Tratamento de versões em bancos de dados para Sistemas de Informações Geográficas

Luis Mariano del Val Cura

Dissertação de Mestrado

Instituto de Computação
Universidade Estadual de Campinas

Tratamento de versões em bancos de dados para Sistemas de Informações Geográficas

Luis Mariano del Val Cura[1]

Março 1997

Banca Examinadora:

- Prof. Dra. Claudia M. Bauzer Medeiros (Orientadora)
  Instituto de Computação - UNICAMP
- Prof. Dra. Ana Carolina Salgado
  Departamento de Informática - UFPE
- Prof. Dr. Luiz Eduardo Buzato
  Instituto de Computação - UNICAMP
- Prof. Dr. Neucimar Jerônimo Leite (Suplente)
  Instituto de Computação - UNICAMP

Dissertação submetida ao Instituto de Computação da
Universidade Estadual de Campinas, como requisito parcial para
a obtenção do título de Mestre em Ciência da Computação

[1]Esta dissertação foi desenvolvida dentro do projeto do CNPq PROTEM/CC-GEOTEC e financiada parcialmente pelo CNPq e pela FAPESP.

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA DO IMECC DA UNICAMP

Val Cura, Luis Mariano del

V23t

Tratamento de versões em bancos de dados para SIG / Luis Mariano del Val Cura – Campinas,[S.P.: s.n.],1997

Orientador:Claudia M.Bauzer Medeiros

Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Computação.

1.Bancos de dados. 2. Bancos de dados orientado a objetos. 3. Sistemas de informações geográficas. I. Medeiros, Claudia Maria Bauzer. II. Universidade Estadual de Campinas. Instituto de Computação. III. Título

Tratamento de versões em bancos de dados para Sistemas de Informações Geográficas

Este exemplar corresponde à redação final da tese devidamente corrigida e defendida pelo Sr. Luis Mariano del Val Cura e aprovada pela Comissão Julgadora.

Campinas, 20 de março de 1997.

Profa. Claudia M. Bauzer Medeiros
Orientadora

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

Tese de Mestrado defendida e aprovada em 05 de março de 1997 pela Banca Examinadora composta pelos Professores Doutores

Profª. Drª. Ana Carolina Salgado

Profº. Drº. Luíz Eduardo Buzato

Profª. Drª. Cláudia Maria Bauzer de Medeiros

A la memoria de Carlos

Prefácio

Sistemas de Informação têm introduzido modelos e mecanismos de versões para a manutenção de múltiplos estados ou variações das entidades modeladas. As aplicações fundamentais de versões estão associadas à manutenção de alternativas de projetos em sistemas CAD/CASE e na representação histórica de entidades em sistemas temporais.

Esta dissertação apresenta um estudo sobre as aplicações de versões em SIG enfatizando aspectos relativos a aplicações temporais, manipulação de múltiplas representações para as entidades espaciais modeladas e facilidades para manutenção de alternativas de projeto espacial. Os principais resultados são: a proposta de um modelo e mecanismo de versões com recursos básicos para dar apoio a estas aplicações; e uma proposta de extensão a um SGBDOO padrão para permitir a inclusão dos recursos do modelo.

Abstract

Information systems contemplate version models and mechanisms for the management of multiple states of modeled entities. Versions are associated, mainly, to the management of alternatives in CAD/CASE systems and the representation of historical evolution of entities in temporal systems.

This dissertation studies the use of versions in Geographic Information Systems (GIS). The focus of this work is on temporal applications, multiple representations of spatial entities, and the management of alternatives of spatial design. The main results presented are: a model and a mechanism for versions in order to support geographic applications; and the proposal of an extension to a standard OODBMS to support the model.

Agradecimentos

À professora Claudia Bauzer Medeiros, pela sua orientação e interesse no decorrer da tese.

A Marta pela sua paciência e apoio constantes.

A Nevton Cereja e Juliano Lópes de Oliveira pela amizade demonstrada nestes anos e pelas valiosas discussões técnicas.

Aos colegas do grupo de Bancos de Dados nestes dois anos (Fatima, Mariano, Cristina, Ricardo, Marcos Andre, Marcio, Glaucia, Laura, Marcos Antonio, Alexandre, Walter, Sergio).

Aos colegas do mestrado e doutorado em Computação, especialmente da turma 95.

Aos professores e funcionários do Instituto de Computação, pela formação e ajuda recebidas.

Ao CNPq e a FAPESP, ao povo brasileiro, pelo apoio financeiro recebido que permitiu a realização deste projeto.

À minha família, pelo apoio e estímulo.

Aos meus colegas da Universidad de La Habana que realizam o meu trabalho enquanto eu continuo meus estudos.

Conteúdo

Prefácio vi

Abstract vii

Agradecimentos viii

1 Introdução 1

2 Revisão Bibliográfica 5

2.1 Versões: Conceitos gerais e propostas 5

2.1.1 Relacionamentos entre versões de um mesmo objeto 5

2.1.2 Relacionamento entre versões de objetos diferentes. 7

2.1.3 Consistência em um Banco de Dados Multiversão 9

2.2 Versões em sistemas CAD/CASE 15

2.2.1 Suporte ao trabalho cooperativo 16

2.3 Versões em Sistemas Temporais 17

2.3.1 Conceitos de Bancos de Dados Temporais 18

2.3.2 Versionamento Temporal 21

2.4 Resumo 22

3 Versões em Sistemas de Informações Geográficas 24

3.1 Sistemas de Informações Geográficas 24

3.1.1 Arquiteturas e funcionalidade nos SIG 25

3.1.2 Modelos de Dados em SIG 26

3.1.3 Modelo de quatro níveis 28

3.2 Versionamento em SIG 30

3.3 Versões como suporte ao projeto espacial 31

3.3.1 O Sistema SAGRE 33

3.3.2 Requisitos de versões para projetos em SIG 33

3.4 Versionamento como solução de múltiplas representações 34

3.4.1 O problema das múltiplas representações em SIG 34
3.4.2 Soluções para múltiplas representações 39
3.4.3 Versões como solução a múltiplas representações em SIG 40
3.4.4 Requisitos de múltiplas representações. 40
3.5 Versionamento temporal em SIG 41
3.5.1 Mudanças geográficas no tempo 41
3.5.2 Consultas temporais em SIG 42
3.5.3 Modelo de evolução espaço–temporal por versões 43
3.5.4 O trabalho de Botelho 44
3.5.5 Proposta de evolução temporal por eventos 44
3.5.6 Requisitos de versões para a modelagem temporal em SIG 51
3.6 Resumo 51

4 Modelo de Versões Proposto **52**
4.1 Características do modelo proposto 52
4.1.1 Objetos versionáveis 52
4.1.2 Contextos Espaço-Temporais (CET) 53
4.1.3 Relacionamentos entre CET diferentes 55
4.1.4 Relacionamentos de versões de objetos em CET diferentes 55
4.1.5 Comportamento das versões de objetos em CET diferentes 56
4.1.6 Espaços de Trabalho 56
4.1.7 Relacionamentos de dependência entre contextos 57
4.1.8 Consultas sobre contextos 58
4.1.9 Formalização do modelo 58
4.1.10 Operações sobre o modelo 60
4.1.11 Ciclo de trabalho no modelo 61
4.2 O mecanismo de versões subjacente 62
4.2.1 O mecanismo DBV 63
4.2.2 Operações básicas do mecanismo DBV 66
4.3 Mapeamento do modelo proposto para o mecanismo DBV 69
4.3.1 Versionamento seletivo 69
4.3.2 Contextos Espaço Temporais 71
4.3.3 Espaços e Contextos de Trabalho 74
4.3.4 Relacionamentos de dependência entre contextos 76
4.3.5 Operações sobre objetos e versões de objetos 79
4.4 Resumo 80

5 O modelo de versões em um SGBDOO 81
5.1 Componentes do modelo 81
5.1.1 Linguagem de definição de objetos (ODL) estendida 82
5.1.2 Classes e objetos predefinidos 83
5.1.3 Linguagem de aplicações estendida 89
5.1.4 Linguagem de consulta (OQL) estendida 92
5.1.5 Arquitetura do modelo. 94
5.2 Uso do mecanismo proposto em SIG 94
5.2.1 Modelagem de múltiplas representações sobre o modelo de versões . 95
5.2.2 Modelagem temporal sobre o modelo de versões 99
5.3 Resumo 105

6 Conclusões e Extensões 106

Referências Bibliográficas 109

Lista de Figuras

1.1 Taxonomia no uso de versões 3

2.1 Tipos de Versões 6
2.2 Tipos de Configurações 8
2.3 Consistência Parcial 11
2.4 Propagação do versionamento 12
2.5 Configurações intermediárias 13
2.6 Operações de Check-in / Check-out 18

3.1 Evolução de escala 36
3.2 Transformações de detalhe 38
3.3 Topologia espaço temporal 43
3.4 Processos de mudanças básicas 46
3.5 Processos de transformação geométrica 47
3.6 Processos de movimento 47
3.7 Processos de Substituição 49
3.8 Processos de difusão 49
3.9 Processos de evolução da estrutura espacial. 50

4.1 O Modelo Proposto 62
4.2 O Mecanismo DBV 64
4.3 Nova estrutura dos objetos 70
4.4 Contextos Espaço Temporais e Relacionamentos 71
4.5 Modificação do estado de uma versão lógica 77
4.6 Dependências entre versões 78

5.1 Arquitetura ODMG 82
5.2 Classes predefinidas do modelo 89
5.3 Arquitetura sobre um *Object Engine* Estendido 94
5.4 Arquitetura sobre o SGBDOO convencional 95

Capítulo 1

Introdução

Esta dissertação apresenta o estudo do uso de versões em SIG, e propõe um modelo e um mecanismo de versões para serem utilizados neste tipo de sistemas.

Um Sistema de Informações Geográficas (SIG) é um software que gerencia grandes volumes de dados georeferenciados, isto é, ligados à superfície terrestre. O usuário relaciona e combina estes dados através de funções de análise e processamento espacial para garantir os objetivos de sua aplicação.

Com estes sistemas, um amplo conjunto de aplicações baseadas na dimensão espacial pode ser desenvolvido. Estas aplicações incluem, dentre outros, o gerenciamento, consulta e planejamento de tarefas em que componentes geográficas e relacionamentos espaciais são fundamentais.

A manipulação de dados por parte de um SIG típico apresenta atualmente, dentre outras, as seguintes limitações:

- Não considera a variação temporal do mundo.
- Não permite a possibilidade de múltiplas representações espaciais.
- Não suporta o gerenciamento automático de alternativas da realidade geográfica modelada.

Tanto do ponto de vista de pesquisa como de operações do usuário, estes três aspectos em geral são abordados de forma independente, dadas as especificidades semânticas de cada um deles. No entanto, neles está implícita a necessidade da manutenção de múltiplos estados ou versões de uma mesma entidade modelada.

Um sistema de informação com possibilidades para versionamento é aquele no qual podem ser mantidos múltiplos estados ou variações das entidades modeladas. O uso de versões é encontrado em diversos contextos. Inicialmente, surgiu associado basicamente a aplicações em sistemas CAD [CK86, BK85, DL88, Kat90, TOC93] visando a manutenção

das múltiplas alternativas de projetos desenvolvidos por vários usuários. Estes modelos foram sintetizados e generalizados por [Kat90], propondo uma terminologia atualmente utilizada na área. De forma similar, versões têm sido amplamente utilizadas em sistemas CASE para a manutenção de múltiplas configurações de sistemas de software [DGL87, EV94, HK87, SS94, Vic90, BCJ92] envolvendo problemas similares aos encontrados na área de CAD. Necessariamente, muitos dos mecanismos de versões desenvolvidos nesse contexto refletem características da semântica desses sistemas e em geral não possuem uma abordagem de integração a um sistema de gerenciamento de bancos de dados(SGBD), isto é, o versionamento aparece como funcionalidade *ad hoc*.

Ainda outro contexto de uso de versões é o de *dados temporais*. Sistemas de bancos de dados temporais [Sno90, JJE+94, ATSS94, TK96] têm utilizado formas de manipulação de versões. Estas são, no entanto, orientadas à manutenção dos diferentes estados temporais das entidades, que geralmente são organizados de forma linear.

Mais recentemente, os sistemas de bancos de dados orientados a objeto têm incorporado facilidades para o gerenciamento de versões [BB91, Gol93, BH89, Zdo86, KBC+89, CJ90, Sci91b, CS90]. Alguns modelos adotados por estes sistemas foram influenciados pelas propostas desenvolvidas na área de CAD [KBC+89]. Outros modelos de versões como o proposto em [CJ90] para o sistema O_2 ou em [Sci91b] tentam definir propostas gerais que sejam igualmente aplicáveis a sistemas temporais, CAD ou CASE e ainda outras aplicações.

Em quaisquer dos casos mencionados o uso de versões tem associado a si o problema de definição e manutenção de um critério de consistência dos dados.

O uso de versões em SIG é ainda pouco encontrado. No entanto, vários dos problemas existentes – variação temporal, múltiplas representações, alternativas – seriam solucionados com versões. Neste sentido, devem ser consideradas as especificidades de cada um destes problemas. Em particular, o versionamento temporal em SIG é em geral associado a uma ordem linear entre as versões de acordo com a unidade de tempo utilizada e inclui o problema da variação das estruturas e representações geométricas e dos relacionamentos topológicos no tempo.

No caso de múltiplas representações, o problema a resolver está centrado no armazenamento simultâneo de diferentes representações e na funcionalidade diferente das operações sobre cada uma delas. Em mais detalhes, uma mesma entidade geográfica pode ser representada (e armazenada) usando diversas estruturas e pode ser percebida por diferentes usuários de acordo com critérios de apresentação, visualização e funcionalidade diferentes.

Finalmente, quanto ao terceiro aspecto, a introdução de facilidades para projeto em SIG permitiria manipular simultaneamente diferentes cenários ou variantes da realidade geográfica modelada. Estas facilidades, muito relacionadas com as apresentadas em CAD, devem considerar o trabalho de natureza cooperativa, facilidades para trabalho individual

e de equipe, critérios de organização das diferentes variantes criadas, dentre outras.

Alguns autores têm considerado o uso de versões em SIG. [NTE92, Lan89, RYG94, Peu94] estudam os problemas da modelagem espaço temporal sob diferentes enfoques do ponto de vista geográfico, enquanto em [Lan93b, Lan93a, CT95, Bot95] algumas soluções são apresentadas sobre como utilizar versões nestes casos. [CFS+94, CCH+96] apresentam os problemas de múltiplas representações em SIG em um nível mais conceitual e [MJ93b, MJ93a, MBJ96] apresentam possíveis soluções de implementação a partir do uso de versões. Finalmente [Bat92, VFMa95] apresentam o uso de versões em ambientes SIG com finalidades de projeto, com características muito em comum com as apresentadas em ambientes CAD e CASE.

Em geral o uso de versões em sistemas computacionais tem sido abordado em dois níveis [BH89, Sci94]:

- *Nível do usuário:* O usuário e suas aplicações criam e mantêm as versões e possuem ferramentas para sua manipulação. As aplicações incluem o uso de versões como parte de sua semântica.
- *Nível do sistema:* O versionamento faz parte do sistema computacional. Para o usuário e suas aplicações, o uso de versões é transparente. As versões são criadas e mantidas pelo próprio sistema.

Gançarski [Gan94] refere-se a estes níveis como versionamento aparente e transparente respectivamente.

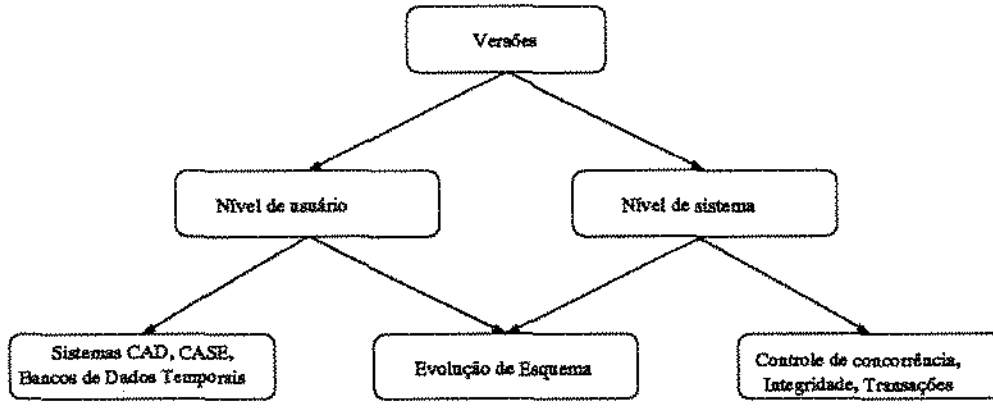


Figura 1.1: Taxonomia no uso de versões

Sobre estes níveis, são descritos três usos fundamentais de versões (Figura 1.1):

- Utilização em sistemas nos quais se deseja a manutenção e manipulação explícita, pelos usuários, de múltiplos estados e variações de objetos que modelam entidades do mundo real (ou abstrações deste). Aqui estão incluídos os sistemas CAD, CASE, sistemas para automação de escritórios, sistemas de bancos de dados temporais. Neste caso as versões estão no nível do usuário.

- Uso como mecanismo para a implementação de mudanças dinâmicas de esquemas em bancos de dados. Neste caso, versões são encontradas tanto para a modificação do esquema de uma relação ou classe quanto para a migração de objetos entre relações ou classes. Aqui o uso de versões ocorre simultaneamente no nível de usuário e de sistema.

- Uso em SGBD, como mecanismo para manter a integridade dos dados em casos de transações longas ou descartadas (*RollBack*), em recuperações devidas a falhas e também como mecanismo para controle e implementação de concorrência. Neste caso o uso de versões é no nível de sistema.

Esta dissertação está centrada no uso de versões no nível de usuário, com especial ênfase na manipulação da multiplicidade de estados de objetos espaciais próprios da modelagem em SIG. O restante do texto está organizado da seguinte forma. O capítulo 2 apresenta os principais conceitos e problemas de bancos de dados temporais e da manipulação de versões. O capítulo 3 discute os aspectos em SIG que podem ser facilitados com o uso de versões, definindo um conjunto de requisitos para um modelo de versões voltado a SIG. A seguir, o capítulo 4 introduz o modelo de versões proposto, incluindo a sua funcionalidade, e descrevendo o mecanismo utilizado para suportar o modelo. O capítulo 5 propõe extensões a um SGBDOO padrão para o suporte do modelo de versões. Finalmente, o capítulo 6 apresenta as conclusões e extensões do trabalho.

Capítulo 2

Revisão Bibliográfica

Neste capítulo será apresentada a revisão bibliográfica realizada. Esta revisão esteve centrada nas diferentes abordagens de versões com seus problemas fundamentais e também em Sistemas Temporais onde versões são utilizadas como forma de implementação.

2.1 Versões: Conceitos gerais e propostas

Os objetos modelados em sistemas com versionamento representam em geral entidades bem definidas e com identidade própria no universo da aplicação. Nas diferentes abordagens, estes objetos conceituais são chamados de Objetos Multiversão [CJ90], Molecular [BK85], de Projeto [Kat90], Versionável [GSdS95] dentre outros. Aqui será usado o termo **Objeto Versionável**. Para cada um destes objetos existe seu **Conjunto de Versões** os quais podem ser vistos por sua vez como objetos que descrevem variações do objeto conceitual. Aqui se faz diferença entre o objeto conceitual modelado e as diferentes versões existentes para ele.

Bancos de dados que contêm múltiplas alternativas dos objetos modelados são chamados de **Bancos de Dados Multiversão** [CJ90].

2.1.1 Relacionamentos entre versões de um mesmo objeto

As versões de um mesmo objeto versionável podem se organizar e relacionar de múltiplas formas. No entanto, o relacionamento mais usual em mecanismos de versões é a *derivação*. No momento da sua criação, uma versão obtém estrutura e valores de outras versões ancestrais das quais é derivada. Posteriormente, a nova versão pode evoluir de maneira independente. Chamamos de **versões origem** às versões ancestrais no relacionamento de derivação.

O processo de derivação é descrito pela **Hierarquia de Derivação de Versões**,

que estrutura o conjunto de versões de um objeto como uma árvore [CK86, Kat90] ou um grafo acíclico dirigido (GAD) [TOC93, Zdo86]. A derivação representada como GAD corresponde a um processo de combinação ou mistura de versões, onde devem-se definir critérios de seleção de valores dentre as versões ancestrais para a criação da nova versão. Em [TOC93] o critério corresponde a definir uma das versões ancestrais como prioritária. Em geral duas versões podem ser classificadas como **derivativas** se existe um caminho desde a raiz da árvore ou GAD que inclua a ambas. No caso contrário são chamadas de **alternativas** ou **variantes** (figura 2.1).

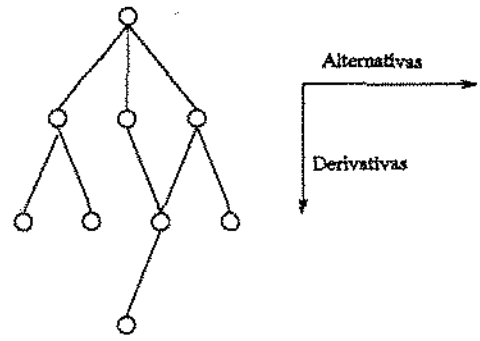


Figura 2.1: Tipos de Versões

O relacionamento de derivação representa adequadamente a semântica da evolução dos projetos em sistemas CAD, onde existem múltiplas alternativas que reusam parte importante do trabalho anterior, em um processo de refinamento e modificação. Em geral o desenvolvimento de projetos de engenharia é bem descrito pelas hierarquias de derivação, as quais podem ser diretamente manipuladas e consultadas pelos usuários através da funcionalidade do sistema.

[Sci91a, Sci94] tenta criar um modelo de versões independente da área de aplicação e apresenta o conjunto de versões de um objeto como um espaço de múltiplas dimensões definidas pelo usuário para a sua aplicação. Uma versão pode ser referenciada e identificada como um "ponto" no espaço multidimensional. Neste modelo não existe nenhum relacionamento explícito entre as versões (por exemplo, derivação), exceto a relação de ordem implícita que as dimensões estabelecem. Qualquer relacionamento explícito entre versões deve ser mantido pelo programador através de atributos nas estruturas dos seus objetos, isto é, referências diretas entre as versões. Por exemplo: derivação entre versões em CAD, ou sucessão linear de versões em Sistemas Temporais.

[GJZ95] sugere utilizar o relacionamento por derivação somente para a criação de novas versões e o diferencia dos relacionamentos associados à semântica da aplicação. (Por exemplo, relação de sucessão em versões temporais ou a própria derivação em CAD).

Para a implementação do conjunto de versões de um objeto, muitas abordagens diferenciam o objeto conceitual versionável das suas versões. A forma fundamental desta abordagem é a criação de objetos genéricos [CVJ94, TOC93, Sci91a, BK85] associados à identidade dos objetos conceituais. Estes objetos genéricos estão relacionados por sua vez com outro conjunto de objetos que correspondem às versões propriamente ditas. A criação de um novo objeto conceitual vai implicar a criação de seu objeto genérico e de uma primeira versão que é associada a esse objeto genérico.

Em [Sci91a, Sci94] atributos de um objeto conceitual podem ser ou não versionáveis. Este objeto conceitual é mapeado para um objeto genérico que vai ter associado atributos e funções compartilhados por todas as versões, enquanto os atributos versionáveis estão armazenados em outros objetos que são as versões. O relacionamento entre versão e objeto genérico é modelado através de herança por delegação que implica que quando um atributo ou função não é encontrado em uma versão, então é procurado no objeto genérico associado.

[GSdS95] estende o modelo anterior permitindo vários níveis de versionamento, isto é, cada versão é considerada por sua vez um objeto genérico que tem suas próprias versões. Esta extensão permite representar a vários níveis o compartilhamento dos mesmos estados entre diferentes versões de um objeto.

Do ponto de vista do seu armazenamento, duas formas básicas são apresentadas [Gan94]: armazenamento *extensivo e diferencial*. No armazenamento extensivo cada versão é armazenada de forma separada, enquanto no armazenamento diferencial são armazenadas somente as diferencias ou *deltas* entre duas versões derivativas ou as sucessivas transações entre uma e outra versões. O armazenamento diferencial consegue um uso mais eficiente da memória mas tem como inconveniente o maior tempo para o acesso ao estado de uma versão dada.

2.1.2 Relacionamento entre versões de objetos diferentes.

Embora o relacionamento e organização das versões de um mesmo objeto tenham relevância nos modelos de versões, o objetivo básico nestes modelos é facilitar o gerenciamento de unidades complexas. Estas unidades estão compostas por versões dos diferentes objetos modelados, formando estruturas consistentes com a semântica da aplicação, isto é, versões que podem ir juntas. Por exemplo, em um sistema CAD para projeto de circuitos, variantes formadas pela combinação de versões de diferentes elementos eletrônicos projetados, conformam diferentes variantes de circuitos, cujo projeto é o objetivo fundamental da aplicação. No entanto, a combinação de versões deve manter determinados critérios de consistência semântica, pelos quais não todas as possíveis combinações de versões são válidas.

Nas diferentes abordagens estas unidades semânticas são chamadas de *contextos, slices,*

configurações. Aqui será utilizado o termo *configuração*. Uma mesma versão de um objeto pode formar parte de diferentes unidades semânticas.

Um dos problemas básicos de um mecanismo de versões é a criação e manutenção destas unidades de consistência. Como caso particular deste problema está o versionamento de *objetos complexos compostos*, os quais estão recursivamente relacionados por composição com outros objetos em vários níveis. Neste contexto, a relação de composição é diferenciada da relação de agregação entre objetos. A relação de composição é entendida como relação *IS PART OF* [KBC+87, KBG89], isto é, relacionamento semântico de dependência entre objetos que não podem existir independentemente uns dos outros e muitas vezes a composição é exclusiva. Em [KBC+87, KBG89] o conjunto dos objetos relacionados por composição é conhecido coletivamente como *Composite Object*. O relacionamento de agregação não impõe estas restrições e simplesmente modela qualquer referência entre objetos diferentes.

Para o estabelecimento dos relacionamentos entre as versões de componentes de um objeto composto em uma configuração existem duas formas de ligação: **ligação estática** (precoce) ou **ligação dinâmica** (tardia) (figura 2.2).

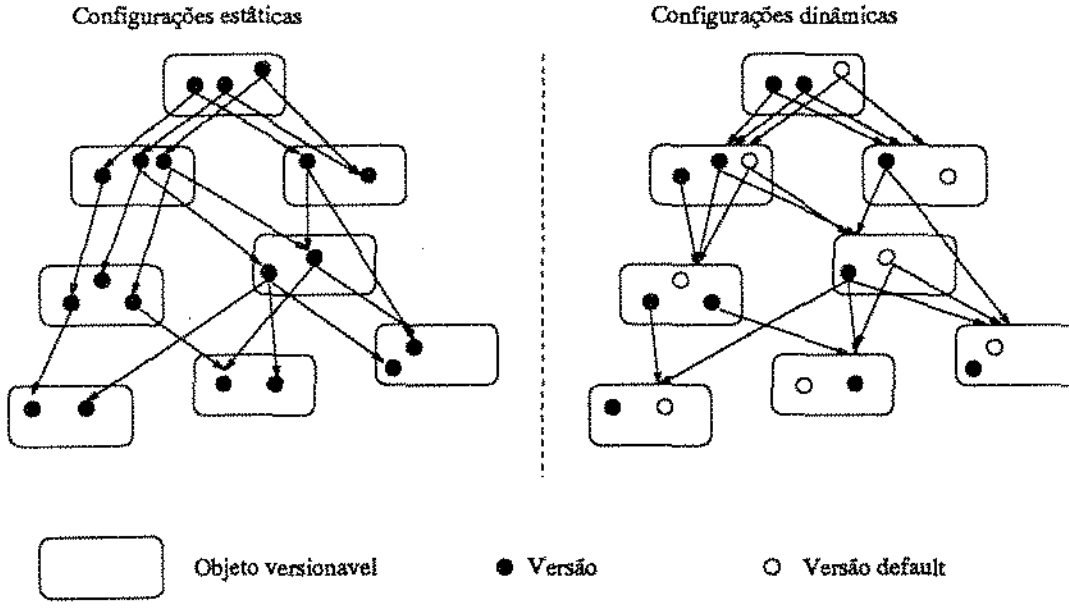


Figura 2.2: Tipos de Configurações

Na ligação estática, as relações de composição entre as versões se estabelecem com relacionamentos diretos, isto é, uma versão composta contém referências diretas às versões componentes. Neste caso temos **configurações estáticas**. No caso de ligação dinâmica em geral o conceito de objeto genérico é utilizado e a composição é estabelecida como

referências explícitas entre as versões e os objetos genéricos componentes (**referências genéricas**), isto é, uma versão de um objeto referencia a todo o conjunto de versões de cada um dos objetos componentes. As versões em uma configuração são estabelecidas dinamicamente a partir de algum critério de seleção em cada objeto. Em geral, uma versão *default* é indicada para cada objeto, na ausência de critérios de seleção. Esta pode ser a versão semanticamente mais completa para o gerenciador do projeto [CK86, Kat90], a última criada [Gol93], dentre outros possíveis critérios. Neste caso temos **configurações dinâmicas**.

Para a manipulação de configurações, em [EV94] é proposto um modelo de versões baseado em grafos AND/OR. Um grafo AND/OR é um grafo acíclico dirigido, composto por nós chamados AND e OR, os quais se vão alternando por níveis. Este grafo modela o espaço de versões de um projeto. Um nó OR representa o objeto genérico de um objeto versionável. Um nó AND representa uma versão desse objeto genérico e contém referências aos objetos que formam sua estrutura de agregação. Neste modelo uma configuração é um subgrafo tal que todos os nós OR contêm somente um nó descendente AND. Este modelo permite interpretar as transações sobre um espaço de versões como operações em grafos.

O uso de configurações dinâmicas é muito comum em sistemas CAD e CASE, pois permite modelar o trabalho de projeto por tarefas. Neste caso por exemplo, um projetista ou grupo utiliza como componente do seu próprio projeto um objeto que é projetado por outro grupo de trabalho e cuja versão final é desconhecida. Esta versão final é associada por meio de ligação dinâmica.

2.1.3 Consistência em um Banco de Dados Multiversão

Na definição tradicional, um banco de dados é chamado de *consistente* se representa um estado válido do mundo real que modela. A consistência é garantida se cada transação leva os dados de um estado consistente a outro estado consistente. Este critério define a base para a descrição de restrições de integridade que descrevem parte importante da semântica de aplicações.

No caso de bases de dados multiversão, a definição anterior não é aplicável, pois a existência de múltiplas versões para cada objeto impede garantia de consistência entre todos os dados simultaneamente, isto é, em conjunto não pode representar um estado do mundo real. O critério de **Consistência parcial** [CJ90] então pode ser aplicado, pelo qual um banco de dados multiversão será considerado consistente se representa múltiplos estados consistentes do mundo real. Neste caso, transações devem levar o banco de dados entre conjuntos de diferentes estados consistentes do mundo real. Um estado consistente do mundo real pode ser identificado com uma configuração consistente. Isto implica que toda versão de um objeto deve pertencer a alguma configuração.

Para garantir consistência parcial nestes bancos de dados, o modelo de versões deve providenciar um modo pelo qual o sistema possa identificar e gerenciar, em cada momento, quais conjuntos de versões conformam as diferentes configurações. Transações que modificam o banco de dados devem garantir a consistência dessas configurações. Ao ser criada uma nova versão de um objeto o sistema precisa associá-la a uma nova configuração.

Existem três enfoques para a identificação das configurações:

- O enfoque associado a transações [Zdo86] considera uma configuração como o conjunto das versões de objetos que são criadas como resultado da mesma transação.
- O enfoque associado a objeto composto, típico das aplicações CAD e CASE, [Kat90, KBC+89] identifica uma configuração com uma versão de um objeto composto que modela a entidade que é projetada. Estes dois enfoques determinam a configuração depois do processo de criação de versões.
- O enfoque com configurações *a priori* considera a existência de configurações definidas inicialmente sobre às quais são associadas as novas versões dos objetos que são criadas.

A figura 2.3 representa no lado esquerdo um banco de dados multiversão e do lado direito as configurações consistentes que podem ser identificadas. O Banco de Dados é consistente enquanto todo o conjunto de versões pode ser organizado em configurações. Neste caso é utilizado o enfoque de configuração associada a objeto composto.

A partir da sua granularidade, três formas de versionamento são definidas [WR95]: versionamento a nível de objeto simples, de objeto composto e de Banco de Dados, descritas a seguir.

Versionamento no nível de objeto simples

Nesta abordagem as versões de um mesmo objeto conceitual estão associadas por um objeto genérico. A criação de uma nova versão de um objeto corresponde à sua inclusão no conjunto de versões associado. A manutenção de integridade depende do tipo de ligação. Com ligação dinâmica uma versão de cada objeto é selecionada para integrar a configuração com as quais são resolvidas as referências genéricas entre os objetos. Esta configuração é a única consistente e a Consistência Parcial não pode ser garantida.

No caso de ligação estática, as diferentes configurações podem ser obtidas percorrendo todas as referências estáticas entre as versões dos objetos, mas nada garante que todas as versões de objetos estejam em uma configuração ou que as dependências de um objeto composto sejam mantidas.

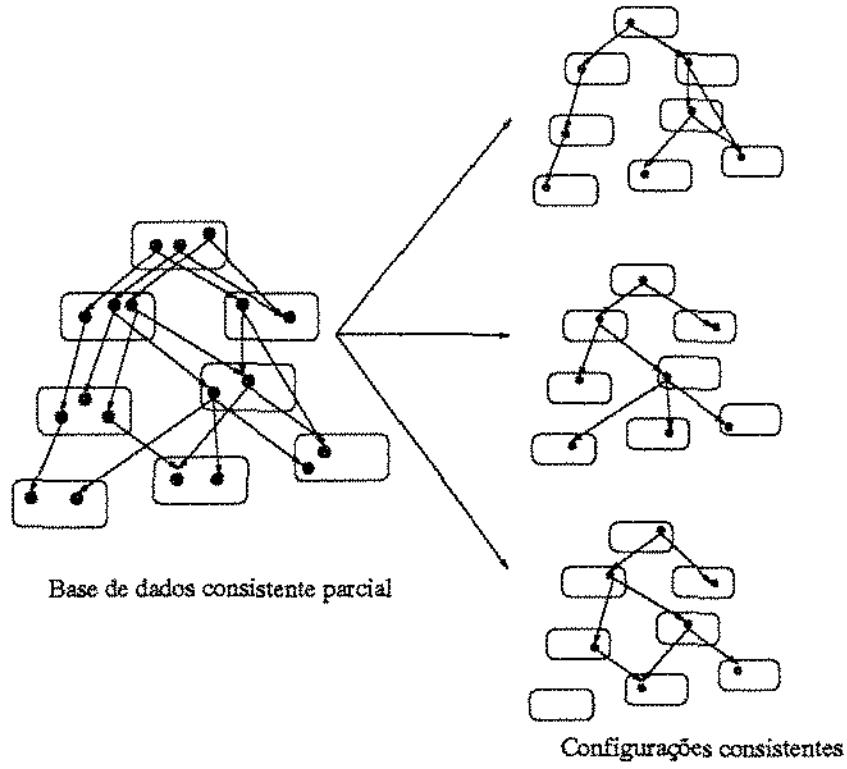


Figura 2.3: Consistência Parcial

Em [Sci91a] é definido um mecanismo de resolução de referências genéricas a partir de *visões* dos conjuntos de versões. Uma visão é um predicado associado a uma referência genérica e seu resultado é o subconjunto que satisfaz esse predicado. Normalmente, o predicado deve dar como resultado um subconjunto unitário para resolver a referência.

Versionamento no nível de objeto composto

Esta abordagem está muito relacionada com o enfoque onde uma configuração é considerada equivalente a uma hierarquia de composição de um objeto complexo composto, isto é, todas as versões em uma configuração estão relacionadas por algum tipo de relação de composição.

Para manter consistência, a criação da versão de um objeto componente deve provocar a criação de versões dos outros objetos a ela relacionados por composição para garantir uma nova configuração consistente. Em [Kat90, TOC93] a solução adotada é a **propagação**. Nesta solução ao ser criada uma nova versão de um objeto o sistema gera novas versões dos objetos com os quais a versão origem tem relacionamentos IS PART OF e estas novas versões por sua vez repetem recursivamente o mesmo processo, em uma propagação do versionamento. Essas novas versões são derivadas a partir de outras versões

existentes.

Por exemplo, na figura 2.4 a criação de uma nova versão de E implica o versionamento de outros objetos (A e B) para obter as diferentes versões das componentes de uma nova versão do objeto composto.

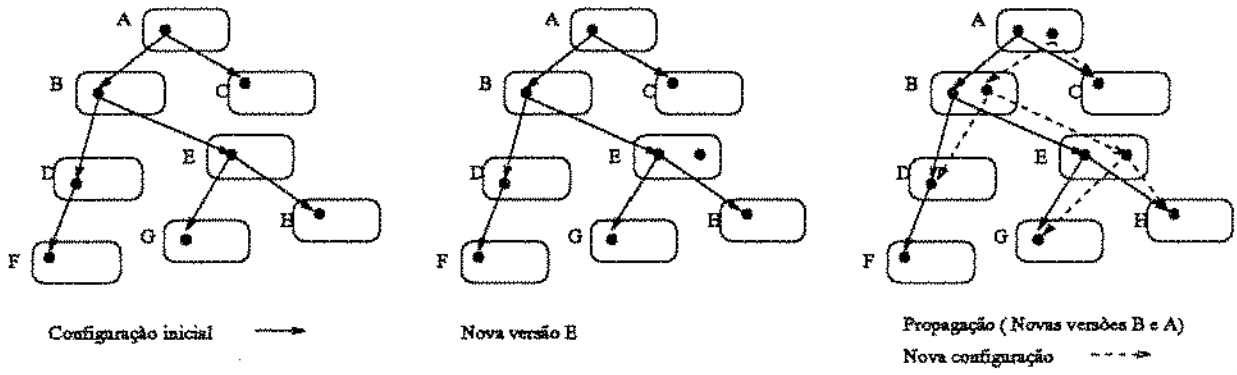


Figura 2.4: Propagação do versionamento

Esta técnica possui dois problemas:

- *A geração de um número grande de versões e configurações intermediárias não usadas.* A criação de uma nova configuração muitas vezes é o resultado da criação sucessiva de várias novas versões de objetos diferentes, que ao final deseja-se formem parte da mesma configuração. Neste processo, o sistema vai gerar novas configurações intermediárias cada vez que uma nova versão individual é criada, e como resultado são criadas versões que não são utilizadas e que aparecem como resposta do mecanismo de propagação. Por exemplo na figura 2.5 o usuário deseja uma nova configuração com novas versões dos objetos E e D (configuração 3). Fazendo o versionamento de cada objeto passo o passo, a configuração 2 é somente um passo intermediário para criar a versão do objeto E. Nesta configuração 2, como parte da propagação foram criadas versões dos objetos B e A que não serão utilizados.

  [Kat90] propõe um mecanismo de versionamento em grupo para a criação simultânea de versões que formarão parte de uma mesma configuração, introduzindo o problema de como criar por propagação, uma configuração consistente que inclua simultaneamente a todas as novas versões.

- *Ambigüidade na propagação.* O fato de que uma versão simultaneamente pode formar parte de varias configurações implica que esta é referenciada por varias outras

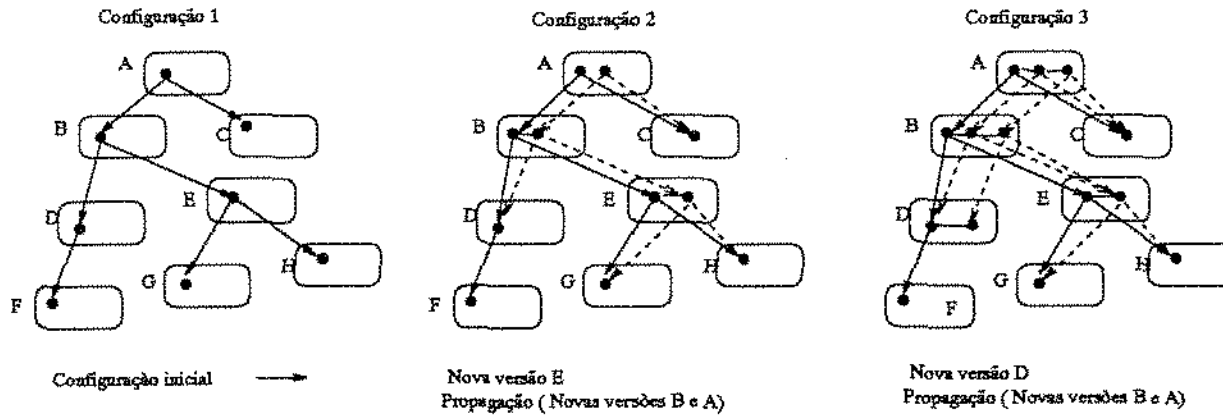


Figura 2.5: Configurações intermediárias

versões em diferentes configurações. No processo de propagação, neste caso, várias alternativas de replicação dos relacionamentos criam ambigüidade. Por exemplo: na figura 2.5 quando uma nova versão de D é criada (configuração 3) o mecanismo de propagação provoca uma nova versão de B, a qual pode-se derivar de duas possíveis versões existentes

Com o objetivo de diminuir estas limitações, em [Zdo86, TOC93] as versões são definidas pelo projetista como sensíveis ou não. O processo de propagação será feito somente para as versões sensíveis. No caso de [TOC93] também se inclui a possibilidade de fazer manualmente a propagação.

O compartilhamento de uma versão por várias configurações gera problemas de consistência semântica no banco de dados. Considerando que cada configuração representa um estado do mundo real, as modificações consistentes feitas em uma delas pode ter como efeito colateral a modificação não consistente em outras configurações.

Como solução são introduzidos mecanismos de **notificação** [CK86, KBC+89] que correspondem ao envio de mensagens ou sinais aos usuários das configurações alteradas ou com algum das suas componentes versionadas. A notificação baseada em sinais informa ao usuário da alteração somente no momento em que este utiliza a versão transformada [CK86].

Este critério de consistência em [CK86] supõe que toda versão é responsabilidade de um projetista e tem associado dois valores de tempo: tempo de modificação(TM) que indica o último tempo onde a versão foi modificada e o tempo de aprovação (TA) que corresponde ao último tempo onde o projetista da versão aprovou uma modificação. Uma versão V é chamada **consistente por referência** sempre que TA $\geq$ TM para V e para todas as versões referenciadas por V.

Outro elemento a considerar no mecanismo de notificação é seu escopo. Suponha que a versão V_i faz referência a V_j, V_j faz referência a V_k e que V_k seja modificada ou

eliminada. Quais versões são notificadas?. Em [CK86] é adotado o critério de somente fazer notificação ao primeiro nível ancestral (ou seja, V_j).

Em geral, a abordagem de versionamento no nível de objeto composto precisa da manutenção de relacionamentos complexos entre os objetos e suas versões para sua implementação. Aliás, esta abordagem tem a limitação de que parte de considerar que uma configuração corresponde sempre a um objeto composto. Em um banco de dados formado por múltiplos objetos complexos compostos não relacionados entre eles por composição, mas que em conjunto formam um estado do mundo real, não seria possível estabelecer quais das suas versões seriam mutuamente consistentes.

Versionamento no nível de Banco de Dados

Em uma abordagem diferente [CJ90, GJ94, Gan94, GJ96] define um mecanismo onde a unidade de versionamento é *todo o banco de dados*, isto é, define um banco de dados multiversão como um conjunto de Versões de Banco de Dados (DBV). Cada DBV contem uma versão diferente de cada um dos objetos versionáveis. Se um objeto não existe em uma DBV então sua versão lógica nela é vazia. Uma nova DBV é criada como derivação de outra DBV já existente. Inicialmente, as versões de todos os objetos na nova DBV tomam os valores das versões na DBV ancestral. Do ponto de vista físico, na realidade as versões não são duplicadas, enquanto diferentes valores não forem associados às duas versões.

O processo de derivação produz uma estrutura de derivação de árvore entre as DBV. Em [WR95] esta estrutura é estendida a um Grafo Acíclico Dirigido(DAG).

As vantagens fundamentais do mecanismo são:

- *Garantia da consistência parcial:* Neste caso o banco de dados multiversão é considerado consistente se cada DBV é consistente. Como a criação da versão de um objeto somente é possível como parte da criação de uma versão do banco de dados completo, isto garante que todos os relacionamentos desse objeto são mantidos na nova configuração.

- *Restrições de integridade:* Segundo [DGJM96, MCJ93] o mecanismo DBV satisfaz as propriedades requeridas por um modelo de versões para integrar restrições de integridade em um SGBD:

  - O critério de consistência no banco de dados multiversão deve estar bem definido.

  - A organização de versões gerada pelos mecanismos de derivação não deve estar limitada à ordem seqüencial.

- *Controle da concorrência.* No mecanismo DBV, a introdução do versionamento não implica maior complexidade do controle de concorrência [CJ94]. Transações diferentes sobre duas DBV diferentes não compartilham informação e portanto não é necessário sincronização entre elas. O problema do controle de transações sobre a mesma DBV é equivalente ao mesmo problema para um banco de dados convencional monoversão.

O mecanismo DBV será apresentado em mais detalhes no capítulo 4 pois será a base da proposta apresentada nesta dissertação.

2.2 Versões em sistemas CAD/CASE

Os sistemas CAD em geral estão orientados a tarefas de projeto ou planejamento e seu objetivo fundamental é a obtenção de um projeto final de um objeto de engenharia. Este objeto geralmente está composto por uma considerável quantidade de módulos e componentes a vários níveis, com relacionamentos muito complexos (por exemplo, um circuito integrado) e com múltiplas alternativas para cada um desses módulos. Os sistemas CASE possuem em geral características similares aos sistemas CAD no uso de versões, mas seu objetivo é o desenvolvimento e manutenção de diferentes configurações de software. Nestes sistemas, os objetos modelados são módulos de programas, dados de teste, módulos compilados e outros.

Em geral, o processo de desenvolvimento dos objetos em CAD e CASE precisa dar apoio às seguintes características [Kat90]:

- *Manipulação de objetos complexos*: Os objetos envolvidos em geral são compostos, complexos e organizados hierarquicamente. Como resultado, a modelagem orientada a objetos aparece como a ideal nestes casos.

- *Natureza iterativa e tentativa do processo de trabalho*: Implica que os novos dados não substituem os anteriores; todos são mantidos como diferentes versões e combinados entre si.

- *Natureza cooperativa do trabalho*: Muitas pessoas participam simultaneamente no desenvolvimento do projeto e podem acessar concorrentemente os mesmos dados e criar suas próprias interpretações dos objetos modelados. Aqui ocorre diferenciação do trabalho individual e de equipe.

Segundo [Kat90], existem três elementos característicos do versionamento em CAD:

- *Derivação entre versões:* Em geral uma nova versão utiliza parte importante dos valores e estrutura de outras versões anteriores.

- *Propagação:* O ponto anterior implica que na criação de uma nova versão seja necessária paralelamente a criação de versões adicionais de outros objetos, para "replicar" exatamente as relações das versões ancestrais, mantendo assim a consistência dos dados. Esta propriedade realmente tem a ver com a consistência do banco de dados como já foi analisado.

- *Classificação de versões:* Em geral, os mecanismos para o trabalho cooperativo, promovem a classificação das versões de acordo com seu estado de evolução no desenvolvimento do projeto.

2.2.1 Suporte ao trabalho cooperativo

Modelos e mecanismos de versões surgiram fundamentalmente associados a sistemas com possibilidades para o trabalho cooperativo, mas não são propriedades exclusivas destes sistemas. No entanto, muitos dos modelos de versões foram definidos em combinação com conceitos e mecanismos para o trabalho cooperativo.

A dinâmica em muitos destes sistemas pode ser entendida como o trabalho de várias pessoas que criam suas próprias versões sobre um projeto ou partes componentes deste, intercambiam estas versões a partir da estrutura de uma organização geralmente hierárquica, repetindo este processo ciclicamente, compondo por sua vez versões finais do objeto de projeto alvo. Nesta metáfora, 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 estruturam o banco de dados em vários níveis que refletem uma organização e direito de acesso às diferentes partes do projeto cooperativo. Em [CK86] estes níveis são: banco de dados público, bancos de dados privados e bancos de dados de projeto. No primeiro estão armazenadas as versões de uso geral no sistema. Nos bancos de dados privados estão as versões dos projetistas que não podem ser acessadas pelo resto dos usuários. Um banco de dados de projeto contém versões não privadas que só podem ser acessadas pelos membros de um grupo de projeto.

Dentro de seu nível de trabalho um projetista cria novas versões e como conseqüência novas configurações. Para intercambiar versões entre os diferentes níveis são realizadas copias destas utilizando igualmente os mecanismos de versionamento. O termo *Check-out* é utilizado para descrever o processo de derivação e criação de novas configurações para um nível inferior na hierarquia de organização e o termo *Check-in* no caso contrario (figura 2.6)

Um modelo de organização similar é apresentado em [Kat90] que chama os diferentes níveis de espaços de trabalho (*Workspaces*) e os classifica em espaços de trabalho de arquivos, privados ou de grupo. Associados a esta estrutura são definidas duas operações para a manipulação de versões entre os espaços de trabalho. A operação *check-out* de

uma versão Vi cria uma cópia desta como versão Vj no banco de dados privado de um projetista. A operação *check-in*, por sua vez, cria uma cópia de uma versão Vj do banco de dados privado como versão Vk no banco de dados de projeto ou público. Estas operações definem o modo de trabalho do projetista, isto é, ele obtém versões privadas do resultado do projeto coletivo em um dado momento, desenvolve individualmente seu trabalho e posteriormente incorpora o resultado aos dados de acesso coletivo.

Paralelamente, as diferentes versões são classificadas de acordo com diferentes critérios em geral associados ao seu estado de terminação ou estabilidade dentro do projeto. Mecanismos para promoção dentro da classificação são definidos e em geral a classificação de uma versão está associada a quais dos diferentes níveis do banco de dados pode armazenala assim como que possibilidades tem de ser modificada, eliminada, dentre outras.

Em [CK86, KBC+89] as versões são divididas em *transient, working e released.* Uma versão *transient* está em um estado não estável e caracteriza um projeto em desenvolvimento. Uma versão *working* caracteriza a versão que está num estado estável, isto é, não pode ser modificada mas pode ser eliminada do projeto por seu criador. Uma versão *released*, por sua vez, além de não poder ser modificada não pode ser eliminada, pois corresponde a uma versão final. As promoções de um estado a outro são feitas pelos projetistas diretamente, mas uma versão *transient* da qual é derivada outra, é promovida automaticamente a versão *working.* Em [TOC93] as versões são classificadas em temporárias e permanentes. Estas últimas não podem ser modificadas.

Em todos os casos, o objetivo é organizar e proteger as versões criadas por um projetista individual ou realizadas no contexto das diferentes estruturas de organização (grupos, subprojetos) e estabelecer critérios do estado de uma versão.

Em [CVJ94] é proposta uma organização lógica do banco de dados em unidades chamadas de constelações (*constellations*). Uma constelação é um subconjunto de objetos de projeto com suas versões e representa o espaço de trabalho de um projetista. Sobre uma constelação o projetista cria configurações. O modelo permite organizar o banco de dados em subconjuntos não necessariamente disjuntos que representem os acessos permitidos a cada projetista. Um projeto pode ser organizado hierarquicamente definindo níveis de constelações onde cada nível inclui todos os objetos de projeto das constelações do nível mais baixo, criando-se assim uma árvore.

2.3 Versões em Sistemas Temporais

Embora do ponto de vista da implementação de versões os sistemas temporais tenham muito em comum com os sistemas CAD ou CASE, a abordagem conceitual e semântica é

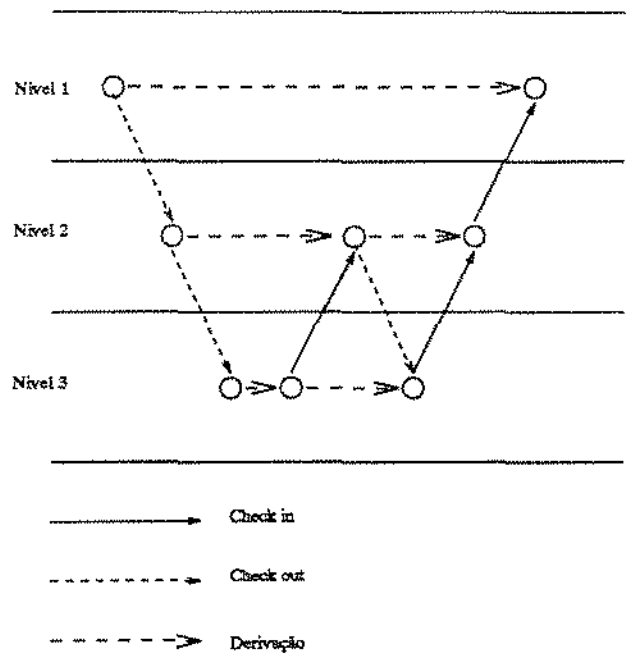


Figura 2.6: Operações de Check-in / Check-out

diferente. Nesta seção serão apresentados os conceitos fundamentais de bancos de dados temporais e as abordagens e problemas no caso do uso de versões.

2.3.1 Conceitos de Bancos de Dados Temporais

O problema da representação temporal dos dados tem sido extensamente estudado nos últimos anos [Sno90], basicamente na definição de modelos, relacionamentos e extensões a linguagens de manipulação de bancos de dados para a descrição da semântica temporal. Em [JJE+94] aparece uma descrição detalhada dos principais conceitos e terminologia em Bancos de Dados Temporais.

Densidade temporal

Embora o tempo conceitualmente seja continuo, existem três visões:

- *Discreta:* Isomorfa aos números naturais. Permite definir o conceito de sucessor no tempo.
- *Densa:* Isomorfa aos números racionais. Sempre é garantido um valor de tempo entre quaisquer outros dois valores.
- *Contínua:* Isomorfa aos números reais. Não existem "buracos"entre pontos de tempo.

Ponto de tempo ou *Chronon*

Em geral um SGBD só pode representar a densidade discreta e o *Chronon* ou Ponto do Tempo é a menor duração de tempo que pode ser representada. Um *chronon* em particular é definido em [JJE+94] como um subintervalo de duração fixa no eixo do tempo que não pode ser decomposto.

Marca de Tempo ou *Time Stamp*

Alguns fenômenos ocorrem de uma maneira discreta e brusca provocando mutações instantâneas dos objetos, enquanto outros ocorrem como processos de evolução contínua. No entanto, independentemente da forma de evolução das entidades, o seu registro será discreto

A associação de valores de tempo às entidades modeladas pode ocorrer de duas formas: associada a atributos ou associada a relações ou objetos. O valor temporal associado é chamado de marca de tempo (*TimeStamp*). Existem três formas de associar marcas de tempo a dados:

- *Evento:* Um evento corresponde a um instante de tempo associado a um *chronon*
- *Intervalo:* Representa a sucessão de tempo entre dois eventos, isto é, um conjunto de *chronon* consecutivos limitados pelos *chronon* associados aos eventos.
- *Elemento temporal:* Corresponde a uma união finita de regiões no tempo que podem ser por sua vez eventos ou intervalos.

O caso da representação de marcas de tempo em intervalo pode ter duas interpretações:

- *Em escada*: As propriedades da entidade são consideradas válidas para o período de tempo compreendido no intervalo. Cada intervalo representa um estado da evolução da entidade.
- *Contínua*: As propriedades de uma entidade, nos diferentes registros discretos ou eventos que definem o intervalo, são utilizadas como base para algoritmos de interpolação ou generalização temporal. Estes calculam o valor das propriedades de um objeto para um instante de tempo não registrado.

Ordem temporal

Existem três formas de estruturar o tempo:

- *Linear:* O tempo é considerado linearmente ordenado, isto é, para dois pontos no tempo sempre é possível estabelecer uma relação de ordem entre eles

- *Ramificado:* O tempo é considerado com ramificações, podendo ser modelado como um grafo acíclico. Esta estrutura é considerada em casos de modelagem de alternativas. Entre os valores de tempo pode-se estabelecer uma relação de ordem parcial.
- *Circular:* O tempo é considerado recorrente, isto é, os fenômenos são periódicos e não pode ser estabelecida qualquer relação de ordem.

Dimensões temporais

Em Bancos de Dados Temporais são modeladas basicamente duas dimensões temporais: **Tempo de Transação**, o qual está associado ao momento em que o novo estado da entidade é armazenado no banco de dados; e **Tempo Válido**, que corresponde ao tempo em que ocorre o novo estado da entidade no mundo real. A manutenção de ambos os tempos em bancos de dados dá lugar aos modelos **bitemporais**.

Em um banco de dados bitemporal, cada novo estado ou versão é associado a um valor de tempo válido, geralmente representado como um intervalo de tempo, no qual esse estado ocorre no mundo real. O valor especial NOW representa um estado válido no tempo atual. O tempo de transação para cada estado é associado aos dados pelo próprio sistema no momento em que estes sejam armazenados e termina quando seu valor é substituído por outro.

Granularidade temporal

Diferentes entidades podem mudar em diferentes escalas temporais pois os fenômenos ou processos em que eles intervêm mudam também em diferentes escalas. Uma escolha errada da escala temporal pode gerar perda de informação sobre a evolução dos fenômenos, ou gerar falsas percepções sobre a coexistência de dois objetos em um instante de tempo quando isto na realidade não ocorre. A combinação de objetos com escalas temporais muito diferentes pode provocar igualmente distorções nos resultados

Exemplo: As mudanças de relevo de uma região como parte do processo de evolução geológica são estudadas em escalas de milhões de anos, enquanto o estudo do crescimento de um rio durante um período de chuvas usa escalas de dias ou semanas.

Restrições de integridade em modelos temporais

Em modelos temporais as restrições de integridade geralmente aparecem como um elemento adicional e não como um tópico de primeira ordem. Em [Wij95] relações de equivalência são utilizadas para expressar que atributos devem evoluir de forma síncrona. Em [JJE+94] dependências funcionais convencionais são estendidas para expressar dependências entre atributos em qualquer momento do tempo.

A maioria de trabalhos na área estão associadas à busca de mecanismos para expressar as restrições sobre a base de lógicas temporais e de primeira ordem, mas não aos mecanismos de implementação e verificação.

Em [DFG+96] é apresentada uma taxonomia das restrições a partir da semântica dos eventos mencionados:

- *Históricas:* As restrições que envolvem eventos históricos são aquelas nas quais os eventos podem acontecer somente se alguma condição é verificada. (Por exemplo: uma pessoa precisa ter sido assistente para ser professor)
- *Síncronas:* Envolvem eventos que devem acontecer sincronamente (Por exemplo: modificar endereço implica modificar telefone)
- *Periódicas:* Envolvem eventos que acontecem periodicamente.(Por exemplo: O salário de um professor deve ser incrementado cada ano)
- *Dependente de roteiros:* Este tipo de restrição não pode ser verificada de forma usual porque a sua violação não depende da atualização realizada pela transação, senão do tempo real. Para este tipo de restrição temporal podem ser usados mecanismos ativos. (por exemplo, Um contrato deve ser assinado dentro das três semanas seguintes à proposta dos parceiros, de outra forma é invalidado.)

2.3.2 Versionamento Temporal

Os sistemas temporais utilizam mecanismos de versões como base para a implementação e manutenção da evolução histórica das entidades modeladas. Existem duas abordagens de versionamento: versionamento de atributos ou versionamento de tuplas (ou objetos). Cada um dos elementos versionados terá associada uma marca de tempo *timestamp* que indica a validade temporal desse estado. Usualmente são utilizados intervalos como formas de representação da marca de tempo.

Do ponto de vista dos problemas associados aos mecanismos de versões e discutidos anteriormente, a modelagem temporal linear, tradicionalmente a mais usada, simplifica os problemas de versões por duas razões:

- *O versionamento no nível de objeto simples garante consistência parcial.* A relação de ordem total associada à semântica temporal permite estabelecer e controlar os diferentes estados do mundo real armazenados no banco de dados a partir dos *timestamps* das versões. Toda nova versão temporal de um objeto será automaticamente associada a uma configuração a partir dos valores de tempo associados, sem precisar criar versões de outros objetos.

- *Relacionamento implícito entre versões.* Nas abordagens de versões analisadas, a derivação aparece como a operação básica com duas funções: descrever a semântica do trabalho no caso de sistemas orientados a projetos e permitir o compartilhamento de estados entre versões derivadas. No caso temporal, do ponto de vista semântico o relacionamento de derivação torna-se desnecessário, pois o relacionamento semântico fundamental entre versões de um objeto é a sucessão temporal, a qual pode se estabelecer a partir dos seus *timestamps* sem derivação.

No entanto, a utilização da plena potencialidade do mecanismo de versões, pode introduzir maiores possibilidades na modelagem temporal [GJZ93] por duas razões básicas:

- *Para modelagem de tempo não linear.* Nesta modelagem, a consistência não é garantida com a abordagem a nível de objetos simples, pois as diferentes configurações não podem ser automaticamente estabelecidas a partir de que para um objeto podem existir mais de uma versão com o mesmo valor de tempo. Neste caso outras abordagens de versões permitiriam a manutenção de configurações alternativas para o mesmo valor de tempo.

- *Para compartilhar estados temporais não consecutivos.* Na abordagem anterior de versionamento temporal, a única forma de compartilhar estados temporais está associado à linearidade do versionamento, isto é, duas versões consecutivas na ordem linear são o reflexo de alguma mudança entre elas, caso contrário existiria somente uma versão. Podem no entanto existir aplicações em que seja mais conveniente compartilhar estados de versões temporais não consecutivas em relação ao tempo.

  Por exemplo, suponha uma aplicação temporal na qual os estados temporais dos mesmos dias da semana sejam muito similares, enquanto dias consecutivos possuem estados diferentes. Poderia ser criada uma versão padrão para cada dia da semana, considerada *default* para aquele dia. No caso que um dia em particular não corresponda com esse padrão, seria criada uma nova versão derivada da versão *default*, mas pela natureza do problema isso não vai ser comum. Neste caso, o total de versões existentes poderia ser muito menor do que em um versionamento linear. No entanto, para esta solução é necessário utilizar o relacionamento de derivação, pois as versões além de manter o relacionamento temporal linear implícito, precisam outro tipo de relacionamento explícito como forma de representar o compartilhamento de estados.

2.4 Resumo

Este capítulo apresentou uma revisão bibliográfica sobre os principais problemas na manipulação de versões no nível de usuário, tanto do ponto de vista da sua inserção em

sistemas de bancos de dados como do apoio ao desenvolvimento. Foram destacados os principais problemas associados à definição de um modelo de versões e os diferentes enfoques da sua solução. Estes problemas são: a organização, manipulação e armazenamento de versões de objetos, o gerenciamento de configurações e os problemas de consistência entre as diferentes versões. Finalmente foram analisadas as características e os problemas da inserção de versões em sistemas de apoio a ambiente de projeto e de sua utilização em sistemas para o gerenciamento da evolução temporal.

Capítulo 3

Versões em Sistemas de Informações Geográficas

Este capítulo apresenta um estudo sobre o uso de versões em SIG. Inicialmente são descritas as características fundamentais dos SIG assim como o modelo de dados espacial adotado. A seguir, é discutida a utilização de versões para apoio projeto, múltiplas representações e modelagem temporal em SIG.

3.1 Sistemas de Informações Geográficas

De modo geral, um Sistema de Informações Geográficas (SIG) pode ser definido como um sistema de informação digital cujos registros são de alguma forma georeferenciados, isto é, possuem referências a coordenadas da superfície terrestre [MP94].

Não existe ainda uma definição padrão aceitável para Sistemas de Informações Geográfica tendo em vista as diferentes abordagens por áreas de aplicação, modelos de dados, funcionalidade, dentre outros, mas seu elemento mais significativo é a sua capacidade para manipular a dimensão espacial.

Existem diversas formas de caracterizar um SIG. Em [CCH+96, Cif95] são apresentadas três abordagem: de banco de dados, de *toolbox* e de processos. A abordagem de banco de dados define o SIG como um SGBD não convencional que garante o gerenciamento de dados geográficos. A abordagem de *toolbox* considera um SIG como um conjunto de ferramentas e algoritmos para manipular dados geográficos e produzir mapas. Finalmente, o enfoque de processos considera um SIG como um conjunto de subsistemas com algum grau de integração onde os dados vão passando por um fluxo de processos de conversão.

O tipo de dados armazenados em SIG é diferente dos encontrados em outros sistemas. Basicamente são considerados três categorias de dados [MP94]:

- **Dados convencionais** : Valores alfanuméricos tradicionais associados a objetos geográficos e que descrevem suas propriedades.
- **Dados espaciais**: Dados que descrevem a geometria e a localização no espaço de um objeto ou fenômeno e relacionamentos topológicos. A geometria consiste em um conjunto de coordenadas espaciais relativas a um sistema de projeção cartográfica.
- **Dados gráficos**: Atributos com imagens da superfície terrestre.

3.1.1 Arquiteturas e funcionalidade nos SIG

A partir da caracterização de um SIG como um banco de dados, [G94] define diferentes arquiteturas em função do relacionamento do SIG com o SGBD subjacente.

- *Arquitetura em camada:* Nestes sistemas um SGBD relacional subjacente é utilizado para representar a informação espacial a qual pode ser acessada diretamente pelos usuários. Sobre este é definida uma camada que interpreta e oferece a funcionalidade espacial. Nesta arquitetura as possibilidades de indexação espacial são limitadas.
- *Arquitetura dual:* Nesta arquitetura uma camada espacial integra dois módulos independentes. Um deles corresponde a um SGBD convencional que manipula atributos não espaciais enquanto o outro módulo corresponde a um subsistema que manipula estruturas espaciais diretamente sobre o sistema de arquivos. As informações nos dois módulos estão relacionadas por enlaces lógicos mantidos pela camada superior. O subsistema espacial implementa estruturas eficientes assim como algoritmos e índices para o processamento de consultas dentro deste subsistema.
- *Arquitetura integrada:* Estas arquiteturas estão baseadas em Sistemas de Bancos de Dados Relacionais Estendidos ou Orientados a Objetos nos quais as estruturas espaciais podem ser modeladas e implementadas diretamente no SGBD de forma eficiente.

Do ponto de vista de sua funcionalidade, SIG incluem elementos não presentes nos sistemas de bancos de dados tradicionais. As funcionalidades oferecidas são diferentes de acordo com o modelo de dados adotado, a área de aplicação e as funções específicas para o usuário final. Não há um conjunto de operações e primitivas espaciais básicas, mas estas em geral podem ser classificadas [MP94] nos grupos de: captura, validação e edição de dados; estruturação e armazenamento; reestruturação e transformação de dados; e análise.

Para a obtenção, validação e edição dos dados geográficos são necessárias metodologias e funcionalidades especiais. A coleta dos dados, além de ser um trabalho muito custoso, introduz erros de precisão que provoca um elemento de incerteza nas consultas.

Adicionalmente, os SIG possuem funcionalidades para a manipulação e transformação dos dados geográficos (por exemplo, processamento cartográfico e modificação da geometria através de operações de mudança de escalas, rotações, traslações).

As linguagens de análise e consulta em um SIG precisam dar apoio a facilidades de acesso espacial [Ege94, C95]. Isto implica a necessidade de novos operadores sobre os dados espaciais. Os operadores *geométricos* retornam valores escalares e estão associados a métricas dos objetos. Os operadores *topológicos* retornam valores booleanos e estão associados a relações topológicas entre objetos espaciais. Finalmente, os operadores *espaciais* retornam objetos geográficos a partir dos existentes.

As extensões às linguagens de consulta com operadores espaciais precisam de mecanismos eficientes para o acesso aos dados e otimização de consultas. Estruturas de dados espaciais são introduzidas para a criação de índices com métodos de acesso eficientes. Estas estruturas são classificadas em dois grupos de acordo com a abordagens para decompor o espaço indexado [Sam89, MP94]:

- Agrupamento dos dados geográficos, construindo um Retângulo Mínimo Envolvente *Minumum Bounding Rectangle (MBR)* para cada objeto . As coleções de MBR são organizadas em hierarquias. As estruturas mais representativas deste grupo é a família das árvores-R.

- Divisão do espaço em células disjuntas. Cada objeto espacial é armazenado na célula correspondente, organizadas por *buckets*. Exemplos de estruturas neste grupo são *GridFiles*, *QuadTrees*, *Spatial k-d trees*

3.1.2 Modelos de Dados em SIG

Um modelo de dados é um conjunto de conceitos e mecanismos para descrever o relacionamento entre um universo de dados, a semântica destes, além de suas restrições de consistência. Esta descrição pode ser vista como uma abstração do mundo real.

Os modelos de dados tradicionais não são adequados para o tratamento de dados geográficos. Segundo [MP94] o problema fundamental está em que os dados geográficos são validados em termos de sua localização geográfica, do tempo e da compatibilidade da sua coleta, o que não ocorre para dados convencionais.

A modelagem de dados geográficos é baseada em duas percepções do mundo: modelo baseado em objetos e modelo baseado em campos [MP94, RM92, CFS+94, Peu94]. No modelo baseado em objetos, o espaço geográfico é representado como uma superfície

formada por objetos bem definidos, com identidade própria e que existem independentemente (uma casa, uma rua). Vários objetos podem ocupar o mesmo espaço e não precisam estar associados a um tema específico. A precisão da geometria e topologia dos objetos é um elemento fundamental neste modelo.

Para o modelo baseado em campos o espaço geográfico é representado como diferentes camadas associadas à distribuição de grandezas contínuas ou discretas em uma região no espaço. Cada grandeza representa um tema de interesse (por exemplo, relevo, vegetação) e pode ser descrita como uma função matemática cujo domínio é a região geográfica ou campo e o contradomínio é o conjunto de valores da grandeza que o campo pode tomar.

[Peu94] identifica os modelos de campo e objeto como duas visões do mundo chamadas de *objetiva* e *subjetiva*. Na visão objetiva, o mundo é modelado como pontos de um universo de dimensões espaço-temporais que existe independentemente dos objetos que o ocupam. A modelagem da realidade geográfica é entendida como associação de grandezas nesses pontos. Na visão subjetiva, o mundo é modelado como um conjunto de objetos que, dentre outros, possuem atributos que correspondem ao espaço e tempo que ocupam, isto é, os objetos existem independentemente do espaço e tempo.

Tradicionalmente os modelos baseados em campos e objetos têm sido identificados com os formatos de representação *raster* e *vetorial*, respectivamente [Cou92]. O termo *raster* é utilizado para descrever matrizes de células retangulares que formam uma cobertura ou subdivisão do espaço. Cada célula tem associado um valor para uma grandeza determinada. O termo *formato vetorial* é usado para identificar a representação dos objetos segundo pontos, linhas e polígonos descritos por listas de coordenadas geográficas exatas.

Nenhum dos dois modelos - campos e objetos - é ideal e em geral eles se complementam. A utilização destes modelos está em geral associada à área de aplicação e as necessidades do usuário. [Cif95] classifica as aplicações dos SIG em aplicações urbanas/rurais e aplicações ambientais e de recursos naturais baseado fundamentalmente nos modelos de dados adotados. Nas aplicações urbanas e rurais predomina o uso do modelo de objetos, os quais geralmente são compostos e complexos e descrevem entidades construídas ou geradas pelo homem. Estas aplicações estão associadas a tarefas de administração, planejamento de redes, organização de cadastro e mapeamento urbano, controle populacional, dentre outras.

As aplicações ambientais e de recursos naturais utilizam fundamentalmente o modelo de campos e representam as informações em várias camadas, que correspondem geralmente a informação da natureza. Incluem a administração de recursos hídricos, flora e fauna, solos, clima, monitorização de catástrofes naturais, dentre outros.

3.1.3 Modelo de quatro níveis

Em geral os modelos de campos e objetos são uma abstração adequada e útil da realidade geográfica, mas a sua identificação com as representações em formato vetorial e raster nem sempre é adequada. Por exemplo, isolinhas são representadas de maneira vetorial, como sucessões de coordenadas geográficas, no entanto descrevem a distribuição de uma grandeza no espaço cuja modelagem conceitual deve ser como um campo.

[CFS+94, C95, CCH+96] apresentam um modelo que tenta resolver o problema definindo quatro camadas que descrevem diferentes níveis de abstração. Estes níveis são:

- *Nível do mundo real*: Corresponde aos elementos da realidade geográfica a ser modelada, por exemplo, rios, ruas, temperatura.
- *Nível Conceitual*: Corresponde à modelagem formal desses elementos da realidade. No caso dos elementos de natureza espacial, são utilizados os modelos de campos e objetos antes descritos. Neste nível são definidas as operações sobre os elementos modelados a partir das necessidades das aplicações.
- *Nível de Representação*: Corresponde ao mapeamento das entidades formais e suas propriedades espaciais para estruturas de representação geométrica e topológica.
- *Nível de Implementação*: Corresponde às estruturas de dados e algoritmos para a implementação das representações. Uma representação pode ter múltiplas formas de implementação

Neste modelo, a modelagem das entidades espaciais em nível conceitual, baseada nos modelos de campos e objetos, é realizada independente da estrutura adotada para a sua representação. A modelagem no nível conceitual define os conceitos, entidades e operações que devem ser refletidas nas interfaces e ferramentas apresentadas ao usuário final, abstraindo detalhes de representação.

Nível conceitual

O nível conceitual do modelo é baseado no paradigma orientado a objetos, onde os elementos da realidade geográfica são modelados a partir de classes. Estas são divididas em classes *convencionais* e classes *georeferenciadas* ou *geoclasses*. As classes convencionais descrevem elementos da realidade geográfica que não possuem natureza espacial e suas instâncias são chamadas de *objetos convencionais* (por exemplo, proprietários dos lotes na modelagem do cadastro do uso do solo).

As *geoclasses* descrevem elementos de natureza espacial a partir dos modelos de campos e objetos apresentados. As instâncias destas classes são chamadas de **GeoCampos** ou **GeoObjetos** respectivamente e têm associadas regiões da superfície da terra.

Em geral **GeoCampos** possuem atributos convencionais e os seguintes atributos espaciais [CCH+96]:

- *Localização:* Descreve a região R ou localização da instância.
- *Contradomínio:* Descreve um conjunto de valores V, chamado de contradomínio da instância.
- *Mapeamento:* Descreve uma função $f : R \rightarrow V$ chamada de mapeamento da instância, a qual modela um campo geográfico R que toma valores em V.

No modelo são apresentadas três especializações de GeoCampos:

- GEOCAMPO TEMÁTICO: O contradomínio V é um conjunto finito arbitrário. Cada um dos valores de V define um tema do geocampo.
- MODELO NUMÉRICO: O contradomínio V é um subconjunto dos números reais. Um geocampo deste tipo permite modelar grandezas com distribuição contínua no espaço.
- DADOS DE SENSOREAMENTO REMOTO: Corresponde a uma especialização de modelo numérico, onde o contradomínio V é um subconjunto dos números naturais. Em geral está associado a dados obtidos pela discretização de informação recebida por um sensor.

Um **GeoObjeto** por sua vez tem uma estrutura básica apresentada em [MBJ96, Bot95, CCH+96]:

- *Componente localização:* Corresponde à descrição espacial e topológica do GeoObjeto.
- *Componentes espaciais:* Correspondem a componentes que são por sua vez geobjetos e que estão relacionados de alguma forma ao GeoObjeto composto.
- *Componentes convencionais:* Correspondem a outros dados de natureza não espacial relacionados com o GeoObjeto.

Um **GeoObjeto** é *elementar* se não possui componentes espaciais e neste caso a sua localização é explicitamente armazenada. Um **GeoObjeto** *composto* contém componentes espaciais e sua localização pode ser explicitamente armazenada ou calculada a partir das localizações das suas componentes.

Este modelo espacial será o adotado por este trabalho.

Nível de Representação

Este nível está relacionado com as estruturas nas quais são representados os objetos modelados no nível conceitual, isto é, geocampos e geobjetos. No modelo apresentado, as diferentes representações estão estruturadas em duas hierarquias de classes cujas raízes são denominadas: **RepGeoCampo** e **RepGeoObjeto**

A partir da classe **RepGeoObjeto** são definidas subclasses que descrevem diferentes representações de objetos geométricos simples (por exemplo: linhas, polígonos, etc) ou combinações destes em estruturas mais complexas. Estes dois casos são representados por um primeiro nível de herança nas subclasses **RepComplexa** e **RepElementar**.

Uma instância da classe **RepComplexa** pode conter múltiplos elementos simples, podendo servir como estrutura de representação de todo um conjunto de geobjetos a nível conceitual. As representações de um geocampo devem garantir a especificação da função de campo definida conceitualmente.

As hierarquias de representações de geobjetos e de geocampos podem incluir representações tanto de tipo vetorial como *raster*. Por exemplo dentre as subclasses da classe **RepGeoCampo** podem ser encontradas:

- **RepTesselação**: Esta representação está baseada na subdivisão da região do geocampo em células regulares que a cobrem totalmente. Esta representação é de tipo *raster*

- **RepIsoLinhas**: Esta representação vetorial descreve o geocampo a partir de um conjunto de linhas fechadas que não se cruzam nem se tocam e que descrevem os mesmos valores na região representada para a função. Para um determinado ponto no campo, a representação calcula o valor da função por algum método de aproximação a a partir das linhas representadas.

3.2 Versionamento em SIG

Do modo mais geral, para a definição de um modelo e mecanismo de versões para SIG, devem ser considerados dois aspectos:

- O problema de como versionar dados espaciais.

- Os requisitos deste modelo no contexto de dados espaciais geográficos.

O problema do versionamento de dados espaciais depende da arquitetura do SIG adotado. No caso de uma arquitetura dual um mecanismo de versionamento para o SIG precisaria de algum tipo de versionamento nas estruturas de dados espaciais armazenadas no módulo espacial externo ao SGBD subjacente. Nesta dissertação não será considerado

este caso pois o modelo precisaria de considerações diferentes para cada uma possível estrutura diferente.

Para uma arquitetura por camadas ou estendida, onde a representação de dados espaciais é realizada sobre as mesmas estruturas do modelo de banco de dados subjacente, o versionamento destes dados pode ser realizado a partir do mecanismo fornecido pelo próprio SGBD.

Há poucas publicações sobre utilização ou definição de modelos e mecanismos de versões em SIG. Em [NTE92] são apresentadas possíveis soluções ao problema do versionamento em SIG no nível de sistema no contexto de transações. Uma transação em um SIG em geral precisa de períodos grandes de tempo, dado o volume de dados que manipula e o tipo de análise a realizar. Por outro lado são discutidas possibilidades de inclusão do versionamento tanto para o tratamento temporal quanto para aplicações de planejamento ou criação de alternativas.

[VFMa95, Bat92] apresentam aplicações na área de projeto espacial, com orientação à criação de espaços de trabalho. [VFMa95] utiliza versões como base para transações longas. [Bat92] descreve o GFIS, um gerenciador de objetos geográficos sobre um SGBD relacional que permite criar copias de dados em espaços de trabalho individual dos usuários a partir de derivações.

[MJ93b, MBJ96] abordam a possibilidade de implementar múltiplas representações de objetos geográficos sob forma de versões. Cada versão armazena uma representação distinta de um mesmo fenômeno. Se apresenta o conceito de *working layer* como unidade de trabalho tanto para modelar as diferentes representações de um objeto como para o gerenciamento de versões como alternativas, formando cenários particularmente em tarefas de planejamento. Igualmente é apresentado o uso de versões na solução de consultas que envolvem elementos temporais tanto do passado como previsão do futuro.

Do modo mais geral, podem ser identificados três aspectos em SIG onde mecanismos de versionamento podem ser utilizados como solução:

- Facilidades de projeto e planejamento espaciais.
- Manipulação de múltiplas representações para uma mesma entidade espacial.
- Modelagem espaço-temporal.

3.3 Versões como suporte ao projeto espacial

Dentre as possibilidades de uso de um SIG estão a administração, planejamento e manutenção de recursos. Uma funcionalidade importante neste tipo de aplicações é a criação de alternativas, projetos, simulações, dentre outras. O usuário deseja trabalhar sobre a

base dos dados reais, experimentar com eles, estabelecer novos relacionamentos e realizar modificações em seu banco de dados, o que pode implicar transações de longo tempo [NTE92]

Dois aspectos devem ser diferenciados: a manipulação de transações longas e o problema de alternativas, simulações e projetos. Para transações longas versões podem ser utilizadas para criar cópias do banco de dados sobre os quais realizar as transformações e posteriormente atualizar o banco original. Neste sentido, o versionamento é utilizado como uma solução a nível de sistema (capitulo 1) e deve garantir o problema de dependência entre dados, isto é, as modificações que são realizadas sobre o banco de dados original devem ser propagadas para as versões.

O problema de projeto em SIG é muito similar ao caso dos ambientes CAD/ CASE, onde existem múltiplas versões distribuídas cooperativamente entre os usuários. Nestas aplicações a relação de derivação descreve a semântica do processo de trabalho.

Alguns elementos devem ser considerados para a criação de configurações em SIG (*Configuração Espacial*) que garantam critério de consistência

- Uma configuração espacial deve ser considerada como associada a uma área alvo de trabalho, isto é, o usuário poderá transformar somente os objetos dentro de um espaço (região) predeterminado.

- A relação de composição entre objetos complexos em geral se traduz em relacionamentos topológicos de continência, isto é, o objeto composto contém "espacialmente" as suas componentes espaciais, embora possam existir outras entidades relacionadas por agregação, pois a relação de composição é dependente da modelagem do usuário.

- Para manter a consistência parcial de uma configuração espacial sobre uma área alvo, não é suficiente replicar os relacionamentos de composição, isto é, uma configuração não pode ser identificada com uma versão de objeto composto. Dentro da configuração podem existir também versões de outros objetos compostos não relacionados entre eles por composição ou agregação, mas relacionados espacialmente, isto é, contidos na área alvo de projeto.

Dado o volume de informação manipulada em aplicações em SIG, deve se evitar a criação de uma configuração que implique replicação física de muitos objetos. Pode-se considerar igualmente a existência de determinados dados espaciais que não mudam mas que pertencem a toda configuração, por exemplo que definem um contexto geográfico de referência, mapa de fundo, etc.

3.3.1 O Sistema SAGRE

Um estudo detalhado de versionamento em SIG é descrito em [VFMa95, CCH+96]no contexto do projeto **SAGRE**. O **SAGRE** é um ambiente para o trabalho cooperativo de gerenciamento de redes de telefonia, baseado no SIG *VISION*. O ambiente contempla o trabalho de operação e de projeto, isto é, a utilização do banco de dados para consultas e gerenciamento, e simultaneamente para planejamento e projeto de novas estruturas telefônicas. Os dados são organizados em um banco de dados central que contém os dados reais para operação e em bancos de dados de trabalho que contêm as versões em projeto. O sistema funciona como um esquema de gerenciamento de transações longas incluindo possibilidades de versionamento.

Um esquema de *check-out* é utilizado para a criação das versões nos bancos de dados de trabalho a partir do banco de dados central, copiando os objetos cuja geometría está incluída em uma área de projeto definida. Nestes bancos de trabalho os projetistas podem criar múltiplas versões. Ao término do trabalho, estas podem ser implantadas no banco de dados central com operações de *check-in*.

Um problema neste sistema é a manutenção da consistência entre as versões e o banco de dados central, pois mudanças neste último devem ser atualizadas nas versões em trabalho. Mecanismos de bloqueio de registros são introduzidos para garantir esta consistência. Dentre os problemas resolvidos neste sistema estão incluídos a possibilidade do compartilhamento de dados entre versões e o banco de dados central e a introdução de restrições espaciais na criação de novas versões. No entanto não fica claro como é garantida a consistência das configurações no caso de objetos não espaciais.

3.3.2 Requisitos de versões para projetos em SIG

Em geral, um modelo e mecanismo de versões para projetos em SIG deve garantir um conjunto de possibilidades que são similares às necessárias em ambientes de CAD. Estes requisitos são:

- Configurações espaciais: Criação de configurações com restrições espaciais e consistentes por composição e por relacionamentos espaciais.
- Trabalho cooperativo: Possibilidade para o trabalho simultâneo de múltiplos usuários em diferentes configurações espaciais, onde os dados utilizados nestas podem ter determinados graus de dependência. No caso dos SIG uma configuração pode envolver grande quantidade de informação, desde que associada a uma área alvo.
- Combinação de versões: Como parte da criação das configurações de trabalho para cada usuário, devem poder ser combinadas alternativas criadas por outras pessoas.

- Classificação de versões: Possibilidade de classificar as diferentes versões de acordo com critérios da semântica do projeto e associar diferentes atributos a sua descrição.
- Estruturação do banco de dados. Definição de espaços de trabalho individuais e coletivo e possibilidades de combinar os resultados desse trabalho (*merge*).
- Operações de *check-in, check-out*: Possibilidade de cópias entre espaços de trabalho através de operações de *check-in, check-out*. Estas operações devem garantir a transferência de toda a informação necessária para o contexto de trabalho, onde neste caso as restrições sobre os dados copiados podem incluir critérios espaciais de seleção.

3.4 Versionamento como solução de múltiplas representações

3.4.1 O problema das múltiplas representações em SIG

O problema da representação múltipla das entidades envolvidas na modelagem em um banco de dados está presente em diversas áreas de aplicação [NM95, MBJ96]. A motivação desta necessidade está em que em geral diferentes usuários percebem, manipulam e utilizam de maneira diferente um mesmo conceito. Em termos de modelagem conceitual de dados, esta multiplicidade se traduz em possíveis valores, esquemas ou funcionalidade diferentes em cada representação.

No caso da modelagem espacial o problema da manipulação de múltiplas representações é complexo. A forma fundamental de apresentação dos objetos georeferenciados é através de um mapa. Um mapa pode ser visto como uma camada base, onde estes objetos ocorrem e sobre o qual atuam muitas das operações que caracterizam um SIG. Sobre o mesmo conjunto de dados espaciais podem ser criados vários mapas que refletem abstrações e pontos de vista diferentes. Um objeto espacial pode ter diferentes formas de representação que correspondem a variações nos requisitos dos usuários, como diferentes resoluções, escalas, projeções e representação geográfica utilizada, dentre outras.

Partindo do modelo de dados de quatro níveis descrito na seção 3.1.3, a interação a nível conceitual de um usuário com um objeto espacial (geocampo ou geobjeto) é feita a partir de um conjunto de operações que devem abstrair as estruturas utilizadas a nível de representação e as implementações realizadas para esse objeto no nível de implementação. Igualmente as diferentes visualizações ou apresentações desses objetos devem corresponder a abstrações das representações. Por exemplo, para determinados usuários a apresentação do mapa das ruas de uma cidade como um grafo pode ser mais conveniente que como coleção de linhas poligonais.

Os seguintes aspectos estão envolvidos na manipulação de múltiplas representações em SIG:

- *Estruturas espaciais:* A descrição geométrica de um objeto espacial conceitual pode estar associada a estruturas espaciais diferentes, cada uma das quais pode ser mais conveniente para determinada funcionalidade ou para determinada abstração do usuário a nível conceitual. Podem não existir procedimentos de conversão de uma estrutura de representação para outra.

- *Propriedades espaciais:* Como parte das múltiplas representações estão incluídas não somente a variação em estruturas de representação mas também propriedades espaciais da representação, por exemplo escala ou projeção.

- *Consistência entre representações:* O fato que múltiplas representações sejam mantidas para um mesmo objeto espacial conceitual introduz o problema da manutenção da consistência entre elas. Múltiplas representações de um mesmo objeto conceitual devem evoluir simultaneamente no caso de operações de modificação da geometria ou de outros parâmetros que estejam envolvidos na representação.

- *Funcionalidade diferente:* Operações definidas para um objeto conceitual devem poder ser mapeada para funções diferentes em representações diferentes, pois algumas funções podem não ser possíveis para algumas representações. Por exemplo, a busca de caminho mínimo em uma cidade não pode ser realizada em uma representação raster e deve ser realizada em uma representação vetorial.

  [CFS+94] sugere a possibilidade de manter múltiplas variantes de uma mesma operação em representações diferentes e escolher dinamicamente a mais adequada. Por exemplo, dois objetos espaciais diferentes em um momento dado não necessariamente possuem as mesmas formas de representação. Desta forma, para realizar uma operação espacial binária entre elas é razoável tentar procurar uma representação comum.

- *Relacionamentos com outros objetos:* Operações sobre um objeto conceitual espacial podem envolver outros objetos conceituais espaciais (por exemplo, operações binárias de distância, adjacência, dentre outras). No caso em que um objeto utilize uma representação para a realização dessa operação, é razoável que a representação utilizada pelo outro objeto seja a mesma, em particular quando ambos podem estar utilizando um mesmo objeto complexo como estrutura de representação. Deve existir alguma forma de estabelecer um contexto de representação comum entre os diversos objetos incluídos em uma mesma operação.

A seguir será apresentado um caso de propriedade espacial que pode ter múltiplas representações: *escala*.

Múltiplas representações das propriedades espaciais. Escala

A mudança de escala é um processo que envolve um conjunto de entidades geográficas em uma região determinada (um mapa), que são transformadas simultaneamente permitindo obter maiores ou menores detalhes da informação, revelar ou ocultar entidades e seus relacionamentos, dentre outros. Mudanças de escala não envolvem necessariamente só mudanças nos aspectos da localização da entidade geográfica ou da sua apresentação gráfica. Pelo contrario, podem envolver mudanças na estrutura e relacionamentos das entidades.

Na figura 3.1 cada coluna corresponde a um mapa diferente com os mesmos três objetos representados em escalas distintas. Cada uma destas colunas é uma versão do mapa que agrega versões dos objetos nesta escala.

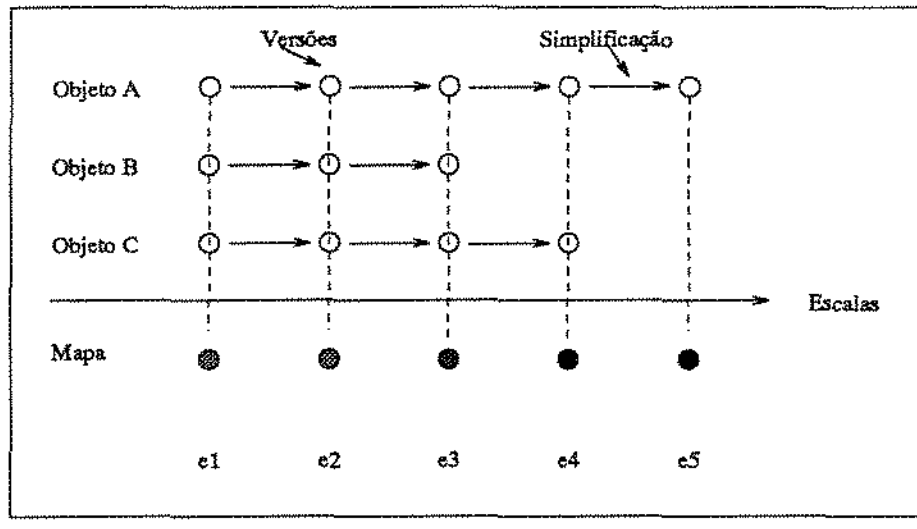


Figura 3.1: Evolução de escala

As transformações de escala podem ser vistas como processos matemáticos de natureza contínua [PD95] que transformam as entidades de um valor de escala para outro desejado ou como resultado de distintas coletas de dados em resoluções diferentes. Em qualquer destes casos devem ser garantidos certos critérios de consistência entre as representações em escalas diferentes.

Estes critérios de consistência são **topológicos** ou **métricos** [PD95]. O primeiro está associado com a manutenção dos relacionamentos topológicos espaciais entre as entidades

em diferentes escalas.A consistência métrica, por sua vez, está associada à manutenção da forma das entidades, tamanho relativo entre elas, etc.

Em [PD95] é apresentado um modelo formal para descrever as transformações de escala em sistemas geográficos. Segundo este modelo, uma mudança de escala pode provocar dois tipos de transformações:

- **Detalhe:** Corresponde a mudanças na dimensão geométrica entre duas representações. Normalmente implicam transformações nos relacionamentos com outras entidades que também mudam de escala, estando associadas à manutenção da consistência topológica. Mudanças de detalhe podem ser descritas em termos qualitativos.

- **Precisão:** Para a sua representação em um mapa, a localização e extensão de uma entidade geográfica são aproximadas pela localização e extensão de outra entidade que representa o objeto em uma escala determinada. A precisão é estabelecida pelo erro de aproximação de acordo com algum critério de distância entre essas duas localizações. A localização da representação obtida para um valor de escala deve minimizar esse erro. Quanto maior seja a resolução, menor será o erro e por conseqüência maior a precisão. Transformações de precisão são expressadas em termos quantitativos e estão associadas à manutenção da consistência métrica.

No caso das transformações de detalhe em [PD95] descreve-se qualitativamente as mudanças como resultados de processos de simplificação (figura 3.2). Estes processos são:

- **Redução:** O objeto aparece representado em uma dimensão geométrica menor depois da simplificação

- **Imersão:** O objeto não aparece representado depois da simplificação e fica imerso em outro objeto maior.

- **Preservação:** O objeto mantém a mesma dimensão geométrica depois da simplificação, mas provavelmente com outra geometria.

O caso da escala ilustra os problemas apresentados de múltiplas representações. Em primeiro lugar para duas escalas diferentes um mesmo objeto pode estar representado por estruturas diferentes. Por exemplo uma cidade pode estar representada por uma rede que descreve as ruas, por um polígono em escala menor e ainda por um ponto. Similarmente, diferentes operações definidas sobre uma cidade precisariam ser implementadas em representações diferentes: por exemplo a busca de um endereço na cidade poderá ser realizado

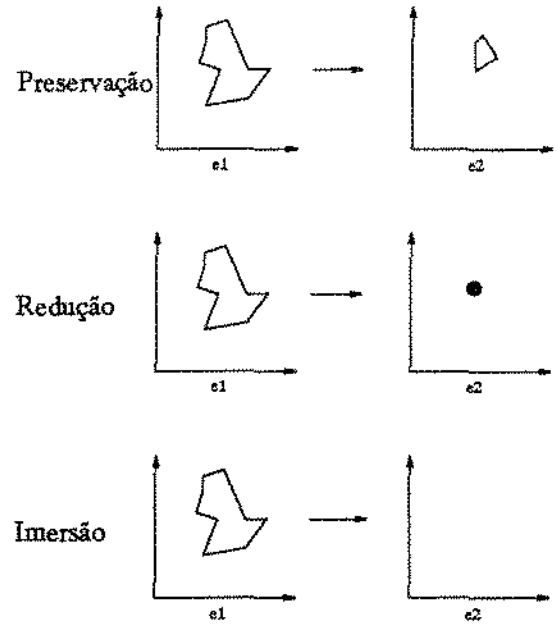


Figura 3.2: Transformações de detalhe

só sobre a rede, no entanto o cálculo da área poderia ser mais eficiente sobre o polígono. A visualização do resultado de uma operação pode implicar também escolha de diferentes representações.

A representação em escala também ilustra o problema da manutenção de consistência entre representações. Com a mudança de uma relação topológica ou a geometria de um objeto em uma representação para uma escala determinada, as outras representações em outras escalas devem garantir consistência topológica e métrica. Finalmente, as diferentes representações nos diferentes objetos devem se coordenar. Para o cálculo de distância entre duas cidades, se a representação escolhida em uma cidade é o ponto, então a outra cidade deve estar representada também por um ponto e o algoritmo de cálculo deve poder realizar a conversão de representação.

Múltiplas representações em SPRING

No Instituto Nacional de Pesquisas Espaciais foi desenvolvido um SIG (SPRING) o qual fornece suporte a múltiplas representações de entidades georeferenciadas. Em [C95] é apresentado o modelo de dados do SPRING baseado em quatro níveis (seção 3.1.3) no qual um banco de dados espacial é definido como um conjunto de planos de informação. Cada plano de informação é equivalente a um mapa e pode ser dos seguintes tipos: modelo de terreno, imagem, mapa temático, objeto cadastral ou rede. Os três primeiros tipos correspondem a especializações do conceito de geocampo antes apresentado, isto é, um geocampo é modelado diretamente em um plano de informação.

O modelo permite a definição de geobjetos, mas sua descrição espacial ou localização sempre precisa formar parte de um plano de informação de tipo cadastral ou rede, os quais são definidos como conjuntos de geobjetos, isto é, a localização de um geobjeto é dependente do plano de informação associado.

A criação de múltiplas representações em SPRING é fornecida a nível de planos de informação e estas podem ser de tipo matricial ou vetorial. Em [CÔ95] é apresentada uma ampla hierarquia de formas de representação destes dois tipos.

Exemplo: Um geocampo de pluviometria modelado como um plano de informação pode ser representado como o conjunto de amostras espaciais de valores de chuvas(vetorial), como uma grade regular(matricial), como uma imagem temática(matricial) ou como um conjunto de isolinhas(vetorial), sendo todas elas representações possíveis do mesmo conceito.

Como um geocampo corresponde sempre a um plano de informação, a criação de múltiplas representações para um geocampo é diretamente suportado. Por outro lado, as múltiplas representações para um geobjeto dependem das múltiplas representações existentes para o plano de informação associado. Geobjetos associados a um mesmo plano de informação terão necessariamente as mesmas representações.

No enfoque apresentado no modelo do SPRING o suporte para múltiplas representações está direcionado a diversas estruturas de representação da localização de um objeto georeferenciado.

3.4.2 Soluções para múltiplas representações

Diversas abordagens têm sido utilizadas para múltiplas representações. O mecanismo de herança múltipla aparece como uma possível solução deste problema. Diferentes classes podem descrever diferentes estruturas e comportamentos que posteriormente são coletados em uma classe herdeira que compõe a descrição de objetos com múltiplas representações. No entanto, esta abordagem possui os conhecidos problemas de conflitos de nomes da herança múltipla e por outro lado não garante independência e proteção para as diferentes representações, as quais precisam coexistir física e logicamente para cada possível usuário.

O uso de modelos e mecanismos de *visões* sobre SGBD pode ser utilizado também como suporte para múltiplas representações. A criação de esquemas e classes virtuais, permitiria modelar diferentes e novos comportamentos e estruturas do mesmo objeto conceitual. No entanto, visões precisam definir a forma de povoamento desses esquemas virtuais como derivação ou transformação de representações já existentes, mas não a partir de inferências ou coletas de novos dados não relacionados semanticamente com os dados dessas representações existentes. Por outro lado, o mecanismo de visões precisaria de mecanismos de conversão entre representações, que nem sempre existem.

Em outras abordagens são apresentadas soluções a partir da extensão do conceito de classe. Em [Per90] diferentes papéis (*roles*) são associados a uma classe. Um papel é definido como um conjunto de atributos e mensagens (métodos) que um objeto da classe pode adotar dinamicamente durante seu tempo de vida. As transições entre papéis podem ser controladas por regras de integridade definidas na própria classe.

3.4.3 Versões como solução a múltiplas representações em SIG

A necessidade de múltiplas representações para entidades geográficas é apresentada em [MBJ96] como a existência de diversas dimensões nas quais as representações evoluem. Estas dimensões podem descrever variações dos requisitos e pontos de vista dos usuários sobre essas entidades (por exemplo, escala, estruturas como no caso de SPRING, dentre outros). Esta abordagem permite a parametrização das representações dos objetos. Cada representação diferente é uma função das dimensões.

[Sci94] apresenta seu mecanismo de versionamento como solução para múltiplas representações. O objeto conceitual é descrito como um objeto genérico que pode ser estendido em múltiplas versões a partir de outros objetos chamados de extensões. Estes últimos ficam relacionados hierarquicamente com a componente fixa (o pai) da qual herdam comportamento e valores de atributos à partir do mecanismo de *herança por delegação.* Todos estes objetos estendidos possuem a mesma estrutura mas com valores diferentes e são chamados de versões do objeto pai. O conjunto das extensões ou versões de um objeto é considerado um espaço multidimensional, onde cada versão é determinada pelos valores das coordenadas em cada dimensão.

Baseado nesta idéia, um objeto geográfico poderia ter diversas descrições e comportamentos para diferentes representações nas diferentes dimensões mas mantendo sempre a mesma identidade. Como cada dimensão corresponderia a uma variação diferente da representação, então cada ponto no espaço multidimensional descreve uma forma diferente de representação. Desta forma, referenciando todos os objetos a partir dos mesmos valores das dimensões pode ser garantido que as representações em cada um deles seja semanticamente a mesma.

A solução ideal seria permitir versionamento de esquema, isto é, que não somente os valores das versões sejam diferentes, mas a sua estrutura e operadores. Neste caso seriam necessários mecanismos para o controle dinâmico do esquema de um objeto em cada versão diferente.

3.4.4 Requisitos de múltiplas representações.

Uma proposta de mecanismo de versões para resolver o problema das múltiplas representações deve considerar os seguintes aspectos:

- Deve garantir transparência da funcionalidade e atributos a nível conceitual sobre as representações que estão sendo utilizadas.
- Deve permitir a uma entidade manipular com flexibilidade suas diferentes representações e as representações equivalentes das outras entidades, assim como incluir dinamicamente novas representações.
- Deve providenciar mecanismos para garantir a consistência entre as diferentes representações ante operações que mudam a entidade conceitual.

3.5 Versionamento temporal em SIG

Conceitualmente, o tempo é uma dimensão essencial para compreender um modelo espacial. A inclusão de dados temporais é um requisito da análise de um processo espacial de modo a representar mudanças, derivar relacionamentos de causa e efeito e compreender a natureza e estrutura dos processos espaciais [Peu94].

Um dos objetivos mais importantes no desenvolvimento atual dos SIG é a inclusão nestes de possibilidades para o seguimento e análise das mudanças espaciais no tempo [Lan89, Lan93b, Lan93a, Peu94, NTE92].

Nos últimos tempos percebe-se um incremento importante de estudos relacionados com bancos de dados e SIG temporais [ATSS94], revelando que a representação dos fenômenos espaciais no tempo é significativamente mais complexa que somente representar fenômenos espaciais [Peu94].

3.5.1 Mudanças geográficas no tempo

A evolução das entidades no tempo pode implicar mudanças em quaisquer das suas propriedades. Do ponto de vista de modelagem, um problema importante é a decisão de quais destas mudanças transformam a entidade em outra diferente, isto é, com outra identidade.

Para [RYG94] uma entidade geográfica possui uma **propriedade essencial** que caracteriza sua identidade semântica. Qualquer mudança desta propriedade implica a destruição da entidade ou a sua transformação em outra de identidade diferente. As mudanças em outras propriedades geram versões temporais da mesma entidade.

A propriedade essencial de uma entidade é definida pelo usuário e pode corresponder igualmente a propriedades espaciais ou convencionais. Assim sendo, a definição de quais mudanças e processos fazem um objeto manter constante a sua identidade – e como resultado criar novas versões dele – deve depender da aplicação.

Por exemplo, no instante t uma loja L_1 fecha em um prédio P_1 e abre em outro prédio P_2. Neste caso, se na modelagem a propriedade essencial para a entidade *loja* é

sua localização, o evento anterior implicará que L_1 no prédio P_1 e L_1 no prédio P_2 têm identidades diferentes, isto é, não correspondem a dois estados (versões temporais) da evolução da mesma entidade.

3.5.2 Consultas temporais em SIG

A inclusão da dimensão temporal em Sistemas de Informações Geográficas estende a riqueza semântica das consultas que podem se realizar.

Langran [Lan93b] apresenta o principio de **dominância temporal** que estabelece que em geral as aplicações sobre SIG temporais possuem uma dimensão (temporal ou espacial) onde são explorados os relacionamentos, enquanto a outra dimensão é fixada. Baseado neste princípio, [Bot95] classifica as consultas em um SIG temporal como Espaciais Atemporais, Temporais e Espaço-Temporais.

Consultas Espaciais Atemporais

As consultas espaciais atemporais exploram as relações espaciais entre objetos em um tempo fixo e nelas são definidas seleção, projeção e agrupamentos espaciais. Uma seleção espacial é definida como um predicado que filtra geobjetos no domínio espacial utilizando operadores espaciais que atuam sobre a geometria ou localização dos geobjetos envolvidos. Por sua vez a projeção espacial destaca atributos dos objetos obtidos pela seleção espacial incluindo atributos convencionais e espaciais.

Consultas Temporais

Estas consultas exploram relacionamentos temporais sem considerar a dimensão espacial e sobre elas são definidas seleção, projeção e agrupamento temporais. Uma seleção temporal corresponde a um predicado que recupera geobjetos relacionados em estados temporais diferentes sem considerar seus relacionamentos espaciais. Uma projeção temporal destaca atributos dos objetos recuperados na seleção temporal, onde estes atributos podem ser temporais. O agrupamento temporal combina valores de um mesmo atributo dos diferentes objetos selecionados onde podem ser aplicados operadores de agrupamento convencionais, espaciais ou temporais.

Consultas espaço-temporais

Estas consultas permitem explorar as dimensões espacial e temporal simultaneamente. Uma seleção espaço-temporal recupera objetos em estados temporários diferentes considerando relacionamentos espaciais entre eles. Igualmente são definidas projeção e agru-

pamento espaço temporal onde neste último caso podem ser considerados operadores de agrupamento espaciais ou temporais.

3.5.3 Modelo de evolução espaço–temporal por versões

Em [Lan93b] se apresenta um modelo de evolução espaço-temporal a partir do uso de versões. A figura 3.3 descreve este modelo no qual parte-se da visão de um projeto ou banco de dados geográfico como um mapa que tem associados um conjunto de entidades geográficas. Cada entidade geográfica modelada possui um conjunto de *versões* que descrevem os diferentes estados dessa entidade no tempo. Cada versão tem associado um intervalo de tempo. Para cada entidade geográfica é definida sua linha de evolução como a sucessão das suas diferentes versões ordenadas pelos valores dos seus intervalos, que descrevem uma topologia temporal. Cada entidade evolui de uma versão a outra através de *mutações* que podem provocar a eliminação permanente ou temporária da entidade no tempo, como o exemplo do objeto B na figura 3.3.

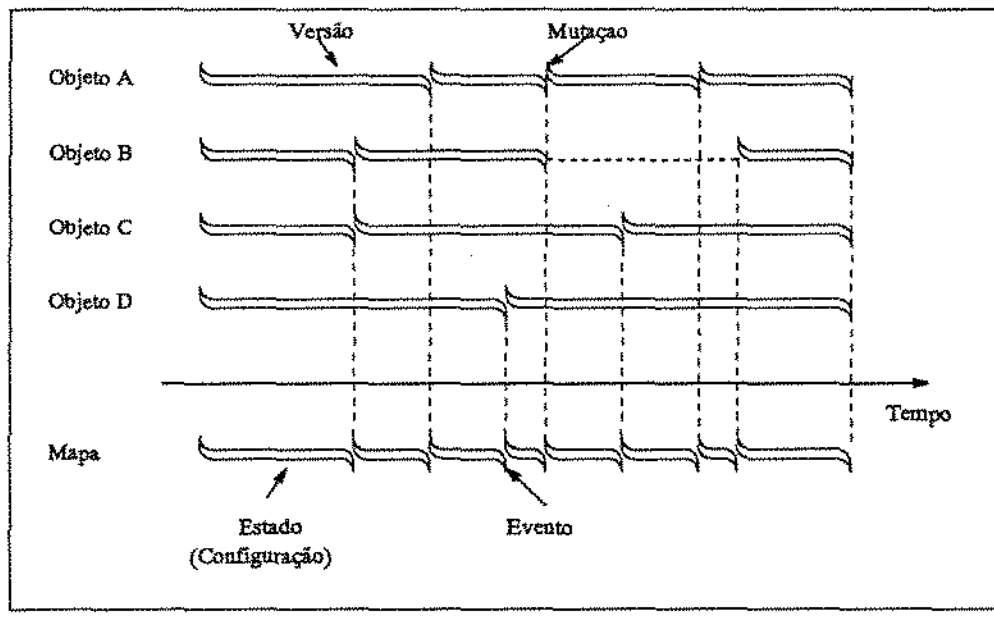


Figura 3.3: Topologia espaço temporal

[RYG94] refina este modelo introduzindo outro nível que decompõe a linha de evolução de uma entidade em outras duas linhas, onde são separadas as mutações de caráter espacial das mutações de propriedades ou atributos convencionais.

A seguir a evolução espaço-temporal da realidade modelada é definida como uma sucessão de *estados* relacionados por *eventos* que provocam mudanças nas propriedades

ou comportamento das entidades modeladas nesse mundo. Cada estado do mapa está composto por uma versão de cada uma das diferentes entidades geográficas modeladas. O estado possui um intervalo de tempo associado, no qual todas essas versões das entidades são válidas e mutuamente consistentes. Os estados são classificados por [RYG94] como **estados históricos** quando representam um estado passado e **estado ativo** quando refere-se ao estado atual, onde ocorrem as mudanças no mundo real.

Um mapa é transformado de um estado temporal para outro através de *eventos*. Um evento pode ser definido como um conjunto de processos que envolvem múltiplas entidades produzindo mutações em uma ou várias delas simultaneamente [CT95]. Neste modelo a mesma versão de um objeto pode ser compartilhada por vários estados e necessariamente cada mutação de uma entidade está associada a um evento e a uma mudança de estado do mapa. Deve-se destacar que nesta abordagem a evolução temporal é analisada no contexto do mapa que o contem, isto é, a evolução de um objeto ocorre fortemente relacionada com o contexto de evolução do mapa.

3.5.4 O trabalho de Botelho

[Bot95] apresenta a inclusão da dimensão temporal no modelo espacial de quatro níveis para o caso de geobjetos, estendendo o modelo temporal geral orientado a objetos definido em [Oli93]. Nesta proposta, os diferentes estados temporais de um objeto são mantidos através de listas de versões para cada um dos atributos. Cada elemento na lista tem associado um intervalo de tempo. O estado de um objeto para um valor de tempo determinado é obtido a partir da composição das versões dos atributos cujos intervalos associados incluem esse valor de tempo.

Nesta abordagem o tempo é considerado linear e são permitidas consultas espaço-temporais. Para o caso do processamento destas consultas a estrutura da localização do objeto é percorrida para construir a geometría do objeto para um valor de tempo determinado. Esta abordagem não considera os diferentes estados do banco de dados nem os eventos que os relacionam.

A abordagem inclui uma hierarquia de classes para a modelagem da geometría dos geobjetos.

3.5.5 Proposta de evolução temporal por eventos

Em [CT95] é apresentada uma abordagem, partindo do modelo de Langran, onde um evento é considerado composto por processos, cada um dos quais envolvem um conjunto especifico dessas entidades.

A abordagem parte de considerar que os eventos no domínio geográfico com seus processos espaço-temporais geram as mudanças de natureza geométrica e topológica .

Como resultado, novos **relacionamentos topológicos espaciais** entre entidades podem ser gerados assim como novas geometrias para as entidades

Exemplo: A mudança nos limites geográficos de uma fazenda gera uma nova representação da sua geometria e a aparição de novos relacionamentos de vizinhança com outras fazendas.

Geralmente as abordagens de modelagem temporal consideram o registro, gerenciamento e manipulação das diferentes versões das entidades modeladas mas não dos eventos que as relacionam temporalmente. De acordo com [Lan93a] os eventos devem ser tratados em SIG como entidades no mesmo nível que o resto das entidades do mundo real. Um evento espaço temporal em geral possui informações geográficas. (por exemplo, a área envolvida) e outras informações que descrevem suas particularidades.

Esses eventos podem também ser fonte de relacionamentos na dimensão temporal que descrevem a interação e evolução histórica das entidades a partir da semântica incorporada nesses eventos. Estes **relacionamentos topológicos temporais**, em geral, não puderam ser inferidos ou obtidos a partir da informação dos diferentes estados de um banco de dados espacial.

Estes relacionamentos ocorrem na dimensão temporal, isto é, são descritos como relações entre objetos em períodos de tempo diferentes a partir de um evento. Como parte da sua abordagem [CT95] apresenta uma taxonomia dos processos espaço-temporais como forma de descrição granular de um evento espaço temporal. Associado a estes processos são estabelecidos os relacionamentos topológicos temporais entre entidades. Estes processos são:

- Processos de evolução de uma única entidade.
- Processos que envolvem relacionamentos temporais entre entidades diferentes.
- Processos de evolução de estruturas espaciais envolvendo múltiplos objetos.

Processos associados à evolução de uma única entidade.

No caso mais simples, um processo espaço-temporal pode envolver uma única entidade, e descrever sua evolução independente entre dois estados. Neste caso, os processos estabelecem relacionamentos temporais triviais entre os diferentes estados da mesma entidade, mas podem ser portadores de informação semântica adicional associada à evolução das entidades que pode enriquecer o análise temporal , por exemplo, mudança nos limites geográficos de uma fazenda.

Estes processos são classificados em:

- Processos de mudanças básicas (figura 3.4): Associados à criação e destruição da entidade (*Aparecer, Desaparecer, Reaparecer*) assim como quaisquer processos que não provocam mudanças espaciais.(*Estabilidade*).
- Processos de transformação (figura 3.5) : Associados a mudanças na geometria da entidade, isto é, mudanças em tamanho e forma :(*Expansão, Contração e Deformação*)
- Processos de movimento (figura 3.6) : Associados a mudanças de posição da entidade: (*Rotação, Deslocamento*)

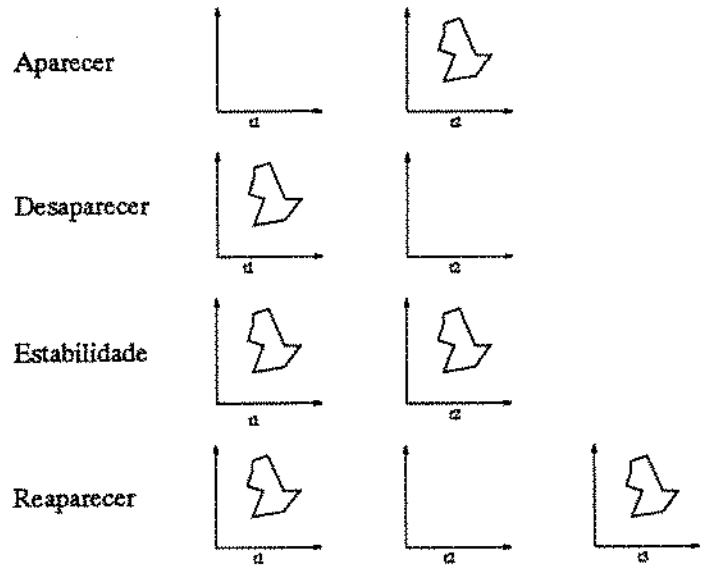


Figura 3.4: Processos de mudanças básicas

Em geral, cada um destes processos permitiria gerar um novo tipo de relacionamento binário entre versões temporais sucessivas da mesma entidade. Estes relacionamentos permitiriam descrever o tipo da mudança sobre um objeto no instante do evento que inclui o processo..

Utilizando basicamente a mesma classificação dos processos de transformação e movimento, em [EAT92] se apresenta um estudo sobre as possíveis seqüências de mudanças nos relacionamentos topológicos espaciais entre duas entidades.

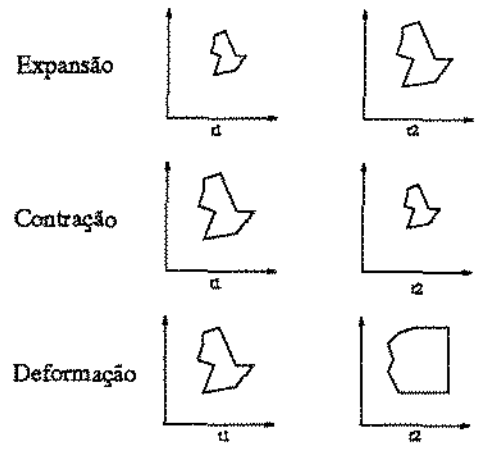


Figura 3.5: Processos de transformação geométrica

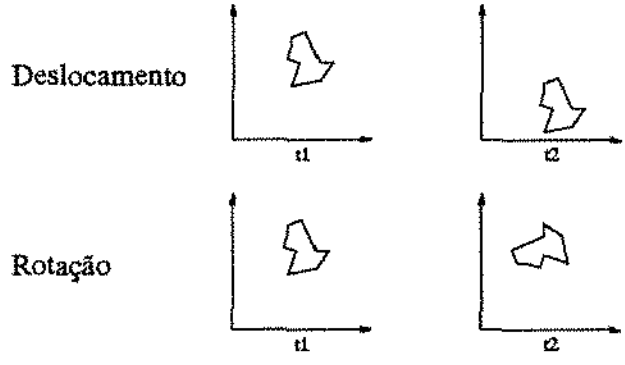


Figura 3.6: Processos de movimento

Processos que envolvem relacionamentos entre entidades diferentes

Um processo pode envolver múltiplas entidades e formar parte da descrição da evolução de todas elas, estabelecendo relacionamentos temporais entre estados diferentes de entidades possivelmente diferentes. Neste caso os relacionamentos não são triviais e dificilmente podem ser obtidos do análise dos estados das entidades em tempos diferentes.

Estes processos estabelecem relações de dependência temporal entre entidades, e impõem restrições de precedência entre elas. Eventos que envolvem múltiplos objetos formam uma rede que define uma topologia temporal e permite verificar relacionamentos de precedência o conectividade de acordo com os processos antes definidos.

Processos que envolvem entidades diferentes são classificados em [EAT92] como processos de substituição e de difusão.

Processos de substituição (figura 3.7): Envolvem sucessões de entidades diferentes de tipos comparáveis que realizam a mesma função ou ocupam a mesma posição no espaço. São classificados por sua vez em processos de sucessão e processos de permutação.

Os processos de sucessão normalmente envolvem dois objetos, o predecessor e o sucessor e o processo não implica necessariamente que o predecessor deixe de existir depois do processo.

A semântica da sucessão e muito dependente do usuário e das propriedades dos objetos consideradas essenciais, isto é, que definem a sua identidade.

Utilizando um exemplo anterior, suponha que uma loja de nome L_1 sofre uma mudança de localização do prédio L_1 ao prédio L_2. Neste casso duas interpretações são possíveis. Considerando o nome L_1 a propriedade essencial da loja, a mudança de prédio corresponde simplesmente a um processo de movimento do mesmo objeto como foi anteriormente descrito. Pelo contrario, se a localização é considerada a propriedade essencial do objeto, o processo de mudança de localização descreve uma sucessão entre dois objetos diferentes: o prédio L_1 e o prédio L_2, os quais por sua vez sofrem transformações em atributos não espaciais, neste caso o nome.

Os processos de permutação podem ser vistos como dois processos de sucessão simultâneos.

Processos de difusão (figura 3.8): Estes processos estão associados à transferência de propriedades entre entidades. Alguns deles estão associados à criação de novas entidades a partir de uma já existente e nesse caso são chamados de **reprodução** quando as novas entidades são de natureza diferente ou de **produção** quando as novas entidades recebem as mesmas propriedades . Por sua vez os processos de **transmissão** estão associados à difusão de propriedades de uma entidade para outras já existentes.

Por exemplo, o processo de cultivo em uma fazenda F_2 em um tempo t_2 das sementes produzidas em outra fazenda F_1 no tempo t_1 anterior.

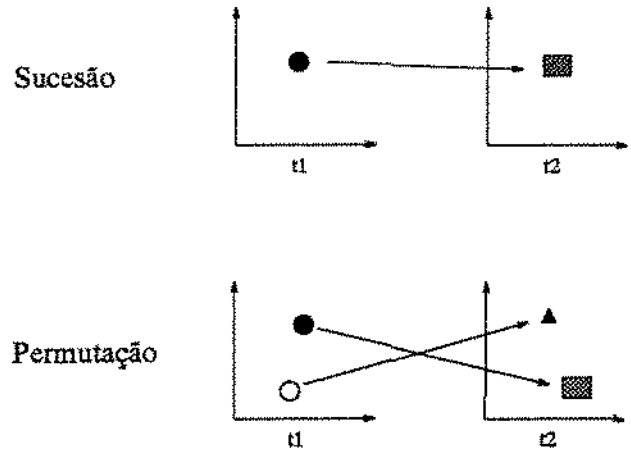


Figura 3.7: Processos de Substituição

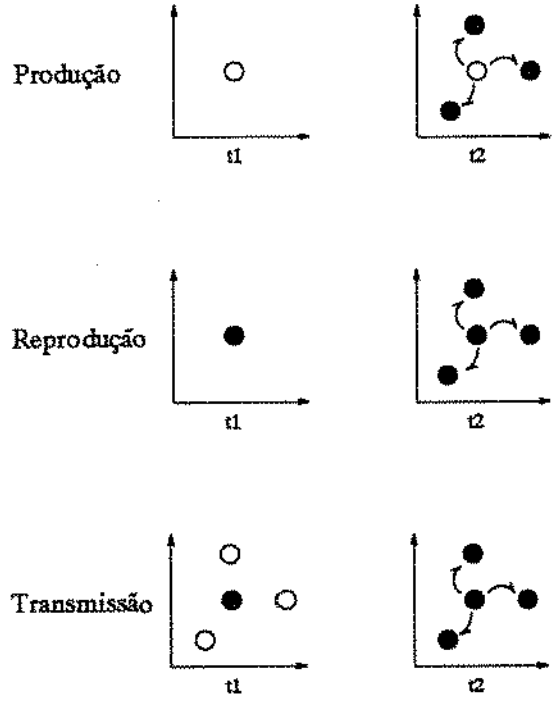


Figura 3.8: Processos de difusão

Processos de evolução de estruturas espaciais envolvendo múltiplos objetos

Estes processos (figura 3.9 estão associados à evolução das estruturas espaciais dos objetos, isto é, os processos não transformam a geometria do objeto, senão que as mudanças ocorrem dentro de outras componentes espaciais que formam parte do objeto e são por ele delimitados. Estas componentes muitas vezes são por sua vez objetos bem identificados, mas possuem relações de dependência espacial (e provavelmente semântica) com o objeto que as contém.

Por exemplo, o relacionamento temporal de entre a geometria dos municípios de um estado e sua redefinição depois de um processo de reforma administrativa mantendo-se os municípios dentro dos limites do estado.

São classificados em processos de união, divisão ou realocação.

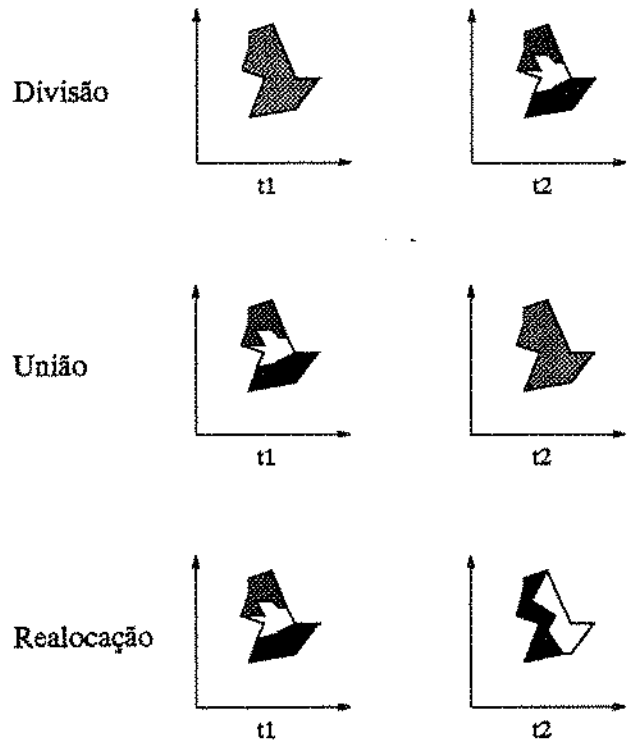


Figura 3.9: Processos de evolução da estrutura espacial.

Novos relacionamentos topológicos temporais

Como resultado da taxonomia [CT95] propõe cinco novos tipos de relacionamentos na topologia temporal. Estes relacionamentos são: Evolução, Sucessão, Produção, Reprodução e Transmissão. O primeiro (Evolução) é associado a versões diferentes da mesma entidade enquanto o resto define relacionamentos entre entidades diferentes.

3.5.6 Requisitos de versões para a modelagem temporal em SIG

Para garantir a modelagem temporal o mecanismo de versões proposto deve considerar os seguintes aspectos:

- Possibilidades para representar a evolução temporal de cada objeto espacial a partir das suas versões.
- Manutenção dos diferentes estados temporais do banco de dados, isto é, o registro de todos os eventos que produzem alguma mudança em alguma entidade modelada. Cada evento deve possuir informação sobre a sua semântica, incluído o conjunto de processos que vão acontecer durante esse evento.
- Manter explicitamente relacionamentos entre entidades diferentes em estados temporais diferentes. Estes relacionamentos devem ser estabelecidos por algum processo associado a um evento.
- Possibilidades para modelagem de tempo não linear. Este requisito implica a busca de algum critério de consistência de configurações temporais para manipular varias versões de um mesmo objeto no mesmo valor de tempo.

3.6 Resumo

Este capítulo apresentou os problemas do tratamento de versões em Sistemas de Informações Geográficas (SIG). Inicialmente foram introduzidas as principais características dos SIG, suas possíveis arquiteturas e especificidades na modelagem de dados. A seguir foram introduzidos os possíveis usos de versões em SIG: seu uso em ambientes de projetos, manipulação de múltiplas representações em SIG e manipulação temporal. Finalmente foi apresentado um modelo de evolução espaço-temporal baseada em eventos adotado neste trabalho. Em todos estes casos foram definidos requisitos de um modelo de versões para a solução dos problemas apontados.

Capítulo 4

Modelo de Versões Proposto

Este capítulo apresenta o modelo de versões proposto. Inicialmente são introduzidos os conceitos básicos do modelo, com as características e recursos que deve oferecer. A seguir é descrito o mecanismo de Versões de Bancos de Dados (*Database Versions (DBV)* [CJ90, GJ94, Gan94], mostrando como deve ser estendido para dar apoio ao modelo de versões proposto.

4.1 Características do modelo proposto

O modelo proposto é orientado a objetos. Isto implica que a modelagem das entidades é feita como objetos que são instâncias de classes. As classes descrevem e encapsulam a estrutura e o comportamento das entidades através de tipos e métodos. As diferentes classes podem se relacionar por meio do mecanismo de herança pelo qual tipos e métodos podem ser herdados por novas classes, as quais são chamadas de subclasses.

O modelo espacial adotado é o modelo de quatro camadas apresentado no capítulo anterior, onde a realidade geográfica é modelada a partir de geocampos e geobjetos, segundo o paradigma orientado a objetos. Estes são descritos a partir de classes espaciais (**GeoClasses**) as quais possuem uma componente de representação espacial como parte da sua estrutura. Objetos de **GeoClasses** podem se relacionar por composição, herança ou agregação.

4.1.1 Objetos versionáveis

O modelo proposto considera a existência de um Banco de Dados (BD) composto por objetos versionáveis. Cada objeto tem múltiplos estados, descritos em diferentes versões. Todas as versões de um objeto compartilham a mesma identidade, isto é, uma versão de um objeto deve ser manipulada com flexibilidade, mas não pode ser manipulada como um

objeto independente. O fato de que objetos complexos referenciam outros objetos implica que as diferentes versões de um objeto representam diferentes relações de composição e agregação com outros objetos também versionáveis.

No Banco de Dados os objetos podem ser *espaciais* ou *convencionais*. Um objeto espacial sempre possui uma componente que referencia uma representação geométrica. Diferentes versões de um objeto espacial podem ter como componentes diferentes representações geométricas as quais por sua vez não necessariamente são versões do mesmo objeto de representação geométrica.

4.1.2 Contextos Espaço-Temporais (CET)

Embora o banco de dados não seja consistente se considerado todo o conjunto de versões (seção 2.1.3), existem subconjuntos de versões de objetos no Banco de Dados que configuram um estado consistente de forma similar a uma configuração CAD. Por exemplo, considere um BD temporal em que cada objeto pode apresentar múltiplos estados (versões) ao longo do tempo. Neste caso, um estado consistente é formado por todas as versões cujo tempo de transação seja o mesmo e cujos períodos de validade coincidam.

Segundo esta mesma idéia, o modelo considera que as múltiplas versões dos diferentes objetos são organizadas em unidades de consistência de informação chamadas de *Contextos Espaço-Temporais (CET)*. Um CET representa uma versão ou estado consistente dentro da realidade geográfica modelada no Banco de Dados e está 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 é outra. Um BD então pode ser considerado como um conjunto de CET.

Para caracterizar cada CET dentro do BD o modelo associa a cada CET um *vetor de dimensões* 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 $[t_1, r_1]$ corresponde ao conjunto de versões de objetos que tenham estado não nulo de representação r_1 no tempo t_1. O vetor de dimensões fornece a *descrição semântica* de um CET. Um CET tem identidade semântica, isto é, o seu vetor é único no conjunto dos CET existentes. Dois CET diferentes devem estar diferenciados pelo menos por um dos valores do vetor de dimensões.

Versões diferentes de um mesmo objeto representam estados semânticos diferentes e devem estar necessariamente associadas a CET diferentes. Além disso, o estado semântico da versão de um objeto em um CET deve ser consistente com a descrição semântica do CET, e portanto os valores dessa descrição semântica do CET são também propriedades das versões de objetos componentes. Isto implica que, além de caracterizar a semântica dos CET, as dimensões representam as linhas de evolução ou versionamento dos objetos.

Todas as versões de objetos em um CET são consistentes em relação a sua descrição semântica. Inicialmente, neste modelo são considerados dois critérios básicos de consistência para serem modelados através de dimensões:

- *Consistência temporal:* Associada ao tempo. O valor da dimensão temporal para um CET é um intervalo de tempo. Para este critério de consistência pode haver tantas dimensões de tempo quantas necessário, cada uma correspondendo a dimensões independentes de consistência. Exemplos padrão são tempo válido e tempo de transação.

- *Consistência de representação espacial:* Associada a aspectos de representação espacial. Pode incluir varias dimensões de descrição como escala, projeção geográfica, estrutura de representação utilizada, dentre outras. Um determinado CET, por exemplo, conterá versões de objetos em uma certa escala e projeção enquanto outro CET referenciará versões dos mesmos objetos em outra escala e projeção. Para cada CET, poderão existir algoritmos espaciais específicos. Por exemplo, o cálculo de distância entre dois objetos necessita que a escala, projeção e provavelmente a estrutura da representação geométrica desses objetos sejam as mesmas. Assim, este cálculo pode considerar a utilização de versões em um mesmo CET.

Estes critérios não são os únicos. Outros critérios de consistência podem ser incorporados ao modelo como dimensões adicionais (por exemplo, identificando versões de trabalho e permanentes)

Objetos distintos podem evoluir de maneira diferenciada para cada dimensão. Em relação a uma dimensão d_i um objeto *ob* pode ser sensível, parcialmente sensível ou não sensível.

- *ob* é **sensível** a d_i: se para quaisquer dois possíveis CET com valores diferentes para d_i, os conteúdos das versões de *ob* são diferentes.

- *ob* é **parcialmente sensível** a d_i: se existe ao menos um par de CET com valores diferentes para d_i, para os quais os conteúdos das versões de *ob* são diferentes.

- *ob* é **não sensível** a d_i se para quaisquer dois CET que diferem apenas no valor para d_i, os conteúdos das versões de *ob* são iguais.

Um objeto *ob* é **versionável** se existe uma dimensão de descrição d_i tal que *ob* é sensível ou parcialmente sensível a d_i.

Em geral, os objetos espaciais são sensíveis para as dimensões que descrevem consistência de representação (por exemplo escala, projeção geométrica) pois suas componentes de representação espacial são diferentes em CET diferentes, isto é, mudanças nos

valores dessas dimensões implicam sempre mudanças nas estruturas que descrevem uma representação espacial.

A introdução de novas versões de objetos no Banco de Dados pode ocasionar aparecimento de novos CET (por exemplo, no caso de criação de uma nova representação). Neste caso um novo CET é criado como derivação de algum outro existente.

Cada CET evolui de maneira independente embora restrições de integridade entre estados de versões em diferentes CET podem existir mas não são consideradas no modelo.

4.1.3 Relacionamentos entre CET diferentes

Muitos modelos de versões incluem o relacionamento de derivação entre versões como parte inerente, sendo mantido e gerenciado este relacionamento pelo próprio modelo. No modelo apresentado, a derivação de um CET a partir de outro não estabelece nenhum relacionamento semântico explícito, se limitando ao processo de criação de um novo CET.

O modelo permite que o usuário estabeleça explicitamente relacionamentos entre diferentes CET, incluída a relação de derivação. Para isto é necessário que os contextos possam ser manipulados como entidades de primeira ordem, isto é, referenciados e manipulados como objetos.

Exemplos de relacionamentos 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.

4.1.4 Relacionamentos de versões de objetos em CET diferentes

O fato de poder manipular os contextos como objetos de primeira ordem permite igualmente relacionar versões de objetos em diferentes CET. A motivação fundamental para a inclusão deste recurso está em permitir modelar os relacionamentos topológicos temporais apresentados na seção 3.5.5. Neste caso eventos espaço-temporais podem ser modelados como relacionamentos entre CET, enquanto no nível de processos os eventos seriam modelados como relacionamentos entre objetos em diferentes contextos.

Este recurso também pode ser utilizado para relacionar versões de objetos que correspondem a estruturas espaciais diferentes associadas às representações geométricas do mesmo objeto espacial em CET diferentes.

4.1.5 Comportamento das versões de objetos em CET diferentes

A introdução de versionamento de comportamento visa solucionar os problemas de múltiplas representações. Do ponto de vista da modelagem de múltiplas representações para uma mesma entidade, a solução ideal em um modelo orientado a objetos seria permitir versionamento de esquemas, isto é, versões diferentes do mesmo objeto descritos por diferentes classes com tipos e funcionalidade diferentes. A abordagem adotada nesta dissertação não considera este tipo de recurso, permitindo apenas versionamento mantendo o mesmo esquema. No entanto, é permitido associar diferentes corpos aos métodos. Os corpos dos métodos de uma classe são consistentes em relação aos CET, isto é, todas as instâncias de uma classe em um CET têm o mesmo comportamento.

Este recurso, juntado aos mecanismos de polimorfismo e herança próprios de um modelo orientado a objetos, introduz possibilidades para a solução dos problemas de múltiplas representações apresentados no capítulo anterior. Em primeiro lugar, os valores e os objetos referenciados nas múltiplas versões de um mesmo objeto podem ser diferentes, incluindo objetos referenciados de classes diferentes [1]. Além disso, o comportamento dos objetos para cada um destes poderia ser também diferente.

Sejam os CET C_1 e C_2 que diferem em relação a representação dos seus objetos e considere um objeto *ob* com versões ob_1 em C_1 e ob_2 em C_2. O modelo considera que um método m de *ob* pode ter duas versões – m_1, em C_1 e m_2, em C_2 – embora com a mesma assinatura. Cada versão de m será aplicada a uma versão diferente de representação de *ob*.

4.1.6 Espaços de Trabalho

Versões são muito usadas em ambientes de projeto. O modelo introduz o conceito de *Espaço de Trabalho (ET)* para permitir o trabalho orientado a projeto espacial cooperativo, simulações ou experimentação. 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*. Cada contexto de trabalho possui seu próprio *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 de cada CTR.

[1] Sempre que formem parte de mesma hierarquia de classes

A criação de um novo ET pode ser realizada a partir de um conjunto de CET ou a partir de outro ET existente. No primeiro caso são seguidos os seguintes passos:

1. Seleção dos CET $c_1, c_2, \cdots, c_k$ que serão usados no ET.
2. Seleção espaço-temporal dentro de cada CET c_i 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 $ct_1, ct_2, \cdots, ct_k$ copiados a partir de $c_1, c_2, \cdots, c_k$, usando as restrições espaço-temporais de (2) (*check out*).
4. Associar a $ct_1, ct_2, \cdots, ct_k$ os mesmos vetores de dimensões de $c_1, c_2, \cdots, c_k$ respectivamente.

Suponha, por exemplo, que o Banco de Dados tem definidas duas dimensões (tempo, escala) e três CET $C_1[t_1, e_1], C_2[t_1, e_2], C_3[t_2, e_1]$ com as classes para representar vegetação, solo e hidrografia e suponha um ET para estudo hidrográfico na escala e_1. Como se trata de estudo hidrográfico, apenas fenômenos relativos à hidrografia são considerados. Neste caso, o ET é criado selecionando C_1 e C_3 e a seguir somente os objetos da classe hidrografia os quais são copiados para dois CTR, T_1 e T_2 que são criados dentro do ET. Finalmente a T_1 e T_2 são associados os *vetores de dimensões* de C_1 e C_3 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 T_1 e T_2.

A operação de criação de um ET e_1 a partir de outro ET e_2 é similar. Um ET é similar a um BD de trabalho em situação de projeto.

O resultado do processo de projeto em um ET pode ser re-incorporado ao Banco de Dados com a operação de *check in*. Esta operação atualiza os CET que compartilham os mesmos vetores de dimensões com os CRT do ET. Se o CET com o mesmo vetor de dimensões não existe, então um novo é criado. Novos objetos criados no processo de projeto são também incorporados ao CET atualizado.

A operação de *check in* será detalhada na seção 4.3.3.

4.1.7 Relacionamentos de dependência entre contextos

Para o trabalho cooperativo de diferentes usuários em seus ET, podem ser estabelecidos relacionamentos de dependência entre versões de um mesmo objeto em diferentes contextos, tanto CET ou CTR. 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 de trabalho.

O estabelecimento do relacionamento de dependência entre dois CTR de dois ET diferentes pode ser realizado de forma tal que todos os objetos que pertencem à intersecção das Extensões dos ET sejam automaticamente definidos como dependentes.

Este recurso pode ser utilizado para modelar a propriedade de um objeto não ser sensível para uma dimensão, estabelecendo relações de dependência entre as versões do objeto em CET que possuam o mesmo valor para essa dimensão.

4.1.8 Consultas sobre contextos

Com a existência de contextos (CET como CTR), novos tipos de consulta podem ser introduzidos no banco de dados. Como resultado de uma consulta poderão ser obtidos tanto objetos isolados como contextos. Um predicado da consulta pode agora incluir duas partes opcionais:

1. Condições sobre um conjunto de contextos a partir dos valores nos seua vetores de descrição ou dos relacionamentos entre contextos.(Por exemplo, seleção do contexto em que a dimensão t_1 tenha valor 30-40)

2. Condições sobre as versões de objetos dentro de um conjunto de contextos selecionados, incorporando nessas condições possíveis relacionamentos de natureza espacial e temporal.

De acordo com a complexidade introduzida pelos contextos, as consultas são assim classificadas em:

- As que envolvem um único contexto.
- As que envolvem múltiplos contextos.
- As que envolvem múltiplos contextos e seus relacionamentos.

Exemplos serão vistos no próximo capítulo.

4.1.9 Formalização do modelo

Esta seção descreve o modelo mais formalmente.

Um *Banco de Dados Espacial Multiversão* B é definido como $B = (S, O, C, T, D)$ onde:

- S: Esquema do Banco de Dados.
- $O = O_E \cup O_C$: Conjunto de objetos versionáveis e não versionáveis componentes de B, onde:

  - O_E: Conjunto de objetos espaciais. Estes objetos se caracterizam por possuir uma componente *loc* que descreve sua representação geomêtrica.
  - O_C: Conjunto de objetos convencionais.

- C: Conjunto de Contextos Espaço Temporais.
- T: Conjunto de Espaços de Trabalho.
- $D = \{D_1, D_2, \cdots, D_n\}$ é o conjunto de *dimensões de descrição* de contextos do banco de Dados.

Um *objeto versionável* $o = (oid_o, V_o) \in O$ é um par formado pelo identificador do objeto denotado por oid_o e um conjunto finito $V_o = \{v_1, v_2, \cdots, v_n\}$ de versões.

Um *Contexto Espaço-Temporal (CET)* $C_E \in C$ é definido como $C_E = (Cid, d_E, Vers_E)$ onde:

- Cid é o identificador do C_E, sendo o contexto considerado como um objeto não versionável do Banco de Dados, $C_E \in O$
- $d_E = (d_1, d_2, \cdots, d_n)$ é a *descrição semântica* de C_E , onde , $d_i \in D_i$
- $Vers_E \subset \bigcup_{o \in O} V_o$ é o conjunto de versões de objetos associadas a C_E. $Vers_E$ contém exatamente uma versão de cada objeto em O. Esta versão em alguns casos pode ser nula. As versões em $Vers_E$ devem corresponder a estados de objetos compatíveis semanticamente com d_E.

Um *Espaço de Trabalho* E_T é definido como $E_T = (A_T, S_T, O_T, C_T, D_T,)$ onde:

- A_T: é a *área alvo de* E_T. Corresponde à região geográfica de trabalho.
- O_T: é a *Extensão de trabalho de* E_T. $O_T \subset O$.
- S_T: é o esquema de E_T, tal que $S_T \subset S$.
- C_T: é o conjunto de Contextos de Trabalho de E_T
- D_T: Corresponde ao conjunto de *dimensões de descrição do Espaço de Trabalho* $D_T = (D_T^1, D_T^2, \cdots, D_T^k)$ onde $D_T \subset D$.

Um *Contexto de Trabalho* C_{TR} em um espaço de trabalho E_T é definido como $C_{TR} = (Ctrid, E_T, d_{TR}, Vers_{TR})$ onde:

- *Ctrid*: Identificador de Contexto de Trabalho. Um contexto é considerado como um objeto não versionável do Banco de Dados, tal que $C_{TR} \in O$
- E_T: Espaço de trabalho ao qual C_{TR} está associado.
- $d_{TR} = (d^1_{TR}, d^2_{TR}, \cdots, d^k_{TR})$ corresponde à descrição semântica do Contexto de Trabalho, onde , $d^i_{TR} \in D^i_T$.
- $Vers_{TR}$: Conjunto de versões tais que se $v \in Vers_{TR}$ é versão do objeto *ob* então:
  - $ob \in O_T(E_T)$.
  - Se $ob \in O_E$ (é um objeto espacial) então a geometria associada ao estado de v está incluída espacialmente em A_T.
  - $Vers_{TR}$ contém apenas uma versão de cada objeto *ob*.

Finalmente uma versão V_i^{ob} de um objeto *ob* pode ser definida como $V_i^{ob} = (oid_{ob}, ctxid, valor)$ onde:

- oid_{ob}: identificador do objeto *ob*
- *ctxid:* Identificador do contexto Espaço Temporal ou de Trabalho ao qual V_i^{ob} está associada.
- *valor:* estado da versão.

Duas versões $v_1^{ob1} = (oid_1, ctxid_1, valor_1)$ e $v_2^{ob2} = (oid_2, ctxid_2, valor_2)$ mantêm uma *relação de dependência* quando sempre é verdade que $valor_1 = valor_2$.

4.1.10 Operações sobre o modelo

As operações sobre o modelo podem ser classificadas em:

- *Operações sobre Contextos Espaço Temporais (CET):*
  - Criação ou eliminação de CET onde a criação pode ser: por derivação desde outro CET existente ou por combinação desde dois CET existentes.
  - Definição e eliminação de relacionamentos entre CET.
  - *Check in* desde um espaço de trabalho.
- *Operações sobre Espaços de Trabalho (ET).*

- Criação de um novo espaço de trabalho a partir de uma área geográfica alvo, o conjunto de objetos da *Extensão do espaço de trabalho* e um conjunto de dimensões de descrição.
- Inserção e extração de objetos na *Extensão do espaço de trabalho*
- *Check out* sobre um ET a partir de um conjunto de CET selecionados ou de CTR selecionados em outro ET existente.

- *Operações sobre Contextos de Trabalho (CTR):*
  - Criação de um CTR a partir de outro no mesmo ET.
  - Definir relacionamento de dependência entre dois CTR em ET diferentes.
  - Eliminação de um contexto de trabalho.

- *Operações sobre objetos e versões de objetos:*
  - Criação de um objeto em um CET, em todo o conjunto de CET e em um ET
  - Leitura, modificação e remoção da versão de um objeto em um CET ou CTR.
  - Comparação e cópia de versões de objetos tanto em CET como em CTR.
  - Relacionamento de dependência entre versões de objetos em CET ou CTR diferentes.

4.1.11 Ciclo de trabalho no modelo

A figura 4.1 descreve as componentes do modelo e como os usuários trabalham sobre ele.

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, atualização e consultas. O fato de que o mundo real seja versionado introduz mais riqueza de representação e recuperação de informação.

Equipes de usuários, ou usuários independentes podem trabalhar paralelamente sobre o Banco de Dados em ET, onde são desenvolvidos e testados novos projetos, os quais finalmente podem ser incorporados ao conjunto de CET como atualizações do mundo real. Os ET igualmente podem servir de plataforma para a experimentação ou simulação a partir dos dados da realidade armazenados. Novos ET podem ser criados a partir de ET existentes.

Para o caso do trabalho em equipe, os espaços de trabalho associados a diferentes usuários dentro de um projeto podem estruturar espacial e hierarquicamente as componentes do projeto onde cada usuário está envolvido. Os relacionamentos de dependência

garantem a possibilidade do intercâmbio de informação entre os diferentes usuários garantindo também a privacidade do trabalho sobre as versões de objetos que são exclusivos do trabalho de um usuário.

O próximo capítulo apresenta um exemplo de modelagem temporal e de múltiplas representações em SIG usando este modelo de versões.

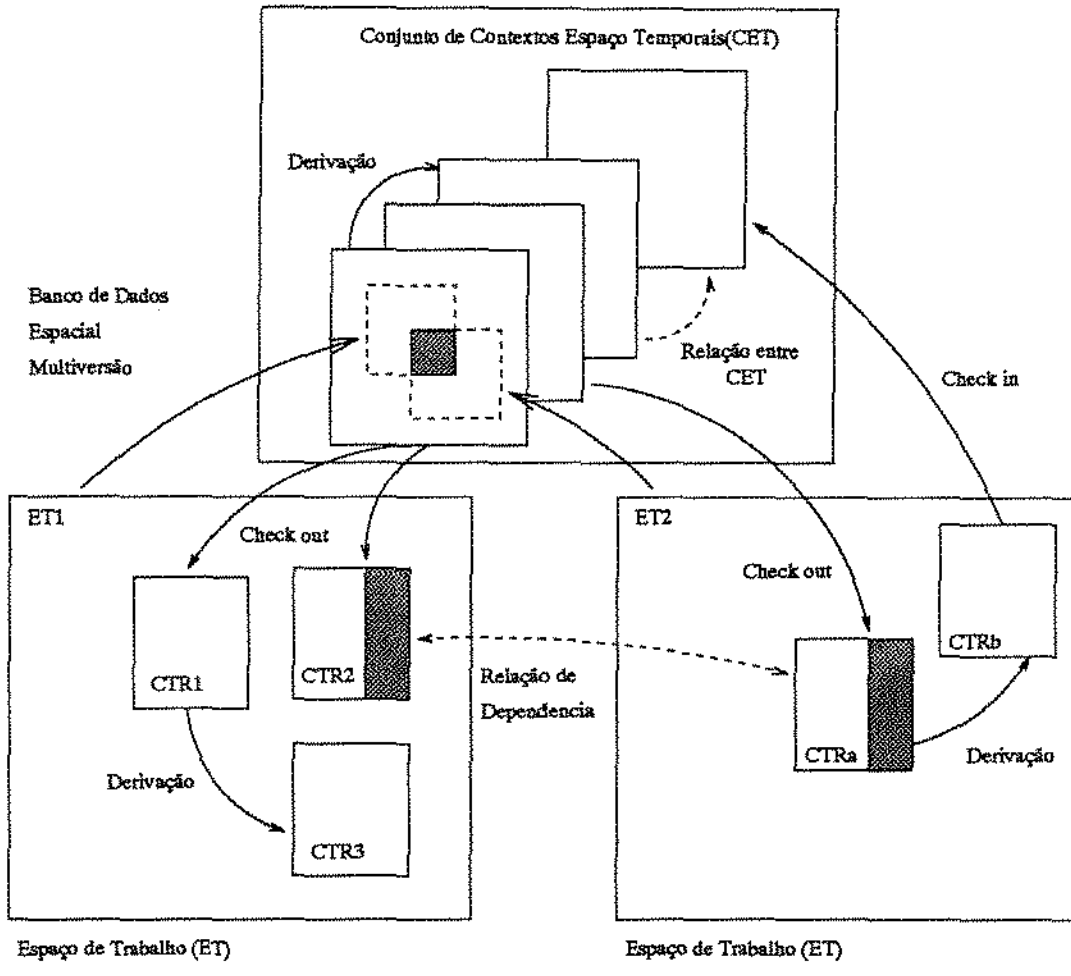


Figura 4.1: O Modelo Proposto

4.2 O mecanismo de versões subjacente

O modelo de versões descrito na seção 4.1 exige um mecanismo que o suporte. Dentre os mecanismos de versões estudados, o de Versões de Banco de Dados (*DataBase Versions*) [GJ96, CJ90, Gan94, GJ94] aparece como o mais adequado para suportar as características

descritas. Nesta seção este mecanismo é apresentado, sendo discutidas as suas limitações e propostas de extensões necessárias para permitir suporte ao modelo proposto.

4.2.1 O mecanismo DBV

A característica fundamental do mecanismo de Versões de Bancos de Dados é a distinção entre nível lógico e físico de versionamento. O nível lógico corresponde à visão do usuário enquanto o nível físico corresponde à visão de implementação.

No nível lógico um Banco de Dados Multiversão é entendido como um conjunto de Versões de Banco de Dados (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 e corresponde a uma forma natural de identificar configurações em um sistema com versionamento. A figura 4.2 mostra o modelo, considerando um conjunto de versões de banco de dados (DBV), cada uma como uma camada diferente.

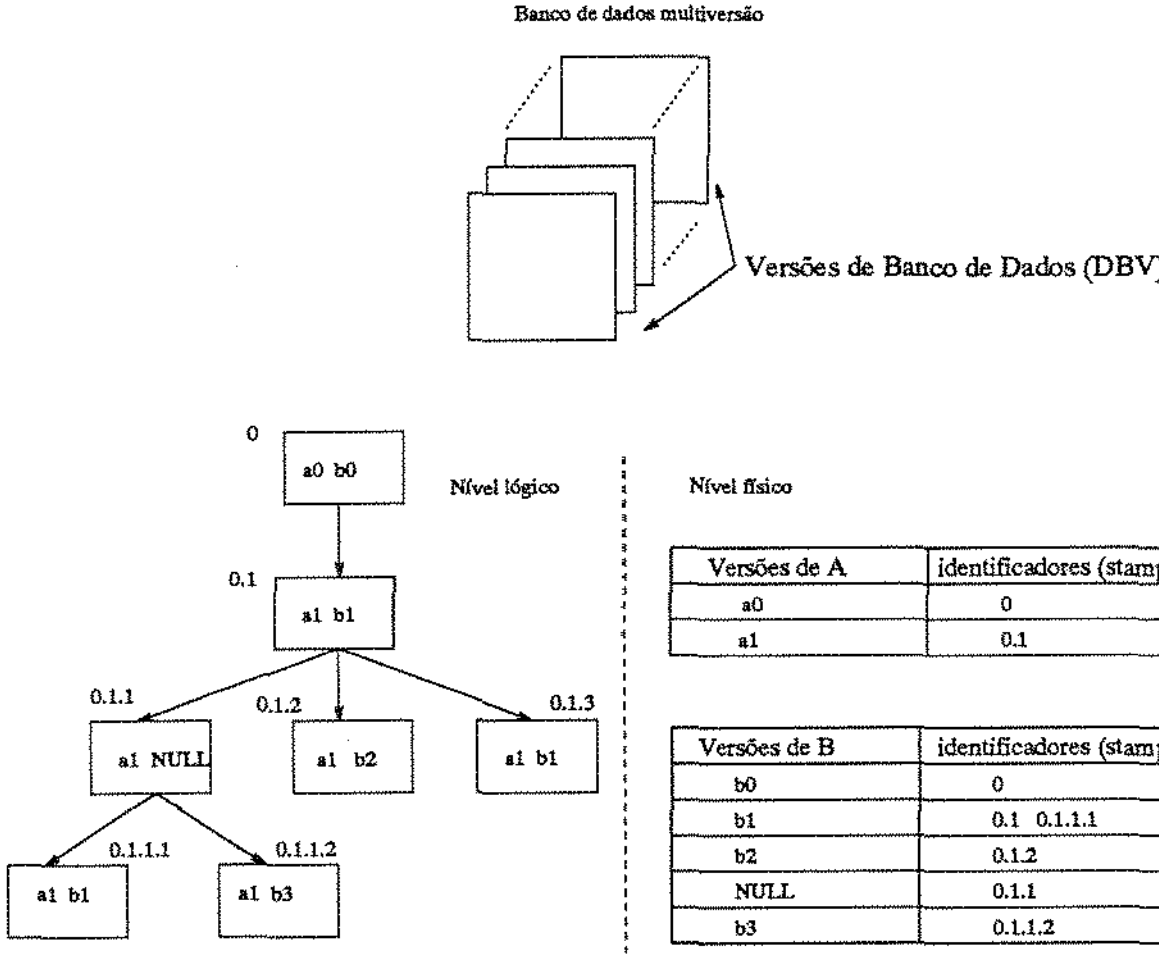
Cada DBV está composta por uma versão lógica de cada objeto existente no banco de dados. Cada versão lógica de um objeto pode ser identificada pelo par (*oid, dbvid*) onde: *oid* corresponde ao identificador do objeto e *dbvid* corresponde ao identificador do DBV no qual aquela versão ocorre. Se um objeto não existe em uma DBV então o valor da sua versão lógica é considerado nulo. O usuário manipula as versões lógicas dos objetos a partir de uma DBV. Por exemplo, a figura 4.2 representa à esquerda as diferentes DBV, cada uma com uma versão dos objetos A e B. No caso da DBV 0.1.1 o valor da sua versão de A é *a1* e da sua versão de B é nulo.

Em geral uma nova DBV é criada como uma cópia lógica ou derivação a partir de uma DBV já existente. A derivação de uma DBV δ_2 a partir de uma DBV δ_1 pode ser descrita como uma transação com duas partes:

- *Cópia:* A DBV δ_1 é copiada para uma nova DBV δ_2.
- *Versionamento:* Um ou mais objetos em δ_2 são versionados.

O identificador de uma versão (*Version Stamp*) é definido como o identificador da DBV ancestral concatenado ao número da versão derivada ordenada entre as suas irmãs. Por exemplo, na figura 4.2 as DBV derivadas de 0.1 são 0.1.1 , 0.1.2 e 0.1.3.

No nível físico, cada objeto está 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 se estabelecer explicita ou implicitamente:



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 4.2: O Mecanismo DBV

- *Explicitamente:* A tabela de associação estabelece o relacionamento entre uma versão física e versões lógicas, armazenando os identificadores de DBV cujas versões lógicas compartilham a mesma versão física. Quando uma versão lógica é modificada e compartilha a mesma versão física com outras versões, então uma nova entrada na tabela de associação deverá ser criada para a nova versão física resultante da modificação.

- *Implicitamente:* A versão lógica de um objeto *ob* em uma DBV δ_1 é compartilhada com todas as versões lógicas de *ob* nos DBV derivados de δ_1, até que *ob* seja modificado por versionamento em alguma DBV derivada de δ_1. Desta forma, quando o identificador de uma DBV não está explicitamente associado a uma versão física na tabela de associação, pode-se interpretar que compartilha o mesmo valor físico com sua DBV ancestral, o qual por sua vez também pode compartilhar o valor com sua ancestral e assim por diante.

Na figura 4.2 as versões lógicas 0.1 e 0.1.1.1 do objeto B compartilham explicitamente a mesma versão física, neste caso *b1*, enquanto as versões lógicas 0.1 e 0.1.1 do objeto A compartilham implicitamente a mesma versão física *a1*.

Para o gerenciamento do modelo é necessário manter a nível físico a *Árvore de derivação de DBV* que é representada na figura 4.2.

O algoritmo para o acesso a uma versão lógica de um objeto *oid* em uma DBV com identificador *dbvid* é:

Algoritmo 1

1. *Buscar o identificador* **dbvid** *na tabela de associação do objeto oid.*

2. *Se existir, a versão física dessa entrada na tabela corresponde à versão lógica* **(oid, dbvid)** *procurada.*

3. *Se não existir, determinar o identificador* **dbvid'** *da DBV ancestral e repetir para ele os passos do algoritmo.*

Este mecanismo garante consistência parcial, pois cada DBV pode ser vista como um estado consistente do mundo real. A manutenção da consistência é obtida de maneira natural pois a criação de uma nova DBV, ocorre a nível lógico sem replicação física de informação. Novas versões físicas são criadas somente no processo de evolução do banco de dados permitindo manter estados físicos comuns o maior tempo possível. A divisão dos níveis lógico e físico permite adicionalmente transparência para o trabalho sobre as diferentes DBV.

A desvantagem fundamental do modelo está na sobrecarga associada ao acesso a cada versão de um objeto, embora esta sobrecarga seja diferente para cada objeto em função do seu grau de versionamento. Algumas operações (como eliminar uma DBV) são particularmente custosas pois implicam percorrer todo o banco de dados para reestruturar a tabela de associação de cada objeto persistente.

4.2.2 Operações básicas do mecanismo DBV

[GJ96] apresenta o conjunto de operações básicas do mecanismo DBV. As operações em geral podem ser divididas nas que atuam sobre uma DBV e as que atuam sobre um objeto versionado.

Operações sobre DBV

Estas operações atuam de maneira geral sobre toda a DBV.

- *Criação de uma DBV* : A criação de uma nova DBV pode ocorrer de duas formas:
  - Como derivação de outra DBV existente: Neste caso todas as versões lógicas da DBV fonte são replicadas na nova DBV criada.
  - Como DBV vazia: Neste caso é criada uma DBV na qual a interpretação de todos os objetos no Banco de Dados será vazia.
- *Eliminação de uma DBV ou um conjunto de DBV:* A eliminação de DBV no nível lógico deve ser interpretada como a eliminação das versões lógicas que ela contém. As DBV eliminadas são marcadas como tal na árvore de derivação mas não são eliminadas fisicamente. No entanto, de tempos em tempos o banco de dados deve ser reprocessado para a eliminação dos identificadores de DBV em cada um dos objetos e a reestruturação das tabelas de associação. Esta operação é muito custosa, pois precisa do processamento de cada objeto no banco de dados, embora seja necessária quando a evolução da banco de dados tenha gerado muitos identificadores de DBV inúteis e mantidos fisicamente.
- *Comparação de duas DBV:* Esta operação indica se as versões lógicas dos objetos em duas DBV são as mesmas. Em geral, esta operação fornece informação sobre quais objetos estão presentes em uma DBV e não em outra, e quais são as diferenças entre dois objetos que aparecem nas duas DBV.
- *Combinação (Merge)* : Esta operação está associada à criação de uma nova DBV a partir de mais de uma das DBV existentes. A operação de combinação é difícil

de automatizar pois em geral precisa de decisões do usuário sobre quais versões de objetos em cada uma das DBV deverá ser incluída na nova DBV. Simplificando o problema para o caso de duas versões ancestrais, [Gan94] propõe automatizar parcialmente a combinação, copiando para a nova DBV aqueles objetos que são iguais nas duas versões. Para o resto dos objetos o usuário deve tomar as decisões. Esta abordagem utiliza a operação de comparação definida anteriormente.

Operações sobre objetos

Operações sobre objetos podem ser divididas em operações que atuam sobre um objeto versionável e as que atuam sobre uma versão do objeto em uma DBV determinada. Estas últimas precisam dos dois argumentos *(oid, dbvid)*

As operações sobre um objeto são:

- *Criação de um objeto:* Esta operação pode ser interpretada como a criação (inserção) de uma versão lógica do objeto em cada uma das DBV já existentes. Inicialmente o estado dessas versões em todas as DBV é o mesmo, mas cada uma poderá evoluir de forma independente.

- *Eliminação de um objeto:* Corresponde à eliminação de todas as versões de um objeto no Banco de dados.

As operações sobre versões dos objetos fornecem a funcionalidade fundamental do modelo de versões. Estas operações são:

- *Leitura, modificação e eliminação de uma versão:* Operação de acesso ao estado de um objeto em uma DBV

- *Comparação de versões:* Em um banco de dados orientado a objetos monoversão a relação de igualdade entre dois objetos complexos pode ter as seguintes interpretações:

  - *Idênticos:* No caso que os identificadores dos objetos sejam o mesmo.
  - *Iguais em superfície:* No 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.
  - *Iguais em profundidade:* No 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 só compartilham a estrutura do grafo de composição e os valores das suas folhas.

onde:

Igualdade em identidade → Igualdade em superfície → Igualdade em profundidade

sendo → indicador de implicação, pois em geral em um BD essas relações são válidas.

Para objetos versionados estas interpretações não podem ser aplicadas da mesma forma e em geral a relação anterior não é satisfeita. Por exemplo igualdade em superfície não vai garantir o mesmo grafo de composição de dois objetos complexos.

[GJ96] apresenta critérios de igualdade para objetos versionados segundo DBV. 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. 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.

Partindo da definição anterior definem-se dois critérios de igualdade em DBV:

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

Estes critérios permitem manipular com flexibilidade os diferentes casos de relacionamentos de igualdade entre versões.

- *Cópia de versões:* Para a cópia entre objetos versionáveis podem existir três possibilidades [Gan94]:

  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 uma 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 operações 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 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.

4.3 Mapeamento do modelo proposto para o mecanismo DBV

Esta seção descreve como realizar o mapeamento do modelo proposto para o mecanismo DBV. Embora muitas das características do modelo sejam suportadas diretamente pelo DBV, outras precisam da introdução de extensões que são apresentadas a seguir.

A hipótese básica para o mapeamento é que sempre haverá um único BD subjacente, com múltiplas DBV. A grosso modo, cada CET e cada CTR é mapeado em uma DBV. Embora um ET (com seu conjunto de CTR) corresponde a uma unidade para trabalho cooperativo, a hipótese é que este BD não esteja separado em nenhum momento do BD principal.

Suponha que o BD tem apenas duas DBV, uma das quais corresponde a um CET e outra a um CTR. As seguintes possibilidades de atualização existem:

- Inserção do objeto no CET, sempre refletido no CTR como um objeto nulo.
- Inserção de objeto no CTR refletida como nulo no CET.
- Eliminação de versões de objeto no CET, e eliminado do CTR se existe dependência entre as versões de CTR e CET.
- Eliminação da versão do objeto no CTR, eliminada do CET se existe dependência.

Modificações de estado podem ser coordenadas de forma análoga e modificações de esquema não são permitidas.

A partir desta visão geral, serão detalhadas a seguir todas as modificações necessárias no mecanismo DBV para dar apoio ao modelo proposto.

4.3.1 Versionamento seletivo

No modelo de versões proposto podem existir objetos não versionáveis, isto é, objetos não sensíveis a nenhuma das dimensões de descrição de contextos. Considerar que um objeto

não muda é equivalente a supor que ele existe simultaneamente em todos os contextos ou não existe. Por exemplo, o modelo proposto considera os CET, os CRT e os ET como objetos não versionáveis incluídos no banco de dados e cujos estados não mudam mesmo que mude o contexto onde o usuário esteja trabalhando.

A um nível de granularidade menor, pode acontecer que apenas alguns dos atributos de um objeto sejam sensíveis às dimensões enquanto outros não mudam nunca. Por exemplo, [RYG94] chama de atributos essenciais na modelagem temporal àqueles que não mudam no tempo e que descrevem a identidade semântica de um objeto.

O mecanismo DBV considera que todos os objetos são versionáveis e possuem uma interpretação para cada DBV. Objetos não versionáveis do modelo proposto podem ser representados no DBV como objetos multiversão com o mesmo estado das suas versões lógicas nas diferentes DBV. No entanto esses objetos ou atributos de objetos podem ser representados a nível físico de uma maneira mais simples e eficiente para o acesso, sem a sobrecarga de processamento introduzida pela estrutura dos objetos do modelo DBV.

O mecanismo DBV deve assim ser estendido para incluir objetos não versionáveis e objetos com apenas alguns atributos não versionáveis. Isso pode ser realizado identificando, na estrutura de um objeto multiversão, atributos *não versionáveis* e *versionáveis*. Estes últimos continuariam representados como o conjunto de versões físicas junto à tabela de associação. A figura 4.3 mostra a estrutura de dois objetos A e B utilizando esta extensão ao DBV. O objeto A possui atributos não versionáveis os quais são representados separadamente dentro da estrutura do objeto. O objeto B, totalmente versionável, continua com a estrutura do modo definido no DBV. Um objeto não versionável é representado como um único estado, no modo tradicional em Bancos de dados Orientados a Objetos.

A partir de agora o termo DBV sub-entende que o conceito original de DBV está estendido com a separação entre atributos e objetos versionados e não versionados.

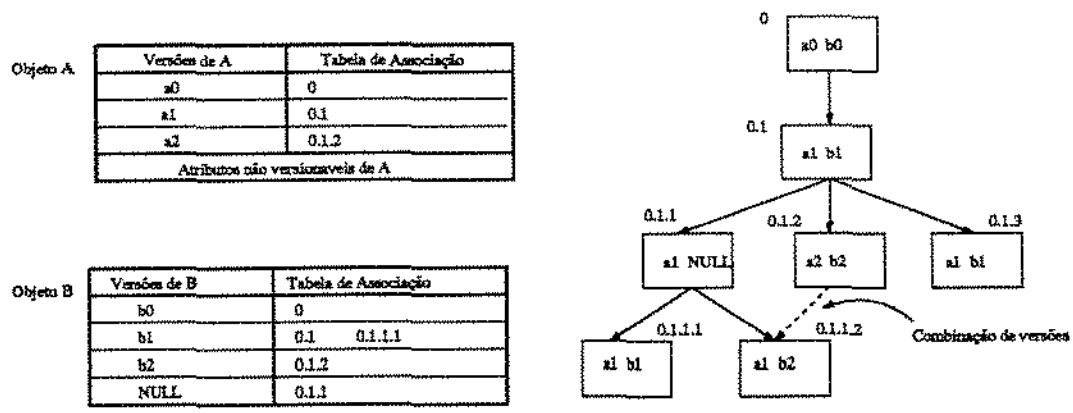


Figura 4.3: Nova estrutura dos objetos

4.3.2 Contextos Espaço Temporais

Um CET pode ser associado de maneira natural a uma DBV estendida. No modelo proposto os CET são representados como objetos que possuem como atributos um vetor de dimensões e uma referência à estrutura, no modelo subjacente, da DBV associada. Qualquer operação definida no modelo sobre um objeto CET deve implicar seu mapeamento em operações sobre a DBV associada.

A figura 4.4 apresenta três objetos CET com vetores de dimensões $C_1[t_1, e_1]$, $C_2[t_2, e_2]$, $C_3[t_1, e_2]$, 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 esta modelagem permite que um mesmo CET possa estar associado em momentos diferentes a diferentes DBV.

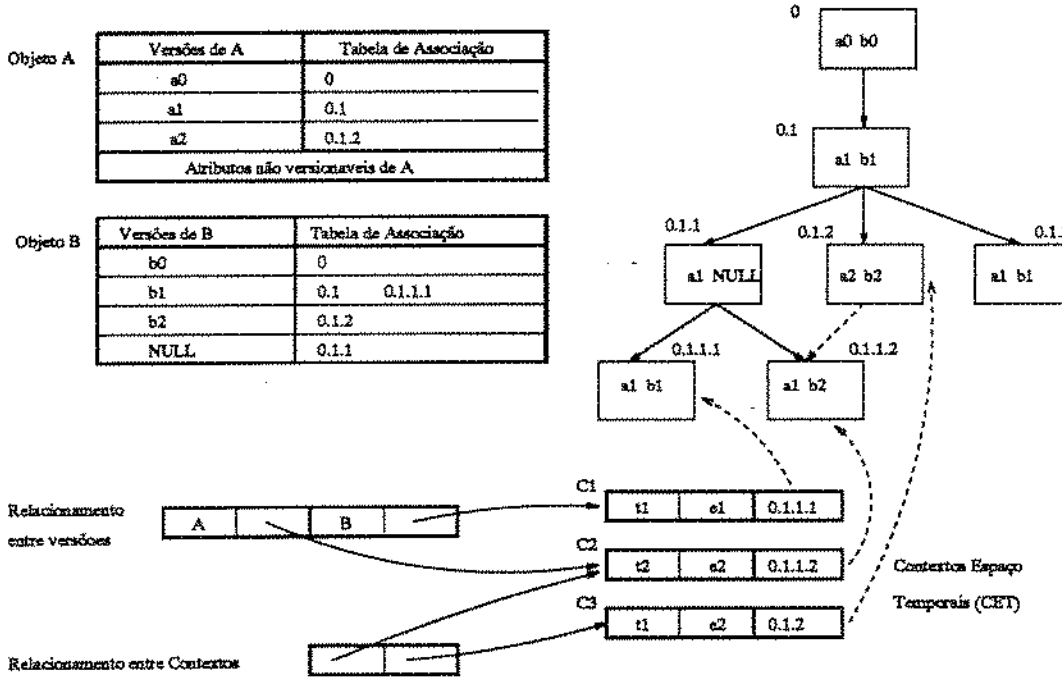


Figura 4.4: Contextos Espaço Temporais e Relacionamentos

As operações de criação por derivação e a eliminação de um CET podem ser mapeadas para as operações equivalentes em DBV. A criação de um CET exige além disso a criação de um objeto CET associado. No caso da eliminação, os objetos CET são destruídos, enquanto as estruturas das DBV são marcadas para sua eliminação em um processamento posterior, onde a árvore de derivação de DBV e as tabelas de associação dos objetos sejam reprocessados.

Combinação de CET

Além de criação por derivação, um CET pode ser criado por combinação a partir de outros contextos. Para permitir a combinação de CET (mapeada em combinações de DBV) são analisados a seguir alguns problemas e soluções.

A proposta de combinação de DBV apresentada em [Gan94], considera que a DBV resultante é criada a partir da cópia das versões daqueles objetos que são iguais nas duas DBV ancestrais, deixando ao usuário a decisão sobre o resto das cópias. Um dos problemas desta solução é que o usuário, além de ser o responsável pela decisão, tem que garantir que os relacionamentos de composição e agregação entre as versões escolhidas sejam consistentes. Este problema pode ser resolvido, no modelo proposto, utilizando operações de cópias de versões em profundidade, isto é, a cópia de uma versão de um contexto para outro reproduzindo também os relacionamentos de composição e agregação a partir do seu grafo de composição. No entanto, conflitos podem aparecer quando dois grafos de composição de objetos diferentes em contextos (e por tanto DBV) diferentes possuem um mesmo objeto em comum. Neste caso, o primeiro objeto copiado ficaria inconsistente.

A solução aqui proposta para resolver este problema é o critério adotado em [WR95]: a combinação de n DBV é reduzida ao problema de combinar duas DBV sucessivamente. Uma das DBV fontes é definida como principal e a outra como 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.

Este critério garante a replicação consistente dos relacionamentos de composição embora seu resultado nem sempre corresponda ao desejado pelo usuário, pois a estrutura de composição de versões de objetos obtidos da DBV secundária pode incluir versões de objetos da DBV principal. No entanto, este critério pode ser muito útil no caso de combinação de DBV que não possuam objetos em comum (contextos sem intersecção).

No nível físico, o uso deste critério de combinação de DBV implica modificações na estrutura da *Árvore de derivação de DBV*, a qual é transformada em um *Grafo de derivação de DBV* com estrutura de Grafo Acíclico Orientado (*DAG*).

Para o acesso à versão física de uma versão lógica **(oid,dbvid)** o algoritmo 1 (pág 65) é modificado segundo [WR95], ficando com os seguintes passos:

Algoritmo 2

1. *Buscar o identificador* **dbvid** *na tabela de associação do objeto oid.*
2. *Se existir, a versão física dessa entrada na tabela corresponde à versão lógica* **(oid, dbvid)** *procurada.*

3. *Se não existir, determinar o identificador dbvid' da DBV ancestral e repetir novamente os passos 1 até 3 para o novo identificador* **dbvid'** *até encontrar uma DBV ancestral* **dbvid'** *com seu identificador explicitamente armazenado em alguma entrada da tabela de associação.*

4. *Se a versão física dessa entrada não é nula, então ela corresponde à versão lógica* **(oid, dbvid)** *procurada.*

5. *Se a versão física encontrado é nula então, determinar no grafo de derivação se existe alguma DBV ancestral* **dbvid'** *secundária. Caso exista, repetir os passos 1 e 2 para esse valor de* **dbvid'**. *A entrada encontrada finalmente na tabela de associação corresponde à versão lógica* **(oid, dbvid)** *procurada.*

Com a introdução, aqui, de combinação entre DBVs, a sobrecarga de processamento para o acesso a uma versão lógica aumenta para o caso de versões de objetos em DBV que compartilham seu estado físico com alguma das versões lógicas de uma DBV secundária ou quando o estado da DBV ancestral é nulo.

A figura 4.4 mostra o CET C_2 criado como combinação das CET C_1 e C_3. Quando o estado da versão lógica do objeto B no CET C_2 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 C_2. Como não existe, então é procurado na tabela, um dos seus ancestrais, encontrando a entrada para a DBV 0.1.1. No entanto, como o seu valor é nulo, então é realizado o mesmo processo para a DBV ancestral secundária, neste caso 0.1.2, que aparece na tabela com versão física b_2 a qual é o valor da versão lógica procurada.

Note que a inclusão de combinação entre DBV introduz apenas modificações na estrutura de derivação (agora um grafo) e no algoritmo de leitura de uma versão lógica, mas não muda o resto do modelo de armazenamento físico.

Relacionamentos entre contextos

Uma das idéias do modelo de versões proposto é a introdução de relacionamentos entre contextos, sejam CET ou CTR. Como forma de modelar estes relacionamentos, são introduzidos objetos que estabelecem ligações entre objetos CET e CTR. A semântica associada a estes enlaces será dada pelo usuário. Similarmente, para o relacionamento binário entre versões de objetos em diferentes contextos são introduzidos outros objetos que enlaçam os dois objetos de contexto e os dois objetos versionáveis envolvidos no relacionamento. Igualmente a semântica deste relacionamento será responsabilidade do usuário.

A figura 4.4 mostra um objeto que relaciona dois CET, neste caso C_2 e C_3 e um objeto que relaciona as versões dos objetos A e B nos CET C_2 e C_1 respectivamente.

4.3.3 Espaços e Contextos de Trabalho

Espaços de Trabalho(ET) e Contextos de Trabalho(CTR) são modelados como objetos não versionáveis de forma similar ao caso dos CET, permitindo a sua manipulação pelos usuários. Um objeto ET tem a seguinte estrutura:

- Um conjunto de referências aos objetos incluídos na sua *Extensão*,
- Uma representação geométrica da sua área alvo.
- O conjunto de CTR associados.

De maneira similar aos CET, os CTR são associados a DBV. Um objeto CTR tem uma estrutura muito similar a um CET: vetor de dimensões, referência à DBV associada e finalmente uma referência ao objeto ET do qual forma parte.

Uma DBV associada a um CTR é criada copiando os objetos de interesse do BD subjacente e tornando nulos os demais. A interpretação deste valor nulo, no entanto, é diferente da interpretação dada por DBV tradicionais. Aqui, este estado nulo deve ser interpretado como que *sempre* esse é seu valor, e que não pode ser modificado, enquanto o estado nulo convencional de uma versão pode mudar. Desta forma, o mecanismo DBV deve ser estendido para introduzir 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.

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.

Operações de *check out*

O modelo proposto considera duas variantes de *check out* para um ET: Uma a partir de um conjunto de n CET e outra a partir de um conjunto de n CTR em outro ET.

A operação de *check out* para um ET a partir de um conjunto de n CET pode ser mapeada em DBV como uma transação que envolve os seguintes passos:

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.

A operação de *check out* sobre um ET a partir de outro ET pode ser realizada com os mesmos passos considerando neste caso as n DBV associadas aos CTR do ET fonte. A operação de *check out* é das mais custosas do ponto de vista de seu mapeamento em DBV.

Operação de *check in*

Dentre as operações incluídas no modelo proposto está o *check in* de objetos de um ET para objetos de CET (BD permanente) que tenham o mesmo vetor de dimensões. A operação de *check in* é feita da seguinte forma:

1. Para cada versão do objeto *oid* em um CTR, $Ctr_i[V_i]$ no ET, buscar um contexto CET $C_i[V_j]$, onde $V_i = V_j$ sendo V os vetores de dimensões.

2. Se este contexto C_i foi encontrado, então mudar o estado de *oid* em C_i para o estado em Ctr_i.

3. Se o contexto não for encontrado, criar um novo contexto $C_i[V_j]$ no Banco de dados e inserir nele o objeto *oid* com o estado em Ctr_i.

Esta operação de *check in* pode ser mapeada no modelo DBV como a criação de uma nova DBV como combinação da DBV associada ao CRT (considerada como DBV principal) e a DBV associada ao CET (considerada secundaria). A nova DBV criada será então associada ao CET enquanto a DBV anterior é marcada para ser eliminada. Este processo garante uma operação de *check in* de baixo custo.

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 na qual devem ser mudados os estados *nulos permanentes* para *nulos temporários* daqueles objetos não contidos na *Extensão* do ET.

Modificação da Extensão de um ET

No modelo proposto duas operações são associadas à extensão de um ET: Inserir e Eliminar um objeto. No caso da operação inserção de um novo objeto na extensão, o mapeamento para o modelo DBV corresponderá 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.

4.3.4 Relacionamentos de dependência entre contextos

A introdução do conceito de relacionamento de dependência no modelo proposto garante que os estados de versões de objetos em diferentes contextos possam evoluir em conjunto. Um relacionamento de dependência corresponde a uma relação de equivalência, isto é, verifica-se transitividade. (ou seja, a atualização de versões que mantenham dependência é realizada em cascata)

O mecanismo DBV não permite esta dependência. Existem duas formas de evolução de uma DBV: atualização das versões dos objetos em uma DBV ou cópia de uma DBV a partir de outra. Se uma destas operações é aplicada sobre uma versão lógica que compartilha uma mesma versão física com outra versão, isto é, compartilham a mesma entrada na tabela de associação (implícita ou explicitamente), então necessariamente uma entrada nova com uma nova versão física é criada para a versão lógica transformada (Ao passo que no modelo proposto se houver dependência, a versão física é modificada e não há criação física de objeto).

Por exemplo, na figura 4.5 as versões lógicas de A nas DBV 0.1 e 0.1.2.1 compartilham explicitamente a mesma versão física, neste caso a_1, enquanto as versões lógicas das DBV 0.1.1, 0.1.3, 0.1.1.1 e 0.1.1.2 também compartilham implicitamente essa mesma versão física. Uma mudança no estado da versão lógica de A na DBV 0.1 para um novo estado a_3 provoca uma atualização na tabela de associação garantindo agora que as versões lógicas 0.1.1, 0.1.3 e 0.1.2.1 explicitamente compartilhem a mesma versão física a_1 enquanto a versão da DBV 0.1 evolui independentemente em uma nova entrada.

Para permitir o relacionamento de dependência do modelo, o mecanismo DBV precisa ser estendido para restringir essa evolução independente. Do ponto de vista lógico, relacionamentos de dependência entre versões de um mesmo objeto podem ser vistos como relacionamentos de equivalência. Desta forma, 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.

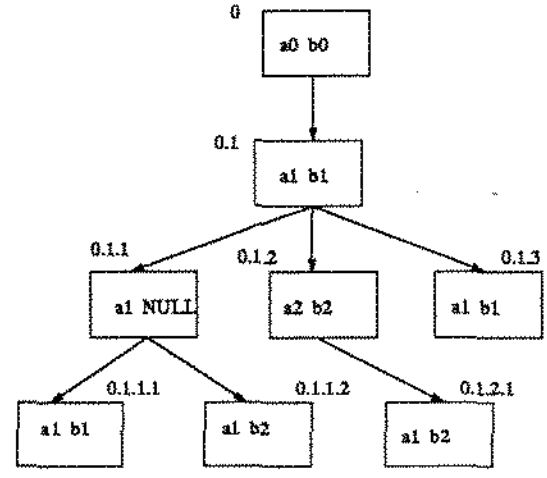
Do ponto de vista físico, é preciso garantir que todas as versões lógicas em uma mesma classe de equivalência estejam associadas sempre à mesma entrada na tabela de associação. Para isto, a tabela de associação de um objeto é modificada, sendo criadas subtabelas para cada entrada, contendo cada uma delas conjuntos de identificadores de DBV equivalentes. Todos os diferentes conjuntos de DBV em uma mesma entrada compartilham a mesma versão física. A mudança do estado de uma versão lógica em uma classe de equivalência pode provocar a criação de novas entradas na tabela de associação mas mantendo sempre todas as versões lógicas na mesma classe de equivalência na mesma entrada.

Objeto A

Versões de A	Tabela de Associação
a0	0
a1	0.1 0.1.2.1
a2	0.1.2
Atributos não versionaveis de A	

Objeto B

Versões de B	Tabela de Associação
b0	0
b1	0.1 0.1.1.1
b2	0.1.2 0.1.1.2
NULL	0.1.1



ANTES

DEPOIS

Objeto A

Versões de A	Tabela de Associação
a0	0
a1	0.1.1 0.1.3 0.1.2.1
a2	0.1.2
a3	0.1
Atributos não versionaveis de A	

Objeto B

Versões de B	Tabela de Associação
b0	0
b1	0.1 0.1.1.1
b2	0.1.2 0.1.1.2
NULL	0.1.1

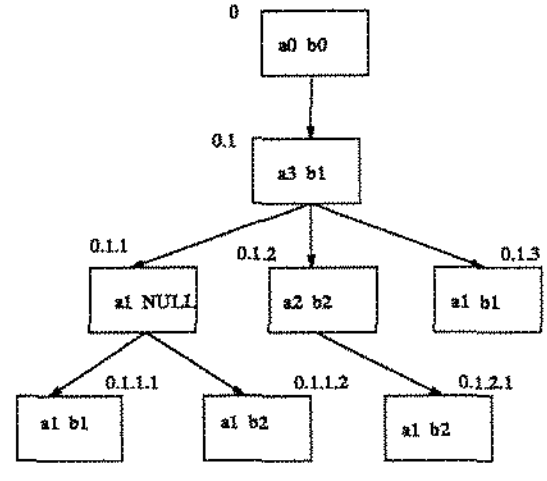


Figura 4.5: Modificação do estado de uma versão lógica

Seguindo o mesmo exemplo anterior, suponha agora (figura 4.6) que existe um relacionamento de dependência entre as versões do objeto 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, traduzido nas tabelas no lado, onde as linhas estão particionadas em classes de equivalência de instâncias.. Suponha que de maneira similar ao exemplo anterior, o estado da versão lógica de A na DBV 0.1 (CET) é mudado para um novo estado a_3. Neste caso, a tabela de associação é modificada garantindo que 0.1 e 0.1.3 continuam compartilhando a nova versão física a_3, enquanto as versões lógicas 0.1.1 e 0.1.2.1 explicitamente compartilham a mesma versão física a_1 mas em classes diferentes. A versão da DBV 0.1 evolui independentemente em uma nova entrada.

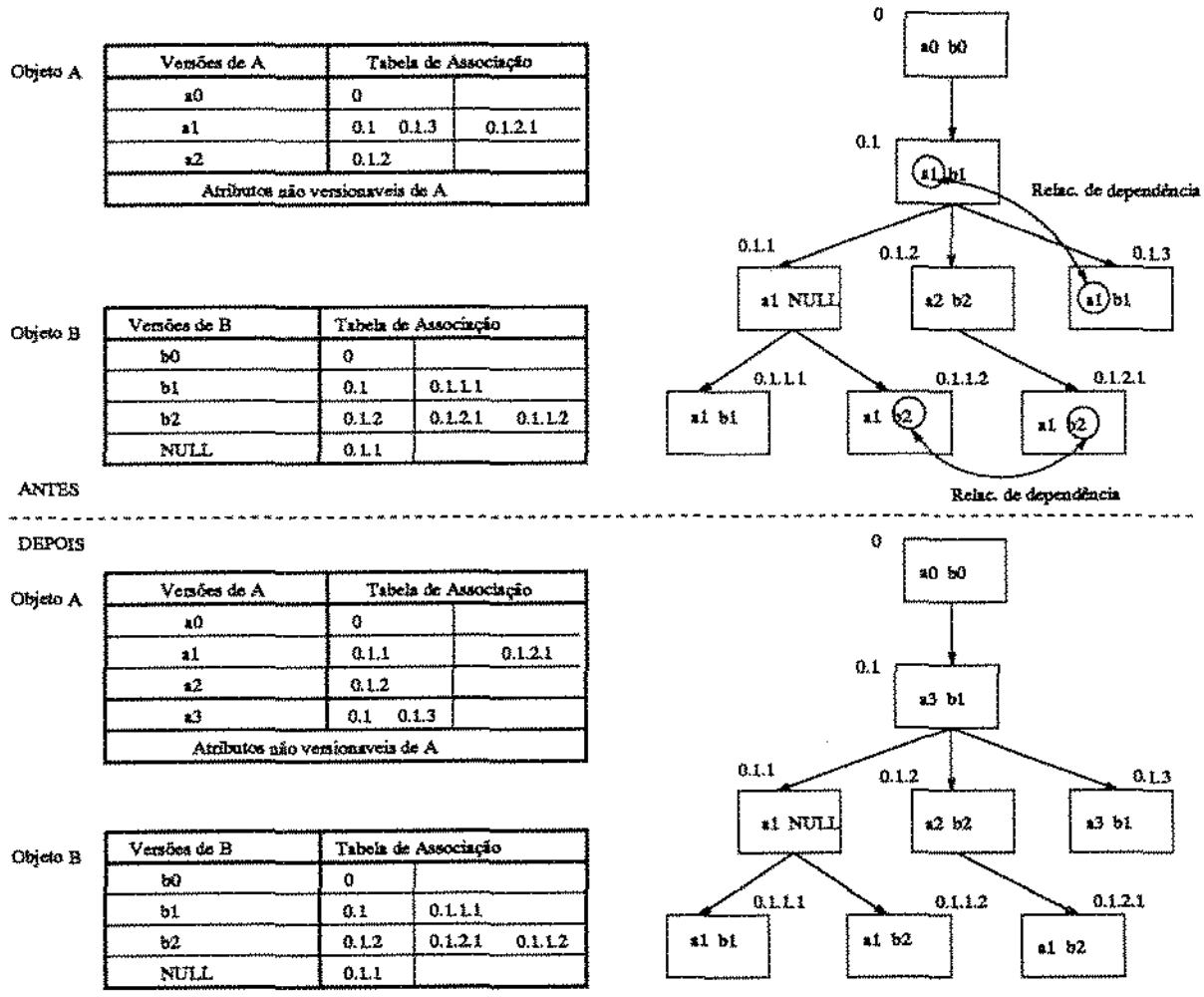


Figura 4.6: Dependências entre versões

4.3.5 Operações sobre objetos e versões de objetos

A criação de um novo objeto no modelo proposto pode ocorrer com as seguintes variantes:

- Apenas para um CET.
- Para todos os CET.
- Apenas para um CTR em um ET.
- Para todos os CTR de um ET.

A criação de um objeto no modelo DBV é realizada associando uma versão física inicial à versão lógica da raiz do grafo de derivação. Isto implica que de maneira implícita todas as versões lógicas para as diferentes DBV compartilham inicialmente um mesmo estado. Vale lembrar que tanto CET quanto CTR correspondem a DBV no banco de dados. Isso significa que a criação de um objeto em um CET/CTR qualquer, para o mecanismo DBV, vai inserir o objeto em todas as CET ou CTR no modelo. Para manter a semântica da inserção, no modelo proposto. é preciso atribuir adequadamente *nulos temporários* e *nulos permanentes* às DBV existentes. Cada uma das formas de criação apresentada para o modelo pode ser mapeada para DBV como transações que seguem os seguintes passos.

1. Criação de um objeto segundo o mecanismo DBV.
2. Associação de valores *nulo permanente* ou *nulo temporário* a determinadas versões do objeto em diferentes DBV. Esta associação dependerá do tipo de criação segundo os casos seguintes:
   - *Criação apenas para um CET:* Associar o valor *nulo temporário* às versões lógicas do objeto das DBV associadas às demais CET. Associar o valor *nulo permanente* às versões lógicas do objeto em todas as CTR (DBV correspondentes).
   - *Criação para todos os CET:* Atribui valor ao objeto na CET desejada associa o valor *nulo permanente* às versões lógicas do objeto das DBV associadas a todos os CTR e o estado inicial do objeto a todas as versões lógicas de DBV associadas às CET.
   - *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 e todos CET

- *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.

Em geral as operações sobre versões de objetos em CET ou CTR (leitura, eliminação, modificação, comparação e cópia) são mapeadas para as correspondentes operações sobre as DBV associadas a essas CET ou CTR, de forma análoga ao descrito aqui.

4.4 Resumo

Este capítulo apresentou o modelo de versões proposto, introduzindo os seus conceitos básicos (Contextos Espaço-Temporais (CET), Contextos de Trabalho (CTR) e Espaços de Trabalho (ET)). Os conceitos correspondem aos recursos para o trabalho orientado a projeto, para a modelagem de múltiplas representações e a modelagem temporal. A seguir foi descrita uma proposta de mecanismo de versões, baseada em extensões do mecanismo DBV de gerenciamento de versões. Esta extensão permite atender os requisitos dos novos conceitos introduzidos.

Capítulo 5

O modelo de versões em um SGBDOO

Neste capítulo são apresentadas extensões às componentes básicas de um SGBDOO segundo o padrão ODMG para dar apoio ao modelo de versões proposto. Estas extensões incluem novas classes predefinidas assim como novos recursos nas linguagens associadas ao SGBDOO. Finalmente são utilizadas estas extensões para apresentar uma proposta de modelagem temporal e de múltiplas representações em SIG.

5.1 Componentes do modelo

Dado que o modelo de versionamento proposto é orientado a objetos ele pode ser definido como uma extensão sobre um SGBDOO considerando o padrão ODMG de arquitetura de sistema [Ban94]. Este padrão considera a estrutura de um SGBDOO como duas componentes básicas de gerenciamento de objetos (*Object Engine*) e de manipulação em memória secundária (*Object Store*)

Sobre estas componentes são definidas linguagens de definição de objetos (*Object Definition Language*- ODL) e de consulta (*Object Query Language* - OQL), assim como linguagens de aplicações (figura 5.1). Ao longo deste trabalho serão utilizadas as linguagens e convenções do SGBDOO O_2 como base para as extensões propostas, pois é totalmente compatível com o padrão ODMG.

A linguagem de definição de objetos permite a descrição do esquema do banco de dados – classes, tipos e raízes de persistência – e incorpora as características mais relevantes do paradigma de orientação a objetos (como herança e encapsulamento). Os métodos são descritos a partir da suas assinaturas.

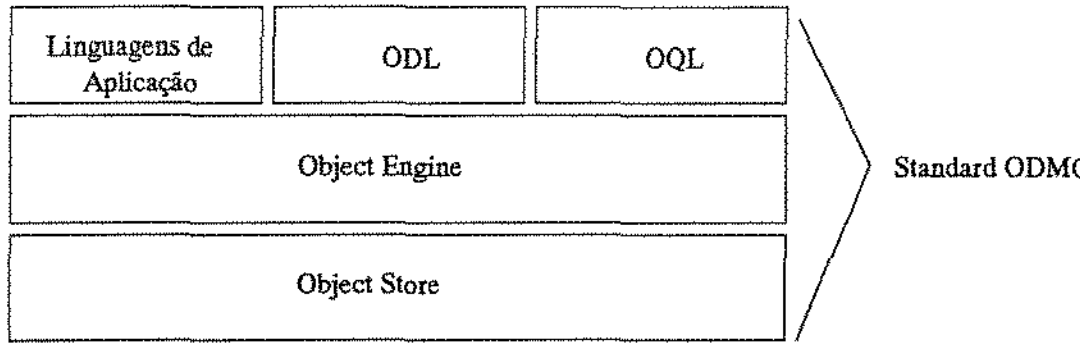


Figura 5.1: Arquitetura ODMG

A linguagem de consulta OQL corresponde a uma extensão de SQL permitindo a consulta do banco de dados e a navegação pela estrutura complexa dos objetos envolvidos desde as raízes de persistência definidas como parte do esquema na ODL.

Finalmente, o ODMG define duas linguagens de aplicação, uma interna e outra externa. A linguagem interna corresponde geralmente a uma extensão de uma linguagem de programação no contexto do SGBDOO e sobre ela são definidos os corpos dos métodos de classes descritos em ODL. A linguagem externa é uma linguagem de programação orientada a objetos que possua mecanismos de interface com o SGBDOO. Sobre estas duas linguagens são desenvolvidas as aplicações sobre o banco de dados.

O modelo de versionamento proposto estende as componentes de ODMG incluindo novos recursos para a manipulação de versões. Estas extensões incluem:

- Extensões à linguagem de definição de objetos (ODL) para incorporar às classes a propriedade de ser versionável.
- Definição de classes e objetos predefinidos, para manipular os conceitos de CET, CTR e ET.
- Extensões às linguagens de aplicação e consulta para fornecer as operações sobre versões definidas no modelo.

5.1.1 Linguagem de definição de objetos (ODL) estendida

A extensão da ODL consiste basicamente em incluir um qualificador **versionable** para indicar quais classes ou quais atributos do tipo de uma classe são versionáveis. Este qualificador pode ser aplicado sobre o tipo da classe ou sobre elementos na descrição dentro do tipo *tuple*. Dizemos que uma classe é versionável se ao menos um dos seus atributos é versionável. Por definição, uma subclasse de uma classe versionável será também versionável.

A propriedade de ser versionável poderia ter sido associada às instâncias de uma classe no momento da sua criação e não como uma propriedade genérica da classe. No entanto, consideramos que com isto perder-se-ia a vantagem da implementação de manipular mais eficientemente atributos não versionáveis.

No exemplo a seguir de uso do qualificador, os atributos *Owner* e *Location* da classe **Farm** e a classe **Company** são versionáveis. No caso, **Company** constitui o exemplo padrão (todo o objeto é versionável) enquanto que apenas dois componentes de **Farm** são versionáveis.

```
class Farm public
  type
     tuple(
            BirthYear: Date;
            BuildComp: Company;
     versionable
            Location: Geometry;
            Owner: Person;
           )
method GetValue: float;
...
end;

versionable class Company public
  type
     tuple( Name : String;
            President : Person;
            ...
          )
...
end;
```

5.1.2 Classes e objetos predefinidos

O modelo provê dois tipos de classes e objetos predefinidos: aqueles para suportar versões (por exemplo: ET) e os relativos à dimensão espacial (GeoClasses).

Classes da modelagem Espacial

Para a modelagem de objetos espaciais será adotado o modelo de quatro níveis apresentado no capítulo 4 no qual um objeto espacial tem associado um estado e funcionalidade a *nível*

conceitual que podem ser representados e implementados de múltiplas formas no *nível de representação.*

Nesta dissertação será abordado somente o caso da modelagem de GeoObjetos. Além de atributos não espaciais, um *GeoObjeto* contém uma componente de localização, a qual está associada uma forma de representação espacial. Adicionalmente um *GeoObjeto* pode conter um conjunto de componentes espaciais que também são *GeoObjetos.*

Para a modelagem de *GeoObjetos* conceituais é introduzida uma classe abstrata **GeoObject** que define a estrutura e funções básicas para a modelagem. Por sua vez a classe **GeoObjectRepr** define a classe raiz dentro de uma hierarquia de classes que estabelecem as diferentes formas de representação para *GeoObjetos*, de forma similar à apresentada em [CFS+94].

```
versionable class GeoObjectRepr public
      // raiz da hierarquia de representações de GeoObjetos.
type

...
end;

versionable class GeoObject private
 type
   tuple(
          Comp: set (GeoObject);
          location: GeoObjectRepr;
         )
method public
       // Funcionalidade de um Geobjeto no nível conceitual
        IsInside( geom: Geometry): boolean;
end;
```

GeoObjectRepr e **GeoObject** devem servir como superclasses para a definição das próprias classes dos usuários na modelagem das entidades geográficas em suas aplicações.

Seguindo o modelo de quatro níveis, a idéia da modelagem é considerar a existência de um estado e uma funcionalidade conceitual de um *GeoObjeto* no nível conceitual, os quais são refletidos para um usuário por meio da interface da classe. Esta interface está descrita por métodos incluídos nas subclasses de **GeoObject**. A declaração como *private* da componente de localização e dos geobjetos componentes tenta garantir que operações sobre estas só possam ser realizadas pelos métodos que refletem a modelagem conceitual. Sob o ponto de vista do nível conceitual dois tipos de métodos constituem a funcionalidade de uma classe da hierarquia **GeoObject**

- Métodos de acesso ao estado conceitual de um objeto e ao relacionamento com outros *GeoObjetos*, isto é, acesso as suas componentes conceituais, propriedades da sua geometria, relacionamentos espaciais (topológicos, métricos e de direção) ou outros relacionamentos próprios da aplicação.

- Métodos de transformação do estado conceitual do *Geoobjeto*. Inclui as transformações da geometria do *GeoObjeto*, de seus relacionamentos espaciais, dentre outros.

Internamente, estes métodos podem utilizar formas de representação diferentes, transparentes ao usuário. A introdução de versionamento permite que objetos das subclasses de **GeoObject** tenham múltiplas versões, em que a componente de localização esteja associada a instâncias diferentes das classes de representação (subclasses de **GeoObjectRepr**) (ou ainda a versões diferentes de uma mesma representação).

A inclusão do método *IsInside* tem como objetivo garantir que toda classe da hierarquia **GeoObject** defina um método que permita verificar se a geometria do *GeoObjeto* esta incluída em uma geometria dada como parâmetro (área alvo). Este método será utilizado posteriormente na modelagem de Espaços de Trabalho.

Classes da modelagem de versões

Como foi definido na descrição do modelo, os CET, CTR e ET são manipulados como objetos não versionáveis do Banco de Dados. Estes objetos são instâncias de um conjunto de classes predefinidas básicas que são: **StContext, WrkContext, WrkSpace** modelando os CET, CTR e ET respectivamente:

```
Class StContext private
...
method private
       EqualDescriptor(StContext);
method public
        Derive (SourceCtx: StContext);
        Merge(SourceCtx1:StContext; SourceCtx2:StContext);
        Equal(Ctx: StContext): boolean;
        CheckIn(ws:WrkSpace);
        Delete;
...
end;

Class WrkContext
```

```
 type WrkSp: WorkSpace;
...
method private
       Create( WS: WorkSpace);
       EqualDescriptor(WrkContext);
method public
        Derive (SourceCtx: WrkContext):boolean;
        Merge(SourceCtx1:WrkContext; SourceCtx2:WrkContext):boolean;
        Equal(Ctx: WrkContext): boolean;
        Delete;
...
end;
```

A classe **WrkSpace** modela um ET e estabelece o meio para o acesso aos CTR incluídos nele. Cada objeto desta classe é definido sobre uma área geográfica alvo a qual é definida como um objeto da classe **Geometry** e contém um conjunto de CTR (instâncias da classe **WrkContext**) e um conjunto de objetos que formam a sua *extensão*.

Os métodos sobre a classe **WrkSpace** permitem, dentre outros, inserção de objetos espaciais (*InsertSpatialObj*). Esta operação só pode ser realizada se a localização do objeto a ser inserido estiver contido na área alvo do ET. Esta verificação utiliza o método *IsInside*.

```
Type LstStContext = list(StContext);

Class WrkSpace private
type
   tuple( Ctxs: set( WrkContext);
          Area: Geometry;
          Extension: set(Object);
        )
...
method
    create( WrkArea: Geometry);
    InsertSpatialObj( obj:GeoObject): boolean;
    InsertObj ( obj: Object);
    DeleteObj( obj: Object);
    Checkout ( Source: SetStContext );
   // métodos de acesso aos CTR locais;
...
```

```
end;
```

A classe **System** oferece funcionalidade para o acesso ao conjunto de CET e ao conjunto de ET do Banco de Dados e para a definição do contexto ativo, o qual é um contexto (CET ou CTR) que é considerado *default* sempre que uma interpretação desse tipo seja necessária.Esta classe garante persistência de todos os CET e ET.

A inicialização de um Banco de Dados segundo o modelo de versões adotado corresponde à criação de uma instância de **System** e sua atribuição a um objeto persistente *Sys*. Esta inicialização gera um primeiro CET primário.

```
Type LstWrkSpace = list(WrkSpace);

Class System private
Type
    tuple  ( Ctxs : LstStContext);
             WkSp : LstWrkSpace);
           )
...
method
       SetActiveStCtx(ctx: StContext);
       SetActiveWrkCtx(WS : WrkSpace; ctx: WrkContext);
       GetStCtx : LstStContext;
       GetWrkSpace : LstWrkSpace;
...
end;

name Sys: System;
```

Instâncias das classes **StContext** ou de suas subclasses, são objetos normais em um Banco de Dados. No entanto, uma instância deste tipo só é inicializada como repositório de versões (novo CET), quando executado um método de derivação ou combinação (*Merge*). Esta inicialização também verifica que o novo CET incorporado não tenha o mesmo descritor de dimensões que algum dos CET já existentes. Para isto esses métodos utilizam internamente o método *EqualDescriptor*. O critério de igualdade entre dois vetores poderá ser estabelecido pelo usuário segundo a semântica do seu problema a partir da redefinição deste método. Esta inicialização também atualiza *Sys*.

Por exemplo, suponha que se deseja criar um novo CET como derivação de outro CET que ocupa a primeira posição na lista mantida em *Sys*. Note que o processo de formação de um novo CET tem duas etapas: a criação do objeto propriamente na classe **STContext** e a sua inicialização por derivação ou combinação.

```
02 StContext ctx1, ctx2;

ctx1 = new StContext;                    // Criação do objeto para o novo CET
ctx1->derive( Sys->GetStCtx->first); // Ativação do novo ET como derivado do
                                         // primeiro ET registrado em Sys
```

De maneira similar, uma instância de **WrkSpace** pode ser criada como um objeto do Banco de Dados inicializando sua área alvo (*Area*) e a sua Extensão(*Extension*). Posteriormente é ativada com uma operação de *check out* a qual cria seu conjunto de CTR locais, e o registra como ET em *Sys*.

O usuário interage com as versões a partir da seleção de instâncias das classes **StContext**, **WrkContext** ou de suas subclasses. A seguir o usuário pode relacionar as instâncias desejadas. O acesso a um CTR Ctr_i exige inicialmente acesso a *Sys* e, em seguida, seleção do ET ao qual Ctr_i pertence. Esta hierarquia de acesso facilita restrições de acesso segundo diferentes critérios, visando segurança e privacidade no uso de espaços de trabalho.

A figura 5.2 descreve os relacionamentos entre as diferentes classes predefinidas, onde **Object** é a classe genérica do O_2 que é raiz de todas as classes de um BD, assim **GeoObject** é subclasse de **Object**. Uma instância da classe **System** possui uma agregação de instâncias das hierarquias **StContext** e **WrkSpace**. Por sua vez uma instância de **WrkSpace** possui uma agragação de instâncias de **WrkContext** e os objetos versionáveis que pertencem à sua extensão estão incluídos espacialmente na área alvo associada. Estes objetos são versionados nos diferentes contextos.

Na realidade, todas estas classes de modelagem espacial e versões, devem ser entendidas como uma base para modelar uma aplicação específica. A modelagem da aplicação deve criar novas subclasses para cada um destes conceitos incorporando a semântica própria, mas reutilizando todo o mecanismo de versões subjacente. A redefinição de métodos nessas novas subclasses, deve incluir no seu corpo sempre uma chamada ao método original da superclasse para manter em todo momento a consistência com o modelo de versões encapsulado na funcionalidade das classes básicas apresentadas. Em particular, para modelar a descrição semântica dos CET e CTR, subclasses de **StContext** devem ser definidas em cada aplicação incorporando em sua estrutura atributos que correspondam às dimensões de descrição de contexto. Mais adiante será mostrado a utilização destas classes.

Finalmente, relacionamentos semânticos entre CET diferentes podem ser modelados com classes que referenciam objetos que representam CET ou CTR. Igualmente, para modelar relacionamentos entre versões em diferentes contextos, basta manter referências aos objetos das versões relacionadas e aos objetos de contexto respectivos.

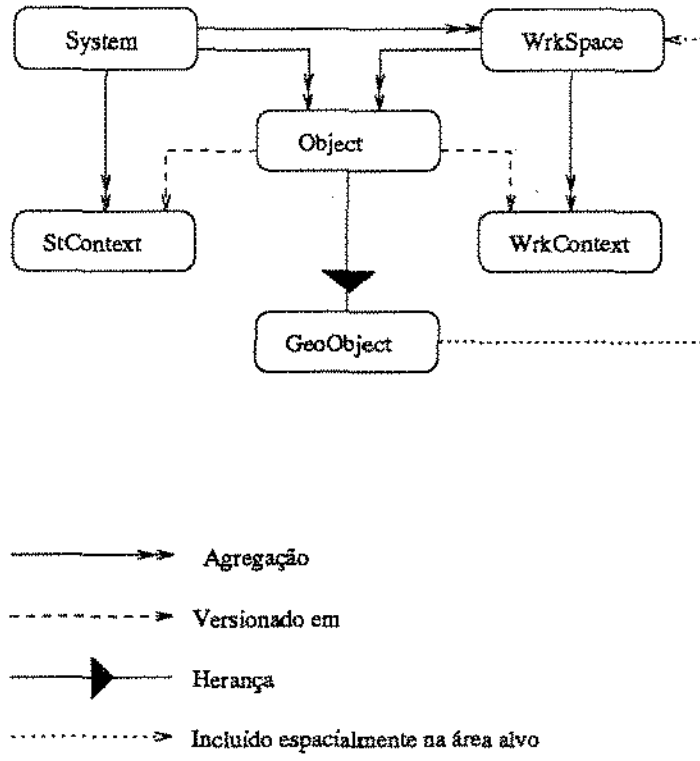


Figura 5.2: Classes predefinidas do modelo

5.1.3 Linguagem de aplicações estendida

Para modelar as operações que atuam sobre conjuntos de versões de objetos, é necessário estender a linguagem de aplicações interna. As extensões introduzidas permitem o acesso e manipulação das versões dos objetos utilizando as instâncias das hierarquias **StContext, WrkContext.**

As extensões introduzidas estão relacionadas a três aspectos: criação de objetos versionáveis, acesso às versões de objetos nos contextos e versionamento de corpos de métodos.

Criação de um objeto multiversão

A criação de um objeto multiversão pode ser realizada de várias formas:

- A criação do objeto somente no CET ou CTR no qual o usuário está trabalhando (*default*). A sintaxe tradicional de criação de objetos será utilizada para esta variante

```
O2 Farm c = new Farm;
```

- A criação do objeto somente em um CET ou um CTR que não é o *default*. Neste caso é introduzida uma nova sintaxe incluindo uma variável *ctx*, das hierarquias

StContext, **WrkContext**, que referencia o CET ou CTR sobre o qual o objeto é criado.

```
O2 Farm c = new Farm [ctx];
```

- A criação do objeto em todos os CTR de um ET. É utilizada a mesma sintaxe do caso anterior, mas a variável *wk* utilizada referencia um objeto da hierarquia **WrkSpace**.

```
O2 Farm c = new Farm[wk];
```

- A criação do objeto com o mesmo estado inicial em todos os CET registrados em *Sys*.

```
O2 Farm c = new Farm in all;
```

Estas formas de criação determinam os estados iniciais das versões de um objeto. No entanto, posteriormente novas versões podem ser associadas a um objeto em outros contextos, respeitando sempre a semântica dos ET:

Manipulação das versões de objetos

De maneira similar ao caso da criação, o acesso à versão de um objeto em um contexto é realizado usando uma indexação com uma variável de contexto.

A sintaxe introduzida é:

```
obj[ ctx ]...
```

onde:
- obj : referência a uma instância de uma classe versionável.
- ctx : referencia a uma instância das hierarquias **StContext, WrkContext**

Esta notação permite adotar várias formas de referência aos recursos de uma classe (atributos e métodos). Será considerada a existência do contexto ativo ou *default* em todo momento. Um contexto é selecionado usando métodos aplicados sobre *Sys*:

1. Referência ao recurso *rcs* de *obj* no contexto ativo. obj → rcs
2. Referência ao recurso *rcs* de *obj* no contexto *ctx*. obj[ctx] → rcs
3. Referência ao recurso *rcs* de *self* no contexto ativo. rcs
4. Referência ao recurso *rcs* de *self* no contexto *ctx*. self[ctx] → rcs

Nos casos 2 e 4, se *rsc* corresponde a um método então durante a sua execução será considerado a *ctx* como contexto ativo. Estas formas de referência, podem ser combinadas para a formulação de referências compostas de versões. A expressão:

```
obj1[ctx1]->obj2[ctx2]->p
```

é uma expressão de caminho que "parte" do contexto *ctx1* do *obj1*, referencia o recurso *obj2* no contexto *ctx2* e tem como terminal o valor p (um atributo ou método). A notação deve ser lida em analogia à notação

```
obj1->obj2->p
```

exceto que as referências são resolvidas em diferentes contextos.

O indexador de versão permite navegar por todo o espaço de versões com grande flexibilidade, embora em geral, esta não deve ser a forma de trabalho usual sobre os contextos. O natural deve ser trabalhar sobre um contexto ativo, navegando entre versões de objetos mutuamente consistentes. Por exemplo:

```
Sys->SetActiveStCtx(ctx);
obj->rcs->p;
....
```

A partir da ativação de um contexto, todas as referências que não utilizam explicitamente o indexador de versão são interpretadas como referências à versão no contexto ativo. Isto permite que uma seqüência de código possa ser aplicada sobre as versões em contextos diferentes apenas mudando previamente o contexto ativo.

Algumas operações adicionais serão necessárias para completar as operações do modelo sobre as versões de objetos:

- **Null(obj, ctx)**: Permite testar se o estado do objeto **obj** no contexto **ctx** é nulo.
- **ValueAssign(obj, ctx1, ctx2)**: Cópia em superfície da versão do objeto referenciado por **obj** no contexto **ctx1** para a sua versão no contexto **ctx2**.
- **DeepValueAssign(obj, ctx1, ctx2)**: Similar à anterior com cópia por valor, em profundidade.
- **DeepStrongAssign(obj1, ctx1, obj2, ctx2)**: Cópia forte em profundidade, da versão do objeto **obj1** no contexto **ctx1** para a versão do objeto **obj2** no contexto **ctx2**.
- **Share(obj, ctx1, ctx2)**: Define relação de dependência entre as versões do objeto **obj** nos contextos **ctx1 e ctx2**.

- **UnShare(obj1, ctx1, ctx2)**: Desfaz uma relação de dependência entre duas versões.
- **Equal/StrongEqual(obj1, ctx1, obj2, ctx2)**: Operadores booleanos de relacionamentos de igualdade fraca e forte em superfície entre duas versões.
- **DeepEqual/ StrongDeepEqual(obj1, ctx1, obj2, ctx2)**: Operadores booleanos de relacionamentos de igualdade fraca e forte em profundidade.

Versionamento de métodos

No SGBDOO O_2, um método pode ter várias versões identificadas por um nome. De maneira estática, um usuário pode selecionar na sua sessão de trabalho qual versão do método definir como ativa.

Por exemplo, partindo da definição da classe **Farm**, para o método *Getvalue* poderia ser definida uma versão *v1* :

```
method body GetValue: float
version v1 in class Farm
{
...
}
```

Generalizando esta idéia, diferentes versões do corpo de um método de uma classe versionável poderiam ser associadas dinamicamente a contextos diferentes com uma sintaxe similar para o acesso a versões em um contexto:

```
Classe[ contexto CJ ].metodo M  = nome de versão
```

interpretada como "no contexto **CJ** o método **M** está associado à versão indicada".

Utilizando o mesmo exemplo, poderia ser definido dinamicamente que *v1* é a versão do método *GetValue* utilizada no contexto *ctx1* para as instâncias de **Farm**.

```
Farm[ctx1].GetValue = v1;
```

5.1.4 Linguagem de consulta (OQL) estendida

Consultas em OQL convencional utilizam variáveis que percorrem coleções de objetos. Em sua forma mais simples, uma consulta sobre uma coleção avalia o predicado da consulta sobre cada objeto da coleção a partir da variável associada a essa coleção. De forma mais geral, várias variáveis sobre coleções diferentes podem ser usadas simultaneamente,

criando-se um produto cartesiano como combinação dos objetos das diferentes coleções. O predicado da consulta é avaliado sobre cada elemento desse produto cartesiano

A extensão da linguagem de consulta para o modelo proposto deve permitir que diferentes versões de objetos possam ser utilizadas como parte de uma consulta. Isto é realizado usando o mesmo conceito de variável de contexto para indexar a versão desejada.

Como, no modelo, os contextos são objetos, coleções de contextos podem ser utilizadas nas consultas. Assim, o produto cartesiano entre uma coleção de objetos versionáveis S_o e uma coleção de contextos S_c, permite, percorrer todos os estados das versões dos objetos em S_o para cada um dos contextos em S_c. Com a introdução, nos predicados, de referencias indexadas, podem ser combinadas diversas coleções de contextos com diversas coleções de objetos, permitindo expressar condições acerca de relacionamentos de versões de objetos de classes diferentes em contextos diferentes.

O predicado pode referenciar uma versão em um contexto particular, mas o resultado de uma consulta deve recuperar objetos como um todo e não versões de objetos.

Por exemplo, suponha a existência de coleções persistentes *TheFarms* e *TheCompanies* das classes **Farm** e **Company** definidas em 5.1.1. A consulta "Selecionar as fazendas construídas no ano 1950 que em alguns dos CET do Banco de Dados tenham "X" como proprietário" pode ser expressada como:

```
SELECT h
FROM h in TheFarms ,d in Sys->GetStContext
WHERE  (h.BirthYear = 1950) and
       (h[d]. Owner.name = "X")
```

Note que a consulta navega sobre todos os contextos CET através da aplicação de *GetStContext* sobre *Sys*.

No exemplo a seguir, a indexação é utilizada em uma referência composta para recuperar objetos associados a versões diferentes do mesmo objeto. Suponha a consulta "Quais os presidentes em um CET das companhias que nesse mesmo CET têm como nome "Y" "

```
SELECT c[d]->President
FROM c in TheCompanies ,d in Sys->GetStContext
WHERE  c[d]->Name = "Y"
```

No caso que uma referência composta não utilize o indexador será adotado sempre o CET *default* no Banco de dados nesse momento.

A indexação de uma referência a um objeto não versionável será permitida por generalidade embora sempre implique uma referência à mesma versão do objeto.

5.1.5 Arquitetura do modelo.

Duas alternativas de arquitetura do modelo de versões podem ser propostas. A primeira alternativa considera que a componente *Object Engine* do modelo ODMG é estendida com a funcionalidade básica para a manipulação de objetos segundo o modelo DBV estendido (figura 5.3). Nesta variante, as linguagens OQL e ODL estendidas poderiam ser processadas por compiladores que as mapeariam diretamente sobre a *Object Engine*. A funcionalidade das classes predefinidas poderia ser implementada utilizando recursos dessas linguagens estendidas ou utilizando diretamente a funcionalidade do *Object Engine* com versões que em geral seria oferecida como uma interface de programas de aplicação (*API*).

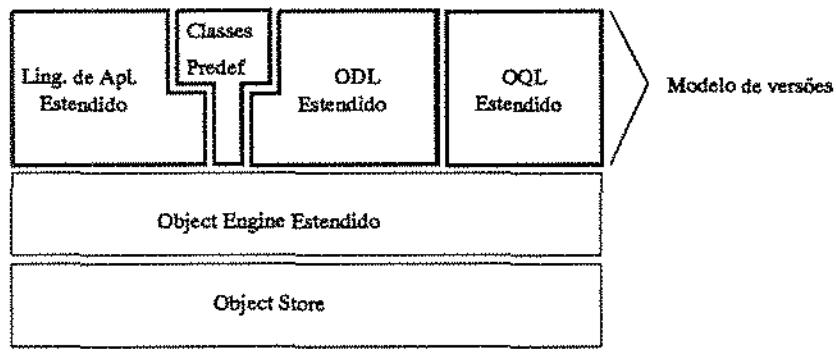


Figura 5.3: Arquitetura sobre um *Object Engine* Estendido

A segunda variante considera o caso onde a funcionalidade do DBV estendido deve ser mapeada diretamente sobre o próprio SGBDOO base. Neste caso, os compiladores das linguagens devem converter as especificações para as linguagens do SGBDOO (figura 5.4), isto é, como um conjunto de objetos e classes. A implementação das classes predefinidas deve interagir com essas especificações.

5.2 Uso do mecanismo proposto em SIG

Esta seção mostra como o modelo de versões proposto pode ser usado como solução para o gerenciamento de versões temporais e de representação. O texto utiliza as extensões de OQL e ODL propostas.

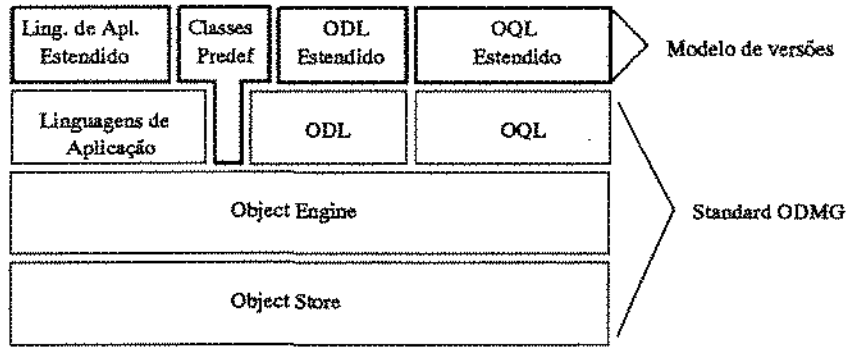


Figura 5.4: Arquitetura sobre o SGBDOO convencional

5.2.1 Modelagem de múltiplas representações sobre o modelo de versões

O problema de múltiplas representações é resolvido organizando a multiplicidade de formas de representação espacial a partir de dimensões. Cada dimensão descreve um aspecto diferente de mudança de representação, e a combinação dos valores em cada uma das dimensões determina uma forma especial de representação. Por exemplo, se uma modelagem espacial considera diferentes escalas, projeções geográficas e estruturas de representação, uma forma de representação necessariamente deve estar associada a um valor de escala, a uma projeção e a uma estrutura. Estas diferentes dimensões de representação corresponderão a diferentes dimensões de descrição semântica.

Para concentrar a apresentação no problema de múltiplas representações são consideradas duas simplificações:

- Considerar somente a existência de dimensões de descrição associadas às formas de representação. A introdução de novas dimensões manterá válidas as soluções apresentadas nesta seção.
- Considerar somente o uso de CET. Tanto os ET como os CTR estão associados, fundamentalmente, a tarefas de projeto e sua introdução posterior não modifica as soluções apresentadas.

A introdução das diferentes dimensões de representação implica que cada CET descreve objetos espaciais de acordo com uma forma particular de representação. Todas as versões de objetos espaciais em um mesmo CET são consistentes quanto a uma representação. Cada objeto pode ter associada uma representação espacial diferente em cada CET, embora não deva necessariamente possuir uma versão para todos os contextos definidos. Considere a seguinte definição:

```
class ReprContext inherit StContext
public type
  tuple (
          Scale: ScaleType;
          Proj: GeogrProj;
          Str : ReprStruct;
        )
...
end;

method body init( sc : ScaleType; pr: GeogrProj; st: ReprStruct)
in class ReprContext
{
StContext@init;
Scale = sc;
Proj = pr;
Str = st;
}

method body EqualDescriptor ( ct: ReprContext): boolean
in class ReprContext
{
 return ...../// Comparação dos valores de tipo
             // ScaleType, GeogrProj e ReprStruct.
}
```

Os CET para representação são modelados como instâncias da classe **ReprContext**, subclasse de **StContext**, onde são introduzidas as dimensões de descrição de escala (*Scale*), Projeção (*Proj*) e estrutura de representação (*Str*).

Quando uma nova forma de representação é introduzida no banco de dados, os contextos de representação permitem determinar a qual das versões do objeto deve ser associada a nova representação. No caso de estruturas espaciais complexas (por exemplo, os chamados planos de informação em [CFS+94]) este aspecto vai ser essencial pois garante que todos os objetos de um contexto referenciem uma mesma representação.

O modelo de versões permite que cada instância de **GeoObject**, possa ter uma versão diferente em cada CET. Em cada versão, a componente *location* descreve uma forma de representação consistente com esse CET.

Um dos problemas básicos para múltiplas representações, (vide seção 3.4), é permitir que a funcionalidade e o estado de um objeto conceitual possam ser manipulados pelo

usuário de forma abstrata sem considerar a forma de representação utilizada. Isto inclui a necessidade de usar operações diferentes para representações distintas.

A modelagem espacial a partir da classe **GeoObject**, considera o ocultamento total da estrutura interna e somente oferecer métodos a nível conceitual que foram classificados em: métodos de acesso ao estado conceitual ou relacionamentos do objeto e métodos de transformação do estado conceitual e relacionamentos do objeto. A implementação destes métodos dentro da classe **GeoObject** ou de alguma classe derivada, explorando as múltiplas representações é descrita a seguir.

Múltiplas representações em métodos

O modelo considera que cada representação tem métodos próprios. Por exemplo, o método *dist (cid1, cid2)* que calcula a distância entre as cidades *cid1* e *cid2* tem implementações distintas se as cidades são representadas por pontos ou áreas.

A solução apresentada é utilizar a possibilidade de associar diferentes versões dos métodos aos diferentes contextos. Desta forma, como parte da definição do comportamento de uma classe, cada método pode ter uma versão diferente em cada contexto, o que permite descrever o comportamento de um método conceitual sobre uma forma concreta de representação.

No entanto, embora esta abordagem já permita que cada *GeoObjeto* conceitual adote automaticamente o comportamento que precisa de acordo com forma de representação desejada, não garante transparência, pois para invocar um método sobre um *GeoObjeto* conceitual o usuário do método precisa conhecer e decidir qual é a representação adequada. Mapeado ao modelo de versões, deve saber qual CET deve ser definida como ativa, antes da invocação do método. Além disso, o usuário teria que determinar se o objeto possui uma versão para essa forma de representação particular.

A solução encontrada é definir para cada método Mt, um método de representação Mt_R. O corpo de método Mt_R é versionado para cada CET i como Mt_R^i o qual descreve o comportamento de Mt para a representação associada a i. O método conceitual Mt determina a representação a ser utilizada. De maneira geral seu corpo é:

Corpo do método Mt

1. Decidir sobre qual representação será realizada a operação, isto é, selecionar um CET i.
2. Definir i como CET ativo (*default*).
3. Executar o método Mt_R. Será executada a versão Mt_R^i do corpo de Mt_R.

A partir destas soluções, pode ser estabelecida uma camada sobre o Banco de Dados Multiversão, sobre a qual o usuário trabalha sobre definições de objetos em um nível conceitual de acordo com a modelagem de seu problema e sem mexer com as representações. No nível mais baixo, o nível de representação é sustentado sobre o modelo de versões.

Para ilustrar, considere a classe **ReprContext** simplificada à dimensão Escala (*Scale*) e uma classe **City** que modela as cidades:

```
class City inherit GeoObject
...
method
   distance( ct:city): float;
   area : float;
...
end;
```

Suponha que existem duas escalas de representação (1:20.000 e 1: 1.000.000) associadas a dois CET (**ReprContext**) diferentes referenciados pelas variáveis de contexto *C1* e *C2*. Como representação das cidades nessas escalas são utilizados polígonos e pontos respectivamente e para o cálculo da distância entre duas cidades podem ser aplicadas duas implementações diferentes para cada uma das escalas. No entanto, o cálculo da área somente pode ser feito sobre a representação em escala 1:20.000.

Utilizando as idéias anteriores, na classe **City** pode ser introduzido um novo método *distanceR* cujo corpo é versionado[1] como *V1* e *V2* para os casos de cálculo da distância por polígonos ou pontos, respectivamente. A versão *V1* será associada a *distanceR* no CET *C1* e a versão *V2* no CET *C2*, Em outras palavras,

```
City[C1].distanceR = v1;
City[C2].distanceR = V2;
```

Como resultado o método *distance* pode ser implementado de forma a decidir qual das duas versões de cálculo de distância utilizar.

```
method body distance (ct:City): float in class City
{
 ...
 if ( "calcular na escala 1:20000")
    Sys->SetActiveStCtx(C1);
```

[1]Note-se que não se considera versionamento de assinatura, pois parte-se do principio que a mudança de assinatura constitui a criação de um *novo* método e não um versionamento.

```
 else
    Sys->SetActiveStCtx(C2);
 return( distanceR(ct) );
}
```

A implementação do método *area* deve garantir que a versão da representação para a cidade no contexto *C1* não seja nula.

5.2.2 Modelagem temporal sobre o modelo de versões

Outro aspecto de utilização de versões em SIG é a modelagem espaço temporal. A solução proposta está baseada em mapear, sobre o modelo de versões definido, o modelo temporal para SIG, baseado em eventos, e descrito na seção 3.5.

Para isto é introduzida a dimensão temporal de descrição de contextos. Cada estado temporal do mundo real modelado é representado por um CET. Duas formas de descrição de tempo podem ser associadas a um CET: um instante ou um intervalo. No modelo apresentado é utilizado um intervalo. De novo, para facilitar o entendimento, será suposto apenas evolução temporal de CET e tempo válido unicamente.

A modelagem do tempo válido é considerada a partir de um instante de tempo t_I. O valor temporal t_F representa o último valor de tempo considerado na modelagem (presente ou futuro). A cada CET é associado um intervalo de tempo de forma tal que todas (e somente) as versões nesse CET são consistentes com aquele intervalo de tempo. Em outras palavras o conjunto de versões de um CET é auto contida do ponto de vista temporal. Entre os diferentes CET é estabelecida uma relação de ordem que reflete a sucessão linear temporal de estados do banco de dados a partir dos intervalos associados.

Para a modelagem temporal uma nova classe **TemporalCtx** é criada como subclasse de **StContext**.

```
class TimeInterval
tuple { InitTime : Time;
        FinalTime: Time)
...
end;

class TemporalCtx inherit StContext
public type
   tuple( ValidTime : TimeInterval
          )
```

```
method
   now: boolean
// Funcionalidade sobre intervalos
   BeforeInt( ti : TimeInterval) : boolean;
   AfterInt( ti : TimeInterval) : boolean;
   DuringInt( ti: TimeInterval): boolean;
   OverlapInt ( ti : TimeInterval) : boolean;
...
// Funcionalidade sobre instantes de tempo
   Before( ti : Time) : boolean;
   After( ti : Time) : boolean;
   Inside( ti: Time): boolean;
...
end;

method body EqualDescriptor ( ctx : TemporalCtx):boolean
in class TemporalCtx
{
return ...// Verdadeiro só quando não existe
          //intersecção temporal entre os intervalos
}
```

A classe **TimeInterval** modela um intervalo de tempo. A classe **TemporalCtx** especializa um CET com a dimensão temporal incluindo um atributo para representar o intervalo de tempo do contexto. Ela provê os operadores temporais tradicionais (por exemplo, *before, after, inside* dentre outros).

A redefinição do método *EqualDescriptor* garante que dois CET não representem simultaneamente o estado em um mesmo intervalo de tempo ou parte dele, garantindo assim a consistência da extensão em relação ao intervalo.

Eventos e processos espaço temporais

O modelo espaço temporal baseado em eventos, apresentado no capítulo 3 considera a evolução temporal como sucessão de *eventos espaço temporais* que marcam os pontos no tempo de mudanças em alguma das entidades espaciais modeladas. Por sua vez, cada *evento* é descrito por *processos espaço temporais*, que foram classificados segundo sua natureza em 3.5.5 e que descrevem as mudanças e relacionam temporalmente os objetos envolvidos no evento.

O resultado de um *evento* sobre o modelo de versões pode ser visto como um relacionamento entre dois CET **TemporalCtx** sucessivos segundo a relação temporal linear. O evento é modelado como uma instância da classe **TemporalEvent**, na qual são incorporados atributos para descrever a semântica do evento. Um destes atributos representa o conjunto dos *processos* associados ao evento. Estes processos por sua vez são modelados como relacionamentos entre versões de objetos em CET diferentes. Finalmente, um evento tem um atributo temporal que descreve o momento onde o evento aconteceu. Nesta abordagem consideramos que este valor temporal será um instante de tempo.

São considerados os cinco tipos de relacionamentos topológicos temporais entre objetos em tempos diferentes, que são apresentados na seção 3.5.5. Estes relacionamentos são: *Evolução, Sucessão, Produção, Transmissão e Reprodução.*

Dois tipos de métodos são introduzidos na classe **TemporalEvent** para gerenciar relacionamentos e processos: para definir um relacionamento entre dois objetos associados a um evento e métodos para consultar se um tal relacionamento existe.

```
class TemporalEvent public
type tuple( Descr: string;
            TmpCtx1, TmpCtx2 : TemporalCxt:
            TimeStamp : Time;
            )
...
method init( ctx1, ctx2: TemporalCtx; descr : string);
       DefineEvolution( obj1 GeoObject);
       DefineSuccession( obj1, obj2: GeoObject);
...
       Evolution(obj : GeoObject): boolean;
       Succession( obj1, obj2: geoObject) : boolean;
...
end;

name Events : list(TemporalEvent);
```

A raiz de persistência *Events* armazena o conjunto de eventos registrados. A classe **TemporalEvent** pode ser estendida de forma a especializar a natureza dos relacionamentos temporais. Por exemplo, o relacionamento de evolução inclui todas as formas de relacionamento geradas por processos sobre uma mesma entidade, mas poderiam ser considerados relacionamentos mais específicos como evolução por movimento, por transformação, etc.

Consultas Temporais

Em geral dois tipos de consultas básicas são definidas em sistemas que manipulam tempo: as que recuperam objetos modelados e as que recuperam valores temporais. Como na modelagem apresentada o tempo é manipulado como objetos, uma consulta do último tipo pode ser entendida como a recuperação de instâncias da classe **TimeInterval**.

Uma consulta temporal que recupera objetos modelados inclui uma seleção caracterizada por dois tipos de predicados: condições que determinam o período de tempo sobre o qual é feita a consulta e condições impostas ao estado ou aos relacionamento desses objetos nesses períodos de tempo selecionados. Na linguagem de consulta sobre versões, um predicado para determinar o período de tempo para a consulta é equivalente a selecionar um conjunto de CET a partir de:

- Condições sobre os valores dos intervalos de tempo associados aos CET
- Condições sobre eventos onde estão envolvidos os CET.

As condições sobre o estado ou relacionamento dos objetos nos períodos de tempo selecionados podem ser descritas a partir da indexação das variáveis de coleções desses objetos sobre os CET.

Por exemplo, suponha a existência de classes de geoobjetos **Farm** e **Road** para descrever as fazendas e rodovias em uma aplicação espaço temporal. Adicionalmente existem duas raízes de persistência *Farms* e *Roads* que contêm as instâncias dessas classes definidas na aplicação

```
class Farm inherit GeoObject
  type
     tuple(
            BirthYear: Date;
            BuildComp: Company;
     versionable
            Owner: Person;
           )
...
method
    intersect(GeoObject): boolean;
    area : float;
end;
```

```
class Road inherit GeoObject
  tuple (
         BirthYear: Date;
       versionable
         Name: String;
...
method
   intersect(GeoObject):boolean;
end;

name Farms : set(Farm);
name Roads : set(Road);
```

Diferentes tipos de consultas de natureza espaço temporal podem ser mapeadas na linguagem de consulta do modelo de versões

- *Consulta em um estado temporal constante que não explora propriedades espaciais:* "Os proprietários entre 1950 e 1960 das fazendas construídas em 1920".

```
SELECT f[t].Owner
FROM  f in Farms , t in Sys->GetStContext
WHERE ( ( t->OverlapInt( TimeInterval(1950, 1960))
            or
          ( t->DuringInt( TimeInterval(1950, 1960) ) )
        and
       ( f->BirthYear = 1920 )    )
```

  Note de novo que o resultado é um conjunto de pessoas, a partir da indexação da variável de contexto t ao longo do conjunto de CET contidos em *Sys→GetStContext*

- *Consulta em um mesmo estado temporal explorando propriedades espaciais:* "A área das fazendas construídas em 1920 em um momento onde as fazendas interceptavam uma rodovia".

```
SELECT f[t].area
FROM  f in Farms , r in Roads, t in Sys->GetStContext
WHERE ( f->BirthYear = 1920 )  and
      ( f[t]->intersect(r) )
```

- *Consulta em vários estados temporais explorando propriedades espaciais em cada um dos estados:* "Os proprietários atuais de fazendas com área maior de 1000 m^2 que em algum momento interceptavam uma rodovia"

```
SELECT f[t2].Owner
FROM  f in Farms , r in Roads, t1 in Sys->GetStContext,
      t2 in Sys->GetStContext
WHERE  ( f[t1]->intersect(r) ) and
       ( t2 ->now ) and
       ( f[t2]->area > 1000 )
```

Em geral as consultas anteriores recuperam objetos e propriedades de objetos versionáveis. De forma similar, essas mesmas consultas podem recuperar os estados temporais utilizados. Por exemplo, a consulta: "Quais os momentos em que a localização de alguma fazenda interceptava a geometria da rodovia "X" ", é expressada:

```
SELECT t1->ValidTime
FROM  f in Farms , r in Roads, t1 in Sys->GetStContext
WHERE  ( f[t1]->intersect(r) ) and
       ( r[t1].name = "X" )
```

Finalmente, são consideradas consultas que permitem explorar os relacionamentos topológicos temporais introduzidos pelos processos espaço temporais e mantidos pelos eventos espaço temporais no modelo. Seguindo o mesmo exemplo anterior, suponha o registro na raiz de persistência *Events* de eventos associados a mudanças nas fazendas e rodovias modeladas. Possíveis consultas podem ser:

- Fazendas que mudaram como parte do evento "inundação X"

```
SELECT f
FROM  f in Farms , ev in Events
WHERE ( ev->Descr = "inundacao X") and
      ( ev->Evolution(f) )
```

- Fazendas que tenham sido consideradas sucessoras da fazenda com nome "Y" em 1980

```
SELECT  f2
FROM  f1 in Farms , f2 in Farms,  ev in Events, t in Sys->GetStContext
WHERE  ( t->Inside(1980) ) and
       ( f1[t]-> Name = "X") and
       ( ev->Succession( f1, f2))
```

- Eventos em que aumentou a área da fazenda de nome "Y" em 1996.

```
SELECT ev
FROM f in Farms, t in Sys->GetStContext, ev in Events
WHERE ( t->Inside(1996) ) and
      ( f[t]->Name = "Y" ) and
      ( f[ev->TmpCtx2]->area  > f[ev->TmpCtx1]->area )
```

Atualização de CET

Uma das possíveis operações sobre um banco dados modelando o tempo é a atualização dos estados já existentes (modificação dos estados do passado) ou a inclusão de novos estados. No modelo apresentado isto pode ser realizado como operações sobre o conjunto das CET.

No caso de atualizações sobre o passado, novos CET podem ser criados. Suponha que existe um CET com intervalo associado $[t_1, t_2]$ e que as atualizações ocorrem no tempo $t_i \in [t_1, t_2]$. Novos contextos com intervalos $[t_1, t_i]$ e $[t_i, t_2]$ devem ser criados. Todo o processo de atualização pode corresponder a múltiplas transações para a introdução da nova informação mantendo consistência dentro e entre os novos contextos criados.

Dado que estas transações podem ser extensas e de longa duração a atualização direta sobre os CET pode implicar que estes fiquem bloqueados durante um longo tempo. O uso de ET pode ser uma solução efetiva. Os novos estados temporais podem ser manipulados e criados no ET e atualizados nos novos CET.

5.3 Resumo

Este capítulo apresentou uma proposta de mapeamento do modelo de versões proposto no capítulo 4 para um SGBDOO. Este mapeamento é baseado na extensão da linguagem de definição de objetos (ODL), linguagem de consulta (OQL) e a linguagem de aplicações. Parte do mapeamento foi realizado através de um conjunto de classes predefinidas que modelam as entidades espaciais e os conceitos básicos do modelo. Estas classes podem ser estendidas por meio da herança para as aplicações dos usuários. Finalmente, foi ilustrado como os recursos do modelo podem ser utilizados para resolver problemas de múltiplas representações e de modelagem temporal em SIG.

Capítulo 6

Conclusões e Extensões

Esta dissertação discutiu a utilização de versões em SIG e apresentou um modelo e um mecanismo para a sua introdução, como parte de um SGBDOO com facilidades espaciais. O modelo de versões proposto está baseado fundamentalmente em:

- Considerar que o versionamento de estruturas espaciais ocorre da mesma maneira que o resto dos objetos modelados.
- Organizar o espaço de versões através de dimensões que permitem estabelecer um critério de consistência semântica entre grupos de versões de diferentes objetos. Este critério é a base do suporte de facilidades temporais e de múltiplas representações.
- Considerar estruturas de organização e funcionalidade para o trabalho de projeto cooperativo, baseadas em um mesmo Banco de Dados com acesso controlado às diferentes versões. Estas estruturas podem ser manipuladas através de objetos associados que formam parte do próprio Banco de Dados.

O capítulo 2 apresentou os conceitos básicos sobre modelos de versões, e os problemas que devem ser resolvidos, em geral, sobre estes. Este capítulo apresenta as características do uso de versões em sistemas CAD/CASE e temporais.

O capítulo 3 apresentou uma visão geral dos SIG e em particular foram discutidos os problemas que podem ser resolvidos com o uso de versões, destacando três aspectos: suporte a trabalho de projeto, suporte temporal e suporte a múltiplas representações. Foram definidos requisitos desejáveis em um modelo de versões para suportar soluções a esses problemas.

A partir dos elementos apontados nos capítulos anteriores, o capítulo 4 introduziu um modelo e um mecanismo de versões para suportá-lo. Este mecanismo é baseado no mecanismo *DBV*, aqui estendido para permitir suportar funcionalidades do modelo.

O capítulo 5 apresentou uma possível especificação do modelo a partir da extensão das linguagens das componentes básicas de um SGBDOO. Esta especificação foi baseada na introdução de extensões mínimas a essas linguagens, incorporando como parte essencial da especificação, o uso dos conceitos de orientação a objetos como é herança e polimorfismo.

Dentre as contribuições fundamentais do trabalho podem ser indicadas:

- Estudo das aplicações de versões em SIG, estabelecendo um conjunto de requisitos de um modelo de versões para estas aplicações
- Proposta de um modelo de versões com recursos básicos que pode apoiar a modelagem temporal, de múltiplas representações e de projeto em SIG.
- Introdução do modelo proposto em um SGBDOO com facilidades espaciais, a partir de um mínimo de modificações nas linguagens de de definição e consulta (OQL e ODL), considerando o uso das facilidades de herança e polimorfismo.

Extensões do trabalho.

Como futuras extensões do trabalho podem ser colocadas as seguintes:

- *Mecanismos de visões sobre o modelo de versões:* A partir da noção aqui introduzida de que um Banco de Dados é considerado como um conjunto de contextos e espaços de trabalho pode ser proposto um modelo de visões que permita a manipulação de partes deste banco de dados (por exemplo, somente sobre um conjunto de dimensões) ou de novos elementos virtuais (incluindo contextos e espaços de trabalho virtuais)
- *Inclusão de possibilidades de versionamento de esquema.* O trabalho não considera esquemas versionados. A possibilidade de incluir versionamento de esquema permitiria que um mesmo objeto em contextos diferentes pudesse ter estrutura e comportamento diferentes.
- *Implementação do modelo proposto sobre um SGBDOO convencional:* Uma implementação sobre um SGBDOO convencional pode ser realizada baseada em mapear cada classe versionável como uma classe no SGBDOO que implemente a estrutura de objetos do mecanismo DBV estendido, isto é, por meio de listas de referências às diferentes versões físicas implementadas como objetos diferentes sobre o SGBDOO.
- *Implementação de consultas no modelo:* As extensões propostas à linguagem de consultas introduzem novos problemas quanto a técnicas de otimização e planejamento da execução da consulta, já que agora estas envolvem o acesso a múltiplos estados de um mesmo objeto que devem ser armazenados juntos de acordo com o mecanismo DBV adotado.

- *Mecanismos de interface:* O modelo de versões proposto introduz problemas interessantes em relação a mecanismos de visualização e interação gráfica com objetos versionados e em particular quando estes são de natureza espacial. Necessariamente os contextos devem estar envolvidos na visualização de um objeto. Este mesmo problema pode ser encontrado para o caso dos resultados de consultas

Referências Bibliográficas

[ATSS94] K. Al-Taha, R. Snodgrass, e M. Soo. Bibliography on spatiotemporal databases. *International Journal of Geographical Information Systems*, 8(1):95–103, 1994.

[Ban94] F. Bancilhon. Object database systems: the ODMG standard. Technical Report 12, O_2 Technology, 1994.

[Bat92] P. Batty. Exploiting relational database technology in a GIS. *Computers and Geosciences: An International Journal*, 18(4):453–462, 1992.

[BB91] M. Borhani e J.P. Barthes. Versions in object-oriented databases. Technical report, Universite de Technologie de Compiegne, 1991.

[BCJ92] MJ. Blin, W. Cellary, e 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. 338–347, dezembro de 1992.

[BH89] A. Bjornerstedt e C. Hulten. Version control in an object-oriented architecture. In W. Kim e K. Lochovsky, editores, *Object-Oriented Concepts, Databases and Applications*, capítulo 18, pp. 451–485. ACM Press, 1989.

[BK85] D. Batory e W. Kim. Modelling concepts for VLSI CAD objects. *ACM Transactions on Database Systems*, 10(3):322–346, setembro de 1985.

[Bot95] M. Botelho. Incorporação de facilidades espaço-temporais em bancos de dados orientados a objetos. Master's thesis, Universidade Estadual de Campinas, IMECC, DCC, 1995.

[CÔ95] G. Câmara. *Modelos, Linguagens e Arquiteturas para Bancos de Dados Geográficos*. PhD thesis, Instituto Nacional de Pesquisas Espaciais, dezembro de 1995.

[CCH+96] G. Câmara, M.A. Casanova, A. Hemerly, G. C. Magalhaes, e C. B. Medeiros. *Anatomias de Sistemas de Informação Geografica.* Instituto de Computação, UNICAMP, 1996.

[CFS+94] G. Câmara, U. Freitas, R. Souza, M. Casanova, A. Hemerly, e C.B. Medeiros. A model to cultivate objects and manipulate fields. In *Proc. 2nd ACM workshop on advances in GIS, Maryland, USA*, pp. 30–37, dezembro de 1994.

[Cif95] R.R. Ciferri. Um benchmark voltado à análise de desempenho de Sistemas de Informações Geográficas. Master's thesis, Universidade Estadual de Campinas, IMECC, DCC, 1995.

[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.

[CJ94] W. Cellary e G. Jomier. Apparent versioning and concurrency control in object-oriented databases. In *Proc. of the International Conference on Computing and Information, (ICCI'94), Peterborough, Ontario, Canada*, 1994.

[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.

[Cou92] H. Couclelis. People manipulate objects (but cultivate fields): Beyond the raster-vector debate in gis. In U. Frank e I. Campari, editores, *Proc.. International Conference on GIS - From Space to Territory: Theories and methods of spatio-temporal reasoning , Pisa ,Italia*, volume 639 de *Lecture Notes in Computer Science*, pp. 65–77, setembro de 1992.

[CS90] M. Caruso e E. Sciori. The VISION object-oriented database system. In F Bancilhon e P. Buneman, editores, *Advances in Database Programming Languages.* Addison Wesley, 1990.

[CT95] C. Claramunt e M. Theriault. Managing time in GIS. An event-oriented approach. In *Recent Advances in Temporal Databases*, pp. 23–42, setembro de 1995.

[CVJ94] W. Cellary, G. Vossen, e G. Jomier. Multiversion object constellations: A new approach to support a designer database work. *Engineering with Computers*, pp. 230–244, 1994.

[DFG+96] A. Doucet, M. Fauvet, S. Gançarski, G. Jomier, e S. Monties. Using database versions to implement temporal integrity constraints. In *Proc. of CP'96 Workshop on Constraints and Databases, Cambridge, Massachusetts,USA*, agosto de 1996.

[DGJM96] A. Doucet, S. Gançarski, G. Jomier, e S. Monties. Integrity constraints in multiversion databases. In *Proc. of Fourteeth British National Conference on Databases, Edinburgh, UK*, volume 1094 de *Lectures Notes in Computer Science*, pp. 56–73, julho de 1996.

[DGL87] K. Dittrich, P. Gotthard, e P. Lockemann. DAMOKLES: The database system for the Unibase software engineering environment. *IEEE Data Engineering*, 10(1):37–47, 1987.

[DL88] K. Dittrich e R. Lorie. Version support for engineering databases systems. *IEEE Transactions on Software Engineering*, 14(4):429–437, abril de 1988.

[EAT92] M Egenhofer e K. Al-Taha. Reasoning about gradual changes of topological relationships. In A. U. Frank e I. Campari, editores, *Proc. International Congference on GIS - From Space to Territory: Theories and methods of spatio-temporal reasoning, Pisa, Italia*, volume 639 de *Lectures Notes in Computer Science*, pp. 169–219, setembro de 1992.

[Ege94] M. Egenhofer. Spatial SQL: A query and presentation language. *IEEE Transactions on Knowledge and Data Engineering*, 6(1):86–95, 1994.

[EV94] J. Ebert e G. Vossen. Object configurations in software enginnering databases. Technical Report N515/94-I, University of Munster, agosto de 1994.

[Gü94] R. Güting. An introduction to spatial database systems. *The VLDB Journal*, 3(4):357–400, outubro de 1994.

[Gan94] S. Gançarski. *Versions et Bases de Donnes: modèle formel, supports de langage et d'interface-utilisateur.* PhD thesis, Paris-Sud University, Centre d'Orsay, France, 1994.

[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 un langage de manipulation pour bases de données multiversions. In *Proceedings 12e journées Bases de Données Avancées, Cassis, France*, 1996.

[GJZ95] S. Gançarski, G. Jomier, e M. Zamfiroiu. A framework for the manipulation of a multiversion database. In *Proc. Workshop of Databases and Expert Systems Applications Conference, London, UK*, pp. 247–256, 1995.

[Gol93] L. Golendziner. Um estudo sobre versões em bancos de dados orientados a objetos. Technical report, Universidade Federal de Rio Grande do Sul, 1993.

[GSdS95] L. Golendziner e C. Saraiva dos Santos. Uma abordagem multi-nível para suporte a versões em bancos de dados orientados a objetos. In *PANEL '95 - XXI Conferencia Latino Americana de Informatica*, pp. 1127–1138, setembro de 1995.

[HK87] S.E. Hudson e R. King. Object-oriented database support for software environments. *ACM SIGMOD Conference*, pp. 491–503, maio de 1987.

[JJE+94] C. Jensen, Clifford J., R. Elmasri, G. Shashi, P. Hayes, e S. Jojodia. A consensus glossary of temporal database concepts. *ACM SIGMOD Record*, 23(1), março de 1994.

[Kat90] R. Katz. Toward a unified framework for version modelling in engineering databases. *ACM Computing Surveys*, 22(4):375–408, dezembro de 1990.

[KBC+87] W. Kim, J. Banerjee, H. Chou, J. Garza, e D. Woelk. Composite object support in an object oriented database system. In *Proc. of OOPSLA '87*, pp. 118–125, outubro de 1987.

[KBC+89] W. Kim, N. Ballou, H. Chou, J Garza, e D Woelk. Features of the ORION object-oriented database system. In W. Kim e F. Lochovski, editores, *Object-Oriented Concepts, Databases and Applications*, pp. 251–282. ACM Press, 1989.

[KBG89] W. Kim, E. Bertino, e J. Garza. Composite objects revisted. *SIGMOD Records*, 18(2):337–347, junho de 1989.

[Lan89] G. Langran. A review of temporal databases research and its use in GIS applications. *International Journal of Geographical Information Systems*, 3(3):215–232, 1989.

[Lan93a] G. Langran. Issues of implementing a spatiotemporal system. *International Journal of Geographical Information Systems*, 7(4):305–314, 1993.

[Lan93b] G. Langran. *Time in Geographic Information Systems*. Taylor and Francis, 1993.

[MBJ96] C. B. Medeiros, M. Bellosta, e G. Jomier. Managing multiple representations of georeferenced elements. Notas pessoais., 1996.

[MCJ93] C. B. Medeiros, W. Cellary, e G. Jomier. Maintaining integrity constraints across versions in a database. In *Proc. 8vo Simposio Brasileiro de Bancos de dados*, pp. 83–97, 1993.

[MJ93a] C. B. Medeiros e G. Jomier. Managing alternatives and data evolution in GIS. *Proc. ACM/ISCA Workshop on Advances in GIS, Arlington, Virginia, USA*, pp. 34–37, 1993.

[MJ93b] C. B. Medeiros e G. Jomier. Using versions in GIS. Technical Report 94-05, IMECC - UNICAMP, 1993.

[MP94] C.B. Medeiros e F. Pires. Databases for GIS. *ACM SIGMOD Record*, 23(1):107–115, março de 1994.

[NM95] H. Naja e N. Mouaddib. The multiple representation in an architectural application. In *Proc. COSIT'95*, volume 988 de *Lectures Notes in Computer Science*, pp. 237–246, 1995.

[NTE92] R. Newell, D. Theriault, e M. Easterfield. Temporal GIS-modeling the evolution of spatial data in time. *Computers and Geosciences.*, 18(4):427–433, março de 1992.

[Oli93] L.M. Oliveira. Incorporação da dimensão temporal a bancos de dados orientados a objetos. Master's thesis, Universidade Estadual de Campinas, IMECC, DCC, junho de 1993.

[PD95] E. Puppo e G. Dettori. Toward a formal model for multiresolution spatial maps. In M. Egenhofer e J. Herring, editores, *Proc. Advances in Spatial Databases.*, volume 951 de *Lectures Notes in Computer Science*, pp. 152–169, agosto de 1995.

[Per90] B. Pernici. Objects with roles. *ACM/IEEE office information systems in SIGOIS Bulletin*, pp. 205–215, 1990.

[Peu94] D. Peuquet. It's about time: A conceptual framework for the representation of temporal dynamics in GIS. *Annals of the Association of American Geographers*, setembro de 1994.

[RM92] J. Raper e D. Maguire. Design models and functionality in GIS. *Computers and Geosciences*, 18(4):387–394, 1992.

[WR95] W Wieczerzycki e J. Rykowsky. Version support for CAD CASE. In *Proc. COSIT'95*, volume 988 de *Lecture Notes in Computer Science*, pp. 249–260, 1995.

[Zdo86] S. Zdonik. Version management in an object-oriented database. In *Proc. Intern. Workshop on Advanced Programming Environments*, volume 244 de *Lecture Notes in Computer Science*, pp. 405–422, 1986.