ao qual serão atribuídas permissões, estão localizados no espaço, políticas de controle de acesso devem atribuir permissões baseadas nas localizações dos objetos, bem como dos usuários.

2.1.2.4 Controle de acesso temporal para bancos de dados

Em muitas situações do mundo real, permissões têm uma dimensão temporal. Ou seja, elas são válidas apenas por um determinado período de tempo. Em um SGBD relacional típico, por exemplo, não é possível especificar que o usuário tem permissão de acesso a uma relação por apenas um dia ou uma semana. Uma solução é implementar um gerenciador de autorizações no nível da aplicação. No entanto, esta solução é inadequada, por tornar a lógica da aplicação complexa e o gerenciamento das autorizações fora do escopo do SGBD.

Dado este problema, [BBFS96b] apresenta um modelo de controle de acesso temporal seletivo, baseado em dois conceitos: intervalo temporal para autorizações e dependência temporal entre autorizações. O primeiro conceito define que a autorização tem um intervalo de tempo associado a ela. O segundo conceito define regras de derivação automática de novas autorizações, expressando dependências temporais entre autorizações. Por exemplo, uma regra de derivação pode definir que um usuário pode ler um objeto contanto que outro usuário também possa lê-lo, em dado intervalo de tempo.

No modelo, autorizações positivas e negativas são suportadas, sendo que em caso de conflitos, utiliza-se a regra de precedência das autorizações negativas. Além disso, ele considera privilégios administrativos especiais que permitem aos usuários concederem e revogarem autorizações. Por exemplo, um usuário com privilégios de administrador sobre um objeto pode conceder e revogar autorizações de qualquer modo de acesso sobre este objeto.

A autorização é definida pelo modelo como uma tupla da seguinte forma < tempo, autorização >, ou melhor, < (*ti*,*tf*), (*s*, *o*, *m*, *pn*, *g*) >, onde:

- *ti*: tempo inicial
- *tf*: tempo final
- *s*: usuário que recebe a autorização
- *o*: objeto

- *m*: modo de acesso
- *pn*: se *pn* = +, *s* recebe a autorização, se *pn* = -, a autorização é negada
- *g*: usuário que concede a autorização

Como exemplo, considere as autorizações e regras de derivação da figura 2.3, retirada de [BBFS96b]. Segundo o artigo, as seguintes regras podem ser derivadas:

- ([10,20], (Chris, o1,read,+,Sam)) e
  ([30,35], (Chris,o1,read,+,Sam)) das autorizações A1 e A2 e regra R1.
- ([10,20], (Matt,01,read,+,Sam)) da autorização A1 e regra R2.
- ([5,9], (John,o1,read,+,Sam)),
  ([21,29], (John,o1,read,+,Sam)), e
  ([41,∞], (John,o1,read,+,Sam)) da regra R3.
- ([5,9], (Bob,01,read,+,Sam)) da regra R4.
- ([5,9], (Jim,o1,read,+,Sam)), das regras R4 e R5.

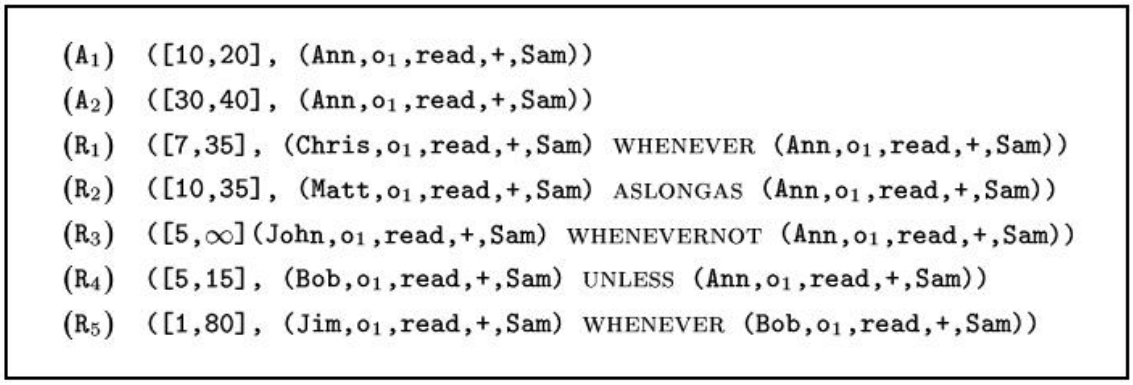


Figura 2.3: Um exemplo de regras de autorização e derivação [BBFS96b]

Este modelo, porém, não provê autorizações periódicas. Em algumas organizações, as autorizações concedidas a usuários devem ser feitas sob medida para o padrão de atividades de cada usuário dentro da organização. Portanto, usuários devem ter autorizações de acesso somente para o período de tempo no qual se espera que eles necessitem dos dados. Um exemplo de autorização é "um funcionário deve ter acesso aos dados da empresa todos os dias úteis das 08h00 às 18h00". Além disso, autorizações periódicas também são muito importantes quando se tratam de autorizações de execução de aplicações. O controle do período de tempo em que uma determinada aplicação pode ser invocada é útil para a otimização de recursos. Programas cuja execução necessita de muitos recursos poderiam ser alocados para períodos nos quais outros programas não fossem executados.

Neste contexto, [BBFS96a] propõe uma extensão ao modelo de controle de acesso temporal para prover autorizações e regras para acesso periódico. Isto permite que autorizações possam ser periódicas e ter um tempo limitado de validade. Dessa maneira, uma autorização é automaticamente concedida no tempo especificado por uma expressão periódica (definida pelo modelo) e revogada quando esta expira. Assim

sendo, a nova regra de autorização é formada por uma tupla da forma < tempo, período, autorização >, onde:

- tempo: intervalo de tempo [min, max], min e max denotam instantes de tempo inicial *ti* e final *tf* com $0 \leq ti \leq tf$.
- período: uma expressão periódica.
- Autorização: autorização da forma < *s*, *o*, *m*, *pn*, *g* >.

A expressão periódica pode ser: o conjunto das segundas-feiras, os grupos das terceiras horas dos primeiros dias de cada mês, etc. Elas são representadas formalmente por uma expressão definida como *P*. Portanto, < [*min*,*max*], *P*, (*s*,*o*,*m*,*pn*,*g*) > denota que o usuário *g* concedeu uma autorização com privilégio *m* ao usuário *s* para acessar o objeto *o*, que é válida para cada instante *P* limitado pelo intervalo *min* e *max*. O atributo *pn* indica se a autorização é *positiva* ou *negativa*.

Por fim, [BBFS97] estende o modelo [BBFS96b] para considerar uma administração descentralizada para o modelo de controle de acesso temporal. A política administrativa é estendida para permitir que o proprietário do objeto e usuários com o privilégio de administrador possam conceder e revogar permissões de qualquer tipo sobre o objeto. Além disso, o proprietário pode seletivamente conceder autorizações com a permissão de *grant*, permitindo delegar administrações de um tipo de privilégio (leitura, escrita) para determinados intervalos de tempo.

Outros modelos de controle de acesso foram desenvolvidos para bancos de dados relacionais [EN97, KPSN96, AJ96], orientados a objetos [TS96, EBFS94] e para a Web, envolvendo documentos HTML [SBJ96] e XML [DSCP01].

2.2 Sistemas de Informação Geográfica

A partir dos anos 90, o espaço tornou-se objeto de estudo em bancos de dados. Dados espaciais descrevem fenômenos aos quais esteja associada alguma dimensão espacial. Uma classe particular destes dados corresponde aos dados geográficos, também chamados geoespaciais, que descrevem fatos, objetos e fenômenos associados a uma localização na superfície terrestre.

Sistemas de Informação Geográficas (SIG) são sistemas automatizados usados para armazenar, analisar, manipular e visualizar dados geográficos, ou seja, dados que representam objetos e fenômenos em que a localização geográfica é uma característica inerente à informação e indispensável para analisá-la.

Os modelos de dados mais comuns em SIG são o modelo *raster* ou *matricial* e o modelo *vetorial* [EGSV98]. O formato matricial é caracterizado por uma matriz de células de tamanhos regulares, onde a cada célula é associado um valor representando as características geográficas da região correspondente. Quanto maior for a dimensão de cada célula (resolução), menor é a precisão ou detalhe na representação do espaço geográfico. O formato vetorial centra-se na precisão da localização dos elementos no espaço, descrevendo dados espaciais como uma combinação de formas geométricas (pontos, linhas e polígonos).

Os dados geográficos são geralmente caracterizados a partir de dois componentes fundamentais: *atributo* e *localização*. O componente *atributo* descreve as

propriedades temáticas de uma entidade geográfica, como o nome, por exemplo. O componente *localização* informa a localização espacial do fenômeno, ou seja, o seu georreferenciamento, associada às propriedades geométricas e topológicas. Por exemplo, a região da bacia amazônica possui atributos descritivos como nome e clima dominante, mas também os atributos espaciais que determinam sua posição geográfica, extensão e geometria.

Os SIGs permitem compatibilizar a informação proveniente de diversas fontes, como informação de sensores espaciais (detecção remota / sensoriamento remoto), informação recolhida com GPS ou obtida com os métodos tradicionais da Topografia.

Entre as questões em que um SIG pode ter um papel importante, encontram-se:

- Localização: Inquirir características de um lugar concreto
- Condição: Cumprimento ou não de condições impostas aos objetos.
- Tendência: Comparação entre situações temporais ou espaciais distintas de alguma característica.
- Rotas: Cálculo de caminhos ótimos entre dois ou mais pontos.
- Modelos: Geração de modelos explicativos a partir do comportamento observado de fenômenos espaciais.

Os campos de aplicação dos Sistemas de Informação Geográfica, por serem muito versáteis, são muito vastos, podendo-se utilizar na maioria das atividades com uma componente espacial. A profunda revolução que provocaram as novas tecnologias afetou decisivamente a evolução da análise espacial.

2.2.1 Operadores espaciais

Além de predicados convencionais, as consultas em bancos de dados geográficos utilizam operadores espaciais. Estes relacionamentos espaciais são classificados em topológicos, métricos e direcionais [Gut94].

Os relacionamentos topológicos, como “dentro de” e “superposto a”, retornam um valor booleano e são invariantes em face de transformações de escala, translação e rotação. A definição de um conjunto mínimo de relacionamentos topológicos é objeto de muito debate na literatura. Freeman [Fre75] define até treze tipos de relacionamentos: “à esquerda de”, “à direita de”, “acima”, “abaixo”, “atrás”, “próximo a”, “longe de”, “ao lado de”, “tocando”, “dentro de”, “fora de” e “entre”. Há ainda outros que listam seis [Feu93] e outros que listam oito [EH90].

Os relacionamentos métricos envolvem medidas como distância ou perímetro. Já os relacionamentos direcionais envolvem um marco de referência, um objeto de referência e o objeto em questão. O marco de referência determina a direção na qual o objeto em questão está localizado em relação ao objeto de referência.

2.2.2 O trabalho de Sasaoka

[Sasaoka02] propõe um modelo de controle de acesso genérico para bancos de dados geográficos, podendo ser adaptado segundo as necessidades de uma determinada área de aplicação. O modelo considera que todos os dados são armazenados em um banco de dados espacial e que todas as regras de autorização são armazenadas em um repositório especial. A estrutura adotada para representar uma autorização é baseada na tripla $< s, o,$

m >, onde "s" é o sujeito que recebe a autorização, "o" o objeto ao qual se terá autorização e "m" o modo de acesso. Somente autorizações positivas são suportadas, ou seja, autorizações que atribuam diretamente permissões aos objetos.

A granularidade do modelo é definida pelo próprio objeto a ser acessado. É possível acessar objetos do tipo ponto, linha, polígono ou conjunto deles. A solução proposta considera duas representações alternativas de regras: < *s*, *o*, *m* > e < *s*, *C*, *m* >. A primeira representação referencia explicitamente o identificador do objeto, enquanto que a segunda contém uma especificação de consulta SQL espacial que define os objetos a serem controlados pela regra. Por exemplo, a autorização "Ana tem acesso de leitura sobre o rio Tietê do estado de São Paulo." referencia explicitamente o identificador do objeto "rio Tietê". Já no caso da autorização "Ana tem acesso de leitura sobre todos os rios do estado de São Paulo" referencia todos os objetos resultantes da consulta "selecionar todos os rios do estado de São Paulo.".

A figura 2.4, retirada do modelo, mostra como consultas espaciais envolvem relacionamentos entre diferentes tipos de objetos espaciais. Estas consultas retornam um resultado, que é o alvo do controle e acesso, o objeto *o* da tupla < *s*, *o*, *m* >.

A apresentação do caractere *C* em uma célula da tabela representa um tipo de consulta. Por exemplo, *C2* corresponde a consultas envolvendo relacionamentos entre pontos e linhas ou linhas e pontos. Consultas podem retornar atributos descritivos, espaciais ou ambos. Podem também retornar objetos espaciais e não espaciais. Alguns exemplos de consultas são:

- consulta *Cx* "Quais os assinantes cadastrados no banco de dados" retorna objetos não espaciais (assinantes).
- consulta *Cy* "Quais os tipos de cabos instalados no bairro Cambuí" retorna atributos descritivos (tipo de cabo).
- consulta *Cz* "Supermercados com mais de 5 telefones instalados" retorna objetos espaciais (supermercados), supondo que têm um componente espacial.

As consultas *Cx* e *Cz* não utilizam predicados espaciais, enquanto que a consulta *Cy* utiliza predicado espacial.

	Ponto	Linha	Polígono
Ponto	*C1*	*C2*	*C3*
Linha	*C2*	*C4*	*C5*
Polígono	*C3*	*C5*	*C6*

Figura 2.4: Objetos espaciais e seus inter-relacionamentos [Sasaoka02].

O modelo pressupõe uma administração centralizada de autorizações, ou seja, apenas o administrador pode conceder e revogar permissões, assim como define algoritmos para atualização da base de regras e leitura dela quando há pedidos de acesso. O direito de acesso é concedido apenas se existir uma regra explícita autorizando o sujeito a acessar aquele objeto com aquele modo de acesso.

A questão do conflito de regras de autorização, como por exemplo, regras que atuam em objetos que estão totalmente ou parcialmente contidos dentro de outros objetos, também é tratada por este modelo. Neste contexto, existem diversas alternativas propostas pelo modelo que podem ser adotadas para resolver o conflito entre as diferentes regras.

A principal diferença do modelo de [Sasaoka02] em relação ao outros é a caracterização espacial envolvida nas permissões. Ou seja, uma permissão espacial está diretamente relacionada à consulta espacial que ela deve satisfazer. No entanto, há ainda vários pontos a contemplar, como por exemplo, a organização das regras de acesso.

Vale também mencionar um aspecto adicional do modelo de [Sasaoka02]. Existe uma hierarquia para objetos espaciais com relação à permissão de acesso, onde o nível básico é o ponto e o nível superior é o polígono. A hierarquia a partir do seu nível superior para o inferior corresponde a: polígono → linha → ponto. Isto significa que ao se obter acesso a um polígono, obtém-se também acesso a todos os objetos que estiverem contidos neste polígono. Quando se tem acesso a uma linha, tem-se acesso a todos os seus pontos.

2.3 Versões

Os objetos modelados em sistemas com versionamento representam em geral entidades bem definidas e com identidade própria no universo da aplicação [delVal97]. Nas diferentes abordagens, estes objetos conceituais são chamados de Objetos Multiversão [CJ90], de Projeto [Kat90], Versionável [GSdS95] dentre outros. Para cada um destes objetos existe um **Conjunto de Versões** que pode ser visto como objetos que descrevem variações de um mesmo objeto conceitual. Aqui serão usados os termos **Objeto Multiversão** para referenciar tais objetos e **Bancos de Dados Multiversão** [CJ90] para referenciar bancos de dados compostos por objetos multiversão.

As versões de um mesmo objeto possuem, geralmente, um relacionamento de derivação entre elas. Quando uma nova versão é criada, ela obtém estrutura e valores de outras versões ancestrais das quais é derivada. No entanto, sua evolução pode ser independente da versão ancestral. Já o relacionamento entre versões de objetos diferentes define uma relação de composição. Por exemplo, um objeto pode ser formado por versões de diferentes objetos, definindo, assim, uma relação de composição.

2.3.1 Versões em sistemas CAD/CASE

Computer-Aided Design (CAD), ou desenho auxiliado por computador, é o nome genérico de sistemas computacionais utilizados pela engenharia, geologia, arquitetura, e design para facilitar o projeto e desenho técnicos. Como resultado, um sistema CAD produz um objeto de engenharia que pode conter múltiplas alternativas para os seus módulos. Um sistema CASE (do inglês *Computer-Aided Software Engineering*) é uma classificação que abrange todo sistema que auxilia atividades de engenharia de software, desde análise de requisitos e modelagem até programação e testes. Assim como nos sistemas CAD, o uso de versões em sistemas CASE auxilia o desenvolvimento de múltiplas alternativas de projeto, como, por exemplo, o desenvolvimento e manutenção de diferentes configurações de software.

[Kat90] define que o processo de desenvolvimento dos objetos em sistemas CAD e CASE precisa suportar manipulação de objetos complexos, em que novos dados não substituem os anteriores, mas são mantidos como diferentes versões. Além disso,

deve existir suporte ao trabalho cooperativo, uma vez que pessoas participam simultaneamente no desenvolvimento do projeto e podem acessar concorrentemente os mesmos dados e criar suas próprias interpretações dos objetos modelados.

A dinâmica em sistemas com suporte ao trabalho cooperativo pode ser entendida como o trabalho de várias pessoas e equipes que criam e manipulam versões de objetos diferentes em um projeto. Neste sentido, é necessário estruturar os dados de forma a dar possibilidades para o trabalho individual, o trabalho coletivo e o intercâmbio de informações.

Em geral as propostas de versionamento estruturam o banco de dados em vários níveis que refletem uma organização e direito de acesso às diferentes partes do projeto cooperativo. Em [CK86] os seguintes níveis são definidos:

- Bancos de dados públicos: contém versões de objetos que não podem ser removidas ou alteradas pelos usuários, mas somente acessadas por projetistas autorizados dentro do ambiente de projeto.
- Bancos de dados privados: contém versões de objetos que são modificadas pelo projetista que criou o banco de dados privado (dono). Este banco não é acessível a outros projetistas.
- Bancos de dados de projeto: contém versões de objetos que são acessíveis somente pelos projetistas que trabalham cooperativamente em um projeto.

As versões de objetos são divididas em três tipos, podendo ser promovidas de um estado a outro pelos projetistas diretamente:

- Versões temporárias (*transient*): armazenadas no banco de dados privado. Podem ser removidas e atualizadas apenas pelos projetistas que criaram as versões.
- Versões de trabalho (*working*): podem existir em um banco de dados privado ou de projeto. É considerada estável e não pode ser atualizada. Somente o dono da versão pode removê-la.
- Versões publicadas (*released*): armazenadas no banco de dados público. Não pode ser alterada ou removida por projetista ou usuário algum.

[Kat90] apresenta um modelo de organização similar, que chama os diferentes níveis de espaço de trabalho de privados, arquivos ou de grupo. Os projetistas criam versões de objetos em um espaço de trabalho privado a partir de operações de *check-out* de versões do espaço de trabalho de arquivos. Por outro lado, integram as versões de objetos presentes em seus espaços de trabalho privados no espaço de trabalho de arquivos através de operações de *check-in*. O espaço de trabalho de grupo permite o trabalho cooperativo entre diferentes usuários participantes de um projeto.

2.3.2 Versões no desenvolvimento de aplicações para bancos de dados

Aplicações para banco de dados são programas que consultam ou modificam dados persistentes armazenados em um banco de dados. Exemplos de aplicações são: Sistemas Integrados de Gestão Empresarial (ou ERP, do inglês *Enterprise Resource Planning*) e Gerenciamento de Relacionamento com o Cliente (ou CRM, do inglês *Customer Relationship Management*).

Os ERPs, em termos gerais, são uma plataforma de software desenvolvida para integrar os diversos departamentos de uma empresa, possibilitando a automação e armazenamento de todas as informações de negócios. Já o CRM é um sistema integrado de gestão com foco no cliente, constituído por um conjunto de procedimentos/processos organizados e integrados num modelo de gestão de negócios. Seu objetivo principal é auxiliar as organizações a angariar e fidelizar clientes ou prospectos, fidelizar clientes atuais procurando atingir a sua satisfação total, através do melhor entendimento de suas necessidades e expectativas e formação de uma visão global dos ambientes de marketing.

O desenvolvimento de aplicações para banco de dados possui as seguintes dificuldades [CAASV04]:

- Múltiplos desenvolvedores precisam usar a mesma base de dados. Para não haver interferência entre os trabalhos, os desenvolvedores precisam usar o banco de dados por um tempo determinado ou criar cópias privadas dos dados.
- Paralelismo de testes. Desenvolvedores precisam executar um grande número de testes como parte do conjunto de testes de regressão. A execução de múltiplos testes em paralelo sobre o mesmo banco de dados é um problema, uma vez que mudanças feitas por um teste pode impactar outro teste.
- Rápida restauração do estado inicial do banco de dados após a execução de um conjunto de testes. Quando os desenvolvedores têm um intervalo de tempo reservado para a execução dos testes, as alterações devem ser rapidamente desfeitas após a execução. A demora neste processo pode impactar a execução do ciclo de testes de outro desenvolvedor.

Estas dificuldades levantam a necessidade do uso de versões no desenvolvimento de aplicações para bancos de dados. O uso de versões permite que desenvolvedores diferentes trabalhem em versões diferentes dos mesmos objetos do banco de dados, sem afetar o trabalho de outros desenvolvedores. Os requisitos de versionamento de dados para este tipo de aplicação são:

1. Múltiplos bancos de dados lógicos devem ser suportados sobre o mesmo banco de dados físico. Alterações feitas em uma versão lógica não devem ser visíveis por outra versão do banco de dados.
2. O versionamento dos dados deve ser transparente, ou seja, não devem ser feitas alterações no código da aplicação para suportar o versionamento dos dados.
3. Todas as funcionalidades do banco de dados devem ser suportadas em cada uma de suas versões lógicas, como restrições de integridade, por exemplo.
4. As alterações feitas em uma versão lógica devem ser facilmente reversíveis.

A figura 2.5, retirada de [CAASV04], ilustra as necessidades de um desenvolvedor no uso da sua própria versão de código assim como da versão de seus dados. O versionamento de código é feito por sistemas de controle de código fonte, como SCCS e ClearCase® [CW98]. Para o versionamento dos dados, [CAASV04] propõem um framework baseado em transações longas para bancos de dados relacionais.

Em alguns tipos de aplicações para banco de dados, como aplicações CAD e CASE, uma transação pode envolver interação com o usuário e pode durar uma grande quantidade de tempo (transações longas). Cada desenvolvedor pode trabalhar em sua transação longa e isolar o seu trabalho de outros desenvolvedores.

A unidade de versionamento é no nível de linhas [AACSV03]. Para isso, uma coluna *VersionNo* é adicionada em todas as tabelas do banco de dados relacional para indicar a versão associada a cada linha. Sempre que uma linha de uma tabela é alterada por uma transação, uma nova versão é associada a ela. Isto é feito através da inserção de uma nova linha, como cópia da linha original, cujos valores são modificados pela transação em questão e uma nova versão é associada a ela. Uma transação possui um registro de todas as versões que estão associadas às alterações realizadas por ela.

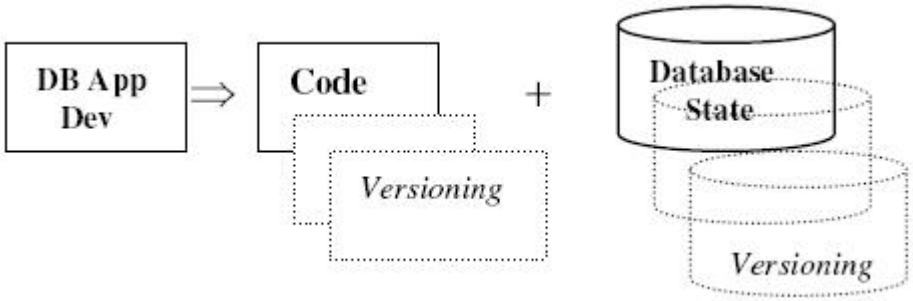


Figura 2.5: Versionamento de código e dados [CAASV04]

Para isolar as transações, uma *visão* é criada em cada tabela, de modo que uma transação de usuário acesse apenas os dados relevantes à sua transação. Ou seja, o mecanismo de visões faz com que uma transação acesse apenas as linhas de uma tabela cujos valores da coluna *VersionNo* estejam presentes no registro de versões da transação. Este mecanismo satisfaz todos os requisitos de versionamento citados anteriormente, pois deixa transparente o acesso aos dados por parte das transações, sem que o usuário tenha que se preocupar com o mecanismo de versionamento.

2.3.3 Versões em bancos de dados para SIGs

O modelo de [delVal97] propõe um mecanismo de versões em bancos de dados para SIGs, com o objetivo de solucionar limitações da manipulação de dados por parte de um SIG típico, como: não permitir a manipulação de dados temporais; não considerar a possibilidade de múltiplas representações espaciais; não suportar gerenciamento de cenários alternativos. A solução apresentada pelo modelo parte da constatação de que, na prática, estas limitações envolvem implicitamente a necessidade de gerenciamento de múltiplos estados (as versões) de uma mesma entidade do mundo real.

O modelo é orientado a objetos e se baseia na idéia da criação de um banco de dados secundário (contexto de trabalho criado a partir do banco de dados original) através da seleção de determinados objetos e suas versões do banco de dados principal, restritos a uma região alvo. Esse banco de dados secundário pode ser manipulado (alteração, visualização, criação e remoção de objetos) por uma determinada equipe e depois integrado novamente à base de dados original, gerando uma nova versão para os objetos modificados. Para isso, o modelo se baseia em três conceitos principais:

contexto espaço-temporal (CET), *espaço de trabalho* (ET) e *contexto de trabalho* (CTR).

Contextos Espaço-Temporais (CET)

Um CET representa uma versão ou estado consistente dentro da realidade geográfica modelada no banco de Dados. Ele é caracterizado por *dimensões de descrição*, cada uma das quais descreve um aspecto semântico da realidade modelada. Por exemplo, o tempo válido é uma dimensão de descrição e o tempo de transação outra. O BD é formado por um conjunto de CETs.

A caracterização dos CET é feita através da associação de um *vetor de dimensões* a cada CET dentro do banco de dados, cujo conteúdo o diferencia dos demais CET. Por exemplo, se os objetos do BD são versionados no tempo (uma dimensão) e em representação (outra dimensão), então o CET [*t1*,*r1*] corresponde ao conjunto de versões de objetos que tenha estado não nulo de representação *r1* no tempo *t1*. Portanto, o vetor de dimensões fornece a descrição semântica de um CET. Versões diferentes de um mesmo objeto representam estados semânticos diferentes e devem estar necessariamente associadas a CET diferentes.

O modelo considera dois critérios de consistência para serem modelados usando dimensões: *Consistência temporal*, associada ao tempo (tempo válido e de transação, por exemplo), e *Consistência de representação espacial*, associada a aspectos de representação espacial (escala e projeção geográfica, por exemplo).

Espaços de Trabalho (ET)

O conceito de ET é introduzido para permitir o trabalho orientado a projeto espacial cooperativo, simulações ou experimentações. Um ET pode ser visto como um Banco de Dados de Trabalho em um ambiente CAD. Como qualquer banco de dados, um ET possui *esquema* e *extensão* (conteúdo) e está composto por diferentes CET, neste caso chamados de *Contextos de Trabalho* (CTR), que correspondem a diferentes versões ou estados consistentes do conjunto dos objetos na *Extensão* do ET. Assim como para cada CET, cada CTR possui um *vetor de dimensões*. Em geral, um CTR pode ser visto como um CET manipulado em um espaço de trabalho.

Os objetos na Extensão de um ET podem ter versões em cada um dos seus CTR. Essas versões são consistentes com a descrição semântica (representada pelo *vetor de dimensões*) de cada CTR. O resultado do processo de projeto em um ET pode ser re-incorporado ao banco de dados. O ciclo de trabalho no modelo é demonstrado pela figura 2.6, retirada de [delVal97].

O conjunto de CET representa o mundo real geográfico modelado sobre o qual basicamente são realizadas as operações tradicionais de um banco de dados: Criação de objetos, atualizações e consultas. O fato de que o mundo real seja versionado introduz mais riqueza de representação e recuperação de informação.

Equipes de usuários, ou usuário independentes, podem trabalhar paralelamente sobre o banco de dados em um ET, onde são desenvolvidos e testados novos projetos, os quais finalmente podem ser incorporados ao conjunto de CET como atualizações do mundo real.

O modelo tem como suporte o mecanismo de Versões em Bancos de Dados (*DataBase Versions - DBV*) [CJ90, Gan94, GJ94, GJZ95]. A seção a seguir apresenta o modelo DBV.

2.3.4 O modelo DBV

O modelo DBV (CJ90] tem por objetivo estabelecer: identificação da versão de cada objeto, consistência entre versões do BD, histórico de cada objeto, histórico do banco de dados, versões e diferenças entre versões do BD. A característica fundamental do modelo é a distinção entre nível lógico e físico de versionamento. O nível lógico corresponde à visão do usuário enquanto que o nível físico corresponde à visão de implementação.

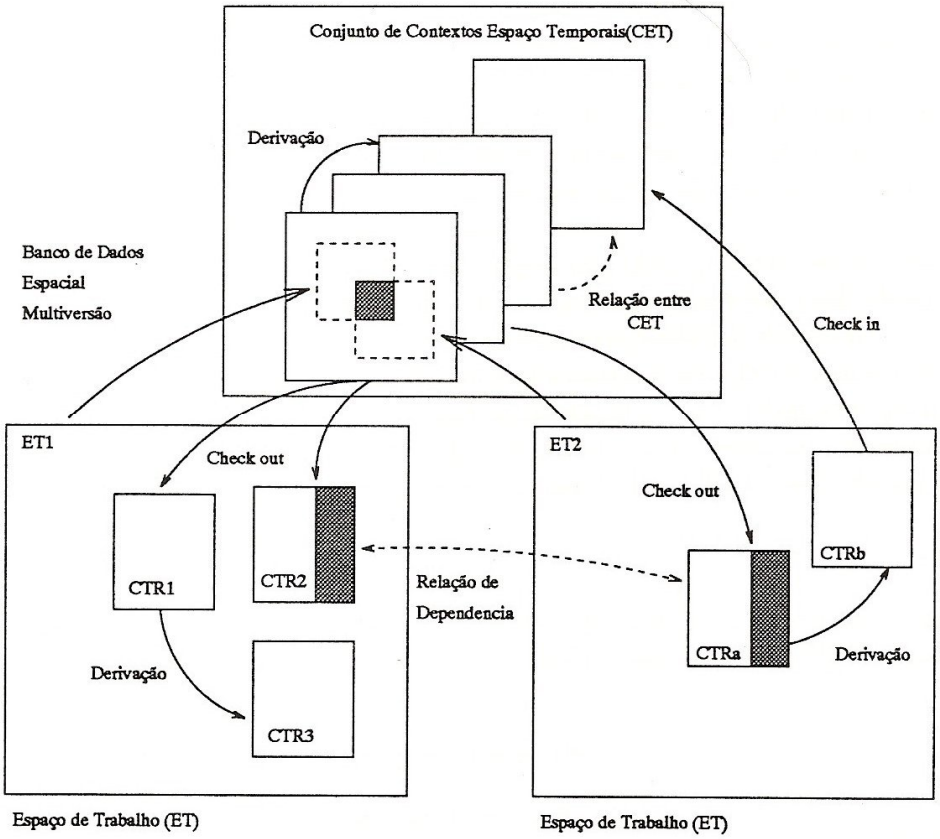


Figura 2.6: O modelo de [delVal97]

No nível lógico, um banco de dados é entendido como um conjunto de Versões de Banco de Dados (*database version-DBV*), isto é, a unidade de versionamento é o banco de dados como um todo. Cada DBV representa um estado diferente do mundo real modelado, contendo versões lógicas dos objetos no BD. A figura 2.7, retirada de [delVal97], apresenta o modelo, para um conjunto de versões de bancos de dados.

Cada versão lógica de um objeto representa o estado de um objeto em uma DBV e é identificada pelo par (*Oid*, *DBVid*), onde: *Oid* corresponde ao identificador do objeto e *DBVid* corresponde ao identificador da DBV na qual aquela versão ocorre. Dois tipos de transações são definidos: transação sobre versões lógicas, que atualiza versões de objetos lógicos em uma ou muitas DBV, e transação sobre DBV, que cria ou remove DBV. A figura 2.6 representa à esquerda sete DBV, cada uma com uma versão dos objetos A e B. No caso da DBV 0.1.1, o valor da versão de A é *a1* e da versão de B é *NULL*.

Em geral, uma nova DBV é criada como uma derivação de uma DBV já existente. O identificador *DBVid* permite determinar o relacionamento de derivação,

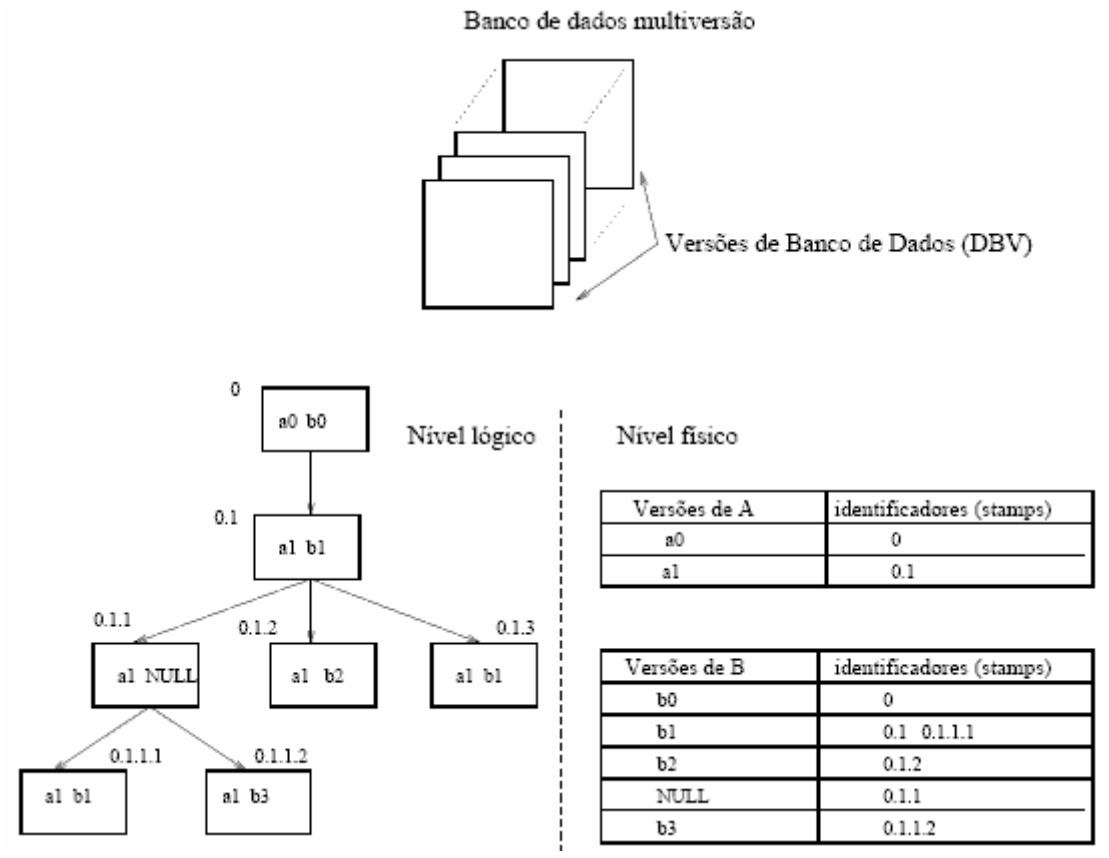
através de seu prefixo. Por exemplo, as DBV derivadas da DBV 0.1 são 0.1.1, 0.1.2 e 0.1.3.

No nível físico, cada objeto é formado pelo conjunto de suas versões físicas e uma *tabela de associação*. Uma mesma versão física de um objeto poderá ser compartilhada por várias versões lógicas. Este relacionamento entre versões físicas e lógicas pode ser explícito ou implícito. No primeiro caso, a DBV é explicitamente associada a uma versão física. No segundo, se não existir uma DBV na *tabela de associação* é porque essa DBV compartilha o mesmo valor físico com sua DBV ancestral, e assim recursivamente.

Na figura 2.7, as versões lógicas 0.1 e 0.1.1.1 do objeto *B* compartilham explicitamente a mesma versão física, neste caso *b1*, enquanto que as versões lógicas do objeto *A* descendentes de 0.1 compartilham implicitamente a mesma versão física *a1*.

O trabalho [PMJV04] apresenta uma implementação parcial de uma aplicação SIG de evolução urbana baseada no modelo multiversão. A ênfase, no entanto, não é nesta evolução, e sim em sua documentação.

[BEKMW04] propõe um modelo de versão para *data warehouse* similar ao modelo DBV, onde a unidade de versionamento também é o banco de dados como um todo. Cada versão do banco de dados é caracterizada por um intervalo de tempo. Ou seja, cada versão é válida em um determinado período de tempo. Fazendo uma relação com o modelo DBV, é como se cada versão do banco de dados tivesse apenas o atributo tempo em seu vetor de dimensões. Outra diferença está no fato de que o modelo de [BEKMW04] permite versionamento do esquema do banco de dados.



Versões de A	identificadores (stamps)
a0	0
a1	0.1

Versões de B	identificadores (stamps)
b0	0
b1	0.1 0.1.1.1
b2	0.1.2
NULL	0.1.1
b3	0.1.1.2

Figura 2.7: Mecanismo e Modelo DBV [delVal97]

Mapeamento do modelo de versões para o mecanismo DBV

O mecanismo de versão de [delVal97] é baseado no modelo DBV. A idéia é associar cada CET e CTR a uma DBV. Embora um ET (com seu conjunto de CTR) corresponde a uma unidade de trabalho cooperativo, a hipótese do modelo é que este BD não esteja separado em nenhum momento do BD principal. Para isso, alterações no modelo DBV são propostas para permitir, dentre outros aspectos: representação de objetos não versionáveis, como CET, ET e CTR; dois estados nulos diferentes: *estado nulo permanente* e *estado nulo temporário*, onde o primeiro deles não pode ser modificado por nenhuma das operações sobre o modelo. Estes estados serão utilizados pelas operações do modelo de versões, detalhadas no capítulo 4.

2.4 Conclusão

Este capítulo apresentou os principais conceitos relativos ao controle de acesso e versões em banco de dados. Todo mecanismo de controle de acesso tem como base um modelo que define como deve ser a concessão de autorizações. Este modelo define obrigatoriamente os objetos a serem acessados, os tipos de acesso permitidos a estes objetos e quem/o que pode acessá-los.

Também foram apresentados alguns trabalhos correlatos nas áreas de controle de acesso e versões, com ênfase nos trabalhos de [Sasaoka02] e [delVal97], que servem de base para a dissertação. O primeiro define um modelo de controle de acesso para bancos de dados geográficos. O segundo define um modelo de versões em bancos de dados para SIGs. Esta dissertação propõe uma extensão do modelo de [Sasaoka02] para suportar múltiplas versões dos objetos do banco de dados, e o aplica no modelo de versões de [delVal97]. A escolha por ambos os modelos se deu pela característica de que eles consideram a questão do trabalho cooperativo no ambiente de projeto. Esta característica é extremamente relevante, pois está presente na maioria dos projetos desenvolvidos pelas organizações atuais.

O próximo capítulo caracteriza o mecanismo de controle de acesso proposto nesta dissertação, segundo o modelo < *s*, *o*, *m* >.

Capítulo 3

Modelo de Controle de Acesso Proposto

Este capítulo apresenta o modelo de controle de acesso para um banco de dados geográficos que contém múltiplas alternativas dos objetos modelados, ou seja, multiversão. Ele adapta o modelo de controle de acesso de [Sasaoka02], que considera um banco de dados geográficos não multiversão, para suportar um banco de dados geográficos composto de objetos com múltiplos estados, descritos em diferentes versões [delVal97]. Os principais itens do modelo de autorização aqui proposto, levando-se em consideração as características dos dados espaciais multiversão, são:

- Estruturas para representar a autorização;
- A determinação do sujeito, objeto e modos de acesso;
- Um conjunto de políticas para gerenciar e administrar autorizações;
- Algoritmos para atualização da base de regras e análise de pedidos de acesso baseando-se nas autorizações.

Além disso, havendo problemas, o modelo aqui proposto pressupõe que cabe ao administrador resolver o problema. Assim, conflitos entre objetos serão desambiguados pelo administrador na inclusão ou atualização de uma regra.

O capítulo está organizado da seguinte forma. A seção 3.1 apresenta as especificações das regras de acesso, sujeito, objeto e modo de acesso. A seção 3.2 especifica o conjunto de políticas para gerenciar e administrar autorizações. A seção 3.3 discute o problema de armazenamento das autorizações. A seção 3.4 propõe uma arquitetura para o controle de acesso no que diz respeito a direito de acesso. A seção 3.5 define os algoritmos para atualização da base de regras de autorização e para verificar um pedido de acesso, baseando-se nas regras de autorização existentes no banco de dados. Finalmente, a seção 3.6 apresenta as conclusões do capítulo.

3.1 Especificação das regras de autorização

A solução proposta considera duas representações alternativas de regras de autorização: $< s, o, m >$ e $< s, C, m >$ [Sasaoka02]. A primeira representação referencia explicitamente o identificador do objeto, enquanto a segunda contém uma especificação da consulta SQL espacial que define os objetos a serem controlados pela regra. Em um

sistema onde os objetos são multiversão e cada versão de um objeto pertence a um contexto (DBV) diferente do banco de dados, as regras de autorização devem ser capazes de referenciar qualquer versão de qualquer objeto em um dado contexto.

Definição 1: (Regra de Autorização) Seja *s* o sujeito, *o* a especificação do conjunto de objetos autorizados e *m* o modo de acesso. Uma regra de autorização é definida pela tupla < *s, o, m* >, onde:

- *s*: sujeito autorizado.
- *o*: conjunto de objetos autorizados, identificados por seus identificadores ou por uma consulta, em um universo versionado, ou seja, < *oid, dbvid* >.
- *m*: modo de acesso.

Uma regra de autorização em bancos de dados multiversão de agora em diante é chamada de "regra de autorização" e é definida pela tupla < *s, o, v, m* >, onde o par {*o, v*} corresponde a uma versão do objeto.

3.1.1 Exemplos

Seguem alguns exemplos de regras de autorização, considerando-se um banco de dados geográficos multiversão, cujo vetor de dimensões é composto apenas por um atributo: escala. Ou seja, cada contexto CET (e portanto DBV) do banco de dados é formado por objetos representados em uma mesma escala.

Exemplo 1 (regra do tipo < *s, o, v, m* >): *R1*: (*s*: Pedro, *o*: cidade de Campinas, *v:* escala 1:1000000, *m*: leitura).

Figura 3.1: Permissão "Pedro tem acesso de leitura à cidade de Campinas - 1:1000000"

Esta regra de autorização define que Pedro tem permissão de leitura à cidade de Campinas na escala 1:1000000. O objeto Campinas é identificado pelo id *oid1* e versão

dbvid1, que corresponde a um ponto (ponto na figura 3.1). Assim sendo, a regra armazenada será: < *Pedro, oid1, dbvid1, leitura* >. Todos os objetos pertencentes ao contexto dbvid1 estão representados na escala 1:1000000.

Exemplo 2 (regra do tipo < *s, o, v, m* >): *R2*: (*s*: Pedro, *o*: cidade de Campinas, *v:* escala 1:50000, *m*: leitura).

Esta regra de autorização define que Pedro tem permissão de leitura à cidade de Campinas na escala 1:50000. O objeto Campinas é identificado pelo id *oid1* e versão *dbvid2*, que corresponde a um polígono (região mais escura na figura 3.2). Assim sendo, a regra armazenada será: < *Pedro, oid1, dbvid2, leitura* >. Todos os objetos pertencentes ao contexto dbvid2 estão representados na escala 1:50000.

Figura 3.2: Permissão "Pedro tem acesso de leitura à cidade de Campinas - 1:50000"

As regras de autorização *R1* e *R2* referenciam versões diferentes de um mesmo objeto (Campinas: *oid1*). Como o identificador do objeto é único no banco de dados, o que varia, neste caso, é o identificador da DBV à qual cada versão está associada. Isto é, cada versão de um objeto representa uma escala do objeto, que é armazenada em uma DBV diferente. Com a mudança de escala, pode-se mudar também a dimensão geométrica do objeto (nos exemplos 1 e 2, mudança de ponto para polígono).

Exemplo 3 (regra do tipo < *s, C, m* >) *R3*: (*s*: Pedro, *C*: bairros da cidade de Campinas na escala 1:50000, *m*: leitura).

Esta regra de autorização especifica que Pedro tem permissão de leitura a todos os bairros da cidade de Campinas na escala 1:50000. Ela pode ser decomposta em:

C: selecione todos os bairros contidos na cidade de Campinas no CET de escala 1:50000.

o: conjunto de polígonos (bairros).

predicado topológico: dentro de (objeto específico).

Suponha agora que Pedro deseje acessar todos os bairros da cidade de Campinas, em todas as escalas. Neste caso, definimos uma permissão *R4*: (*s*: Pedro, *C*: bairros da cidade de Campinas em todas as escalas, *m*: leitura) que referencia todos os contextos (DBV) do banco de dados que contenham uma representação válida em escala do objeto "Campinas". A regra atribui a permissão de leitura para Pedro sobre todas as versões (escalas) de todos os objetos do tipo "Bairro" que estão incluídos dentro da geometria do objeto Campinas.

Com a existência de contextos do banco de dados, as consultas podem envolver um único ou múltiplos contextos. O resultado de uma consulta recupera objetos como um todo e não identificadores de versões de objetos apenas.

3.1.2 Especificação do sujeito

Sujeitos, em um modelo de autorização, representam entidades tentando acessar a base de dados. Pode ser uma pessoa, uma conta de usuário, um grupo de usuários ou mesmo programas em execução sob o comando de um usuário. Sujeitos podem estar associados a papéis ou perfis, tendo associado um conjunto específico de permissões de acordo com cada um deles. Dependendo do tipo de aplicação, informações pessoais do sujeito podem ser usadas para se avaliar o acesso ao banco de dados, através da definição de *credenciais* [BHEA00]. Existem ainda sujeitos com autoridade para gerenciar o banco de dados (administradores). Assim, este modelo considera que sujeitos sejam usuários finais, apenas para facilitar a compreensão. Entretanto, isso pode ser facilmente estendido para incluir papéis ou grupos.

3.1.3 Especificação do objeto

Em um banco de dados não espacial, o controle de acesso é definido a partir de predicados sobre atributos descritivos. Considerando-se bancos de dados geográficos, a definição de controle de acesso pode ser: considerando apenas atributos descritivos; considerando apenas atributos espaciais; ou considerando ambos. Os atributos espaciais considerados são de três tipos: Pontos (poste, lojas), Linhas (ruas, rios) e Polígonos (cidades, bairros). Um objeto espacial sempre possui um atributo que referencia uma representação geométrica.

O modelo proposto considera a existência de um BD geográfico composto por objetos multiversão. O objeto a ser acessado pode ser um componente espacial ou conjunto destes, sendo que cada objeto tem múltiplos estados, descritos em diferentes versões. Além destes objetos, o modelo considera ainda os objetos para representação dos contextos do banco de dados. O modelo de [delVal97] estipula os seguintes objetos para representação de contextos: CET (contexto espaço-temporal), ET (espaço de trabalho) e CTR (contexto de trabalho). Estes objetos não são versionáveis, porém fazem parte do conjunto de objetos do banco de dados e devem estar sujeitos ao controle de acesso.

3.1.4 Especificação do modo de acesso

O *m* da tupla < *s, o, v, m* > corresponde ao modo de acesso, ou seja, o tipo de operação que o sujeito tem permissão de executar sobre o objeto. Este modelo considera os seguintes tipos ou modos de acesso: leitura (*read*), escrita (*write*), remoção (*delete*), e criação (*create*). Escrita subentende-se permissão para alterar os dados. Dependendo do

tipo de aplicação, os tipos de acesso também podem ser facilmente estendidos para incluir novas operações.

3.2 Conjunto de políticas para gerenciar e administrar autorizações

Para simplificar o modelo, pressupõe-se uma administração centralizada de autorizações, onde apenas o administrador pode conceder e revogar permissões. Este modelo não pressupõe a existência de autorizações negativas, já que introduz uma complexidade maior nos algoritmos que avaliam o pedido de acesso.

Seguindo [Sasaoka02], as autorizações espaciais devem ser avaliadas de acordo com a hierarquia decrescente dos objetos multiversão: polígono, linha e ponto. Ou seja, se o usuário tem acesso a um polígono, ele também tem acesso às linhas e pontos contidos nele. Se tiver a uma linha, tem também aos pontos sobre esta linha. Para os objetos representantes de contextos, não existe hierarquia de acesso entre eles.

Uma vez concedido o acesso a um objeto, o tipo de operação concedido se aplica a todas as informações contidas no objeto. Por exemplo, suponha que um usuário tenha acesso de escrita em dado objeto. Uma vez que o pedido de alteração ao objeto foi validado, o usuário poderá alterar qualquer informação contida nele.

A política adotada implica em um alto número de autorizações armazenadas no sistema, uma vez que para acessar uma versão de um objeto qualquer, seja qual for o modo de acesso, é necessário ter uma autorização positiva definindo o acesso. Isto requer armazenamento adicional, uma vez que serão inúmeras as autorizações armazenadas no sistema, e também grande capacidade de processamento, tendo em vista o número de verificações a ser feito.

Uma forma de melhorar estes fatores seria usar autorizações negativas para definir o acesso de leitura dos dados armazenados e autorizações positivas para escrita. Isto é, permitir que tudo possa ser lido a menos que exista autorização negativa e permitir a escrita somente se existir uma autorização positiva possibilitando o acesso. A hipótese é que haverá poucas autorizações negativas.

3.3 Armazenamento de regras de autorização

Dado que o conjunto de usuários e objetos pode ser muito grande, a questão do armazenamento das regras de autorização é um fator que deve ser considerado, tendo-se em vista os seguintes aspectos:

- Capacidade de armazenamento. Um número muito alto de permissões de acesso impacta o armazenamento do sistema.
- Desempenho do ponto de vista de acesso. Um processamento caro para a validação do pedido de acesso pode comprometer o desempenho do sistema.

Levando-se em consideração tais aspectos, [Sasaoka02] estipula que as regras sejam armazenadas em separado dos objetos, ao invés de associar a cada objeto do banco de dados os pares $< s, m >$ correspondentes. Ou seja, propõe armazenar $< s, o, m >$ e $< s, C, m >$ e calcular todas as consultas dinamicamente a cada pedido de acesso.

Partindo-se desta opção, no caso de objetos multiversão há ainda as seguintes opções:

1. armazenar < *s, o, v, m* > e < *s, C, m* > e calcular todas as consultas dinamicamente a cada pedido de acesso.
2. armazenar < *s, o, v1...vn, m* > e < *s, C, m* > e calcular todas as consultas dinamicamente a cada pedido de acesso.

A diferença entre as duas opções está na maneira com que as regras estão associadas às versões de cada objeto. No primeiro caso, podemos ter uma regra para cada versão de cada objeto do banco de dados. Já no segundo, podemos incluir na mesma regra todas as versões de um objeto que estão sujeitas à mesma permissão de acesso.

Por exemplo, suponha que Pedro tenha acesso de leitura às versões dbvid*1* e dbvid*2* do objeto "Campinas" (figuras 3.1 e 3.2). No primeiro caso, teremos duas regras de autorização, uma para cada versão do objeto Campinas (< *Pedro, Campinas, dbvid1, leitura* > e < *Pedro, Campinas, dbvid2, leitura* >). Já no segundo, teremos apenas uma regra para as versões dbvid*1* e dbvid*2* do objeto "Campinas" (< *Pedro, Campinas, dbvid1-dbvid2, leitura* >).

A opção (1) requer maior capacidade de armazenamento, uma vez que para cada versão de um objeto, deve existir uma autorização que possibilite ao usuário o acesso a ela. Suponha que existam no banco de dados 100 versões diferentes do objeto "Campinas". Caso um usuário queira ter acesso a todas as versões deste objeto, devem existir 100 regras de autorização para possibilitar o acesso a todas as versões. Esse número pode crescer ainda mais, caso tenhamos mais de um modo de acesso para as mesmas versões e para um mesmo sujeito. Quando houver uma grande quantidade de objetos e versões de objetos no banco de dados, o número de autorizações será muito grande.

Já a opção (2) requer menos capacidade de armazenamento, por incluir na mesma regra todas as versões de um objeto, para um mesmo usuário e modo de acesso. Porém, esta opção requer mais capacidade de processamento, uma vez que o critério de busca se torna mais complexo por ter que percorrer uma lista de versões ao invés de uma simples comparação. Esta opção torna os algoritmos de Inserção / Eliminação / Modificação de regras e Validação de um pedido de acesso mais complexos. Por exemplo, dado um pedido de acesso *P* = < *usr1, oid1, dbvid1, m1* >, deve-se primeiro localizar uma regra que seja composta por *s:usr1*, *o:oid1, m:m1*. Caso exista uma regra composta por esses valores, deve-se fazer uma busca pela versão *dbvid1* na lista de versões presente na regra. Caso a versão *dbvid1* faça parte desta lista, o pedido de acesso é autorizado.

Para simplificar os algoritmos de gerenciamento de regras de autorização e validação de pedido de acesso, e uma vez que a tecnologia atual nos permite trabalhar com grandes capacidades de armazenamento, optamos por utilizar a opção (1).

3.4 Arquitetura do sistema

Esta seção apresenta uma proposta básica de arquitetura para o controle de acesso, visando o gerenciamento das regras de autorização e verificação do direito de acesso. A figura 3.3 representa a arquitetura do sistema, que tem como base a arquitetura proposta por [Sasaoka02], composta pelos seguintes componentes:

- Gerenciador de Autorizações
- Gerenciador de Conflito entre Objetos
- Gerenciador de Conflito entre Contextos
- Gerenciador de Controle de Acesso
- Gerenciador de Consultas
- Gerenciador de Dados Geográficos

O Gerenciador de Autorizações é responsável por todo o gerenciamento da base de regras de autorização e possui interface para o administrador de segurança. Através deste, o administrador pode inserir, alterar e remover regras de autorização. O Gerenciador de Conflito entre Objetos verifica se há conflito entre os objetos referenciados por uma regra de autorização e outros objetos do banco de dados, no momento em que existe um pedido de atualização da base de regras. O Gerenciador de Conflito entre Contextos verifica se regras de autorização e pedidos de acesso não entram em conflito com as dependências entre contextos definidas no banco de dados. O Gerenciador de Controle de Acesso implementa o algoritmo 3.4 de controle de acesso de acordo com as regras definidas na seção 3.5. O Gerenciador de Consultas processa tanto as consultas dos pedidos de acesso quanto as consultas das regras de autorização, retornando os objetos solicitados na consulta. O Gerenciador de Dados Geográficos é o responsável por executar as consultas geográficas e manipular os objetos espaciais.

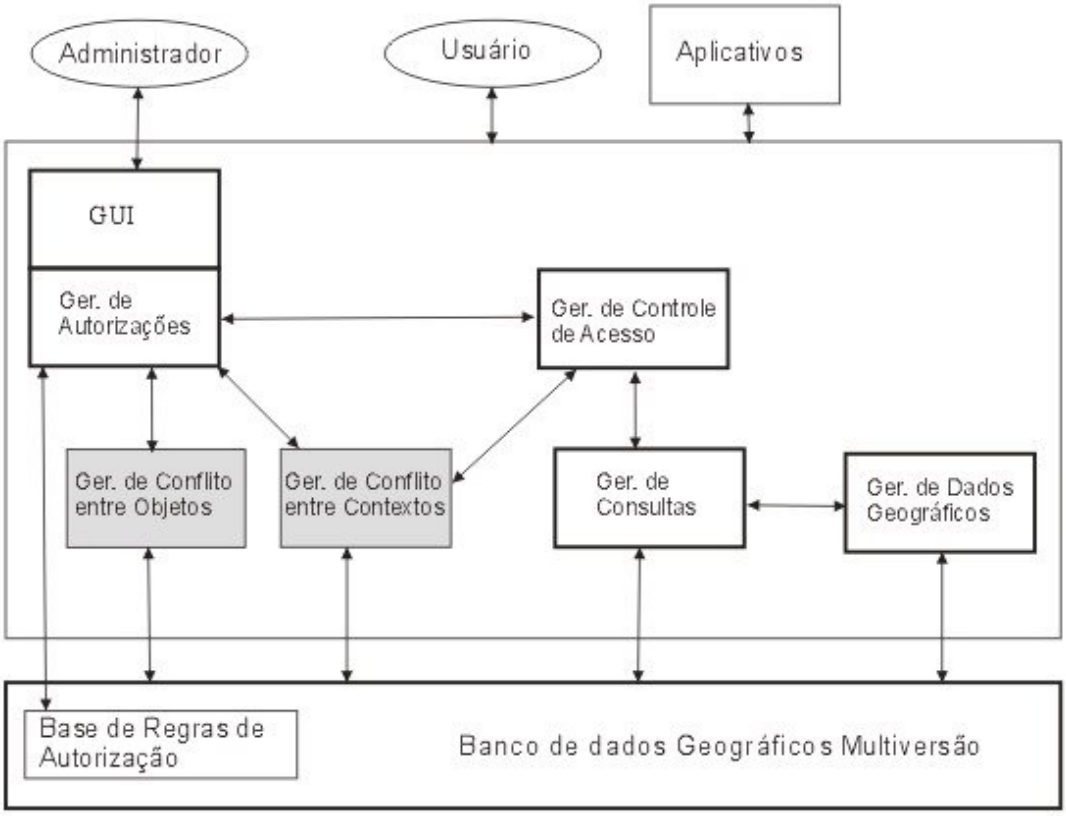


Figura 3.3: Arquitetura do modelo. Os gerenciadores contornados em negrito foram alterados para suportar a adoção de versões pelo modelo. Os gerenciadores em fundo cinza foram criados para verificação de conflitos entre os objetos referenciados por uma regra de autorização e outros objetos do banco, além de dependências entre contextos.

O administrador de segurança é o único usuário que tem permissão para atualizar a base de autorizações. Para isto, ele deve especificar o sujeito, o modo de acesso e o objeto/versão, através de seu identificador ou consultas.

O Gerenciador de Autorizações deve prover uma interface que facilite a identificação de objetos espaciais e suas versões, bem como a identificação dos contextos (CET, ET e CTR) existentes, caso o usuário queria definir alguma regra de autorização usando identificador/versão do objeto. Deve, também, prover uma interface para que se possa especificar as consultas aos objetos sem grandes complicações, incluindo consultas entre contextos.

Suponha o cenário em que o usuário solicita uma inserção de regra do tipo $R = < s, C, m >$ na Base de Regras de Autorização. Os seguintes passos são executados:

1. Uma vez que o Gerenciador de Autorizações recebe o pedido de inserção, ele se comunica com o Gerenciador de Controle de acesso para processar a consulta definida na regra de autorização, recuperando os objetos por ela retornados.
2. Tendo os objetos aos quais a regra de autorização se aplica, o Gerenciador de Autorizações se comunica com o Gerenciador de Conflito entre Objetos para verificar se existe conflito entre os objetos referenciados pela regra a ser inserida e outros objetos existentes no banco. Além disso, ele também se comunica com o Gerenciador de Conflito entre Contextos para garantir que nenhuma dependência entre contextos seja violada devido à nova regra. Ambos os conflitos serão detalhados na seção 3.5.
3. Por fim, tendo todos os conflitos existentes resolvidos, o Gerenciador de Autorizações efetua o armazenamento da nova regra na Base de Regras de Autorização.

Outro cenário é consulta. Suponha que um usuário ou aplicativo faça um pedido de acesso $PA = < s, o, v, m >$. Neste caso, os seguintes passos são executados:

1. O Gerenciador de Controle de Acesso valida ou não a autorização face ao pedido *PA*, executando o algoritmo 3.4 de validação de um pedido de acesso. Ele se comunica com o Gerenciador de Consultas a fim de processar a consulta do pedido de acesso *PA* e a consulta armazenada na regra de autorização.
2. O Gerenciador de Consultas calcula a consulta e recupera os elementos correspondentes. Como a consulta pode envolver um ou mais contextos, a identificação da versão de cada objeto também é recuperada. Caso a consulta envolva predicados geográficos, é enviada ao Gerenciador de Dados Geográficos, que retorna os elementos espaciais resultantes para o Gerenciador de Consultas, bem como os identificadores das versões dos elementos.
3. O Gerenciador de Consultas recebe os elementos já com seus identificadores e suas versões e os devolve para o Gerenciador de Controle de Acesso. Este, então, se comunica com o Gerenciador de Conflito entre Contextos para garantir que a concessão do pedido de acesso não quebre alguma dependência entre contextos existente.

4. Por fim, face ao resultado da verificação de conflitos e regras de autorização, o pedido de acesso é autorizado ou negado.

3.5 Algoritmos para gerenciar regras e analisar pedidos de acesso

Esta seção apresenta os algoritmos para gerenciar regras de autorização e para analisar pedidos de acesso. Além disso, esta seção discute os problemas relacionados a conflitos entre componentes espaciais no momento da atualização da base de regras, bem como os conflitos entre um pedido de acesso e a base de regras na hora da validação do pedido. O controle de acesso exige dois tipos de algoritmos: atualização da base de regras (figura 3.3) e leitura desta base quando há um pedido de acesso. Os algoritmos 3.1, 3.2 e 3.3 permitem o gerenciamento da base de regras, enquanto que o algoritmo 3.4 permite a validação de um pedido de acesso.

3.5.1 Problema geral do controle de acesso

A dissertação propõe que o controle de acesso em bancos de dados geográficos multiversão seja entendido como um processo de três etapas principais:

1 Atualização da base de regras: inserir, remover e atualizar regras do banco de dados.
2 Direito a acesso: verificar se o usuário tem direito a acesso, a partir das regras de autorização armazenadas no banco de dados.
3 Permissão de acesso: caso positivo, verificar se o acesso será concedido, em função de eventuais problemas de concorrência, naquele momento.

A dissertação se atém somente às duas primeiras etapas, supondo que os problemas de concorrência são tratados pelo SGBD. Os problemas relacionados à primeira etapa são esboçados na seção 3.5.1.1, enquanto que os da segunda são esboçados na seção 3.5.1.2.

3.5.1.1 Conflitos entre componentes espaciais na atualização da base de regras de autorizações

No momento da inserção ou atualização de regras de autorização no banco de dados, os objetos referenciados pela regra podem ter interseção com outros objetos do banco de dados. Este problema está resumido graficamente pela figura 3.4. Nela, o objeto *oid1* do contexto *dbvid1* referenciado pela regra de autorização *R1* tem interseção com outros objetos (*oid2, oid3* e *oid4*) do mesmo contexto do banco de dados. O problema, neste caso, está em definir se o usuário definido pela regra *R1* pode ter acesso às áreas *oid1* ∩ *oid2*, *oid1* ∩ *oid3* e *oid1* ∩ *oid4*, no contexto *dbvid1*, com o mesmo modo de acesso definido pela regra.

Definição 2: Seja uma regra representada pela tupla $R1 = < s1, oid1, dbvid1, m1 >$. Dizemos que esta regra pode entrar em conflito de direito de acesso se existe um objeto

oid2 no mesmo contexto tal que *oid1* ∩ *oid2* ≠ 0 (ou seja, o objeto referenciado pela regra de autorização *R1* tem interseção com um outro objeto pertencente ao mesmo contexto do banco de dados).

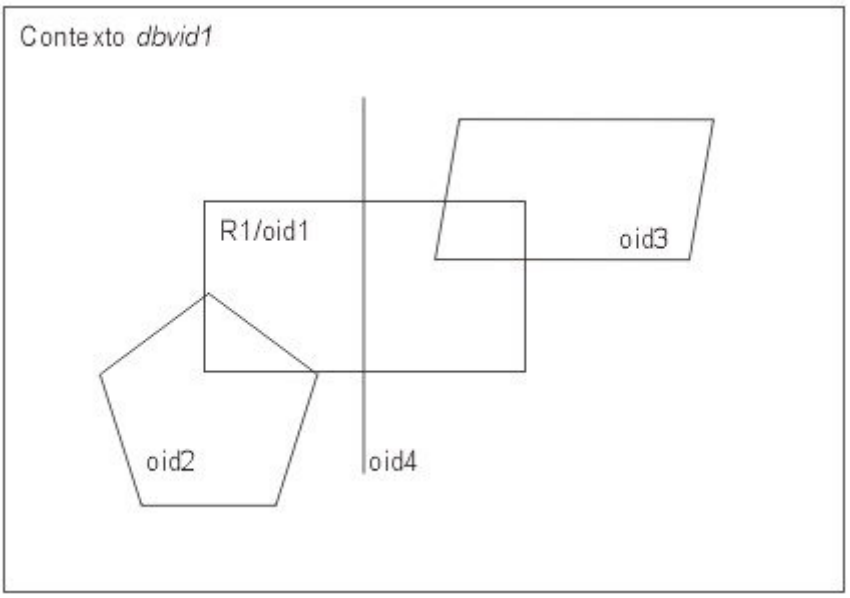


Figura 3.4: Interseção entre uma regra e objetos espaciais em um contexto

Este tipo de conflito pode ser mais complicado caso a regra de autorização envolva objetos em mais de um contexto. Por exemplo, suponha que exista uma regra de autorização *R2* que referencie duas versões do objeto "Campinas" (*dbvid1* e *dbvid2*). Suponha agora que o objeto no contexto *dbvid1* seja representado por um ponto, enquanto que no contexto *dbvid2* seja representado por um polígono. No momento da inserção de *R1* na base de regras, deve-se verificar conflitos em todos os contextos referenciados por *R1*. Neste exemplo, no contexto *dbvid1* não haveria conflito com objeto algum, uma vez que o objeto "Campinas" é representado por um ponto, que não está contido em nenhum outro objeto do banco de dados. Já no contexto *dbvid2*, como o objeto "Campinas" é representado por um polígono, existe conflitos com outros objetos, como representado pela figura 3.4.

Ou seja, em um contexto multiversão, os conflitos entre uma regra de autorização e os objetos do banco de dados podem ser diferentes de acordo com o contexto considerado, sendo necessário tratamentos diferentes para cada contexto.

3.5.1.2 Conflitos de direito de acesso

O problema de direito de acesso no caso de dados geográficos está resumido graficamente pela figura 3.5. A figura mostra que um sujeito *s* pediu acesso a um conjunto de objetos delimitados pela área *PA* (pedido de acesso). Este conjunto pode conter, inclusive, um único objeto pontual. O banco de dados, por sua vez, já tem armazenado diferentes regras de autorização < *s, o, v, m* > (*R1, ..., R4*). O problema de direito de acesso consiste em determinar se o pedido de acesso *PA* é aceitável ou se é conflitante com as regras existentes.

Definição 3: Seja um pedido de acesso representado pela tupla *PA = < s1, oid1, dbvid1, m1 >* e uma regra de autorização existente no banco de dados *R2 = < s2, oid2, dbvid1, m2 >*. *PA* pode estar em conflito de acesso com a regra de autorização *R2* se *s1 = s2*, *m1 = m2* e *oid1 ≠ oid2*, mas *oid1 ∩ oid2 ≠ 0* para o mesmo contexto *dbvid1* do banco de dados (ou seja, existe uma autorização para *s1* com o mesmo tipo de operação *m1*, mas objetos distintos que fazem intersecção).

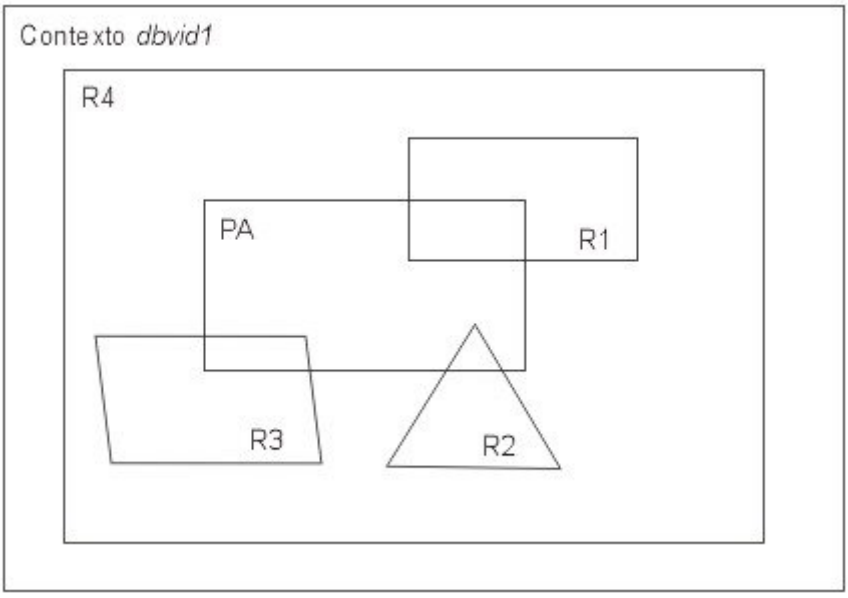


Figura 3.5: Pedido de acesso

Suponha um exemplo onde *s1* deseja acessar *oid1* no contexto *dbvid1*, que é uma linha totalmente contida no polígono *oid2* ao qual *s1* tem acesso. Suponha também que existe uma regra de autorização que define que *s1* tem acesso ao objeto *oid2* no contexto *dbvid1*, mas não existe uma regra que define o acesso ao objeto *oid1* explicitamente. Neste caso, a política mais factível é conceder o direito de acesso ao objeto *oid1*, já que a linha está totalmente contida no polígono. Entretanto, há outros tipos de conflitos especialmente quando *oid1* estiver parcialmente contido em *oid2*.

Da mesma maneira que acontece com regras de autorização, um pedido de acesso *PA* pode envolver mais de um contexto do banco de dados. Neste caso, a verificação do conflito de direito de acesso deve ser executada em cada contexto referenciado pelo pedido de acesso.

A seção 3.5.2 tratará destes problemas específicos e da solução proposta para cada caso de interseção entre componentes espaciais.

3.5.1.3 O problema do relacionamento de dependência entre contextos

Para o trabalho cooperativo de diferentes usuários, podem ser estabelecidos relacionamentos de dependência entre versões de um mesmo objeto em diferentes contextos. Estes relacionamentos indicam o compartilhamento dessas versões nos diferentes contextos, isto é, quando em um contexto o estado da versão é modificado, esta mudança deve ser refletida no estado da versão relacionada no outro contexto.

Uma vez que regras de autorização e pedidos de acesso referenciam objetos que podem definir um relacionamento de dependência entre contextos, o usuário definido pela regra, ou pelo pedido de acesso, deve ter acesso a todos os contextos pertencentes ao relacionamento de dependência.

Por exemplo, suponha um banco de dados multiversão com três contextos (*dbvid1, dbvid2* e *dbvid3*). Suponha agora a existência de um objeto *oid1* no banco de dados que define um relacionamento de dependência entre os três contextos. Caso o estado da versão de *oid1* no contexto *dbvid1* é alterado, essa alteração deve ser propagada para os outros dois contextos *dbvid2* e *dbvid3*. Ou seja, sempre que o estado da versão do objeto muda em um contexto, o estado da versão do mesmo objeto nos outros dois contextos deve ser alterado também.

Suponha agora que exista um pedido do administrador para inclusão da regra de autorização *R1 = < s1, oid1, dbvid1, m1 >*, onde *m1* seja modo de escrita, e que existe um relacionamento de dependência entre os três contextos, já que todos contêm *oid1* e que é compartilhado. Isto implica que a regra *R1* só poderá ser incluída na base de regras caso o usuário *s1* tenha acesso de escrita sobre os outros dois contextos que fazem parte do relacionamento de dependência, uma vez que a alteração em um contexto implica alteração em todos os outros.

O mesmo acontece com o pedido de acesso. Suponha que exista um pedido de acesso *PA = < s1, oid1, dbvid1, m1 >*, onde *m1* seja modo de escrita, e que existe um relacionamento de dependência entre os três contextos, devido ao objeto *oid1*. O pedido só será aceito se o usuário *s1* tiver permissão de escrita nos três contextos.

Em ambos os casos, caso o usuário não tenha permissão *m1* sobre algum contexto pertencente ao relacionamento de dependência, a regra de autorização não é incluída na base de regras ou o pedido de acesso é negado.

Alternativamente, dado o mesmo pedido de acesso *PA = < s1, oid1, dbvid1, m1 >* citado anteriormente, caso o usuário não tenha permissão *m1* sobre algum contexto pertencente ao relacionamento de dependência, este pode querer criar uma nova versão (*dbvid4*) derivada de *dbvid1*. Neste caso, o contexto *dbvid4* não possui relacionamento de dependência com os contextos *dbvid1*, *dbvid2* e *dbvid3*, podendo o usuário acessar o objeto *oid1* do contexto *dbvid4* no modo *m1*.

3.5.2 Resolução de conflitos entre componentes espaciais

Esta seção discute problemas de direito de acesso quando ocorrem os dois tipos de conflitos definidos na seção 3.5.1 (Definição 2 e Definição 3).

A seção 3.5.2.1 apresenta as soluções possíveis para os conflitos segundo a Definição 2, enquanto que a seção 3.5.2.2 apresenta as soluções para os conflitos segundo a Definição 3. As soluções para ambos os conflitos consideram que os objetos em questão não definem relacionamento de dependência entre contextos; se definem, o usuário tem permissão de acesso a todos os contextos dependentes entre si.

A seção 3.5.2.3 discute o problema de regras de autorização e pedidos de acesso que referenciem objetos que definam dependências entre contextos.

3.5.2.1 Conflitos do tipo 1

Aqui serão apresentadas soluções para os conflitos segundo a Definição 2, considerando-se regras de autorização que referenciem apenas um contexto do banco de

dados (caso (a)). Ou seja, como um contexto isolado representa uma instância do banco de dados, este pode ser visto como um banco de dados monoversão. Neste caso, serão adotadas as soluções propostas por [Sasaoka02] que considera um banco de dados sem versões.

O caso (b) apresenta a solução para o caso de uma regra de autorização referenciar objetos em mais de um contexto do banco. Neste caso, a decisão deve levar em conta os conflitos ocorridos em cada contexto referenciado pela regra.

a) Interseção de objetos

Este tipo de interseção espacial envolve dois ou mais objetos, sendo ilustrado na figura 3.6. Por exemplo, seja uma regra de autorização $R1 = < s1, oid1, dbvid1, m1 >$, onde o objeto *oid1* no contexto *dbvid1* é representado por um polígono. Suponha que existem dois objetos *oid2* e *oid3* no mesmo contexto, representados por um polígono e uma linha respectivamente, porém $oid1 \cap oid2 \neq 0$ e $oid1 \cap oid3 \neq 0$ (área mais escura da figura 3.6) e não existe uma regra de autorização que permita o acesso aos objetos *oid2* e *oid3* pelo sujeito *s1*. Deseja-se saber se o usuário *s1* terá acesso aos objetos contidos na área (*oid1* ∩ *oid2* e *oid1* ∩ *oid3*).

A solução adotada estipula que o usuário *s1* pode acessar qualquer objeto de *oid1*, mesmo que este também pertença a algum outro objeto. Esta solução leva em consideração facilidades de implementação, mesmo que possa causar falhas e inconsistências, já que permite acessar dados que também pertencem às áreas de *oid2* e *oid3*.

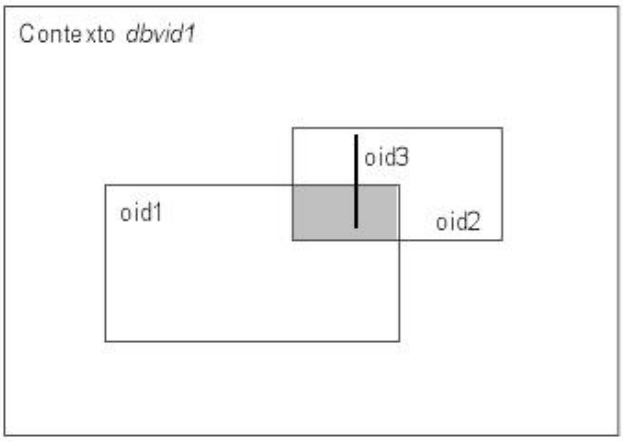


Figura 3.6: Interseção de objetos

b) Regras que referenciem objetos em mais de um contexto

As duas seções anteriores trataram do problema de conflitos entre objetos espaciais, em um mesmo contexto do banco de dados. Porém, dada uma regra de autorização *R* que referencie objetos em mais de contexto, os conflitos podem acontecer em um ou mais contextos simultaneamente. Neste caso, para cada contexto referenciado pela regra *R*, deve-se verificar se não há interseções entre polígonos e interseções entre linhas e polígonos. Por exemplo, dado as figuras 3.1 e 3.2, suponha que exista uma regra de autorização que estipule que um usuário tenha permissão de escrita sobre o objeto "Campinas" em ambas as escalas (*dbvid1* e *dbvid2*). Para cada contexto referenciado pela regra, deve-se verificar se o objeto "Campinas" possui interseção com outros objetos no mesmo contexto. Neste exemplo, não haverá conflito no contexto *dbvid1*, uma vez que o objeto "Campinas" é representado por um ponto. Já no contexto *dbvid2*,

haverá conflito entre o objeto "Campinas" e outros objetos, uma vez que ele é representado por um polígono e existem outros objetos que fazem interseção com ele (por exemplo, rodovia SP330, representada por um objeto do tipo linha).

3.5.2.2 Conflitos do tipo 2

Aqui serão apresentadas soluções para os conflitos segundo a Definição 3 (interseção entre objetos). Este tipo de conflito ocorre quando um sujeito *s1* pede acesso *PA = < s1, oid2, dbvid1, m1 >* a um objeto *oid2* e na base de regras existe apenas uma regra *R2 = < s1, oid1, dbvid1, m1 >* que autoriza *s1* a acessar o objeto *oid1*. Para validar o pedido de acesso *PA*, é preciso verificar o relacionamento espacial existente entre os objetos *oid1* e *oid2*.

Generalizando, este tipo de conflito pode ser entendido como tendo apenas dois casos, dado um mesmo contexto: objetos totalmente contidos em outro ou objetos parcialmente contidos, ou seja, *oid2* total ou parcialmente contido em *oid1*. Os casos (a) e (b) apresentam as soluções adotadas para ambos os conflitos, enquanto que o caso (c) apresenta a solução para o caso de um pedido de acesso *PA* referenciar objetos em mais de um contexto do banco de dados. Da mesma forma que ocorre com os conflitos do tipo 1, as soluções adotadas por este modelo para conflitos do tipo 2, cujo pedido de acesso referencie apenas um contexto do banco de dados, são as mesmas adotadas pelo modelo de [Sasaoka02], já que um contexto isolado pode ser visto como um banco de dados sem versões.

a) Objetos totalmente contidos

Assim como no modelo de [Sasaoka02], mencionado no capítulo 2, existe uma hierarquia para objetos espaciais com relação à permissão de acesso, onde o nível básico é o ponto e o nível superior é o polígono. A hierarquia a partir do seu nível superior para o inferior corresponde a: polígono → linha → ponto. Isto significa que ao se obter acesso a um polígono, obtém-se também acesso a todos os objetos que estiverem contidos neste polígono. Quando se tem acesso a uma linha, tem-se acesso a todos os seus pontos.

Portanto, pode-se definir regras de inferência para hierarquias de objetos totalmente contidos um no outro, em um mesmo contexto do banco de dados. A existência da autorização *< s1, oid1. dbvid1, m1 >* permite inferir a autorização *< s1, oid2, dbvid1, m1 >*, não sendo necessário defini-la explicitamente se *oid2* estiver totalmente contido em *oid1*. As figuras 3.7, 3.8 e 3.9, adaptadas de [Sasaoka02], mostram alguns tipos de continência de objetos.

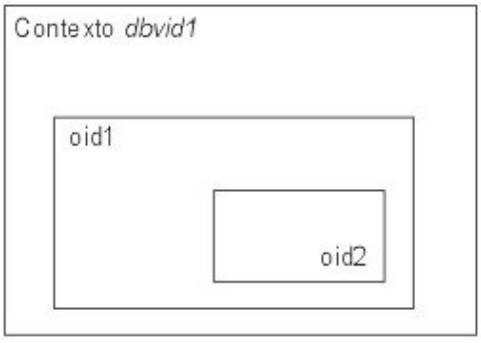


Figura 3.7: Continência de polígono em polígono

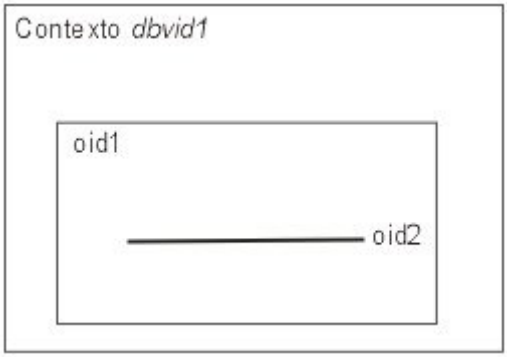


Figura 3.8: Continência de linha em polígono

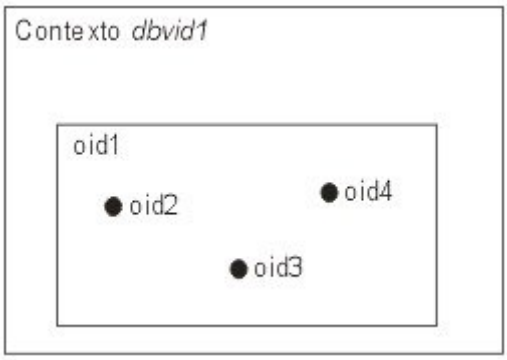


Figura 3.9: Continência de pontos em polígono

b) Objetos parcialmente contidos

A questão é o que fazer quando o objeto *oid2* estiver parcialmente contido em *oid1*. Deve-se ou não permitir o acesso de *s1* a *oid2*? Assim como estipulado no conflito do tipo 1, a solução adotada estipula que o usuário *s1* pode acessar o objeto *oid2* que está parcialmente contido em *oid1*. Isto se deve ao fato de que para analisar o objeto *oid1,* é importante que se conheça e analise todos os objetos que interfiram em *oid1*.

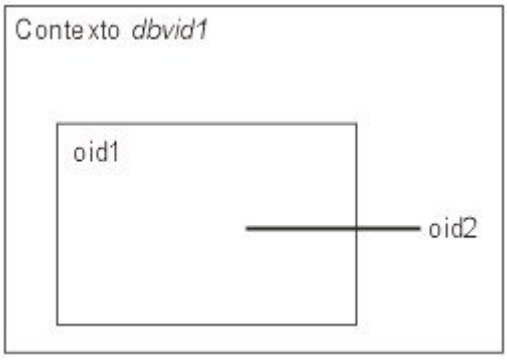


Figura 3.10: Interseção de linha e polígono

c) Pedidos de acesso que referenciem objeto em mais de um contexto

Quando o pedido de acesso envolve objetos em mais de um contexto, a ocorrência de conflitos do tipo 2 deve ser verificada em cada contexto referenciado pelo pedido de acesso. Por exemplo, dado as figuras 3.1 e 3.2, suponha que exista um pedido de acesso

PA do usuário *urs1* para todas as rodovias que cruzem com a cidade de Campinas, em todas as escalas. Porém, na base de regras, suponha que exista apenas uma autorização que garanta ao usuário *usr1* o acesso ao objeto "Campinas", também em todas as escalas. Neste caso, como no contexto *dbvid1* o objeto "Campinas" é representado por um ponto, o usuário não terá acesso às rodovias. Já no contexto *dbvid2*, o usuário terá acesso apenas às partes das rodovias que estejam contidas no polígono que representa o objeto "Campinas". Ou seja, o pedido de acesso deve ser analisado separadamente para cada contexto referenciado por ele, podendo o usuário ter o acesso concedido para alguns contextos específicos, e não para outros.

3.5.3 Inserção de regras

No modelo proposto, considera-se que os conflitos segundo a Definição 2 da seção 3.5.1.1 são resolvidos no momento da inserção de regras e não no momento da validação de pedidos de acesso. Cabe ao administrador decidir se irá permitir regras, mesmo que estas tenham conflitos segundo a Definição 2.

Algoritmo 3.1 Inserção de regras

Hipótese: Pressupõe-se que a inserção não tornará a base inconsistente.

Input: Pedido de inserção da regra $R = < s, o, v, m >$ ou $R = < s, Q, m >$, onde Q é uma consulta que define o conjunto de objetos da regra.

Output: base de regras atualizada ou inserção cancelada.

1. Se a regra referenciar somente um objeto *o*.
   1.1. Se o modo de acesso *m* for escrita, verificar se o objeto *o* define relacionamento de dependência entre contextos.
      1.1.1. Caso defina, verificar se o usuário *s* possui acesso de escrita a todos os contextos pertencentes ao relacionamento de dependência.
      1.1.2. Caso não possua, o pedido de inserção da regra é negado.
   1.2. Verificar se há conflito de objetos no contexto do objeto *o* de acordo com a Definição 2.
   1.3. Se houver conflito, solicitar decisão do administrador sobre a inclusão da regra.
   1.4. Se não houver conflito, insere a regra *R*.
2. Se *R* referenciar uma consulta *Q* em um ou mais contextos
   2.1. Transformar *Q* em consulta *C*, em *SQL* ou *SQL* espacial e executá-la.
   2.2. Seja $O = \{oi, vi\}$, o conjunto de objetos retornados pela consulta *C*. Para cada objeto $\{oi, vi\}$ pertencente a *O*, verificar se há conflito de objetos no contexto de *oi*, de acordo com a Definição 2.
   2.3. Se não houver conflito de nenhum objeto de *O*, insere a regra *R*.

Se houver conflito com algum *oi*, solicitar decisão do administrador sobre inserção ou não de *R*.

3.5.4 Eliminação de regras

A eliminação de regras pode ser efetuada para um objeto apenas ou para um conjunto de objetos, com uma consulta. No caso de uma consulta, são recuperadas todas as regras da base cujo usuário e modo de acesso são os mesmos da especificação da regra. De posse das regras, são verificados os objetos que elas referenciam. Caso haja interseção com os objetos definidos pela consulta da regra a remover, apresentar ao administrador que decidirá quais regras serão eliminadas.

Algoritmo 3.2 Eliminação de regras

Input: Pedido de eliminação da regra $R = < s, o, v, m >$ ou $R = < s, Q, m >$.

Output: base de regras atualizada.

1. Se *R* referenciar somente um objeto, remover a regra da base de dados.
2. Se *R* referenciar uma consulta
   2.1. Transformar *Q* em consulta *C*, em *SQL* ou *SQL* espacial e executá-la.
   2.2. Seja $O = oi$, o conjunto de objetos retornados pela consulta *C*.
   2.3. Recuperar todas as regras $< s, o, m >$ e $< s, C, m >$, tais que $o \cap oi \neq 0$ e $C \cap oi \neq 0$, no contexto de *oi*.

Apresentar estas regras ao administrador, que decidirá quais efetivamente eliminar.

3.5.5 Modificação de regras

Pressupõe-se que toda modificação será realizada através de remoção (algoritmo 3.2) e inserção (algoritmo 3.1) de regras.

Algoritmo 3.3 Eliminação de regras

1. Eliminar a regra atual executando o algoritmo 3.2.
2. Inserir nova regra executando o algoritmo 3.1.

3.5.6 Validação de um pedido de acesso

Dado um pedido de acesso *PA*, verificar se ele será aceito ou não com base nas regras de autorizações presentes na base de regras.

Algoritmo 3.4 Validação de um pedido de acesso

Input:

[1] pedido de acesso $PA = (S, O, V, M)$ ou $PA = (S, Ca, M)$, onde *Ca* é uma consulta que define os objetos relativos ao pedido de acesso *PA*.

[2] conjunto de regras de autorização (s, o, v, m) e (s, C, m) existentes na base de regras

Output:

[1] AUTORIZADO ou [2] NEGADO

1. Se *PA* não envolver uma consulta, mas somente um objeto *O*.
   1.1. Se existir uma regra *R1* = < *S, O, V, M* > na base de regras, autorizar o pedido de acesso e terminar.
   1.2. Selecionar regras de autorização *ri* = {< *s, o, v, m* >} e *rj* = {< *s, C, m* >} da base de regras, tais que *s* = *S* e *m* = *M*. O resultado desta etapa é um conjunto de regras *RA* = < *S, oi, vi, M* > ∪ < *S, Ci, M* >.
   1.3. Calcule as consultas das regras < *S, Ci, M* > para determinar os objetos referenciados, obtendo o conjunto final de regras *RF* = < *S, ok, vk, M* >.
   1.4. Detectar problemas relativos aos conflitos segundo Definição 3 entre *PA* e os objetos de *RF*.
   1.5. Resolver os conflitos segundo política definida em 3.5.2.2 garantindo ou não o direito de acesso.
2. Se *PA* envolver uma consulta.
   2.1. Selecionar regras de autorização *ri* = {< *s, o, v, m* >} e *rj* = {< *s, C, m* >} da base de regras, tais que *s* = *S* e *m* = *M*. O resultado desta etapa é um conjunto de regras *RA* = < *S, oi, vi, M* > ∪ < *S, Ci, M* >.
   2.2. Calcular as consultas das regras < *S, Ci, M* > para determinar os objetos referenciados, obtendo o conjunto final de regras *RF* = < *S, ok, vk, M* >.
   2.3. Calcular a consulta *Ca*, obtendo *PA* = {< *S, oa, va, M* >}, que determina os objetos atingidos pelo pedido de acesso.
   2.4. Detectar problemas relativos aos conflitos segundo Definição 3 entre *PA* e os objetos de *RF*.
   2.5. Resolver os conflitos segundo política definida em 3.5.2.2 garantindo ou não o direito de acesso.

Detalhando melhor os passos Detectar conflitos (1.4 e 2.4) e Resolver conflitos (1.5 e 2.5), temos o seguinte algoritmo:

3. Para cada contexto de *PA*, se *PA* está contido em *RF* então acesso CONCEDIDO.
4. Senão, para cada *ra* = < *S, oa, va, M* > pertencente a *PA – RF,* verificar se *oa* está contido em algum *ok*. Se estiver, o acesso a *oa* é concedido.
5. Se houver pedidos ainda não resolvidos para objetos *oa*, existem conflitos de interseção parcial que devem ser resolvidos segundo políticas da seção 3.5.2.2 (b).

O algoritmo 3.4 funciona da seguinte maneira. Ao receber um pedido de acesso, seleciona todas as regras de autorização contidas na base de regras com o mesmo sujeito e modo de acesso (ou modo de acesso superior). Isto é importante no algoritmo, porque várias requisições já podem ser descartadas neste momento, sem perder tempo em calcular as consultas e recuperar objetos.

No modelo proposto, um sujeito pode ter muitas autorizações. Portanto, pode haver muitas regras de autorização < *s, o, v, m* > e < *s, C, m* > resultantes da consulta anterior. Em seguida, as consultas armazenadas na regra < *s, C, m* > e do pedido de acesso (se aplicável) são calculadas. A partir deste instante, basta determinar se existem regras que autorizem os objetos definidos pelo pedido de acesso. Se sim, o acesso é

CONCEDIDO. Caso contrário, verifica-se se cada objeto do pedido de acesso (que não tiver uma regra equivalente) está contido em algum objeto de alguma regra armazenada. Se todos estiverem, o acesso é concedido. Se houver objetos parcialmente contidos, então os conflitos devem ser resolvidos segundo política da seção 3.5.2.2 (b). Estes passos são executados para cada contexto que o pedido de acesso referenciar. Somente após a concessão de acesso a todos os contextos, é que o pedido de acesso do usuário será aceito.

Não há necessidade de se verificar dependência entre contextos quando do pedido de acesso de escrita a um objeto. O algoritmo de inserção de regras já garante que todos os sujeitos que têm acesso de escrita a um objeto que define relacionamento de dependência entre dois ou mais contextos, têm acesso de escrita a todos os contextos dependentes entre si.

Seja o seguinte exemplo para ilustrar o funcionamento do algoritmo 3.4:

Seja um banco de dados com dois contextos *dbvid1* e *dbvid2* (figuras 3.1 e 3.2), cujo vetor de dimensões contém um atributo: escala. O contexto *dbvid1* contém objetos representados na escala 1:1000000, enquanto que o contexto *dbvid2* na escala 1:50000. Suponha que exista um objeto no banco (*oid5*) que represente a quantidade de rodovias que cruzam com a cidade de Campinas e que existe um relacionamento de dependência entre os contextos *dbvid1* e *dbvid2* através deste objeto. Ou seja, sempre que o valor de *oid5* mudar em um contexto, o outro contexto também deverá ser atualizado com o valor novo.

Base de regras:

R1 = < s, Campinas, dbvid1, m >

R2 = < s, Campinas, dbvid2, m >

R3 = < s, "todas as ruas de Valinhos", dbvid2, m >

R4 = < s2, Valinhos, dbvid2, m >

Caso 1: *PA = < s, Campinas, dbvid2, m >*

As regras *R1, R2* e *R3* são recuperadas, a consulta *R3* é executada e obtemos

RF = R1 ∪ *R2* ∪ *< s, rua1, dbvid2, m >, ..., < s, ruaN, dbvid2, m >.*

Como *PA* está contido em *RF*, acesso concedido pelo passo 3.

Caso 2: *PA = < s, "cidade de Campinas em todas as escalas", m >*

As regras *R1, R2* e *R3* são recuperadas, a consulta *R3* é executada e obtemos

RF = R1 ∪ *R2* ∪ *< s, rua1, dbvid2, m >, ..., < s, ruaN, dbvid2, m >.*

Como *PA* está contido em *RF*, acesso concedido pelo passo 3.

Caso 3: *PA = < s, Barão Geraldo, dbvid2, m >*

Neste caso, como Barão Geraldo está contido em Campinas, o acesso é concedido pelo passo 4.

Caso 4: *PA = < s, SP330, dbvid2, m >*

Neste caso, cai-se no passo 5, que corresponde à situação de interseção parcial polígono-linha. O acesso será dado apenas para a parte da rodovia SP330 que está contida no objeto Campinas, sendo necessário dividir o objeto.

Caso 5: *PA = < s2,Itatiba, dbvid2, m >*

Acesso negado pelo passo 1.1.

3.6 Conclusão

Este capítulo apresentou o modelo de controle de acesso para um banco de dados geográficos multiversão. O modelo tem como base o modelo de controle de acesso de [Sasaoka02] e o estende para suportar um mecanismo de versões dos objetos armazenados no banco [delVal97]. O capítulo propõe uma arquitetura para o sistema e define algoritmos para gerenciar regras de acesso e validar um pedido de acesso. O próximo capítulo utiliza o modelo aqui proposto para definir as condições e algoritmos para a execução das operações sobre o modelo multiversão de [delVal97].

Capítulo 4

O controle de acesso no modelo de versões em bancos de dados para SIGs

O modelo de versões em banco de dados para SIGs escolhido como base foi desenvolvido por [delVal97], por considerar, dentre outros aspectos, facilidades para o trabalho de projeto cooperativo. Ele se baseia em três conceitos principais: *contexto espaço-temporal* (CET), *espaço de trabalho* (ET) e *contexto de trabalho* (CTR). O mecanismo de versões utilizado como base é o de Versões em Bancos de Dados (*Database Versions (DBV)* [CJ90, GJ94, Gan94]), estendido para suportar as funcionalidades do modelo.

Este capítulo apresenta as operações do modelo de versões em banco de dados para SIGs, mostrando como a proposta da dissertação pode ser adotada em cada uma das operações. O capítulo está organizado da seguinte forma. A seção 4.1 apresenta as definições gerais do controle de acesso. A seção 4.2 apresenta as operações do modelo de versões sobre Contextos Espaço Temporais. A seção 4.3 apresenta as operações sobre Espaços de Trabalho. A seção 4.4 apresenta as operações sobre Contextos de Trabalho. A seção 4.5 apresenta as operações sobre objetos e versões de objetos. Cada seção está organizada da seguinte forma. Inicialmente, ela descreve o algoritmo correspondente do modelo de [delVal97]. A seguir, mostra como se deve proceder à verificação de autorização para aquele caso. Finalmente, a seção 4.6 apresenta as conclusões do capítulo.

A notação utilizada nos algoritmos será a seguinte:

- Identificadores das classes de objeto CET, ET e CTR: *CET*, *ET* e *CTR* respectivamente.
- Identificador de objeto CET: *Cid* (*C0.1*, *C0.1.1*, *C2*).
- Identificador de objeto ET: *ETid* (*ET1*, *ET2*, *ET3*).
- Identificador de objeto CTR: *CTid* (*CT0.1*, *CT0.1.1*, *CT3*).
- Identificador de objeto multiversão: *OIDid* (*OID1*, *OID2*, *OID3*).
- Identificador de DBV: *Did* (*D0.1.1*, *D0.1*, *D2*). Para facilitar o entendimento, DBV associados a CET serão identificados por *DCid* e DBV associados a CTR por *DCTid*.

4.1 Definições gerais

Todas as operações do modelo de versões que acessam versões lógicas de objetos precisam passar, inicialmente, por uma verificação que detecte se o pedido tem acesso ao contexto ao qual a versão lógica pertence. Somente após esta validação inicial é que se verifica a permissão de acesso à versão lógica em si. O algoritmo ALG 1 valida o acesso a uma versão lógica de objeto pertencente a um contexto do banco de dados.

Algoritmo ALG 1: Validação de um pedido de acesso a uma versão lógica de objeto.

Input:

[1] usuário *S*, objeto destino *OIDx*, contexto fonte *Dx* e modo de acesso *Mx*.

Output:

[1] AUTORIZADO ou [2] NEGADO.

1. Executar algoritmo 3.4, onde *PA* = (*S*, *Dx*, *v*, *m*), *v* = *NULL* e *m* = *leitura ou escrita*.
2. Executar algoritmo 3.4, onde *PA* = (*S*, *OIDx*, *Dx*, *Mx*).
3. Se Output = AUTORIZADO para os passos 1 e 2, autorizar o acesso.
4. Senão, negar o acesso.

O passo 1 do algoritmo ALG 1 verifica se o usuário tem permissão de acesso (*leitura* ou *escrita*) ao contexto *Dx*, que pode estar associado tanto a um CET quanto a um CTR. O passo 2 verificar se o usuário tem permissão de acesso *Mx* à versão lógica do objeto *OIDx* em *Dx*. Se ambas as condições forem satisfeitas, então o pedido de acesso é autorizado.

Note que se o modo de acesso *Mx* for do tipo *remoção*, ou seja, deseja-se remover a versão lógica do objeto *OIDx* do contexto *Dx*, então o usuário deve ter obrigatoriamente permissão de acesso (escrita) sobre o contexto *Dx* (*m* = *escrita* no passo 1). Isto se deve ao fato de que a remoção de uma versão lógica de objeto, assim como a inclusão, implica em alteração do contexto ao qual ela pertence.

Para as operações que envolvem criação de contexto a partir de outros contextos (fontes) existentes no banco de dados, é necessário, ainda, verificar se o pedido tem permissão de acesso, ao menos, a uma versão lógica pertence a cada um dos contextos fontes. O algoritmo ALG 2 valida o pedido de criação de contexto a partir de um ou mais contextos fontes.

Algoritmo ALG 2: Validação de um pedido de criação de contexto a partir de contextos fontes.

Input:

[1] usuário *S* e contextos fontes *D1*, *D2*, ..., *Dn*.

Output:

[1] AUTORIZADO ou [2] NEGADO.

1. Para cada *D1*, *D2*, ..., *Dn*
   1.1. Executar algoritmo 3.4, onde *PA* = (*S*, *Dn*, *v*, *m*), *v* = *NULL* e *m* = *leitura ou escrita*. Se Output = NEGADO, negar o pedido e terminar.
   1.2. Para cada versão lógica dos objetos *OID1*, *OID2*, ..., *OIDy* em *Dn*, executar algoritmo 3.4, onde *PA* = (*S*, *OIDy*, *Dn*, *m*) e *m* = *leitura*. Se Output = NEGADO para todos *OIDy* em *Dn*, negar o pedido e terminar.
2. Autorizar o pedido.

O algoritmo ALG 2 funciona da seguinte maneira. Para cada contexto fonte, verifica se o sujeito *S* tem permissão de acesso ao contexto (passo 1.1). Se afirmativo, verifica se *S* tem permissão de acesso, ao menos, a uma versão lógica de objeto pertencente ao contexto (passo 1.2). Se o usuário não tiver permissão de acesso a algum contexto ou a nenhuma versão lógica de um contexto, o pedido é negado.

Os algoritmos subseqüentes irão freqüentemente invocar ALG 1 e ALG 2. Além do mais, parte-se do princípio de que novas regras de autorização (*s, o, v, m*) e (*s, C, m*) serão definidas para os objetos criados pelas operações.

4.2 Operações sobre Contextos Espaço Temporais (CET)

Um CET é associado a uma DBV estendida [delVal97], ou seja, todas as operações definidas no modelo sobre um objeto CET são mapeadas em operações sobre a DBV associada. Um CET é representado como um objeto de estado único, uma vez que ele não é versionável.

A figura 4.1, retirada de [delVal97], apresenta um banco de dados com dois objetos A e B versionáveis. Em cada um dos sete contextos do banco de dados, existe uma versão lógica dos objetos *A* e *B*, cujo valor é armazenado na tabela de associação. Há três objetos CET com vetores de dimensões *C1*[*t1,e1]*, *C2*[*t2,e2]*, *C3*[*t1,e2]*, nas dimensões tempo e escala, que têm associados as DBV 0.1.1.1, 0.1.1.2 e 0.1.2 respectivamente.

Note que existem dois objetos na figura 4.1 que definem relacionamento. O primeiro define um relacionamento de dependência entre os objetos *A* e *B* nos contextos *C1* e *C2*. Isto significa que caso a versão lógica de um desses dois objetos mude em um contexto, esta mudança deve ser propagada para a versão lógica do mesmo objeto pertencente ao outro contexto. O segundo objeto define um relacionamento entre os contextos *C2* e *C3*. A semântica para este tipo de relacionamento é dada pelo usuário.

4.2.1 Criação de CET por derivação

A operação de criação de um CET por derivação envolve os seguintes passos:

- Criação de uma DBV por derivação, definida pela cópia de todas as versões lógicas da DBV fonte para a nova DBV criada.
- Criação de um objeto CET associado à nova DBV.

Se agora aplicamos controle de acesso, então a criação de um CET por derivação deve ser precedida de verificação de permissão. O algoritmo 4.1 valida a criação de CET por derivação.

Algoritmo 4.1 Validação de um pedido de criação de CET por derivação.

Input:

[1] usuário *S* e contexto *DCx* associado ao objeto CET fonte *Cx*.

Output:

[1] AUTORIZADO ou [2] NEGADO.

1. Executar algoritmo 3.4, onde *PA* = (*S*, *CET*, *v*, *m*), *v* = *NULL* e *m* = *criação*. Se Output = NEGADO, negar o pedido e terminar.
2. Executar algoritmo ALG 2, sendo *DCx* o contexto fonte. Se Output = NEGADO, negar o pedido e terminar.
3. Autorizar o pedido.

O passo 1 do algoritmo 4.1 verifica se o usuário tem permissão para criar um novo CET. O passo 2 verifica se o usuário tem permissão de acesso (leitura ou escrita) à DBV fonte *DCx*, bem como permissão de acesso, ao menos, a uma versão lógica de *DCx*.

Ressalte-se que ter permissão de acesso à DBV não significa que o usuário tem permissão de acesso às suas versões lógicas. Devem existir regras (*s, o, v, m*) e (*s, C, m*) no banco de dados que concedam ao usuário o acesso às versões lógicas da DBV fonte, de acordo com o algoritmo ALG 2. Somente as versões lógicas às quais o usuário tiver permissão de acesso (leitura ou escrita) na DBV fonte serão replicadas para a DBV destino, sendo atribuído o valor nulo permanente na DBV destino para as versões lógicas às quais o usuário não tem acesso na DBV fonte. Note que se o usuário tiver permissão de acesso (leitura) sobre o DBV fonte e não tiver permissão de acesso (leitura ou escrita) sobre versão lógica alguma do DBV, o pedido de criação é NEGADO.

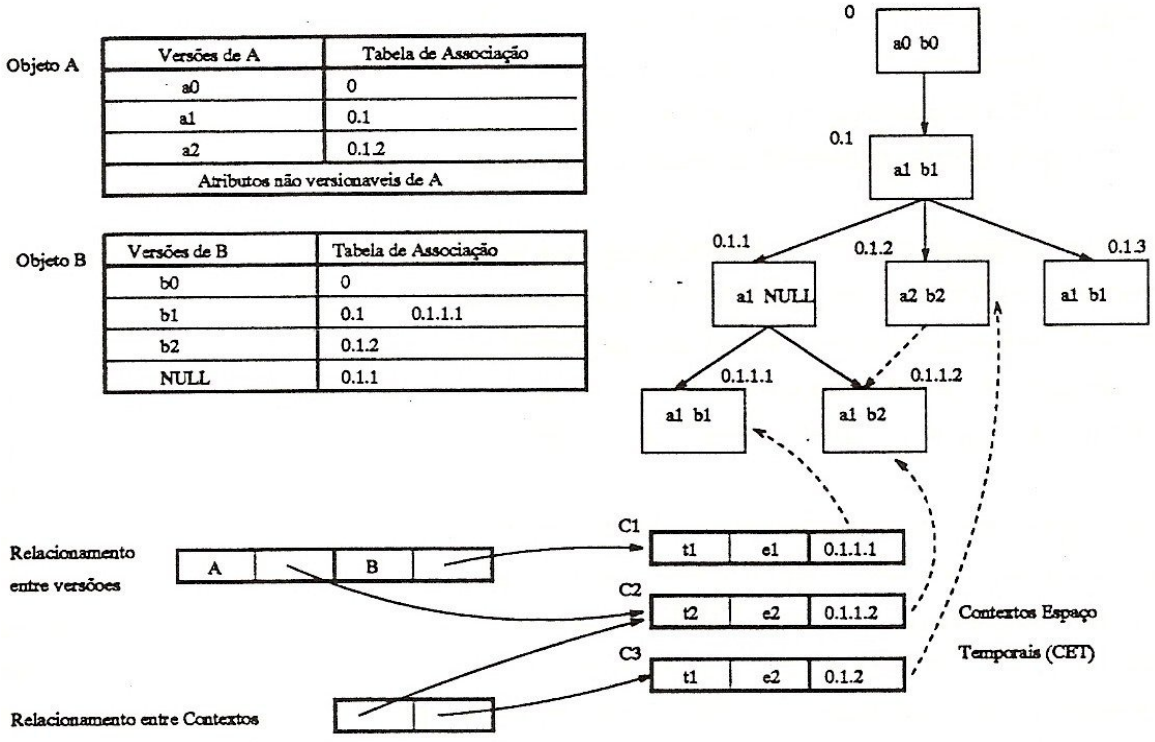


Figura 4.1: Contextos Espaço Temporais e Relacionamentos [delVal97].

4.2.2 Criação de CET por combinação

Além da criação por derivação, um CET pode ser criado a partir de combinação de outros contextos (combinação de DBV). Isto se dá a partir dos seguintes passos:

- Combinação de *n* DBV pela combinação de duas DBV sucessivamente. Uma das DBV fontes é definida como principal e a outra secundária. As versões lógicas dos objetos na nova DBV criada serão as versões lógicas dos objetos na versão principal mais as versões lógicas dos objetos na DBV secundária que não existem na DBV principal.
- Criação de um objeto CET associado à nova DBV, resultante do processo de combinação das *n* DBV.

O algoritmo 4.2 valida um pedido de criação de CET por combinação a partir de dois CET.

Algoritmo 4.2 Validação de um pedido de criação de CET por combinação a partir de dois CET.

Input:

[1] usuário *S* e contextos *DC1* e *DC2* associados aos objetos CET fontes *C1* e *C2* respectivamente.

Output:

[1] AUTORIZADO ou [2] NEGADO.

1. Executar algoritmo 3.4, onde *PA* = (*S*, *CET*, *v*, *m*), *v* = *NULL* e *m* = *criação*. Se Output = NEGADO, negar o pedido e terminar.
2. Executar algoritmo ALG 2, sendo *DC1* e *DC2* os contextos fontes. Se Output = NEGADO, negar o pedido e terminar.
3. Autorizar o pedido.

O passo 1 do algoritmo 4.2 é o mesmo do algoritmo de criação de CET por derivação. O passo 2 verifica a permissão de acesso (leitura ou escrita) às duas DBV associadas aos CET fontes, que serão combinadas para a criação de um novo CET. Ela também verifica se o usuário tem permissão de acesso, ao menos, a uma versão lógica de cada DBV fonte. Por questão de simplicidade, a solução aqui proposta nega o pedido de criação caso o resultado da execução do passo 2 seja NEGADO.

Uma segunda abordagem é a autorização da criação do CET, caso o usuário tenha permissão de acesso a somente uma das DBV fontes. Neste caso, o processo de criação deixa de ser por combinação e passe a ser realizado por derivação da DBV ao qual o usuário tem permissão de acesso. Pode-se, ainda, deixar a critério do usuário essa escolha, sempre que o acesso for negado a uma das duas DBV fontes.

Assim como no processo de criação de CET por derivação, durante o processo de criação de CET por combinação é necessário verificar a permissão de acesso do usuário a cada versão lógica contida em cada DBV fonte. Caso o usuário não tenha permissão de acesso (leitura ou escrita) sobre uma versão lógica da DBV primária (ou

secundária) no processo de combinação, será considerado o valor *nulo* para a realização da combinação.

Como exemplo, a figura 4.1 mostra o CET *C2* criado como combinação dos CET *C1* e *C3*. Para isto, supondo um sujeito *S*, devem existir na base de regras as seguintes regras de autorização:

- Regra que autoriza o sujeito *S* a criar um novo objeto CET no banco de dados: < *S, CET, V, M* >, onde *V* = NULL e *M* = *criação*.
- Regras que autorizem o sujeito *S* a acessar as DBV correspondentes aos CET *C1* e *C3*: < *S, DBVID0.1.1.1, V, M* > e < *S, DBVID0.1.2, V, M* >, onde para ambas as regras *V* = NULL e *M* = *leitura ou escrita.*
- Regras que autorizem o sujeito S a acessar as versões lógicas dos contextos *C1* e *C3*: < *S, A, 0.1.1.1, M* >, < *S, B, 0.1.1.1, M* >, < *S, A, 0.1.2, M* > e < *S, B, 0.1.2, M* >, onde *M* = *leitura ou escrita* para todas as regras. Ao invés de ter regras específicas para cada versão lógica, também seria possível que existissem regras do tipo < *S*, *C*, *M* >, onde a consulta espacial *C* referenciasse as versões lógicas dos objetos *A* e *B* nos contextos *C1* e *C3*.

Quando o estado da versão lógica do objeto *B* no CET *C2* é procurado, primeiramente é analisada a tabela de associação de *B* na procura em uma das suas entradas do identificador da DBV 0.1.1.2 associada ao CET *C2*. Como não existe, então é procurado na tabela um dos seus ancestrais, encontrando a entrada para a DBV 0.1.1. No entanto, como seu valor é nulo, é realizado o mesmo processo para a DBV ancestral secundária, neste caso 0.1.2, que aparece na tabela como uma versão física *b2* a qual é o valor da versão lógica procurada.

Na solução proposta, caso o usuário não tenha permissão de acesso a um dos dois CET fontes (*C1* ou *C3*), o processo de criação de um novo CET por combinação é negado. Caso o usuário não tenha permissão de acesso a uma ou mais versões lógicas presentes nos CET que serão combinados, será considerado o valor *nulo* para tais versões durante o processo de combinação.

4.2.3 Eliminação de CET

A eliminação de um CET é mapeada no modelo DBV como a eliminação de uma DBV. Isto é interpretado como a eliminação das versões lógicas que a DBV contém, além da remoção da DBV e do objeto CET associado. Caso exista um relacionamento de dependência entre as versões lógicas do CET e de um ou mais CTR, a solução adotada estabelece que o usuário deve ter permissão de acesso (leitura) aos CTR dependentes, e permissão de acesso (escrita) às versões lógicas de objetos dos CTRs que possuem relacionamento de dependência com as versões lógicas do CET.

Isto é necessário uma vez que ao se eliminar as versões lógicas de objetos do CET, é preciso remover o registro da dependência com as versões lógicas dos objetos dos CTR. Como as dependências podem ter significado semântico, esta verificação deve ser realizada.

O algoritmo 4.3 verifica um pedido de eliminação de CET.

Algoritmo 4.3 Validação de um pedido de eliminação de CET.

Input:

[1] usuário *S* e contexto *DCx* associado ao objeto CET *Cx*.

Output:

[1] AUTORIZADO ou [2] NEGADO.

1. Executar algoritmo 3.4, onde *PA* = (*S*, *DCx*, *v*, *m*), *v* = *NULL* e *m* = *remoção*. Se Output = NEGADO, negar o pedido e terminar.
2. Para cada *DCT1*, *DCT2*, ..., *DCTn* associado a um CTR diferente que possui um relacionamento de dependência com *DCx*:
   2.1. Executar algoritmo 3.4, onde *PA* = (*S*, *DCTn*, *v*, *m*), *v* = *NULL* e *m* = *leitura*. Se Output = NEGADO, negar o pedido e terminar.
   2.2. Para cada OID*1*, *OID2*, ..., *OIDn* presente em *DCTn* e que possui um relacionamento de dependência com a versão lógica do mesmo objeto em *DCx*, executar algoritmo 3.4, onde *PA* = (*S*, *OIDn*, *DCTn*, *m*) e *m* = *escrita*. Se Output = NEGADO, negar o pedido e terminar.
3. Autorizar o pedido.

Note que caso o usuário tenha permissão para eliminar o CET (passo 1), mas não tenha permissão de acesso (leitura) sobre os CTR (passo 2.1) que possuem um relacionamento de dependência com o CET e sobre uma ou mais versões lógicas que definem o relacionamento de dependência com as versões lógicas do CET (passo 2.2), a operação de remoção do CET não poderá ser efetuada.

4.2.4 Definição e eliminação de relacionamento entre CET

No modelo de versões de [delVal97] é possível estabelecer relacionamentos entre dois CET. A semântica associada a estes enlaces será dada pelo usuário. Exemplos de relacionamento entre CET são:

- Relacionamento de sucessão temporal entre CET que descrevem diferentes estados temporais do banco de dados.
- Relacionamento de equivalência entre CET diferentes que representam o mesmo estado temporal do mundo, mas com diferentes representações espaciais.

Estes relacionamentos semânticos provêem maior riqueza para a modelagem, permitindo novos tipos de consultas e formas de navegação no banco de dados.

O algoritmo 4.4 verifica a permissão para definição ou eliminação de relacionamentos entre CET:

Algoritmo 4.4 Validação de um pedido de definição ou eliminação de relacionamento entre CET.

Input:

[1] usuário *S* e contextos *DC1, DC2, ..., DCn* associados aos objetos CET fontes *C1, C2, ..., Cn* respectivamente.

Output:

[1] AUTORIZADO ou [2] NEGADO.

1. Para cada *DC1*, *DC2*, ..., *DCn*, executar algoritmo 3.4, onde *PA* = (*S*, *DCn*, *v*, *m*), *v* = *NULL* e *m* = *leitura ou escrita*. Se Output = NEGADO, negar o pedido e terminar.
2. Autorizar o pedido.

O passo 1 verifica se o usuário tem permissão de acesso (leitura ou escrita) para todos os CET fontes. Neste caso, um novo objeto é criado para mapear os CET relacionados. Já no caso da eliminação, o objeto que mapeia o relacionamento é removido.

A figura 4.1 mostra um objeto que relaciona dois CET, neste caso *C2* e *C3*. Para o estabelecimento deste relacionamento, dado um usuário *S*, as seguintes regras devem existir no banco de dados para autorizar a operação: *R1* = < *S, DBVID0.1.1.2, v, m* > e *R2* = < *S, DBVID0.1.2, v, m* >, onde *v* = NULL e *m* = *leitura* ou *escrita*.

4.2.5 *Check in* a partir de um espaço de trabalho

A operação de *check in* de objetos de um ET para objetos de CET (BD permanente) que tenham o mesmo vetor de dimensões é feita da seguinte forma:

1. Para cada versão lógica do objeto *OID* em um CTR, *CTi*[*Vi*] no ET, buscar um contexto CET *Ci*[*Vj*], onde *Vi* = *Vj* sendo *V* os vetores de dimensões.
2. Se esse contexto *Ci* foi encontrado, então mudar o estado de *OID* em *Ci*, para o estado de *CTi*.
3. Se o contexto não for encontrado, criar um novo contexto *Ci*[*Vj*] no banco de dados e inserir o objeto *OID* com o estado *CTi*.

Esta operação de *check in* é mapeada no modelo DBV como a criação de uma nova DBV, que será a combinação da DBV associada ao CTR (considerada como DBV principal) e a DBV associada ao CET (considerada secundária), para cada CTR no ET. A nova DBV criada será então associada ao CET enquanto a DBV anterior é marcada para ser eliminada. No caso que a CET com o mesmo vetor de dimensões não exista, um novo CET é criado e junto com ele uma nova DBV derivada.

Com a introdução do controle de acesso no modelo de versões, os passos para a execução de um *check in* passam a ser definidos pelo algoritmo 4.5:

Algoritmo 4.5 Operação de *check in* de um ET.

Input:

[1] usuário *S* e objeto ET *ET*.

Output:

[1] SUCESSO ou [2] FALHA.

1. Para cada CTR *CTi[Vi]* (associado à DBV *DCTi*) contida em *ET*, buscar um contexto CET *Cj*[*Vj*], onde *Vi* = *Vj* sendo *V* os vetores de dimensões.
2. Se esse contexto *Cj (*associado à DBV *DCj)* foi encontrado.
   2.1. Executar algoritmo 3.4, onde *PA* = (*S*, *DCj*, *v*, *m*), *v* = *NULL* e *m* = *escrita*. Se Output = NEGADO, cancelar a transação de *check in* e terminar.
   2.2. Para cada versão lógica dos objetos *OID1*, *OID2*, ..., *OIDn* em *DCTi*, se existir uma versão lógica de *OIDn* em *DCj*, executar algoritmo 3.4, onde *PA* = (*S*, *OIDn*, *DCj*, *m*) e *m* = *escrita*. Se Output = NEGADO, cancelar a transação de *check in* e terminar.
   2.3. Criar uma nova DBV *DCk* como combinação de *DCTi* (considerada como DBV principal) e *DCj* (considerada secundária).
   2.4. Associar *DCk* a *Cj* e marcar a *DCj* para ser eliminada.
   2.5. Copiar as regras de autorização de *DCj* (e de suas versões lógicas) para *DCk*.
3. Se o contexto *Cj* não foi encontrado.
   3.1. Executar algoritmo 3.4, onde *PA* = (*S*, *CET*, *v*, *m*), *v* = *NULL* e *m* = *criação*. Se Output = NEGADO, cancelar a transação de *check in* e terminar.
   3.2. Criar uma nova DBV *DCm* e inserir nela todos os objetos de *CTi* com o estado em *CTi*.
   3.3. Criar um novo objeto CET associado *Cm* e associá-lo à nova DBV *DCm*.
   3.4. Criar regras (*s, o, v, m*) para o novo contexto criado e para suas versões lógicas.
4. Retornar SUCESSO para a operação de *check in*.

O algoritmo 4.5 funciona da seguinte maneira. Ao receber um pedido de *check in* de um ET, para cada CTR existente no ET, seleciona o CET que contém o mesmo vetor de dimensões do CTR (passo 1). Caso o CET exista (passo 2), verifica se o usuário tem permissão de acesso (escrita) sobre o CET (passo 2.1). Caso não tenha, o processo de *check in* é cancelado. Caso tenha, verifica se o usuário tem permissão para alterar o estado dos objetos que pertençam à intersecção dos contextos CET e CTR (passo 2.2). Caso não tenha, o processo de *check in* é cancelado. Caso tenha, os estados dos objetos no CET são atualizados com os estados dos objetos em CTR.

Caso o CET com o mesmo vetor de dimensões do CTR não exista (passo 3), verifica se o usuário tem autorização para criar um novo CET no banco de dados (passo 3.1). Caso não tenha, o processo de *check in* é cancelado. Caso tenha, um novo CET é criado e todos os objetos de CTR são inseridos no novo CET com o estado em CTR.

Uma abordagem alternativa é verificar todas as permissões antes de se iniciar o processo de *check in*. A vantagem desta abordagem é que operações de *check in* grandes não causam *overhead* de criação de contextos e cópia de versões de objetos caso a operação seja cancelada ao final por falta de permissão de acesso.

A solução adotada pela dissertação considera que o processo de localização de contextos com o mesmo vetor de dimensões, a localização e comparação de versões lógicas de objetos entre contextos diferentes não são triviais. Já na segunda abordagem, esses processos são duplicados, uma vez para a validação da permissão de acesso e outra para a realização do *check in*.

4.3 Operações sobre Espaço de Trabalho (ET)

Um ET é representado como um objeto de estado único, ou seja, não versionável. Sua estrutura é formada por um conjunto de referências aos objetos incluídos na sua *Extensão*, uma representação geométrica da sua área alvo e um conjunto de CTR associados.

4.3.1 Criação de ET a partir de um conjunto de CET e de uma área geográfica alvo

A criação de um ET a partir de um conjunto de CET selecionados e de uma área geográfica alvo se dá pelos seguintes passos [delVal97]:

1 Seleção dos CET *C1*, *C2*, ..., *Ck* que serão usados no ET.

2 Seleção espaço-temporal, dentro de cada CET *Ci*, dos objetos a serem manipulados, para definir a *Extensão de Trabalho* do ET. Esta seleção é baseada em critérios espaciais (definição de área geográfica alvo) e semânticos.

3 Criação dos CTR *CT1*, *CT2*, ..., *CTk* copiados a partir de *C1*, *C2*, ..., *Ck*, usando as restrições espaço-temporais de (2) (*check out*).

4 Associação a cada *CT1*, *CT2*, ..., *CTk* dos mesmos vetores de dimensões de *C1*, *C2*, ..., *Ck* respectivamente.

Suponha, por exemplo, que a figura 4.1 represente um banco de dados que tem definidas duas dimensões (tempo e escala) e cujos CET *C1*, *C2* e *C3* possuam classes para representar vegetação, solo e hidrografia. Suponha também um ET para estudo hidrográfico na escala *y*. Como se trata de estudo hidrográfico, apenas fenômenos relativos à hidrografia são considerados. Neste caso, o ET é criado selecionando *C2* e *C3* e a seguir somente os objetos da classe hidrografia os quais são copiados para dois CTR, *CT2* e *CT3* que são criados dentro do ET. Finalmente, a *CT2* e *CT3* são associados os *vetores de dimensões* de *C2* e *C3* respectivamente. Como um ET é um banco de dados de trabalho, dentro deste podem ser gerados novos CTR, associados a *vetores de dimensões* diferentes dos de *CT2* e *CT3*.

O algoritmo 4.6 verifica a permissão para criação de um ET a partir de um conjunto de CET selecionados e uma área geográfica alvo:

Algoritmo 4.6 Validação de um pedido de criação de ET.

Input:

[1] usuário *S*.

[2] contextos *DC1, DC2, ..., DCn* associados aos objetos CET selecionados *C1, C2, ..., Cn* respectivamente.

[3] área geográfica alvo.

Output:

[1] AUTORIZADO ou [2] NEGADO.

1. Executar algoritmo 3.4, onde *PA* = (*S*, *ET*, *v*, *m*), *v* = *NULL* e *m* = *criação*. Se Output = NEGADO, negar o pedido e terminar.
2. Executar algoritmo 3.4, onde *PA* = (*S*, *CTR*, *v*, *m*), *v* = *NULL* e *m* = *criação*. Se Output = NEGADO, negar o pedido e terminar.
3. Para cada *DC1*, *DC2*, ..., *DCn*, selecionar as versões lógicas dos objetos *OID1*, *OID2*, ..., *OIDy* pertencentes à área geográfica alvo. Executar ALG 1, sendo *DCn* o contexto fonte, *OIDy* o objeto alvo e modo de acesso leitura ou escrita. Se Output = NEGADO para um *DCn* ou para todos *OIDy* pertencentes a *DCn*, negar o pedido e terminar.
4. Autorizar o pedido.

O passo 1 do algoritmo 4.6 verifica se o usuário *S* tem permissão para criar um novo ET. O passo 2 verifica se o usuário tem permissão para criar CTR. Finalmente, para autorizar o pedido de criação de ET, o usuário deve ter permissão de acesso (leitura ou escrita), ao menos, a uma versão lógica, pertencente à área geográfica alvo do ET, de cada CET selecionado (passo 3). Isto significa que o pedido de criação de ET será negado nos seguintes casos:

- Caso o usuário não tenha permissão de acesso (leitura ou escrita) a um CET pertencente ao conjunto de CET selecionados.
- Caso o usuário não tenha permissão de acesso (leitura ou escrita) a nenhuma versão lógica de um CET selecionado.

Na prática, serão criadas *n* DBV, sendo cada uma delas associada a um CTR diferente, que representam o conjunto de CET selecionados. As versões lógicas de cada CTR terão o valor *nulo permanente* caso o usuário não tenha permissão e acesso a versão lógica do CET correspondente.

Também é possível criar um ET *e1* a partir de outro ET *e2*. Esta operação é similar ao caso de criação a partir de um conjunto de CET. Um ET, que contém um ou mais CTR, é similar a um banco de dados de trabalho, que contém um ou mais CET. O algoritmo 4.6 também se aplica a este caso. Se o usuário não tiver permissão de acesso ao ET de referência, o pedido de criação é negado.

4.3.2 Inserção e extração de objetos da extensão do ET

No modelo de [delVal97] são associadas duas operações à extensão de um ET: Inserir e Eliminar um objeto. No caso da operação de inserção de um novo objeto na extensão, o mapeamento para o modelo DBV corresponde a mudar o valor *nulo permanente* pelo valor *nulo temporário* em todas as versões desse objeto nas DBV associadas ao CTR dentro do ET. No caso da operação de eliminação será exatamente o contrário.

O algoritmo 4.7 valida um pedido de eliminação de objeto da extensão de um ET:

Algoritmo 4.7 Validação de um pedido de eliminação de objeto da extensão do ET.

Input:

[1] usuário *S*, espaço de trabalho *ETx* e objeto *OIDy*.

Output:

[1] AUTORIZADO ou [2] NEGADO.

1. Executar algoritmo 3.4, onde *PA* = (*S*, *ETx*, *v*, *m*), *v* = *NULL* e *m* = *escrita*. Se Output = NEGADO, negar o pedido e terminar.
2. Para cada *DCT1*, *DCT2*, ..., *DCTn* pertencente a *ET*x e que possua uma versão lógica do objeto *OIDy* (valor diferente de *nulo permanente*), executar ALG 1, sendo *DCTn* o contexto fonte, *OIDy* o objeto alvo e modo de acesso escrita. Se Output = NEGADO, negar o pedido e terminar.
3. Autorizar o pedido.

Para que o usuário possa remover objetos da extensão do ET, ele deve ter permissão de acesso para alterar o ET (passo 1), permissão de acesso para alterar todos os CTR do ET e permissão de acesso para alterar a versão lógica do objeto *OIDy* em cada contexto CTR do ET (passo 2). Caso o usuário tenha permissão de acesso negada para algum destes passos, a operação de extração de objeto da extensão do ET é negada.

A inserção de objetos na extensão do ET segue praticamente os mesmos passos do algoritmo 4.7. A única diferença está no passo 2, onde há a necessidade de se verificar apenas a permissão de acesso (escrita) a todos os CTR do ET. Se esta condição for satisfeita, então o pedido de inserção é autorizado.

Uma segunda opção é não impedir a inserção de um objeto no ET caso o usuário não tenha permissão de acesso (escrita) sobre todos os CTR do ET. Neste caso, o valor *nulo permanente* é alterado para *nulo temporário* somente nos CTR em que o usuário tiver permissão de acesso (escrita). A dissertação não optou por esta solução, uma vez que ambientes de desenvolvimento de projetos podem ser formados por todos os CTR de um ET. Se somente alguns CTR forem atualizados e outros não, pode causar uma inconsistência no ambiente do projeto.

4.3.3 *Check out* sobre um ET a partir de um conjunto de CET selecionado

A operação de *check out* sobre um ET a partir de um conjunto de *n* CET selecionados é mapeada em DBV como uma transação que envolve os seguintes passos [delVal97]:

1. Criação de *n* novas DBV, cada uma delas como derivação de cada uma das DBV associadas às *n* CET, atribuindo em cada uma delas o estado *nulo permanente* às versões dos objetos que não estão contidos na *extensão* do ET.
2. Criação de *n* objetos CTR a cada um dos quais é associada uma das novas DBV e o *vetor de dimensões* da CET associada.
3. Inserção destes objetos no conjunto de CTR do ET.

O algoritmo 4.8 valida um pedido de *check out* sobre um ET a partir de um conjunto de CET selecionados.

Algoritmo 4.8 Validação de um pedido de *check out* sobre um ET a partir de um conjunto de CET selecionados.

Input:

[1] usuário *S*.

[2] espaço de trabalho *ETx*.

[3] contextos *DC1, DC2, ..., DCn* associados aos objetos CET selecionados *C1, C2, ..., Cn* respectivamente.

Output:

[1] AUTORIZADO ou [2] NEGADO.

1. Executar algoritmo 3.4, onde *PA* = (*S*, *ETx*, *v*, *m*), *v* = *NULL* e *m* = e*scrita*. Se Output = NEGADO, negar o pedido e terminar.
2. Executar algoritmo 3.4, onde *PA* = (*S*, *CTR*, *v*, *m*), *v* = *NULL* e *m* = *criação*. Se Output = NEGADO, negar o pedido e terminar.
3. Para cada *DC1*, *DC2*, ..., *DCn*, selecionar as versões lógicas dos objetos *OID1*, *OID2*, ..., *OIDy* pertencentes à extensão de *ETx*. Executar ALG 1, sendo *DCn* o contexto fonte, *OIDy* o objeto alvo e modo de acesso leitura ou escrita. Se Output = NEGADO para um *DCn* ou para todos *OIDy* pertencentes a um *DCn*, negar o pedido e terminar.
4. Autorizar o pedido.

O algoritmo para validação de um pedido de *check out* sobre um ET segue a mesma lógica do algoritmo 4.6. As diferenças estão nos passo 1 e 3. O passo 1 verifica a permissão de acesso (escrita) sobre o ET alvo. O passo 3 verifica a permissão de acesso (leitura ou escrita) sobre as versões lógicas em *DCn* dos objetos pertencentes à extensão do ET.

Também é possível a operação de *check out* sobre um ET a partir de outro ET. Esta operação é similar ao caso de *check out* a partir de um conjunto de CET. Basta que cada uma das *n* novas DBV seja derivada de cada um dos CTR do ET de referência (passo 1 do processo de *check out*). O algoritmo 4.8 também se aplica para este caso.

4.4 Operações sobre Contexto de Trabalho (CTR)

Assim como CET e ET, um CTR é representado como um objeto de estado único, ou seja, não versionável. De maneira similar aos CET, os CTR são associados à DBV.

4.4.1 Criação de um CTR a partir de outro no mesmo ET

A operação de criação por derivação de um CTR, dentro de um ET, a partir de outro CTR é mapeada como uma simples derivação de DBV, associada à criação de um objeto CTR associado. Uma DBV associada a um CTR é criada copiando os objetos de interesse do BD subjacente e tornando os demais objetos *nulos permanentes*.

O algoritmo para validação de um pedido de criação de um CTR a partir de outro no mesmo ET segue as mesmas passos do algoritmo 4.8. O usuário deve ter permissão para alterar o ET em questão (passo 1) e permissão para criar um novo CTR

dentro do ET (passo 2). A diferença está no passo 3. Agora, é necessário que o usuário tenha permissão de acesso (leitura ou escrita) sobre ao menos uma versão lógica de objeto no CTR fonte. Todas as versões do objeto que não são acessíveis pelo usuário, na DBV relacionada ao CTR fonte, recebem o valor *nulo permanente* na nova DBV associada ao novo CTR.

A figura 2.6 (capítulo 2) mostra dois CTR criados por derivação de outro CTR no mesmo ET. No primeiro caso, o CTR *CTR3* é derivado do CTR *CTR1* no espaço de trabalho *ET1*. Já no segundo caso, o CTR *CTRb* é derivado do CTR *CTRa* no espaço de trabalho *ET2*. A derivação de um CTR fonte para um CTR destino não cria relacionamento algum entre eles. Isto deve ser feito explicitamente pelo usuário caso haja a necessidade da existência de um relacionamento entre os contextos.

4.4.2 Definir relacionamentos de dependência entre CTR em ET diferentes

A definição de relacionamento de dependência entre dois CTR em ET diferentes recai sobre a definição de relacionamento de dependência entre contextos. Este relacionamento garante que os estados das versões do objeto nos contextos possam evoluir em conjunto. Mapeando para o modelo DBV, do ponto de vista físico, garante que as versões lógicas em uma mesma classe de equivalência estejam associadas sempre à mesma entrada na tabela de associação. Para cada objeto são definidas classes de equivalência entre as versões nas diferentes DBV de forma tal que uma mudança em uma versão lógica de um objeto em uma DBV será refletida nas versões lógicas das DBV na mesma classe de equivalência.

Por exemplo, na figura 4.2, suponha que existe um relacionamento de dependência entre as versões de *A* nas DBV 0.1 e 0.1.3 e as versões do objeto *B* nas DBV 0.1.2.1 e 0.1.1.2. Isto é, traduzindo nas tabelas ao lado, onde as linhas estão particionadas em classes de equivalência de instâncias. Suponha que o estado da versão lógica de *A* na DBV 0.1 é mudado para um novo estado *a3*. Neste caso, a tabela de associação é modificada garantindo que as versões lógicas 0.1 e 0.1.3 continuam compartilhando a nova versão física *a3*, enquanto as versões lógicas 0.1.1 e 0.1.2.1 explicitamente compartilham a mesma versão física *a1*, mas em classes diferentes. A versão da DBV 0.1 evolui independentemente em uma nova entrada.

Relacionamentos de dependência entre CET são definidos da mesma forma, recaindo sobre definição de relacionamentos entre contextos. As DBV da figura 4.2 podem representar tanto CET quanto CTR.

O algoritmo 4.9 valida um pedido de definição de relacionamento de dependência entre contextos:

Algoritmo 4.9 Validação de um pedido de definição de relacionamento de dependência entre contextos.

Input:

[1] usuário *S*, objeto *OIDx* e contextos *D1*, *D2*, ..., *Dn*.

Output:

[1] AUTORIZADO ou [2] NEGADO.

1. Para cada *D1*, *D2*, ..., *Dn* pertencente à lista de contextos de entrada, executar ALG 1, sendo *Dn* o contexto fonte, *OIDx* o objeto alvo e modo de acesso *escrita*. Se Output = NEGADO, negar o pedido e terminar.
2. Autorizar o pedido.

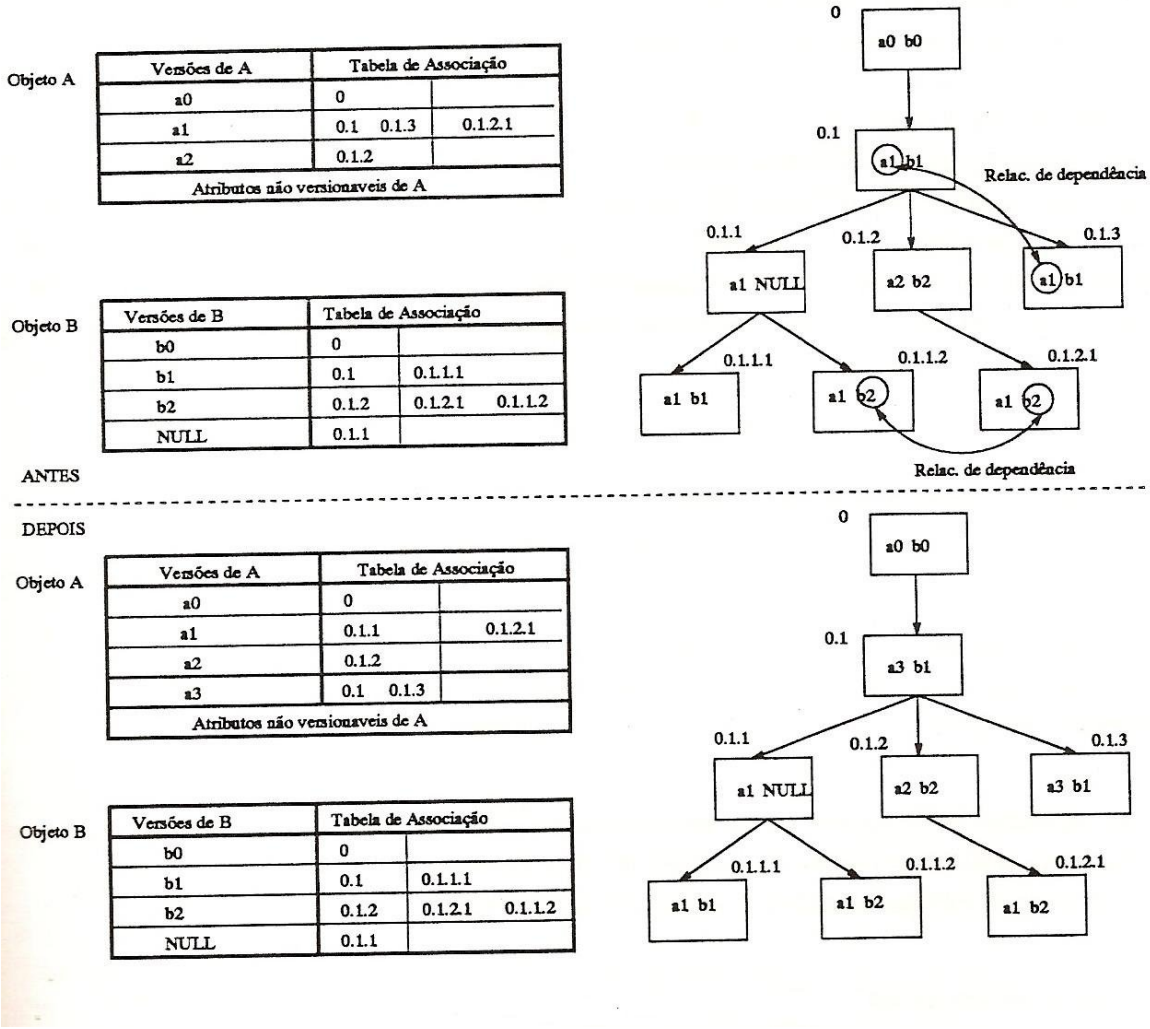


Figura 4.2: Dependência entre versões [delVal97].

Para que um relacionamento de dependência entre contextos seja definido, o usuário deve ter permissão de acesso (leitura ou escrita) sobre cada contexto envolvido, e permissão de acesso (escrita) sobre a versão lógica do objeto *OIDx* em cada contexto *Dn* (passo 1). Se o usuário não tiver permissão de acesso (escrita) sobre a versão lógica em um dos contextos, o pedido de definição de relacionamento de dependência entre contextos é negado.

A figura 2.6 (capítulo 2) mostra um relacionamento de dependência entre os contextos *CTR2* e *CTRa*, pertencentes aos espaços de trabalho *ET1* e *ET2* respectivamente. Para um ou mais objetos de *CTR2*, caso o estado da versão lógica do objeto mude, o estado da versão lógica do objeto em *CTRa* também será alterado (e vice-versa).

4.4.3 Eliminação de um contexto de trabalho

A eliminação de um CTR ocorre da mesma forma que o processo de eliminação de um CET. A operação é mapeada no modelo DBV como a eliminação de uma DBV. Isto é interpretado como a eliminação das versões lógicas que a DBV contém, além da remoção da DBV e do objeto CTR associado. Caso exista um relacionamento de dependência entre as versões lógicas do CTR e as versões lógicas de um ou mais CET, a solução adotada estabelece que o usuário deve, também, ter permissão de acesso aos CET e às versões lógicas dos CET que possuem relacionamento de dependência com as versões lógicas do CTR.

O algoritmo 4.10 valida um pedido de eliminação de CTR:

Algoritmo 4.10 Validação de um pedido de eliminação de CTR.

Input:

[1] usuário *S* e contexto *DCTx* associado ao CTR *CTx*.

Output:

[1] AUTORIZADO ou [2] NEGADO.

1. Executar algoritmo 3.4, onde *PA* = (*S*, *DCTx*, *v*, *m*), *v* = *NULL* e *m* = *remoção*. Se Output = NEGADO, negar o pedido e terminar.
2. Para cada *DC1*, *DC2*, ..., *DCn* associado a um CET diferente que possui um relacionamento de dependência com *DCTx*:
   2.1. Executar algoritmo 3.4, onde *PA* = (*S*, *DCn*, *v*, *m*), *v* = *NULL* e *m* = *leitura*. Se Output = NEGADO, negar o pedido e terminar.
   2.2. Para cada OID*1*, *OID2*, ..., *OIDy* presente em *DCn* e que possui um relacionamento de dependência com a versão lógica do mesmo objeto em *DCTx*, executar algoritmo 3.4, onde *PA* = (*S*, *OIDy*, *DCn*, *m*) e *m* = *escrita*. Se Output = NEGADO, negar o pedido e terminar.
3. Autorizar o pedido.

Caso o usuário tenha permissão para eliminar um CTR, mas não tenha permissão de acesso (escrita) sobre uma ou mais versões lógicas dos CET que possuem relacionamento de dependência com as versões lógicas do CTR, a operação de remoção do CTR não é autorizada.

4.5 Operações sobre objetos e versões de objetos

Esta seção apresenta as operações sobre objetos e versões de objetos e como o controle de acesso proposto pela dissertação atua sobre elas.

Todos os algoritmos desta seção referentes à criação, modificação ou eliminação de uma versão lógica de um objeto devem ter um passo adicional de verificação de integridade com relação a regras pré-existentes e que passam a afetar os objetos (se criados ou modificados). Se eliminados, há possibilidade de um sujeito perder o acesso a objetos que possuam interseção ou que estejam incluídos naquele objeto. Assim sendo, ao final de cada operação sobre versões de objetos, sugere-se a seguinte verificação: se na criação ou modificação de uma versão lógica de objeto *L* aparecerem

novos conflitos com versões lógicas de objetos pertencentes ao mesmo contexto, solicitar decisão do administrador sobre a inclusão ou atualização de *L*. No caso da eliminação de uma versão lógica de objeto *L*, se existir conflito com outras versões lógicas de objetos pertencentes ao mesmo contexto, solicitar decisão do administrador sobre a eliminação de *L*.

4.5.1 Criação de um objeto

A criação de um objeto no modelo DBV é interpretada como a criação (inserção) de uma versão lógica do objeto em cada uma das DBV já existentes para uma versão física inicial nula. Inicialmente o estado dessas versões em todas as DBV é o mesmo, mas cada uma poderá evoluir de forma independente.

A criação do objeto é realizada associando uma versão física inicial à versão lógica da raiz do grafo de derivação. Em um CET/CTR qualquer, para o mecanismo DBV, a criação vai inserir o objeto em todas as CET e CTR do modelo. Para manter a semântica da inserção, no modelo de [delVal97], atribui-se adequadamente *nulos temporários* e *nulos permanentes* às DBV existentes.

As associações de valores *nulo permanente* e *nulo temporário* a determinadas versões do objeto em diferentes DBV dependerá dos seguintes tipos de criação de um objeto:

a) *Criação apenas para um CET*: Associar o valor *nulo temporário* às versões lógicas do objeto nas DBV associadas aos demais CET. Associar o valor *nulo permanente* às versões lógicas do objeto em todos os CTR (DBV correspondente). O algoritmo 4.11 valida um pedido de criação de objeto para um CET.

Algoritmo 4.11 Validação de um pedido de criação de objeto para um CET.

Input:

[1] usuário *S* e contexto *DCx* associado ao objeto CET *Cx*.

Output:

[1] AUTORIZADO ou [2] NEGADO.

1. Executar algoritmo 3.4, onde *PA* = (*S*, *DCx*, *v*, *m*), *v* = *NULL* e *m* = *escrita*. Se Output = NEGADO, negar o pedido e terminar.
2. Autorizar o pedido.

Basta que o usuário tenha permissão de acesso (escrita) sobre a DBV associada ao CET em questão para que o pedido de criação seja aceito.

b) *Criação para todos os CET*: Atribui estado inicial ao objeto na CET desejada, associa o valor *nulo permanente* às versões lógicas do objeto das DBV associadas a todos os CTR e associa o estado inicial do objeto a todas as versões lógicas de DBV associadas às CET. O algoritmo 4.12 valida um pedido de criação de objeto para todos os CET.

Algoritmo 4.12 Validação de um pedido de criação de objeto para todos os CET.

Input:

[1] usuário *S*

Output:

[1] AUTORIZADO ou [2] NEGADO.

1. Para cada *DC1*, *DC2*, ..., *DCn* associado aos objetos CET existentes no banco de dados, executar algoritmo 3.4, onde *PA* = (*S*, *DCn*, *v*, *m*), *v* = *NULL* e *m* = *escrita*. Se Output = NEGADO, negar o pedido e terminar.
2. Autorizar o pedido.

Neste caso, o usuário deve ter permissão de acesso (escrita) a todas DBV associadas aos CET existentes no banco de dados, para que o pedido de criação seja aceito.

c) *Criação em um CTR*: Associar o valor *nulo permanente* às versões lógicas das DBV associadas a quaisquer CTR de outros ET e o valor *nulo temporário* às versões lógicas das DBV associadas aos demais CTR do ET e todos CET. O algoritmo 4.13 valida um pedido de criação de objeto para um CTR.

Algoritmo 4.13 Validação de um pedido de criação de objeto para um CTR.

Input:

[1] usuário *S* e contexto *DCTx* associado ao objeto CTR *CTx*.

Output:

[1] AUTORIZADO ou [2] NEGADO.

1. Executar algoritmo 3.4, onde *PA* = (*S*, *DCTx*, *v*, *m*), *v* = *NULL* e *m* = *escrita*. Se Output = NEGADO, negar o pedido e terminar.
2. Autorizar o pedido.

Similarmente ao algoritmo 4.11, o usuário precisa apenas ter permissão de acesso (escrita) à DBV associada ao CTR em questão para que o pedido de criação seja aceito.

d) *Criação para todos os CTR de um ET*: Associar o valor *nulo permanente* às versões das DBV associadas a quaisquer CTR em outros ET e o valor *nulo temporário* às versões das DBV associadas aos CET. O algoritmo 4.14 valida um pedido de criação de objeto para todos os CTR de um ET.

Algoritmo 4.14 Validação de um pedido de criação de objeto para todos os CTR de um ET.

Input:

[1] usuário *S* e objeto *ETx*.

Output:

[1] AUTORIZADO ou [2] NEGADO.

1. Executar algoritmo 3.4, onde *PA* = (*S*, *ETx*, *v*, *m*), *v* = *NULL* e *m* = *escrita*. Se Output = NEGADO, negar o pedido e terminar.
2. Para cada *DCT1*, *DCT2*, ..., *DCTn* associado aos objetos CTR existentes em *ETx*, executar algoritmo 3.4, onde *PA* = (*S*, *DCTn*, *v*, *m*), *v* = *NULL* e *m* = *escrita*. Se Output = NEGADO, negar o pedido e terminar.
3. Autorizar o pedido de criação de objeto para todos os CTR de um ET.

Para que o pedido de criação seja aceito, o usuário deve ter permissão de acesso (escrita) ao ET (passo 1) e permissão de acesso (escrita) às DBV associadas aos CTR existentes no ET (passo 2).

4.5.2 Leitura, modificação e eliminação da versão de um objeto em um CET ou CTR

Estas operações são mapeadas no modelo DBV como operações de acesso ao estado de um objeto em uma DBV. O algoritmo 3.4 de validação de um pedido de acesso garante que o usuário terá acesso, do modo desejado, aos objetos aos quais ele tem autorização.

Ter autorização significa que há regras no banco de dados (*s, o, v, m*) e (*s, C, m*) que autorizem o usuário a acessar o objeto no modo desejado. Porém, com a existência de relacionamentos de dependência entre contextos, a validação do pedido de modificação e eliminação da versão de um objeto se torna mais complexa. Nestes dois casos, o usuário deve ter permissão de acesso de escrita ou remoção para as versões dos objetos que definem relacionamento de dependência entre contextos. O algoritmo de validação 4.15 valida um pedido de modificação ou eliminação da versão de um objeto em um CET ou CTR.

Algoritmo 4.15 Validação de um pedido de modificação (ou eliminação) da versão lógica de objeto para um CET ou CTR.

Input:

[1] usuário *S*, contexto *Dx* e objeto *OIDx*.

Output:

[1] AUTORIZADO ou [2] NEGADO.

1. Executar algoritmo 3.4, onde *PA* = (*S*, *Dx*, *v*, *m*), *v* = *NULL* e *m* = *leitura* ou *escrita* (para modificação) ou *m* = *escrita* (para eliminação). Se Output = NEGADO, negar o pedido e terminar.
2. Para cada *D1*, *D2*, ..., *Dn* que possui relacionamento de dependência com *Dx*, executar ALG 1, sendo *Dn* o contexto fonte, *OIDx* o objeto alvo e modo de acesso *escrita*. Se Output = NEGADO, negar o pedido e terminar.
3. Autorizar o pedido.

O passo 1 do algoritmo verifica se o usuário tem permissão de acesso (leitura ou escrita) ao contexto de entrada. O passo 2 verifica se o usuário tem permissão de acesso aos contextos que possuem relacionamento de dependência com *Dx* e permissão de acesso (escrita) às versões lógicas do objeto *OIDx* nestes contextos.

Como exemplo, na figura 2.6 (capítulo 2), suponha que o usuário deseje modificar a versão lógica de um objeto pertencente ao contexto *CTR2* do espaço de trabalho *ET1* e que este objeto define o relacionamento de dependência com o contexto *CTRa* do espaço de trabalho *ET2*. Para isto, o usuário deve ter permissão de acesso para alterar ambas as versões lógicas, já que existe um relacionamento de dependência entre elas.

No caso de eliminação, como eliminação de uma versão lógica implica em alteração do contexto ao qual ela pertence, o usuário deve ter permissão de acesso (escrita) sobre ele.

4.5.3 Comparação de versões de objetos tanto em CET como em CTR

Duas versões lógicas de objetos *v* e *v'* são consideradas fortemente (fracamente) iguais no nível *n* do grafo de composição se os valores fortes (fracos) em profundidade são iguais até o nível *n* [delVal97], sendo que:

- *Objetos idênticos*: Caso em que os identificadores dos objetos são os mesmos.
- *Igualdade em superfície*: Caso em que os valores dos objetos são iguais. Esta igualdade é forte, pois dois objetos compartilham o mesmo grafo de composição do objeto, com exceção da raiz.
- *Igualdade em profundidade*: Caso em que os valores em profundidade são iguais. Um valor em profundidade pode ser obtido pela substituição recursiva de todas as referências a um objeto pelo seu valor. Esta igualdade é considerada fraca, pois os dois objetos compartilham a estrutura do grafo de composição e os valores das suas folhas.

Igualdade em identidade → Igualdade em superfície → Igualdade em profundidade, sendo → indicador de implicação.

O valor forte em profundidade de uma versão lógica é o grafo de composição sem a raiz. O valor fraco de um objeto em profundidade é obtido eliminando os identificadores de objetos e considerando somente a estrutura do grafo de composição. No modelo de [delVal97] são definidos dois critérios de igualdade em DBV:

- *Igualdade forte (fraca) em superfície*: Equivalente à igualdade forte (fraca) no nível 1.
- *Igualdade forte (fraca) em profundidade*: Equivalente à igualdade forte (fraca) no nível ∞.

Para questões de comparação de versões de objetos tanto em CET como em CTR, basta que existam regras (*s, o, v, m*) e (*s, C, m*) na base de regras que forneçam ao usuário permissão de acesso (leitura) às versões dos objetos em questão (algoritmo 3.4). No caso de objetos complexos, não há a necessidade da existência de regras para todos os objetos do grafo de composição. Basta que exista uma regra de autorização sobre a

versão lógica raiz do objeto complexo para que o usuário tenha acesso ao objeto como um todo.

4.5.4 Cópias de versões de objetos tanto em CET como em CTR

Para a cópia de objetos versionáveis podem existir três possibilidades [delVal97]:

1. Cópia de uma versão lógica de um objeto em uma DBV (*o, d*) para a versão de outro objeto (*o', d*) na mesma DBV.
2. Cópia de uma versão lógica de um objeto em uma DBV (*o, d*) para a versão lógica do mesmo objeto em outra DBV (*o, d'*).
3. Cópia de ma versão lógica de um objeto em uma DBV (*o, d*) para a versão lógica de outro objeto em outra DBV (*o', d'*).

São definidas as operação de:

- *Cópia de valor*: O valor do estado de uma versão é copiado para a outra versão. Esta operação garante igualdade forte em profundidade para o caso 1 e igualdade forte em superfície para os casos 2 e 3.
- *Cópia de valor profunda*: A estrutura de composição e valores são replicados a partir da criação de novos objetos na DBV destino da cópia. Garante igualdade fraca em profundidade em todos os seus casos.
- *Cópia forte profunda*: Corresponde a replicar a estrutura de composição e valores das versões, mas sobre os mesmos objetos do objeto fonte. Garante igualdade forte em profundidade para os casos 2 e 3. É equivalente à cópia por valor no caso 1.

Para qualquer tipo de cópia, o usuário deve ter permissão de acesso para:

1. Ler a versão lógica do objeto origem. Isto significa, executar ALG 1, sendo *D1* o contexto (origem) fonte, *OID1* o objeto (origem) alvo e modo de acesso *leitura*. Se Output = NEGADO, negar o pedido de cópia.
2. Modificar o estado da versão lógica do objeto destino. Isto significa executar ALG 1, sendo *D2* o contexto (destino) fonte, *OID2* o objeto (destino) alvo e modo de acesso *escrita*. Se Output = NEGADO, negar o pedido de cópia.

Assim como acontece com a comparação de objetos, para que o usuário leia ou modifique versões de objetos complexos, basta que exista uma regra de autorização sobre a versão lógica raiz do objeto complexo para autorizar o usuário a efetuar a operação.

4.5.5 Relacionamento de dependência entre versões de objetos em CET ou CTR diferentes

Dizemos que um contexto *Dx* tem um relacionamento de dependência com o contexto *Dy* se uma ou mais versões lógicas de *Dx* possuir um relacionamento de dependência

com uma ou mais versões lógicas de *Dy*. O relacionamento de dependência entre contextos foi detalhado na seção 4.4.2.

Na figura 4.2 é possível observar um relacionamento de dependência entre as versões dos objetos *A* nas DBV 0.1 e 0.1.3 e entre as versões do objeto *B* nas DBV 0.1.2.1 e 0.1.1.2. Estes contextos podem representar tanto um CET quanto um CTR. Ou seja, um relacionamento de dependência entre versões do objeto *A* em duas DBV diferentes faz com que exista um relacionamento de dependência entre as DBV. O mesmo acontece com o objeto *B*, definindo um relacionamento de dependência entre os contextos 0.1.2.1 e 0.1.1.2.

4.6 Conclusão

Este capítulo apresentou as operação do modelo de versões em bancos de dados para SIGs de [delVal97] e como o modelo de controle de acesso da dissertação se aplica em cada operação. Como o modelo de versões em banco de dados para SIGs tem como base o modelo de Versões em Bancos de Dados (*Database Versions (DBV)*, muitas de suas operações são mapeadas para operações do modelo DBV. O capítulo apresentou os algoritmos correspondentes de [delVal97] para cada operação e como se deve proceder à verificação de autorização para cada uma delas.

Capítulo 5

Conclusões e extensões

5.1 Contribuições

Esta dissertação apresentou um modelo de controle de acesso para bancos de dados geográficos multiversão, estendendo o modelo de [Sasaoka02] para suportar um mecanismo de versões dos dados armazenados no banco de dados. O modelo proposto define os objetos a serem acessados, os tipos de acesso permitidos a estes objetos e quem pode acessá-los. O mecanismo de versões utilizado como referência foi proposto por [delVal97] e, assim como [Sasaoka02], considera, dentre outras, a questão do trabalho cooperativo.

O capítulo 2 apresentou os conceitos básicos sobre controle de acesso e versões, destacando a aplicação destes em bancos de dados para SIG. Alguns modelos de autorização existentes foram apresentados e seus aspectos mais importantes foram detalhados. Também foram apresentadas características do uso de versões em sistemas CAD/CASE e bancos de dados. Os principais aspectos relacionados ao uso de versões são o suporte ao trabalho de projeto e o suporte a múltiplas representações. Ênfase especial foi dada aos trabalhos de [Sasaoka02] e [delVal97] por serem os trabalhos utilizados como base para o desenvolvimento desta dissertação.

O capítulo 3 apresentou o modelo de controle de acesso para bancos de dados geográficos multiversão, baseado na definição de regras de autorização $< s, o, v, m >$. Os principais pontos abordados pelo modelo foram:

- Estruturas para representar a autorização.
- Determinação do sujeito, objeto e modos de acesso.
- Política para gerenciar e armazenar autorizações.
- Algoritmos para atualização da base de regras e análise de pedidos de acesso.

O capítulo 4 apresentou as operações do modelo de versões em bancos de dados para SIGs [delVal97], mostrando como a proposta da dissertação foi adotada em cada uma das operações. Elas são classificadas em: operações sobre Contextos Espaço Temporais (CET); operações sobre Espaços de Trabalho (ET); operações sobre Contextos de Trabalho (CTR); e operações sobre objetos e versões de objetos. Como o modelo de versões tem como base o modelo DBV, muitas de suas operações são mapeadas para o modelo DBV.

As principais contribuições desta dissertação foram:

- Proposta de um modelo de controle de acesso para bancos de dados geográficos com suporte a versionamento dos dados.
- Aplicação do mecanismo de controle de acesso em um modelo de versões em banco de dados para SIGs.

5.2 Extensões

Como futuras extensões a esta dissertação podem ser destacadas as seguintes:

- **Controle de acesso a atributos, métodos e funções de objetos**: Algumas aplicações necessitam restringir o acesso a partes de objetos e não apenas a objetos como um todo. O modelo de [delVal97] define que atributos de um objeto podem ser não versionáveis. Isto pode requerer um controle de acesso cuja granularidade não seja o objeto como um todo, mas sim os seus atributos. O modelo de [GOGF93] define regras de autorização sobre atributos, métodos e objetos para um banco de dados orientado a objetos. Estes fatores indicam a necessidade de se ter um controle de acesso sobre partes de objetos e não apenas objetos como um todo.

- **Mecanismo de rastreabilidade de acesso**: Por questões de segurança, pode ser necessário saber a quais objetos um sujeito requisitou acesso e de que modo este acesso foi solicitado. Neste contexto, o controle de acesso deve prover um mecanismo de LOG sobre todas as validações de requisição de acesso realizadas. Este LOG pode ser usado para verificar tentativas recorrentes de pedidos de acesso de um sujeito sobre objetos não autorizados.

- **Controle de acesso seletivo**: O modelo proposto não se baseia totalmente no controle de acesso seletivo, porque não tem uma administração de autorizações descentralizada. Para isto, é necessário adicionar itens à tupla $< s, o, v, m >$ para permitir concessão e revogação de autorizações não centralizadas. Neste caso, deve ser possível saber, por exemplo, quem concedeu a autorização e se o usuário tem a opção de repassar esta autorização para outros usuários.

- **Controle de acesso com prioridades para sujeitos**: Diferentes usuários podem tentar acessar (no modo *escrita*) uma mesma versão de objeto no mesmo instante de tempo. A classificação de sujeitos por níveis de prioridade determina hierarquia de acesso. Neste caso, o conflito pode ser resolvido dando acesso ao sujeito de maior prioridade e negando ao outro.

- **Controle de acesso espaço-temporal**: As autorizações do banco de dados geográfico multiversão podem estar associadas a um intervalo de tempo. Por exemplo, um sujeito *s1* pode ter acesso a uma versão de objeto $\{o1, v1\}$ no modo *escrita* das 08h00 às 18h00. A tupla $< s, o, v, m >$ deve ser alterada para conter o intervalo de tempo em que ela é valida. Os algoritmos dos capítulos 3 e 4 também devem ser alterados para verificar o intervalo temporal. No capítulo 2 foram apresentados estudos sobre controle de acesso temporal e devem ser levados em consideração na elaboração do controle de acesso espaço-temporal.

- **Suporte a autorizações negativas**: Esta questão gera várias ramificações, pois uma autorização negativa sempre é mais difícil de verificar e manter, razão pela qual não foi considerada no modelo de [Sasaoka02]. Aqui, isto é dificultado pela existência de múltiplas versões. No entanto, seria interessante considerar tal extensão, já que em um determinado espaço geográfico pode-se querer restringir o acesso a apenas alguns objetos nele contidos.

- **Autorizações por intervalo de versões**: Permitir autorizações a um intervalo de versões, ao invés de a uma versão ou todas as versões que satisfaçam a uma consulta. Esta extensão poderia inicialmente ser tratada dentro do caso em que a consulta *C* definiria o intervalo, mas para tanto seria necessário considerar aspectos de linguagem de consulta a versões.

- **Multiversionamento no problema de autorização**: Estudar o impacto do multiversionamento no problema de autorização, tendo em vista que autorizações sobre uma escala podem afetar objetos em outra escala. Por exemplo, o polígono "Campinas" é atravessado pela Rodovia Dom Pedro e, nesta escala, a autorização sobre o polígono pode trazer consigo autorização sobre a rodovia. Em outra escala, "Campinas" seria um ponto e, portanto, a rodovia não seria considerada. Estes aspectos de interação entre contextos devem ser melhor estudados para evitar efeitos colaterais no controle de acesso.

Referências Bibliográficas

[AACSV03] S.Agarwal, G. Arun, R. Chatterjee, B. Speckhard, and R.Vasudevan, "Long Transactions in an RDBMS", GITA Annual Conference, San Antonio, March, 2003.

[AJ96] P. Ammann and S. Jajodia. Ensuring atomicity of multilevel transactions. *IEEE Symposium on Security and Privacy*, 1996.

[BANE87] Banerjee J. et al., "Data Model Issues for Object-Oriented Applications", ACM Transaction on Office Information Systems, Vol 5, #1, April 1987, pp. 3-26.

[BBCDN04] A. Belussi, E. Bertino, B. Catania, M. L. Damiani and A. Nucita. An Authorization Model for Geographical Maps. Proceedings ACM GIS, p.82-91, 2004.

[BBFS96a] E. Bertino, C. Bettini, E. Ferrari, and P. Samarati. Supporting periodic authorization and temporal reasoning in database access control. *Proc. 22nd Int. Conference on VLDB*, páginas 472-483, 1996.

[BBFS96b] E. Bertino, C. Bettini, E. Ferrari, and P. Samarati. A temporal access control mechanism for database systems. *IEEE Trans. Knowledge and Data Eng.*, 8(1):67-80, 1996.

[BBFS97] E. Bertino, C. Bettini, E. Ferrari, and P. Samarati. Decentralized administration for a temporal access control model. *Information Systems*, 22(4):223-248, 1997.

[BCB03] A. Belussi, B. Catania, and E. Bertino. A Reference Framework for Integrating Multiple Representations of Geographical Maps. *In Proc. of ACM GIS*, 2003.

[BCDP05] E. Bertino , B. Catania , ML Damiani , P Perlasca, GEO-RBAC: a spatially aware RBAC. Proceedings ACM SIGSAC, P.29-37, 2005.

[BCJ92] MJ. Blin, W. Cellary and G. Jomier. A model of configurations for hardware/software system deliveries. *In Proc. 5th Int. Conf. on Software Engineering and its Applications, Toulouse, France*, pp. 228-347, December 1992.

[BDPSN96] Ahmad Baraani-Dastjerdi, J. Pieprzyk, and Raihaneh Safavi-Naini. Security in databases: A survey of study. February: 1-39, 1996. *http://citeseer.nj.nec.com/baraani-dastjerdi96security.html*

[BEKMW04] B. Babel, J. Eder, C. Koncilia, T. Morzy, and R. Wrembel. Creation and management of versions in multiversion data warehouse. Proceedings of the 2004 ACM symposium on Applied computing, P 717-723, 2004.

[BHEA00] E. Bertino, M. A. Hammad, W. G. Aref, and A. K. Elmagarmid. An access control model for video database systems. In CIKM, páginas 336-343, 2000.

[BP76] D. E. Bell and L. J. La Padula. Secure computer systems. Unified exposition and multics interpretation. Technical report, The Mitre Corp, 1976.

[CAASV04] R. Chatterjee, G. Arun, S. argawal, B. Speckhard, and R. Vasudvan. Using Applications of Data Versioning in Database Application Development. Proceedings of the 26th International Conference on Software Engineering, P 315-325, 2004.

[CCH96] G. Câmara, M. A. Casanova, A. S. Hemerly, G. C. Magalhães and C. M. B. Medeiros. *Anantomia de Sistemas de Informação Geográfica*. 10ª Escoa de Computação, 1996.

[CJ90] W. Cellary e G. Jomier. Consistency of Versions in Object-Oriented Databases. *In Proceedings International VLDB Conference, Brisbane, Australia*, pp. 432-441, Agosto de 1990.

[CK86] H. Chou e W. Kim. A unifying framework for versions in a CAD environment. In *Proceedings International VLDB Conference*, pp. 336-344, agosto de 1986.

[CW98] R. Conradi, and B. Westfechtel, "Version Models for Software Configuration Management", ACM Computing Surveys, Vol. 30, No. 2, June 1998.

[delVal97] L. M. del Val Cura. Tratamento de versões em bancos de dados para Sistemas de Informações Geográficas. Dissertação de Mestrado, Universidade Estadual de Campinas, 1997.

[Den83] D. E. Denning. *Cryptography and Data Security*. Addison-Wesley Publishing Company, 1983.

[DSCP01] Ernesto Damiani, P. Samarati, S. De Capitani and S. Paraboshi. Controlling Access to XML Documents. *IEEE Internet Computing*, December:18-28, 2001.

[EBFS94] E. Guides, E. B. Fernandez and H. Song. A model for evaluation and administration of security in object-oriented databases. *IEEE Trans. Knowledge and Data Eng.*, 6(2):275-292, 1994.

[EBS95] S. Jajodia, E. Bertino, and P. Samarati. Database security – research and prctice. *Information Systems*, 20(7):537-556, 1995.

[EGSV98] M Erwig, RH Güting, M Schneider, M Vazirgiannis. Abstract and Discrete Modeling of Spatio-Temporal Data Types, Proceedings ACM GIS, p.131-136, 1998.

[EH90] M. Engenhofer and J. Herring. A mathematical framework for the definition of topological relationships. *Proceedings of the 4th International Symposium on Spatial Data Handling*, páginas 803-813, 1990.

[EN97] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems*. Addilson-Wesley, third edition, 1997.

[EV94] J. Ebert and G. Vossen. Object Configurations in software engineering databases. Technical Report N515/94-I, University of Munster, August 1994.

[Feu93] M. Feutchwanger. *Towards a Geographic Semantic Database Model*. Tese de Doutorado, Simon Fraser University, 1993.

[FK92] D.Ferraiolo and Richard Kuhn. Role-Based Access Control. *Proceedings of 15th National Computer Security Conference*, 1992.

[Fre75] J. Freeman. The modeling of spatial relations. *Computer Graphics and Image Processing*, 4:156-171, 1975.

[GJ94] S. Gançarski e G. Jomier. Managing entity versions within their contexts: A formal approach. In *Proc. International DEXA Conference*, volume 856 de *Lecture Notes in Computer Sciences,* pp. 400-409, 1994.

[GJ96] S. Gançarski e G. Jomier. Vers um langage de manipulation pour bases de données multiversions. In *Proceedings 12e jounées Bases de Données Avancées*, *Cassis*, *France*, 1996.

[GJZ95] S. Gançarski, G. Jomier and M. Zamfiroiu. A Framework for the Manipulation of a Multiversion Database. *In Proceedings DEXA International Conference, Workshop*, pages 247-256, 1995.

[GOGF93] N. Gal-Oz, E. Gudes and E. B. Fernandez. A Model of Methods Access Authorization in Object-oriented Databases. *In Proceedings of the 19th VLDB Conference*, 1993.

[GSdS95] L. Golendziner e C. Saraiva dos Santos. Uma abordagem multi-nível para suporte a versões em banco de dados orientados a objetos. *In PANEL'95 – XXI Conferencia Latino Americana de Informática*, pp. 1127-1183, Setembro de 1995.

[Gut94] R. H. Gutting. An Introduction to Spatial Database Systems. *The VLDB Journal*, 3(4):357-400, 1994.

[GW76] P. G. Griffiths and B. Wade. An authorization mechanism for a relational database system. *ACM TODS*, 1(3): 243-255, 1976.

[Kat90] R. Kats. Toward a unified framework for version modeling in engineering databases. *ACM Computing Surveys*, 22(4):375-408, Dezembro de 1990.

[KPSN96] S. Jajodia, K. P. Smith, B. T. Blaustein and L. Notargiacomo. Correctness criteria for multilevel secure transactions. *IEEE Trans. Knowledge and Data Eg*., 8(1):32-44, 1996.

[MBT89] M. B. Thuraisingham. Mandatory Security in Object-Oriented Database Systems. *In Proceeding ACM OOPSLA*, pp. 203-210, 1989.

[PMJV04] M. A. Peerbocus, C. B. Medeiros, G. Jomier and A. Voisard. A System for Change Documentation Based on a Spatiotemporal Database. GeoInformatica, Volume 8, Número 2, páginas 173-204, 2004.

[RBKW91] F. Rabitti, E. Bertino, W. Kim, and D.Woelk. Supporting multiple access control policies in database systems. ACM Transactions on Database Systems, 16(1):88–131, 1991.

[SBJ96] P. Samarati, E. Bertino and S. Jajodia. An authorization model for a distributed hypertext system. *IEEE Trans Knowledge and Data Eng*., 8(4):555-562, 1996.

[Sasaoka02] L. K. Sasaoka. Controle de Acesso em Bancos de Dados Geográficos. Dissertação de Mestrado, Universidade Estadual de Campinas, 2002.

[TOC93] G. Talens, C. Oussalah and M. Colinas. Versions of simple and composite objects. *In Proc. International VLDB Conference*, pp. 62-72, 1993.

[TS96] R. K. Thomas and R. S. Sandhu. A trusted subject architecture for multilevel secure object-oriented databases. *IEEE Trans. Knowledge and Data Eng.*, 8(1):16-30, 1996.