editado e organizado por:

Marco Antonio Casanova
Gilberto Câmara
Clodoveu A. Davis Jr.
Lúbia Vinhas
Gilberto Ribeiro de Queiroz

Bancos de Dados Geográficos

Maio, 2005

Editora	Mundogeo
Coordenação Editorial	
Produção Gráfica	AR Comunicação
Capa	
Preparação	
Revisão	
Editoração Eletrônica	AR Comunicação
Impressão e acabamento	Gráfica Infante

B212b

Bancos de dados geográficos / Organizadores João da Silva... [et al.].– Curitiba: EspaçoGEO, 2005.

504 p.

ISBN

1. Bancos de dados geográficos. 2. Aplicativos geográficos. 3. Geoinformação.

I. Câmara, Gilberto, 1956 – II. Título.

CDD- 629.4

R. Desembargador Hugo Simas, 1231 – Escritório 03
Bom Retiro – Curitiba / PR – 80520-250
Tel.: (41) 3338-7789 Fax: (41) 3338-9237
www.mundogeo.com

Prefácio

Há quase duas décadas, bancos de dados tornaram-se o componente central de sistemas de informação, tanto do ponto de vista de projeto, quanto do ponto de vista de operação. Esta evolução foi possível graças a uma sólida tecnologia desenvolvida para armazenamento e manipulação de dados convencionais, notadamente os chamados sistemas de gerência de bancos de dados objeto-relacionais (SGBD-OR).

O projeto e operação de sistemas de informação geográfica vem seguindo o mesmo rumo, adotando bancos de dados geográficos (BDGs) como ponto central da arquitetura. A relativa demora na adoção de BDGs explica-se pela complexidade de representação e manipulação de dados geográficos. Tal complexidade exigiu desenvolvimentos adicionais da tecnologia dos SGBD-OR até que o nível de funcionalidade e o desempenho fossem satisfatórios para a plena adoção de BDGs.

Mais recentemente, o foco do desenvolvimento de sistemas de informação caminhou na direção de federações de sistemas fracamente acoplados. Os sistemas de informação geográfica acompanharam de perto esta evolução, oferecendo uma gama variada de serviços de intercâmbio e disseminação de dados geográficos, novamente fruto da maturação de novas tecnologias.

Este livro aborda bancos de dados geográficos dentro desta ampla perspectiva, cobrindo desde aspectos de representação dos dados geográficos até a sua disseminação na Internet. O livro está dividido em quatro partes, para facilitar a leitura.

A *Parte I - Fundamentos* examina os problemas básicos de representação computacional de dados geográficos, resume os principais algoritmos geométricos e representações topológicas e trata da modelagem de dados espaço-temporais em geral.

A *Parte II - Persistência e Acesso* apresenta uma visão geral das principais tecnologias especificamente desenvolvidas para sistemas de gerência de banco de dados geográficos. Inclui ainda exemplos de sistemas existentes que oferecem extensões espaciais.

A *Parte III - Interoperabilidade* cobre tecnologias relacionadas a integração e interoperabilidade, e inclui o tema de disseminação de dados geográficos na Internet. Apresenta também as propostas do Open Geospatial Consortium para interoperabilidade.

Por fim, a *Parte IV - TerraLib* descreve os aspectos mais relevantes da biblioteca TerraLib, incluindo o tratamento de dados matriciais. Apresenta ainda o *TerraLib Development Kit - Tdk,* cujo objetivo principal é facilitar o desenvolvimento de aplicativos geográficos que utilizem a TerraLib.

O conteúdo do livro foi balanceado para atender a diferentes comunidades:

- Gerentes de tecnologia de informação interessados em implantar bancos de dados geográficos em suas instituições podem se beneficiar da leitura dos Capítulos 1, 3, 5, 8, 10 e 11.
- Desenvolvedores de sistemas de informação geográfica têm acesso a material sobre o funcionamento interno dos bancos de dados geográficos (Capítulos 2, 6 e 7) e também uma descrição da TerraLib (Capítulos 12, 13 e 14).
- Alunos de pós-graduação podem encontrar uma introdução a temas no estado-da-arte nos Capítulos 4, 9 e 10.

Este livro é resultante da cooperação entre as equipes baseadas nos estados de São Paulo (INPE), Rio de Janeiro (PUC-Rio) e Minas Gerais (UFMG e PUC Minas). Uma parte substancial dos resultados aqui

apresentados resultou de projetos de pesquisa e desenvolvimento, teses e dissertações nessas instituições. Este livro serve como texto básico do curso de pós-graduação em Bancos de Dados Geográficos ministrado no INPE (http://www.dpi.inpe.br/cursos/ser303) e nas disciplinas da Pós-Graduação em Informática da PUC Minas. Este livro também se encontra on-line, com material adicional, no sítio:

http://www.dpi.inpe.br/livros/bdados.

Agradecemos ao CNPq pelo apoio ao projeto TerraLib e o apoio parcial as pesquisas de Gilberto Câmara e Clodoveu Davis Jr; ao INPE pelo apoio na editoração deste livro; ao Dr. Antônio Miguel Vieira Monteiro, chefe da Divisão de Processamento de Imagens; ao TecGraf e seu coordenador Prof. Marcelo Gattass; à PRODABEL e PUC Minas; e, finalmente, à Terezinha Gomes dos Santos pelo apoio logístico.

Esperamos que a comunidade de geoinformação brasileira possa beneficiar-se de nossa experiência, que tentamos transmitir neste livro.

São José dos Campos, Rio de Janeiro, Belo Horizonte, maio de 2005.

Os autores.

Sobre os autores

Alberto H. F. Laender é doutor em Ciência da Computação pela University of East Anglia (UK) e professor titular da UFMG.

Clodoveu A. Davis Jr. é doutor em Ciência da Computação pela UFMG e professor da PUC Minas.

Daniela Francisco Brauner é doutoranda em Informática na PUC-Rio.

Gilberto Câmara é doutor em Computação Aplicada pelo INPE, e pesquisador da Divisão de Processamento de Imagens do INPE.

Gilberto Ribeiro de Queiroz é mestre em Computação Aplicada pelo INPE e engenheiro da Divisão de Processamento de Imagens do INPE.

Karine Reis Ferreira é mestre em Computação Aplicada pelo INPE e engenheira da Divisão de Processamento de Imagens do INPE.

Karla A. V. Borges é doutoranda em Ciência da Computação na UFMG e analista da PRODABEL.

Ligiane Alves de Souza é mestrando em Ciência da Computação na UFMG.

Lúbia Vinhas é doutoranda em Computação Aplicada no INPE e engenheira da Divisão de Processamento de Imagens do INPE.

Marcelo Tílio Monteiro de Carvalho é pesquisador do Grupo de Tecnologia em Computação Gráfica (TecGraf) da PUC-Rio.

Marco Antonio Casanova é doutor em Applied Mathematics pela Harvard University (EUA) e professor da PUC-Rio.

Mário de Sá Vera é pesquisador do Grupo de Tecnologia em Computação Gráfica (TecGraf) da PUC-Rio.

Olga Fradico de Oliveira é doutoranda em Computação Aplicada no INPE.

Paulo de Oliveira Lima Junior é mestre em Computação Aplicada pelo INPE, professor e coordenador do curso de Bacharelado em Sistemas de Informação da UNIPAC (Conselheiro Lafaiete, MG).

Ricardo Cartaxo Modesto de Souza é engenheiro sênior da Divisão de Processamento de Imagens do INPE.

Taciana de Lemos Dias é doutoranda em Computação Aplicada no INPE, professora da PUC Minas e analista da PRODABEL.

Índice

1 *Representação computacional de dados geográficos*

Gilberto Câmara

1.1 Introdução

Este capítulo examina os problemas básicos de representação computacional de dados geográficos, e esclarece questões da seguinte natureza: *Como representar os dados geográficos no computador? Como as estruturas de dados geométricas e alfanuméricas se relacionam com os dados do mundo real? Que alternativas de representação computacional existem para dados geográficos?*

Em seu livro "Olhos de Madeira", Carlo Ginzburg nos traz um fascinante ensaio sobre a origem da palavra 'representação'. A origem do termo remonta ao século XIII, chamando-se *représentation* aos manequins de cera exibidos junto ao cadáver dos reis franceses e ingleses durante as cerimônias funerárias (Ginzburg, 2001). Enquanto o soberano era velado, a presença do manequim era um testemunho à transcendência do rei e a sua presença futura no mundo dos mortos. O manequim tinha a função de lembrar aos presentes que o rei havia assumido uma outra forma e que uma nova vida se iniciava para o morto. Nesta nova forma, apesar de morto o rei continuaria presente para seus súditos ("*re* + *présentation*").

Assim, desde a sua origem a palavra 'representação' está associada a uma forma abstrata de descrição do mundo. O uso do manequim como representação do soberano morto é apenas um exemplo do problema mais geral da construção de abstrações que descrevem o mundo. Para explicar como funcionam os bancos de dados geográficos, este capítulo descreve o processo de transformar aos conceitos abstratos de *espaço*

geográfico no referindo ao *espaço computacionalmente representado*. Para exemplificar, consideremos alguns problemas:

- Uma cientista social deseja entender e quantificar o fenômeno da *exclusão social* numa grande cidade brasileira, através de mapas de exclusão/inclusão social, gerados a partir de dados censitários (Sposati, 1996).
- Uma ecóloga pretende estudar os remanescentes florestais da Mata Atlântica, através de estudos de *fragmentação* obtidos a partir de interpretação de imagens de satélite (Pardini et al., 2005).
- Uma pedóloga pretende determinar a *distribuição* de propriedades do solo numa área de estudo, a partir de um conjunto de amostras de campo (Bönisch et al., 2004).

O que há de comum nesses casos? A especialista lida com conceitos de sua disciplina (*exclusão social, fragmentos, distribuição de propriedades do solo*) e precisa de representações que traduzam estes conceitos para o computador. Após esta tradução, ela poderá compartilhar os dados de seu estudo, inclusive com pesquisadores de outras disciplinas.

1.2 Descrição geral de sistemas de informação geográfica

O termo *sistemas de informação geográfica* (SIG) é aplicado para sistemas que realizam o tratamento computacional de dados geográficos. A principal diferença de um SIG para um sistema de informação convencional é sua capacidade de armazenar tanto os atributos descritivos como as geometrias dos diferentes tipos de dados geográficos. Assim, para cada lote num cadastro urbano, um SIG guarda, além de informação descritiva como proprietário e valor do IPTU, a informação geométrica com as coordenadas dos limites do lote. A partir destes conceitos, é possível indicar as principais características de SIGs:

- Inserir e integrar, numa única base de dados, informações espaciais provenientes de meio físico-biótico, de dados censitários, de cadastros urbano e rural, e outras fontes de dados como imagens de satélite, e GPS.
- Oferecer mecanismos para combinar as várias informações, através de algoritmos de manipulação e análise, bem como para consultar, recuperar e visualizar o conteúdo da base de dados geográficos.

Os componentes de um SIG estão mostrados na Figura 1.1. No nível mais próximo ao usuário, a *interface homem-máquin*a define como o sistema é operado e controlado. Esta interface pode ser tanto baseada na metáfora da "mesa de trabalho" (Kuhn e Frank, 1991) (Richards e Egenhofer, 1995) (Câmara et al., 1999), como adaptada ao ambiente de navegação da Internet (Kraak e Brown, 2001), quanto baseada em linguagens de comando como Spatial SQL (Egenhofer, 1994) e LEGAL (Câmara, 1995). No nível intermediário, um SIG deve ter mecanismos de processamento de dados espaciais. A *entrada* de dados inclui os mecanismos de conversão de dados (Hohl, 1998). Os algoritmos de *consulta e análise espacial* incluem as operações topológicas (Egenhofer e Franzosa, 1991), álgebra de mapas (Tomlin, 1990), estatística espacial (Druck et al., 2004), modelagem numérica de terreno (Li et al., 2004) e processamento de imagens (Mather, 2004). Os mecanismos de visualização e plotagem devem oferecer suporte adequado para a apreensão cognitiva dos aspectos relevantes dos dados pesquisado (MacEachren, 2004) (Tufte, 1983) (Monmonier, 1993). No nível mais interno do sistema, um *sistema de gerência de bancos de dados geográficos* oferece armazenamento e recuperação dos dados espaciais e seus atributos. Cada sistema, em função de seus objetivos e necessidades, implementa estes componentes de forma distinta, mas todos os subsistemas citados devem estar presentes num SIG.

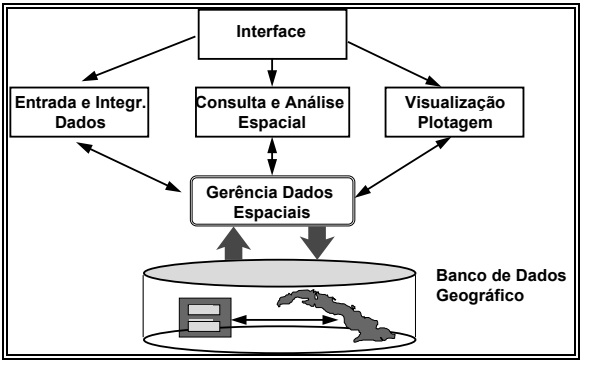


Figura 1.1- Arquitetura de sistemas de informação geográfica.

Do ponto de vista da aplicação, o uso de sistemas de informação geográfica (SIG) implica em escolher as representações computacionais mais adequadas para capturar a semântica de seu domínio de aplicação. Do ponto de vista da tecnologia, desenvolver um SIG significa oferecer o conjunto mais amplo possível de estruturas de dados e algoritmos capazes de representar a grande diversidade de concepções do espaço. Como o presente livro está focado nos diferentes aspectos relacionados com a tecnologia de bancos de dados geográficos, discutiremos em maior detalhe a questão de gerência de dados espaciais. Leitores interessados nos demais aspectos de um SIG poderão consultar as referências acima.

1.3 Traduzindo a informação geográfica para o computador

Para abordar o problema fundamental da Geoinformação, que é a *produção de representações computacionais do espaço geográfico*, usamos o *paradigma dos quatro universos*, proposto inicialmente por Gomes e Velho (1995) e adaptado para a geoinformação por Câmara (1995). Este paradigma distingue quatro passos entre o mundo real e sua realização computacional (ver Figura 1.2).

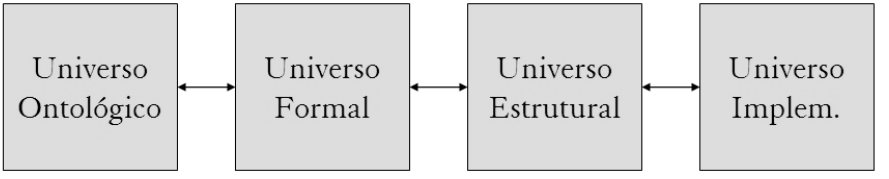


Figura 1.2 - Paradigma dos quatro universos.

No primeiro passo, nossas percepções do mundo real são materializadas em conceitos que descrevem a realidade e respondem a questões como: *Que classes de entidades são necessárias para descrever o problema que estamos estudando?* (Smith, 2003). Criamos assim o *universo ontológico*, onde incluímos os conceitos da realidade a serem representados no computador, como os tipos de solo, elementos de cadastro urbano, e caracterização das formas do terreno.

O segundo universo (*o universo formal*) inclui modelos lógicos ou construções matemáticas que generalizam os conceitos do universo ontológico e dão resposta à pergunta: *Quais são as abstrações formais*

necessárias para representar os conceitos de nosso universo ontológico? Estas abstrações incluem modelos de dados e álgebras computacionais. Exemplos: o modelo entidade-relacionamento (Chen, 1976) e o modelo OMT (Rumbaugh et al., 1991). O caso específico de modelos dados geográficos é tratado em maior detalhe nos capítulos 3 e 4 deste livro e inclui o modelo OMT-G (Davis et al., 2002). A questão de linguagens é tratada no Capítulo 5.

O terceiro universo é o *universo estrutural*, onde as diversas entidades dos modelos formais são mapeadas para estruturas de dados geométricas e alfanuméricas, e algoritmos que realizam operações. Neste universo, respondemos a questões como: *Quais são os tipos de dados e algoritmos necessários para representar os modelos e as álgebras do universo formal?* As estruturas de dados são os elementos básicos de construção dos sistemas computacionais, e serão discutidas em maior detalhe neste capítulo. Aspectos do universo estrutural descritos no livro incluem arquiteturas de SGBD (Capítulo 8), conversão de dados (Capítulo 9), interoperabilidade (Capítulo 10) e disseminação de dados na Internet (Capítulo 11).

O universo de *implementação* completa o processo de representação computacional. Neste universo, realizamos a implementação dos sistemas, fazendo escolhas como arquiteturas, linguagens e paradigmas de programação. Neste livro, as questões de implementação discutidas incluem geometria computacional (Capítulo 2), métodos de acesso (Capítulo 6), processamento de consultas (Capítulo 7), além da descrição detalhada da biblioteca TerraLib (capítulos 12 a 14).

O paradigma dos quatro universos é uma forma de compreendermos que a transposição da realidade para o computador requer uma série complexa de mediações. Primeiro, precisamos dar nomes às entidades da realidade. Depois, geramos modelos formais que as descrevem de forma precisa. A seguir, escolhemos as estruturas de dados e algoritmos que melhor se adaptam a estes modelos formais. Finalmente, fazemos a implementação num suporte computacional apropriado. Nas próximas seções, examinaremos em detalhe cada um destes universos.

1.4 O universo ontológico

Ontologia é o campo da filosofia cujo objetivo é descrever os tipos e estruturas de entidades, eventos, processos e relações que existem no mundo real (Smith, 2003). Sua gênese remonta a Aristóteles, mas o interesse recente por ontologias em sistemas de informação decorre principalmente da necessidade de compartilhar informação de forma eficiente para um público cada vez mais interdisciplinar.

Um sistema de informação pode ser concebido como um mecanismo de comunicação entre duas partes: o produtor e o usuário. Para que funcione, é necessário que haja uma concordância entre os conceitos das partes. Numa perspectiva mais geral, seu sucesso depende da existência de uma comunidade que compartilhe as definições utilizadas para construí-lo. Por exemplo, considere o caso de um estudo sobre *segregação* em áreas urbanas. Existem diferentes conceitos de segregação na literatura sociológica (Caldeira, 2000) (Massey e Denton, 1993) (Torres, 2004) (White, 1983). Para construir um sistema de informação que permita o estudo da segregação urbana, é preciso que o produtor de informação defina qual dos diferentes conceitos estará sendo representado, como esta representação será construída, e como o usuário pode compreender as características e limitações desta representação.

Deste modo, o problema fundamental de um sistema de informação é definir o conjunto de conceitos a ser representado. Se quisermos que estes conceitos sejam compartilhados por uma comunidade interdisciplinar, é fundamental que os conceitos utilizados sejam devidamente explicitados. Assim, surge a pergunta: "*Qual o papel dos conceitos na representação do mundo?*" A melhor forma de responder é baseando-se na perspectiva realista (Searle, 1998):

1. A realidade existe independentemente das representações humanas.
2. Nós temos acesso ao mundo através de nossos sentidos e de nossos instrumentos de medida.
3. As palavras em nossa linguagem podem ser usadas para referir-se a objetos do mundo real.

4. Nossas afirmações são verdadeiras ou falsas dependendo de sua correspondência aos fatos do mundo.
5. Algumas afirmações em nossa linguagem dizem respeito a uma realidade externa e independente ("*há neve no topo do Monte Evereste*"). Outras afirmações dizem respeito a convenções socialmente construídas ("*este papel é uma certidão de nascimento*").

Como nos ensina Searle (1993), esta perspectiva tem conseqüências importantes sobre nossa concepção do mundo:

> *"Apesar de termos representações mentais e lingüísticas do mundo sob a forma de crenças, experiências, afirmações, teorias, etc., há um mundo, 'lá fora', totalmente independente destas representações. A órbita elíptica dos planetas relativamente ao Sol e a estrutura do átomo de hidrogênio são inteiramente independentes das representações que os seres humanos têm de tais fenômenos. Já coisas como o dinheiro, a propriedade, o casamento e os governos são criados e sustentados pelo comportamento cooperativo humano."*
>
> *"Na sua maior parte, o mundo existe independentemente da linguagem (princípio 1) e uma das funções da linguagem é representar como são as coisas no mundo (princípio 3). Um aspecto crucial no qual a realidade e a linguagem entram em contato é marcado pela noção de verdade. Em geral, as afirmações são verdadeiras na medida em que representam com precisão uma característica da realidade que existe independentemente da afirmação (princípio 4)."*.

O projeto de um sistema de informação requer, como passo inicial, a escolha das entidades a ser representados e, se possível, a descrição organizada destas entidades por meio de conceitos. Esta descrição forma uma *ontologia de aplicação*, definida como "*um conjunto de conceitos compartilhados por uma comunidade*" (Gruber, 1995). Para os dados geográficos, uma geo-ontologia tem dois tipos básicos de conceitos: (a) conceitos que correspondem a fenômenos físicos do mundo real; (b) conceitos que criamos para representar entidades sociais e institucionais (Smith e Mark, 1998) (Fonseca et al., 2003). Chamamos o primeiro tipo de *conceitos físicos* e o segundo de *conceitos sociais* (Tabela 1.1). Embora todos os conceitos resultem do uso compartilhado da linguagem, há uma diferença entre conceitos que se referem ao mundo físico ("*A Amazônia*

possui uma floresta tropical") e aqueles que resultam de convenções humanas ("*Esta é uma reserva indígena*").

Nossa geo-ontologia diferencia entre conceitos associados a entidades que pode ser individualizadas e identificadas nominalmente (caso de *lagos* e *lotes*) e aquelas que variam de forma contínua no espaço (caso de *poluição*).

Tabela 1.1 – Tipos de conceitos associados a entidades geográficas

	Realidade física	Realidade social
Entidades individualizáveis	*indivíduos bona fide* (e.g., montanha)	*indivíduos fiat* (e.g., lote)
Entidades com variação contínua	*topografias físicas* (e.g., poluição)	*topografias sociais* (e.g., segregação)

Os conceitos físicos podem ser subdivididos em:

- Conceitos associados a entidades individualizáveis, que possuem uma fronteira bem definida a partir de diferenciações qualitativas ou descontinuidades na natureza. Designados como *indivíduos bona fide* (do latim "boa fé"), sua existência decorre de nossa necessidade de dar nomes aos elementos do mundo natural. Por exemplo, embora a a superfície da Terra apresente uma variação contínua no espaço, nossa percepção do espaço depende da associação de nomes especiais a variações bem definidas no terreno. Daí nascem conceitos como *montanha*, *vale* e *desfiladeiro*.
- Conceitos associados a entidades que tem variação contínua no espaço, associadas aos fenômenos do mundo natural, não estando a princípio limitadas por fronteiras. Chamamos estes conceitos de topografias físicas, onde o termo "topografia" está associado a qualquer grandeza que varia continuamente. Exemplos incluem *temperatura, altimetria, declividade e poluição*.

Os conceitos sociais podem ser subdivididos em:

- Conceitos que descrevem entidades individuais criadas por leis e por ações humanas. Estas entidades possuem uma fronteira que as distingue do seu entorno e tem uma identidade única. Sua existência depende usualmente de um registro legal. Designadas como *indivíduos fiat* (do latim "fazer"), incluem conceitos como *lotes, municípios e países.*
- Conceitos descrevendo entidades que têm variação contínua no espaço, associadas a convenções sociais. Tome-se o caso de *pobreza*, conceito socialmente definido que ocorre no espaço de forma ininterrupta ("*em cada lugar há algum tipo diferente de pobreza*"). Chamamos estes conceitos de topografias sociais. Exemplos incluem: *exclusão social, segregação urbana, desenvolvimento humano.*

Uma geo-ontologia é um conjunto de conceitos e um conjunto de relações semânticas e espaciais entre estes termos. Cada conceito tem um nome, uma definição e um conjunto de atributos. O conjunto das relações semânticas inclui as relações de sinonímia, similaridade, e hiponímia (também dito especialização: *"hospital é um tipo de prédio"*). Por exemplo:

- **rio**: "Curso de água natural, de extensão mais ou menos considerável, que se desloca de um nível mais elevado para outro mais baixo, aumentando progressivamente seu volume até desaguar no mar, num lago, ou noutro rio".
- **riacho**: "rio pequeno, mais volumoso que o regato e menos que a ribeira"
- relação semântica: um riacho *é um* rio. (hiponímia).

O conjunto de relações espaciais inclui as relações topológicas como pertinência e adjacência, relações direcionais como "ao norte de", e relações informais como "no coração de" ou "perto de". Por exemplo:

- **afluente**: "curso de água que deságua em outro curso de água, considerado principal".
- relação espacial: um afluente *está conectado a* um rio.

Na maior parte dos sistemas de informação atuais, as ontologias de aplicação não estão explicitadas, o que reduz o potencial de compartilhamento da informação. Com o advento da Internet, que permite a disseminação de dados forma ampla e para um público heterogêneo, a necessidade de explicitar as ontologias utilizadas tornou-se ainda mais premente. A explicitação das ontologias de aplicação está na base das propostas recentes da "Web Semântica" (Berners-Lee et al., 2001) e de propostas de padrões como OWL. Como resultado de pesquisas recentes, já temos vários sistemas disponíveis na Internet para criação e gestão de ontologias, como o Protegé (Noy et al., 2001). Para dados geográficos, o consórcio OGC ("Open Geospatial Consortium") propôs o formato GML como mecanismo de descrição de ontologias geográficas. Fazemos uma descrição mais detalhada do tema nos capítulos 9 e 10 deste livro.

1.5 O universo formal

O universo formal representa um componente intermediário entre os conceitos do universo ontológico e as estruturas de dados e algoritmos computacionais. Como os computadores trabalham com estruturas matemáticas, a passagem direta de conceitos informais da ontologia de aplicação para estruturas de dados poderia gerar decisões inconsistentes. No universo formal, buscamos estabelecer um conjunto de entidades lógicas que agrupem os diferentes conceitos da ontologia de aplicação da forma mais abrangente possível. Adicionalmente, neste universo definimos ainda como serão associados valores aos diferentes conceitos; ou seja, como podemos medir o mundo real. Deste modo, o universo formal tem duas partes: (a) como medir o mundo real (teoria da medida); (b) como generalizar os conceitos da ontologia em entidades formais abrangentes. Estas duas partes serão discutidas a seguir.

1.5.1 Atributos de dados geográficos: teoria da medida

Para representar dados geográficos no computador, temos de descrever sua variação no espaço e no tempo. Em outras palavras, precisamos poder a perguntas como: "*qual é o valor deste dado aqui e agora?*". Isto requer uma compreensão dos processos de mensuração da realidade, de forma consistente com os dois primeiros princípios de Searle (1998): "*a*

realidade existe independentemente das representações humanas" e "*nós temos acesso ao mundo através de nossos sentidos e de nossos instrumentos de medida*". O processo de medida consiste em associar números ou símbolos a diferentes ocorrências de um mesmo atributo, para que a relação dos números ou símbolos reflita as relações entre as ocorrências mensuradas. Por exemplo, podemos medir a poluição numa cidade através de sensores localizados em diferentes locais. Cada um destes sensores nos dará uma medida diferente. Esta atribuição é denominada *escala de medida*. A referência geral mais importante sobre escalas de medidas é o trabalho de Stevens (1946), que propõe quatro escalas de mensuração: *nominal, ordinal, intervalo e razão*.

Os níveis nominal e ordinal são *temáticos*, pois a cada medida é atribuído um número ou nome associando a observação a um tema ou classe. A *escala nominal* classifica objetos em classes distintas sem ordem inerente, como rótulos que podem ser quaisquer símbolos. As possíveis relações entre os valores são identidade ($a = b$) e dessemelhança ($a \neq b$). Um exemplo é a cobertura do solo, com rótulos como "*floresta*", "*área urbana*" e "*área agrícola*" (ver Figura 1.3).

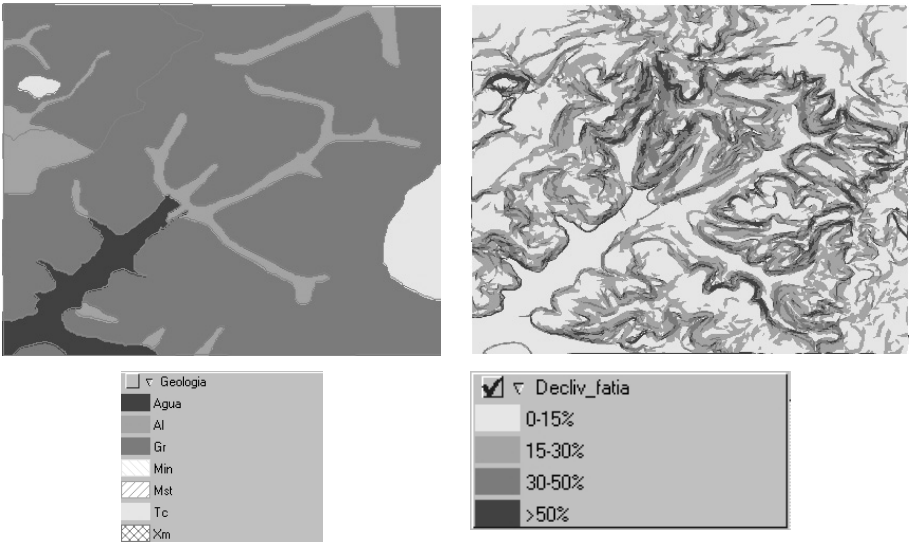


Figura 1.3 – Exemplos de medida nominal (mapa geológico) e medida ordinal (mapa de classes de declividade).

A *escala ordinal* introduz a idéia de ordenação, caracterizando os objetos em classes distintas que possuem uma ordem natural (por exemplo 1 – ruim, 2 – bom, 3 – ótimo ou "0-10%", "11-20%", "mais que 20%"). A distância definida entre os elementos não é significativa. Nesta escala são evidenciadas as relações "<" ou ">", isto implica que para todo a e b, as relações $a < b$, $a > b$ ou $a = b$ são possíveis. Um exemplo é a aptidão agrícola de solos, com rótulos como "muito apto", "apto", "pouco apto", e "inapto".

As medidas temáticas não estão associadas à magnitude do fenômeno. Quando o estudo necessita de uma descrição mais detalhada, que permita comparar intervalo e ordem de grandeza entre eventos, recorre-se aos níveis de medidas denominados de *numéricos*, onde as regras de atribuição de valores baseiam-se em uma escala de números reais.

Existem dois níveis de medidas baseados em escalas de números reais: *escala por intervalo e o escala por razão.* A *escala por intervalo* possui um ponto zero arbitrário, uma distância proporcional entre os intervalos e uma faixa de medidas entre $[-\infty, \infty]$. A temperatura em graus Celsius é exemplo de medida por intervalo, onde o ponto zero corresponde a uma convenção (a fusão do gelo em água). Por ter uma referência zero arbitrária, valores medidos no nível por intervalo não podem ser usados para estimar proporções. Operações aritméticas elementares (adição e subtração) são válidas, porém multiplicação e divisão não são apropriadas. Por exemplo, dados a e b, pode-se ter $a = b + c$, onde c é a diferença entre a e b em alguma unidade padrão. Assim, a temperatura em São Paulo pode ser c graus mais baixa do que a temperatura em Campos de Jordão.

A *escala de razão* permite um tratamento mais analítico da informação, pois nela o ponto de referência zero não é arbitrário, mas determinado por alguma condição natural. Sua faixa de valores é limitada entre $[0, \infty]$. Nesta escala existe um ponto zero absoluto que não pode ser alterado e um intervalo arbitrário com distâncias proporcionais entre os intervalos. Números negativos não são permitidos, pois o número zero representa ausência total daquilo que está sendo medido. Por exemplo, na descrição de atributos como peso e volume de objetos não há valores negativos. No caso de temperatura em graus Kelvin, a condição natural é o ponto de

repouso dos átomos da matéria, a partir do qual não se consegue temperaturas menores. Este ponto é o zero absoluto para temperatura, zero graus Kelvin. O fato de ponto de referência zero ser absoluto permite afirmações tais como *a* é duas vezes mais pesado que *b*. Desta forma, dado *a* e *b* pode-se ter $a = c \times b$, onde *c* indica o número de vezes que *b* vai até *a*, a relação de *a* para *b*. Operações matemáticas de adição, subtração, multiplicação e divisão são suportadas nesta escala.

A Tabela 1.2 apresenta um resumo das escalas de medidas, destaca a característica principal, apresenta algumas operações admitidas e exemplos para cada uma delas.

Tabela 1.2 – Tipos de medidas de dados geográficos

Escala	Características	Exemplos	Operações possíveis
Nominal	Descrição	Tipo de solo, vegetação, uso do solo	Seleção, Comparação
Ordinal	Ordem	Classes de declividade, aptidão de uso	Mediana, Máximo, Mínimo
Intervalo	Distância	Altimetria	Diferença, Soma
Razão	Valores absolutos	Renda, população, taxa de natalidade	Operações aritméticas

1.5.2 Espaço absoluto e espaço relativo

Antes de considerar os diferentes modelos formais para dados geográficos, é necessário analisarmos brevemente os conceitos de *espaço absoluto* e *espaço relativo*. Esta distinção decorre da possibilidade de representarmos no computador a localização dos objetos no espaço ou apenas o posicionamento relativo entre eles, como ilustrado na Figura 1.4. Nesta figura, mostramos à esquerda os distritos da cidade de São Paulo, identificados por suas fronteiras. Neste caso, trata-se de uma representação no espaço absoluto, na qual as coordenadas das fronteiras devem corresponder às estabelecidas na legislação. Do lado direito, mostramos um grafo com as conexões dos distritos, que formam uma rede (repetimos a imagem dos distritos por razões de melhor legibilidade

da figura). No modelo de redes, a localização exata de cada distrito não é armazenada, pois a rede só captura as relações de adjacência. Dizemos então que a rede de conexões dos distritos é um modelo de *espaço relativo*.

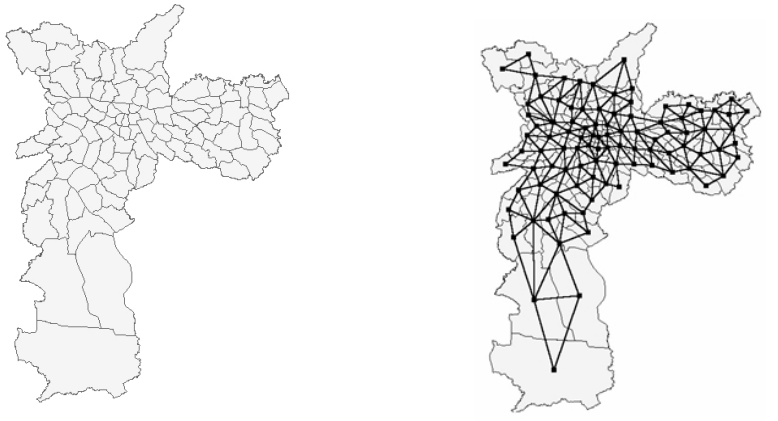


Figura 1.4 – Dualidade entre espaço absoluto e espaço relativo. À esquerda, distritos de São Paulo com suas fronteiras. À direita, grafo mostrando a rede de conectividade entre os distritos (espaço relativo). O mapa da esquerda foi repetido por razões de melhor legibilidade.

A distinção entre espaço absoluto e espaço relativo é de grande importância para a Geografia. Milton Santos (Santos, 1985) refere-se ao "espaço dos fixos" e ao "espaço dos fluxos". Castells (1999) fala em "espaço de lugares" e "espaços de fluxos". Vejam o que Helen Couclelis comenta a respeito do tema:

> *"Espaço absoluto, também chamado cartesiano, é um container de coisas e eventos, uma estrutura para localizar pontos, trajetórias e objetos. Espaço relativo, ou leibnitziano, é o espaço constituído pelas relações espaciais entre coisas" (Couclelis, 1997).*

Uma das escolhas básicas que fazemos na modelagem dos fenômenos geográficos é definir se utilizaremos representações no espaço absoluto ou no espaço relativo. Esta escolha depende primordialmente do tipo de análise que queremos realizar. Usualmente, consultas espaciais que envolvem dois tipos de entidades ("*quais os rios que cruzam esta estação ecológica?*") requerem a representação no espaço absoluto. O mesmo vale

para questões de álgebra de mapas ("*áreas inaptas tem declividade maior que 15% ou solos arenosos*"). Quando os procedimentos de análise envolvem apenas as relações de conectividade ("*como chegar na estação de metrô Clínicas, partindo da estação Liberdade?*" ou "*qual é a média da mortalidade infantil de meus vizinhos?*") podemos utilizar representações no espaço relativo. Quando falamos em entidades como estradas, linhas de transmissão, conexões de água e esgoto, cadeias de mercado e linhas de comunicação, o espaço relativo é na maioria das vezes plenamente adequado.

1.5.3 Modelos no espaço absoluto: geo-campos e geo-objetos

Existem dois modelos formais para entidades geográficos no espaço absoluto: *geo-campos e geo-objetos*. O modelo de geo-campos enxerga o espaço geográfico como uma superfície contínua, sobre a qual variam os fenômenos a serem observados. Por exemplo, um mapa de vegetação associa a cada ponto do mapa um tipo específico de cobertura vegetal, enquanto um mapa geoquímico associa o teor de um mineral a cada ponto. O modelo de geo-objetos representa o espaço geográfico como uma coleção de entidades distintas e identificáveis, onde cada entidade é definida por uma fronteira fechada. Por exemplo, um cadastro urbano identifica cada lote como um dado individual, com atributos que o distinguem dos demais.

Definição 1.1. Geo-Campo. Um geo-campo representa um atributo que possui valores em todos os pontos pertencentes a uma região geográfica. Um *geo-campo gc* é uma relação $gc = [R, A, f]$, onde $R \subset \Re^2$ é uma partição conexa do espaço, A é um atributo cujo domínio é $D(A)$, e a função de atributo $f: R \rightarrow A$ é tal que, dado $p \in R$, $f(p) = a$, onde $a \in D(A)$.

A noção de geo-campo decorre da definição física associada (segundo o Aurélio, "campo é um conjunto de valores de uma grandeza física que, numa região do espaço, dependem só das coordenadas dos pontos pertencentes a essa região"). Em outras palavras, para cada ponto do espaço, um campo terá um valor diferente.

Definição 1.2 Geo-Objeto. Um geo-objeto é uma entidade geográfica singular e indivisível, caracterizada por sua identidade, suas fronteiras, e seus atributos. Um *geo-objeto* é uma relação $go = [id, a_1,...a_n, G]$, onde *id* é um identificador único, G é um conjunto de partições 2D conexas e

distintas $\{R_1,...,R_n\}$ do espaço $\Re^2$, e a_i são os valores dos atributos $A_1,...,A_n$. Note-se que um geo-objeto pode ser composto por diferentes geometrias, onde cada geometria tem uma fronteira fechada (e.g., o Japão com suas diferentes ilhas).

Um exemplo de geo-campo (uma imagem IKONOS da cidade do Rio de Janeiro) e de um conjunto de geo-objetos (os distritos dessa cidade) é apresentado na Figura 1.5. A variável associada à imagem é a reflectância do solo, medida pelo sensor óptico do satélite. Os geo-objetos associados aos distritos de São Paulo são mostrados numa gradação de tons de cinza, cuja intensidade é proporcional ao índice de exclusão social (Sposati, 1996); quanto mais escuro, mais o distrito possui moradores em situação de exclusão social. Os dados na Figura 1.3 acima (geologia e declividade) também são exemplos de geo-campos.

A Figura 1.5 também ilustra uma questão importante: *existem diferenças fundamentais entre geo-campos e geo-objetos? Ou seriam apenas duas maneiras de ver o mesmo tipo de dado?* Considere os retângulos desenhados no interior das duas representações mostradas. Na figura à esquerda, o interior do retângulo tem as mesmas propriedades do geo-campo que o contém. Para cada ponto interior ao retângulo, podemos recuperar o valor do atributo (neste caso, a reflectância da imagem). Verificamos que *uma partição espacial genérica de um geo-campo compõe outro geo-campo com as mesmas propriedades.*

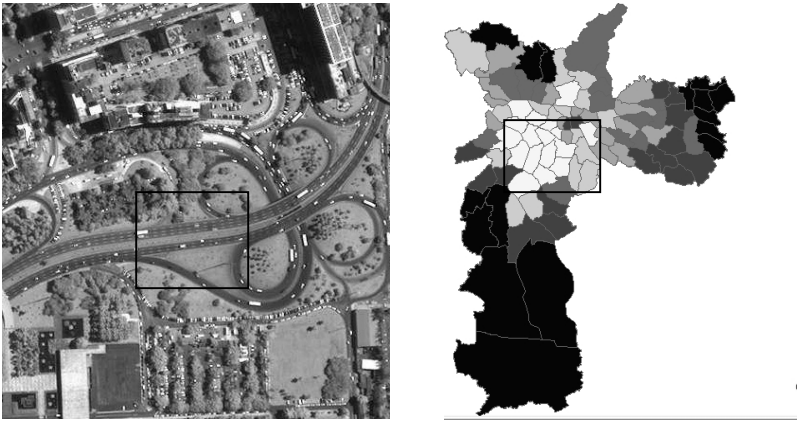


Figura 1.5 – Exemplo de geo-campo (imagem IKONOS do Rio de Janeiro) e de conjunto de geo-objetos (distritos da cidade de São Paulo).

Considere agora a figura da direita (distritos de São Paulo). O interior do retângulo mostrado não define mais um conjunto de geo-objetos com as mesmas propriedades do conjunto completo. O retângulo intercepta parcialmente alguns objetos. Como cada objeto é único e não pode ser dividido sem perder suas características originais, verificamos que *uma partição espacial genérica de um conjunto de geo-objetos não compõe outro conjunto de geo-objetos com as mesmas propriedades.*

A diferença essencial entre um geo-campo e um geo-objeto é o papel da fronteira. A fronteira de um geo-campo é uma divisão arbitrária relacionada apenas com nossa capacidade de medida. Na Figura 1.5, os limites da imagem correspondem apenas a eventuais limitações do instrumento sensor e não do fenômeno medido. Assim, o geo-campo pode ser divido em partes e ainda assim manter sua propriedade essencial (que é sua função de atributo).

Por contraste, um geo-objeto é essencialmente definido por sua fronteira, que o separa do mundo exterior; ele não pode ser dividido e manter suas propriedades essenciais. Dentro da fronteira, todas as propriedades do objeto são constantes. Tomemos um distrito de São Paulo, como a Sé, que tem um código único de identificação no censo do IBGE. Se dividirmos a Sé em duas partes, precisamos de dois novos códigos de identificação para caracterizar os dois novos distritos.

O exame da Figura 1.5 ilustra outra propriedade dos geo-objetos. É bastante comum lidarmos com um conjunto de geo-objetos que representam uma partição consistente do espaço; isto é, os recobrimentos espaciais destes objetos não se interceptam e eles possuem o mesmo conjunto de atributos. Estas características fazem com que possamos agrupar estes objetos numa coleção.

Definição 1.3 Coleção de geo-objetos. Uma coleção de geo-objetos é relação $cgo = [id, o_1,...o_n, A_1,..., A_n]$, onde *id* é um identificador único, e $o_1,...o_n$ são geo-objetos que possuem os atributos $A_1,..., A_n$. Usualmente, se R_i for a região geográfica associada a o_i, temos $R_i \cap R_j = \emptyset$, $\forall\ i \neq j$. Deste modo, uma coleção reúne geo-objetos cujas fronteiras não se interceptam, e têm o mesmo conjunto de atributos.

O uso de coleções de geo-objetos é bastante freqüente em bancos de dados geográficos, pois é muito conveniente tratar geo-objetos similares

de forma consistente. Por exemplo, falamos dos distritos da cidade de São Paulo, dos municípios do estado do Ceará, e das reservas indígenas da Amazônia. A idéia de coleções de geo-objetos é ainda útil para propormos um modelo orientado-a-objetos para dados geográficos, discutido a seguir.

1.5.4 Modelos no espaço relativo: redes

O modelo de redes concebe o espaço geográfico como um conjunto de pontos no espaço (chamados de nós), conectados por linhas (chamados arcos), onde tanto os nós quanto os arcos possuem atributos. Os fenômenos modelados por redes incluem fluxo de pessoas ou materiais, conexões de influência, linhas de comunicação e acessibilidade. Um dos atrativos do modelo de redes é que o suporte matemático para este modelo (*a teoria de grafos*) é uma área de pesquisa consolidada (Bondy e Murty, 1976) (Gross e Yelen, 1998).

O problema que deu início à teoria dos grafos foi uma questão espacial. Em 1736, o matemático Leonard Euler vivia na cidade de Königsberg (na época parte da Prússia; hoje chamada Kaliningrad e pertencente à Rússia) onde haviam duas ilhas próximas no meio da cidade, cruzadas por sete pontes (ver Figura 1.6 à esquerda). Euler se perguntou se havia uma maneira de fazer um circuito fechado (sair e voltar para um mesmo lugar), cruzando cada uma das pontes apenas uma vez. Ele construiu um grafo equivalente (ver Figura 1.6 à direita) e demonstrou que o problema era insolúvel.

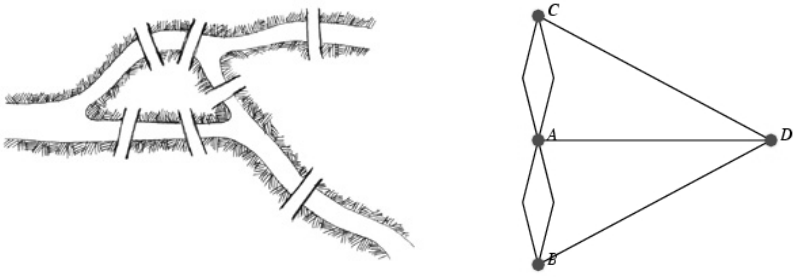


Figura 1.6 – As sete pontes de Königsberg e o grafo equivalente.

Definição 1.4 Redes. Uma rede é uma estrutura geográfica que tem como suporte um grafo $G = [N, A, \varphi]$, onde N é um conjunto de nós, A é um conjunto de arcos (arestas), e $\varphi(a)=(u,v)$ é uma função de incidência que associa cada arco $a \in A$ a um par de nós $(u, v) \in N$. No caso geográfico, os nós podem estar associados a uma localização (x,y) do espaço para fins de referência.

Como os nós de uma rede são abstrações de entidades existentes no espaço, eles podem estar associados aos seus atributos descritivos. Por exemplo, na rede mostrada na Figura 1.4, cada nó está associado a um distrito de São Paulo, e poderia ter diferentes atributos que descrevem este distrito. Também os arcos de uma rede podem ter propriedades, como o custo de percorrimento de um nó a outro. As propriedades mensuráveis das redes incluem operações diretas computáveis sobre a topologia do grafo, como qual o caminho ótimo entre dois nós. Também podemos computar operações matemáticas que envolvem apenas as relações de conectividade, como os indicadores locais de autocorrelação espacial (veja-se a respeito, Druck et al, 2004).

A definição de redes pode ser estendida para considerar o caso de conexões bidirecionais, como no caso de redes de transporte, onde as relações entre os nós não são simétricas, pois os fluxos em sentidos opostos podem ser diferentes. A Figura 1.7 ilustra uma rede simples e uma rede com conexões bidirecionais.

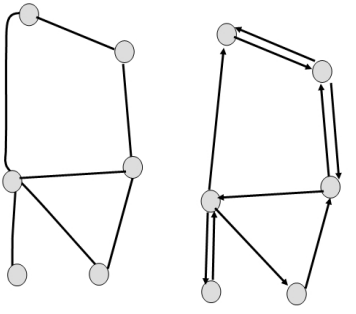


Figura 1.7 – Exemplos de redes simples e de redes com conexões bidirecionais.

Os modelos de rede têm grande utilidade em problemas de geoinformação, incluindo assuntos como gerenciamento de serviços como água, esgoto, eletricidade e telefonia. Para maiores referências, deve-se consultar Birkin et al (1996) e Godin (2001).

1.5.5 Um modelo orientado-a-objetos para dados geográficos

As seções anteriores nos permitem apresentar um modelo orientado-a-objetos que apresenta uma versão unificada dos dados geográficos, com base nos conceitos básicos de *geo-campo*, *coleção de geo-objetos* e *rede*. Para fins de organização lógica, o modelo considera a existência de uma classe genérica, chamada de *plano de informação* (ou *layer*), que é uma generalização destes dois conceitos. O conceito de *plano de informação* captura uma característica comum essencial dos três conceitos básicos: cada instância deles é referente a uma localização no espaço e tem um identificador único. Assim, o uso do conceito de *plano de informação* permite organizar o banco de dados geográfico e responder a perguntas como: "*Quais são os dados presentes no banco, qual o modelo associado a cada um e qual a região geográfica associada?*" Adicionalmente, como cada geo-campo está associado a uma única função de atributo, ele pode ser especializado em *geo-campo temático* (associado a medidas nominais ou ordinais) e *geo-campo numérico* (associados a medidas por intervalo ou por razão). Com estes seis conceitos, construímos um modelo formal básico para dados geográficos, mostrado na Figura 1.8.

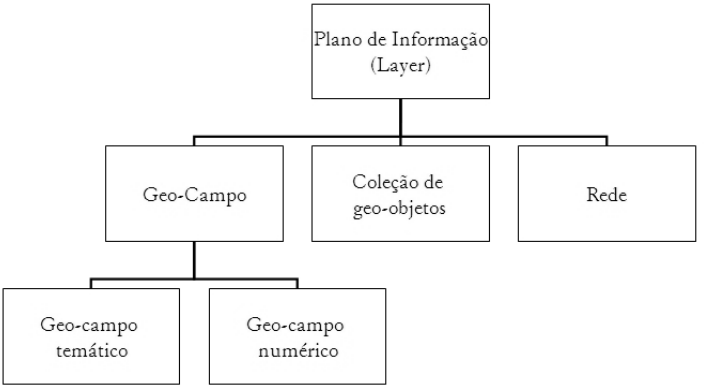


Figura 1.8 – Modelo OO básico para dados geográficos.

O modelo mostrado na Figura 1.8 serve de base para a maioria dos modelos de dados orientados-a-objetos adotados atualmente em geoinformação:

- O software SPRING (Câmara et al., 1996) inclui os conceitos de *rede, geo-campo numérico e geo-campo temático, coleção de geo-objetos* (chamada de *mapa cadastral*). Os geo-campos numéricos admitem as *imagens* como caso particular.
- No ArcGIS (ESRI, 2000), a coleção de geo-objetos é chamada de *features (feições)*. Os geo-campos numéricos são chamados de *surfaces (superfícies)*, e as imagens também são modeladas como caso particular de geo-campos numéricos. As redes (*networks*) também são incluídas.
- No modelo OpenGIS (OGC, 1998), os geo-campos são chamados de *coverage,* e a coleção de objetos é chamada de *feature collection*. O modelo OpenGIS não tem o conceito explícito de *layer,* mas considera que as visões de *feature collection* e *coverage* são complementares.
- Na TerraLib (vide Capítulo 12 deste livro), o conceito de plano de informação (*layer)* é um conceito usado para organizar a informação no banco de dados. Os conceitos de *geo-campos* e de coleções de *geo-objetos* são implícitos. Como se trata de uma biblioteca, os designers da TerraLib quiseram permitir diferentes alternativas de projeto de sistema.

1.6 Do universo ontológico ao universo formal

Para passar do universo ontológico para o universo formal, precisamos responder à pergunta: *como os conceitos da ontologia de aplicação são formalizados?* Colocando o problema de forma mais geral: *Que critérios deve satisfazer um conceito para que seja utilizável em estudos quantitativos associados à geoinformação?* Tais critérios são:

- O conceito deve ser passível de ser associado a propriedades mensuráveis.
- Estas propriedades devem ser medidas no território e devem permitir diferenciar as diferentes localizações.

- Os resultados quantitativos e os modelos matemáticos utilizados devem ser validados em estudos de campo, que devem incluir dimensões objetivas e subjetivas do fenômeno em questão.

Para representar um conceito genérico como "exclusão social", precisamos definir precisamente quais atributos caracterizam a exclusão social e como podemos medi-los no território. Esta caracterização realiza a passagem do universo ontológico para o universo formal. Com base em conceitos bem estabelecidos e associados a medidas quantitativas no espaço, podemos construir *territórios digitais*. O processo pode ser resumido na Figura 1.9.

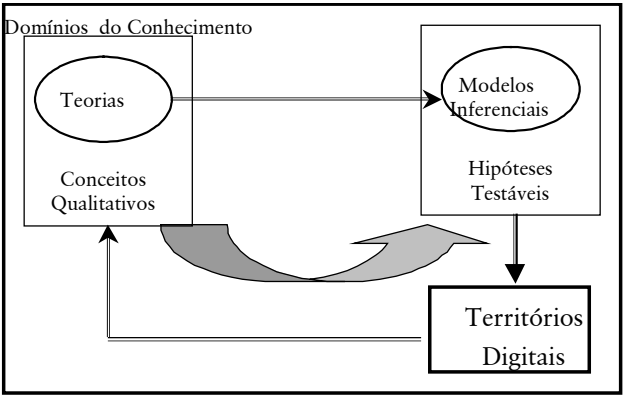


Figura 1.9 – Relação entre a construção dos territórios digitais e as teorias disciplinares (cortesia de Silvana Amaral Kampel).

Os especialistas desenvolvem teorias gerais sobre os fenômenos, que incluem o estabelecimento de conceitos organizadores de sua pesquisa (como "exclusão" ou "vulnerabilidade"). Para passar destas teorias para a construção computacional, é necessário que o especialista formule modelos inferenciais quantitativos. Estes modelos devem ser submetidos a testes de validação e de corroboração, através dos procedimentos de análise quantitativa. Os resultados numéricos podem então dar suporte ou ajudar a rejeitar conceitos qualitativos.

Após definir como que atributos mensuráveis serão associados ao conceito, o projetista do sistema de informação deverá decidir se este conceito será modelado no *espaço absoluto* ou no *espaço relativo*. A decisão deve-se dar essencialmente em função das propriedades que queremos

medir. Se a localização exata é fundamental, ou se precisamos saber o valor do fenômeno em todos os pontos da região de estudo, então é necessário usar os modelos de espaço absoluto. Se o fluxo e as conexões são essenciais, então podemos usar o modelo de rede.

Se precisamos dos dados expressos no espaço absoluto, então devemos escolher ainda qual o modelo apropriado (geo-campo ou geo-objeto). Para isto, a decisão depende essencialmente do papel da fronteira. Se as fronteiras são parte essencial das entidades modeladas, estamos tratando com *indivíduos* e não com *topografias* (vide Tabela 1.1) e o modelo de geo-objetos é o mais adequado. Senão, usaremos os modelos de geo-campos.

1.7 Universo estrutural

As estruturas de dados utilizadas em bancos de dados geográficos podem ser divididas em duas grandes classes: *estruturas vetoriais* e estruturas matriciais.

1.7.1 Estruturas de dados vetoriais

As estruturas vetoriais são utilizadas para representar as coordenadas das fronteiras de cada entidade geográfica, através de três formas básicas: pontos, linhas, e áreas (ou polígonos), definidas por suas coordenadas cartesianas, como mostrado na Figura 1.10.

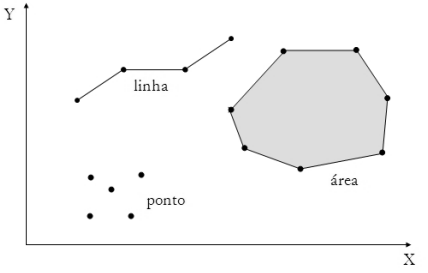


Figura 1.10 – Representações vetoriais em duas dimensões.

Um *ponto* é um par ordenado *(x, y)* de coordenadas espaciais. O ponto pode ser utilizado para identificar localizações ou ocorrências no espaço. São exemplos: localização de crimes, ocorrências de doenças, e localização de espécies vegetais. Uma *linha é* um conjunto de pontos

conectados. A linha é utilizada para guardar feições unidimensionais. De uma forma geral, as linhas estão associadas a uma topologia arco-nó, descrita a seguir. Uma *área* (ou *polígono)* é a região do plano limitada por uma ou mais linhas poligonais conectadas de tal forma que o último ponto de uma linha seja idêntico ao primeiro da próxima. Observe-se também que a fronteira do *polígono* divide o plano em duas regiões: o interior e o exterior. Os polígonos são usados para representar unidades espaciais individuais (setores censitários, distritos, zonas de endereçamento postal, municípios). Para cada unidade, são associados dados oriundos de levantamentos como censos e estatísticas de saúde.

1.7.2 Vetores e topologia: o caso dos geo-objetos

A topologia é a parte da matemática na qual se investigam as propriedades das configurações que permanecem invariantes nas transformações de rotação, translação e escala. No caso de dados geográficos, é útil ser capaz de determinar relações como adjacência ("vizinho de"), pertinência ("vizinho de"), intersecção, e cruzamento.

Objetos de área podem ter duas formas diferentes de utilização: como objetos *isolados* ou objetos *adjacentes*. O caso de objetos isolados é bastante comum em SIG urbanos, e ocorre no caso em que os objetos da mesma classe em geral não se tocam. Por exemplo, edificações, piscinas, e mesmo as quadras das aplicações cadastrais ocorrem isoladamente, não existindo segmentos poligonais compartilhados entre os objetos. Finalmente, temos objetos adjacentes, e os exemplos típicos são todas as modalidades de divisão territorial: bairros, setores censitários, municípios e outros. Neste caso, pode-se ter o compartilhamento de fronteiras entre objetos adjacentes, gerando a necessidade por estruturas topológicas. Estes também são os casos em que recursos de representação de buracos e ilhas são mais necessários.

Quando queremos armazenar as estruturas de dados do tipo polígono no caso de *objetos adjacentes*, temos uma decisão básica a tomar: guardamos as coordenadas de cada objeto isoladamente, e assim duplicamos as fronteiras em comum com outros objetos, ou armazenamos cada fronteira comum uma única vez, indicando a que objetos elas estão associadas? No primeiro caso é chamado de *polígonos sem topologia* e o segundo, de *topologia arco-nó-polígono*, comparados na Figura 1.11.

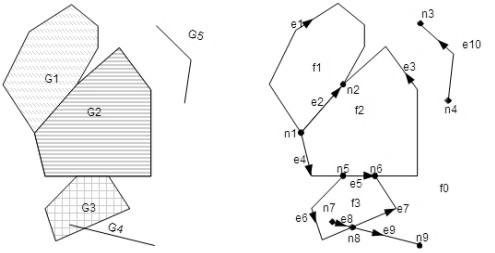
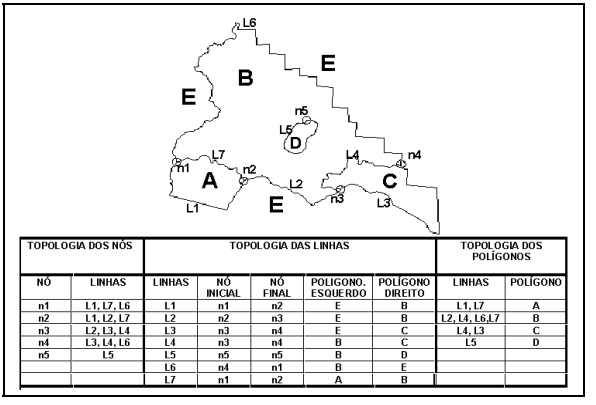


Figura 1.11 – Polígonos sem topologia (à esquerda) e topologia arco-nó-polígono (à direita). (Fonte: Ravada, 2003).



TOPOLOGIA DOS NÓS		TOPOLOGIA DAS LINHAS					TOPOLOGIA DOS POLÍGONOS	
NÓ	LINHAS	LINHAS	NÓ INICIAL	NÓ FINAL	POLIGONO. ESQUERDO	POLÍGONO DIREITO	LINHAS	POLÍGONO
n1	L1, L7, L6	L1	n1	n2	E	B	L1, L7	A
n2	L1, L2, L7	L2	n2	n3	E	B	L2, L4, L6,L7	B
n3	L2, L3, L4	L3	n3	n4	E	C	L4, L3	C
n4	L3, L4, L6	L4	n3	n4	B	C	L5	D
n5	L5	L5	n5	n5	B	D		
		L6	n4	n1	B	E		
		L7	n1	n2	A	B		

Figura 1.12 – Topologia arco-nó-polígono.

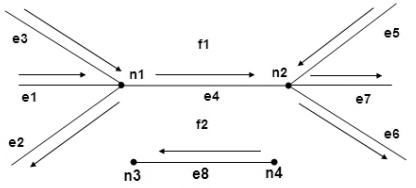
A topologia arco-nó-polígono, como mostrado na Figura 1.12, requer três listas separadas. Os pontos inicial e final de cada linha são chamados de nós. Para cada nó, armazenamos as linhas nele incidentes. Para cada linha, armazenamos os nós inicial e final, permitindo assim que a linha esteja associada a um sentido de percorrimento; guardamos ainda os dois polígonos separados por cada linha (à esquerda e à direita, considerando o sentido de percorrimento). Para cada polígono, guardamos as linhas que definem sua fronteira.

1.7.3 Vetores e topologia: o caso das redes

Objetos de linha podem ter variadas formas de utilização. Analogamente aos objetos de área, podemos ter objetos de linha isolados,

em árvore e em rede. Objetos de linha isolados ocorrem, por exemplo, na representação de muros e cercas em mapas urbanos. Objetos de linha organizados em uma árvore podem ser encontrados nas representações de rios e seus afluentes, e também em redes de esgotos e drenagem pluvial. E podem ser organizados em rede, nos casos de redes elétricas, telefônicas, de água ou mesmo na malha viária urbana e nas malhas rodoviária e ferroviária.

No caso das redes, é fundamental armazenar explicitamente as relações de adjacência, utilizamos a *topologia arco-nó*. Um *nó* pode ser definido como o ponto de intersecção entre duas ou mais linhas, correspondente ao ponto inicial ou final de cada linha. Nenhuma linha poderá estar desconectada das demais para que a topologia da rede possa ficar totalmente definida. Para exemplificar, considere-se a Figura 1.13, que mostra um exemplo de como a topologia arco-nó pode ser armazenada.



EDGE_ ID	START _NODE _ID	END_ NODE_ ID	NEXT_ LEFT_ EDGE_ ID	PREV_ LEFT_ EDGE_ ID	NEXT_ RIGHT _EDGE _ID	PREV_ RIGHT _EDGE _ID	LEFT_ FACE_ ID	RIGHT _FACE _ID	GEOM ETRY
E4	N1	N2	-E5	E3	E2	-E6	F1	F2	(...)
E8	N4	N3	-E8	-E8	E8	E8	F2	F2	(...)

Figura 1.13 – Estrutura de dados para topologia arco-nó no Oracle Spatial SGBD (Fonte: Ravada, 2003).

1.7.4 Vetores e topologia: o caso dos dados 2,5 D

Uma das possibilidades associadas a dados vetoriais é a associação de valores que denotem a variação espacial de uma grandeza numérica. No caso mais simples, associamos a cada localização no espaço um valor numérico de atributo. Neste caso, como os valores de localização estão no plano e o valor adicional descreve uma superfície sobre este plano. Os

dados resultantes são chamados de dimensão "dois e meio", pois não se tratam estritamente de dados tridimensionais, pois o suporte espacial ainda são localizações 2D. A Figura 1.14 ilustra exemplo de dados de dimensão 2,5.

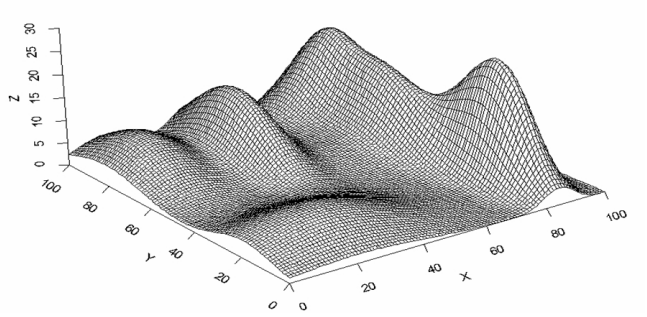


Figura 1.14 – Exemplo de dado com dimensão 2,5 (cortesia de Renato Assunção).

A maneira mais comum de armazenar estes dados é através de estruturas matriciais (vide próxima seção). Temos três alternativas que usam estruturas vetoriais:

- Conjunto de amostras esparsas 2,5D, constituído de pares ordenados *(x,y,z)*, onde *(x,y)* é uma localização no plano e *z* um valor numérico de atributo.
- Conjunto de isolinhas (curvas de nível), que são linhas às quais estão associados valores numéricos. As isolinhas não se cruzam, e são entendidas como estando "empilhadas" umas sobre as outras.
- A *malha triangular* ou TIN (do inglês "triangular irregular network") é uma estrutura do tipo vetorial com topologia do tipo *nó-arco* e representa uma superfície através de um conjunto de faces triangulares interligadas.

A malha triangular é a estrutura vetorial mais utilizada para armazenar dados 2,5D. Cada um dos três vértices da face do triângulo armazenados as coordenadas de localização *(x, y)* e o atributo *z*, com o valor de elevação ou altitude. Em geral, nos SIGs que possuem pacotes para MNT, os algoritmos para geração da malha triangular baseiam-se na triangulação de Delaunay com restrição de região. Quanto mais equiláteras forem as faces triangulares, maior a exatidão com que se descreve a superfície. O valor de elevação em qualquer ponto dentro da

superfície pode ser estimado a partir das faces triangulares, utilizando-se interpoladores. A Figura 1.15 mostra uma superfície tridimensional e a grade triangular correspondente.

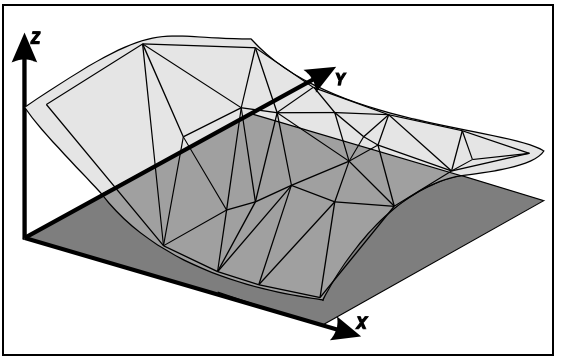


Figura 1.15 – Superfície e malha triangular correspondente. (cortesia de Laércio Namikawa).

1.7.5 Hierarquia de representações vetoriais

Para um entendimento mais detalhado das representações vetoriais em GIS, deve-se inicialmente precisar o que se entende por primitivas geométricas: *coordenadas 2D*, *coordenadas 2,5D*, *nó 2D*, *nó 2,5D, nó de rede*, *arcos*, *arcos orientados*, *isolinhas* e *polígonos*. Dada uma região $R \subset \Re^2$, pode-se definir:

- COORDENADA_2D - Uma coordenada 2D é um objeto composto por uma localização singular $(x_i, y_j) \in R$;
- COORDENADA_2,5D - Uma coordenada 2,5D é um objeto composto por uma localização singular (x_i, y_j, z), onde $(x_i, y_j) \in R$;
- PONTO2D - Um ponto 2D é um objeto que possui atributos descritivos e uma coordenada 2D;
- LINHA2D - Uma linha 2D possui atributos e inclui um conjunto de coordenadas 2D;
- ISOLINHA - uma isolinha contém uma linha 2D associada a um valor real (cota);
- ARCO ORIENTADO - um arco orientado contém uma linha 2D associada a uma orientação de percorrimento;

- NÓ2D - um nó 2D inclui uma coordenada2D $(x_i, y_i) \varepsilon R$ e uma lista L de linhas 2D (trata-se da conexão entre duas ou mais linhas, utilizada para manter a topologia da estrutura);
- NÓ REDE - um nó de rede contém um nó 2D e uma lista de arcos orientados;
- NÓ 2,5D - um nó 2,5D instância desta classe contém uma coordenada 2,5D (x_i, y_i, z_i) e um lista L de linhas 2D (trata-se da conexão entre três ou mais linhas de uma grade triangular);
- POLÍGONO - um polígono pode ser armazenado como uma lista de coordenadas 2D (caso dos geo-objetos sem topologia) ou por uma uma lista de linhas 2D e uma lista de nós 2D (caso de topologia *arco-nó-polígono*).

Uma vez definidas as primitivas geométricas vetoriais, pode ser estabelecida a hierarquia de representações geométricas vetoriais, como mostrado na Figura 1.16, onde distinguem-se os relacionamentos de especialização *é-um* ("is-a"), inclusão de uma instância *parte-de* ("part-of"), inclusão de um conjunto de instâncias *conjunto-de* ("set-of") e inclusão de uma lista de identificadores de instâncias *lista-de* ("list-of").

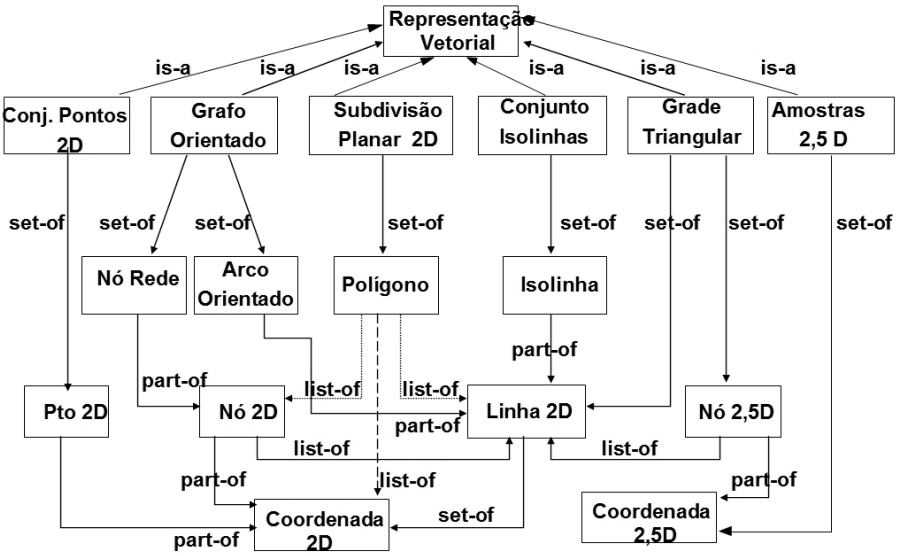


Figura 1.16 – Hierarquia de classes para estruturas vetoriais.

Distinguimos os seguintes tipos de estruturas de dados vetoriais:

- CONJUNTO DE PONTOS 2D - uma instância desta classe é um conjunto de pontos 2D utilizados para guardar localizações isoladas no espaço (p.ex. no caso de poços de petróleo);
- CONJUNTO DE ISOLINHAS - uma instância desta classe é um conjunto de linhas, onde cada linha possui uma cota e as linhas não se interceptam;
- SUBDIVISÃO PLANAR - para uma região geográfica *R* qualquer, uma subdivisão planar contém um conjunto *Pg* de polígonos que não se sobrepõem;
- GRAFO ORIENTADO - uma instância desta classe é uma representação composta de um conjunto de nó de rede e de um conjunto de arco orientado 2D;
- MALHA TRIANGULAR - uma instância desta classe contém um conjunto de nós 2,5D e um conjunto L de linhas 2D tal que todas as linhas se interseptam, mas apenas em seus pontos iniciais e finais;
- MAPA PONTOS 2,5D - uma instância desta classe é um conjunto de coordenadas 2,5D. Trata-se de um conjunto de amostras 2,5D.

1.7.6 Representação matricial

As estruturas matriciais usam uma grade regular sobre a qual se representa, célula a célula, o elemento que está sendo representado. A cada célula, atribui-se um código referente ao atributo estudado, de tal forma que o computador saiba a que elemento ou objeto pertence determinada célula. Nesta representação, o espaço é representado como uma matriz *P(m, n)* composto de *m* colunas e *n* linhas, onde cada célula possui um número de linha, um número de coluna e um valor correspondente ao atributo estudado e cada célula é individualmente acessada pelas suas coordenadas.

A representação matricial supõe que o espaço pode ser tratado como uma superfície plana, onde cada célula está associada a uma porção do terreno. A resolução do sistema é dada pela relação entre o tamanho da célula no mapa ou documento e a área por ela coberta no terreno, como mostrado na Figura 1.17.

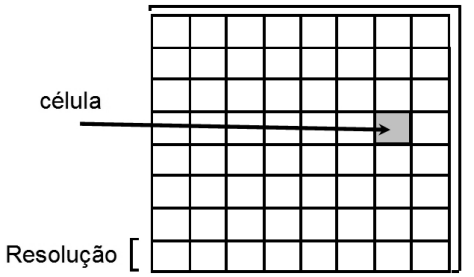


Figura 1.17 – Estrutura matricial.

A estrutura matricial pode ser utilizada para representar diferentes tipos de dados:

- *Grade regular*: representação matricial de dimensão "dois e meio" na qual cada elemento da matriz está associado a um valor numérico, como mostra a Figura 1.18 à esquerda.
- *Matriz temática*: representação matricial 2D na qual cada valor da matriz é um código correspondente à uma classe do fenômeno estudado, como mostra a Figura 1.18 à direita.

15	17	20	21	23
16	18	22	23	24
19	19	22	23	24
23	23	27	28	28
22	22	31	32	33

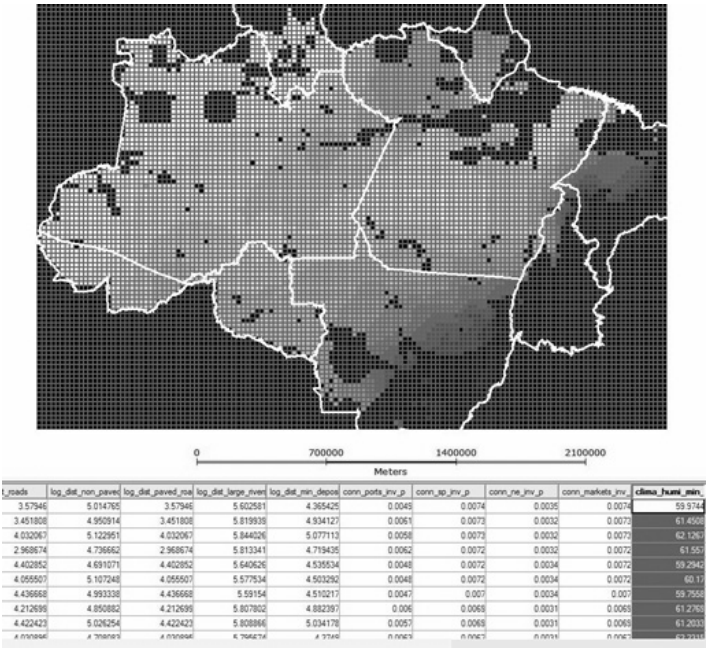
1	1	2	2	2
1	1	2	2	2
1	1	2	2	2
2	2	3	3	3
2	2	3	3	3

Figura 1.18 – À esquerda, grade regular com valores de temperatura em graus Celsius e, à direita, matriz temática com dados classificados (1 = "15-20 graus", 2 = "20-25 graus", 3 = "25-35 graus").

1.7.7 Espaços celulares: generalização de estruturas matriciais

Um *espaço celular* é uma estrutura matricial generalizada onde cada célula está associada a vários tipos de atributos. Os espaços celulares têm várias vantagens sobre estruturas matriciais simples. Usando matrizes com um único atributo (como o caso dos dados mostrados na Figura 1.18), um fenômeno espaço-temporal complexo precisa de várias matrizes separadas para ser representado, o que resulta em maior dificuldade de gerência e de interface. Num espaço celular, a mesma célula está associada a diferentes informações, com ganhos significativos de manuseio dos dados.

Os espaços celulares são muito convenientes para armazenamento em bancos de dados objeto-relacionais. Toda a estrutura de um espaço celular pode ser armazenada numa única tabela, o que faz o manuseio dos dados ser bem mais simples que os dados vetoriais ou mesmo que os dados matriciais indexados. Aplicações como álgebra de mapas e modelagem dinâmica ficam mais simples de implementar e operar. Um exemplo de espaço celular é mostrado na Figura 1.19, onde mostramos uma parte de um banco de dados onde há um espaço celular onde a Amazônia foi dividida em células de 25 x 25 km^2; cada uma delas está associada a diferentes atributos socioeconômicos e ambientais (na Figura 1.19, o atributo visualizado é umidade média nos três meses mais secos do ano). Os espaços celulares ainda não são estruturas de dados comuns nos bancos de dados geográficos, e atualmente apenas a TerraLib tem suporte para este tipo de estrutura (vide Capítulo 12). Com a ênfase crescente dos SIG em modelos dinâmicos, podemos prever que esta estrutura será futuramente amplamente disponível nas diferentes implementações de bancos de dados geográficos.



t_roads	log_dist_non_paved	log_dist_paved_roa	log_dist_large_river	log_dist_min_depos	conn_ports_inv_p	conn_ap_inv_p	conn_ne_inv_p	conn_markets_inv_	clima_humi_min_
3.57946	5.014765	3.57946	5.602581	4.365425	0.0045	0.0074	0.0035	0.0074	59.9744
3.451808	4.950914	3.451808	5.819938	4.934127	0.0061	0.0073	0.0032	0.0073	61.4585
4.032067	5.122951	4.032067	5.844026	5.077113	0.0058	0.0073	0.0032	0.0073	62.1267
2.968674	4.736662	2.968674	5.813341	4.719435	0.0062	0.0072	0.0032	0.0072	61.557
4.402852	4.691071	4.402852	5.640626	4.535534	0.0048	0.0072	0.0034	0.0072	59.2942
4.055507	5.107248	4.055507	5.577534	4.503292	0.0048	0.0072	0.0034	0.0072	60.17
4.436668	4.993338	4.436668	5.59154	4.510217	0.0047	0.007	0.0034	0.007	59.7558
4.212696	4.850882	4.212696	5.807802	4.882397	0.006	0.0069	0.0031	0.0069	61.2765
4.422423	5.026254	4.422423	5.808866	5.034178	0.0057	0.0069	0.0031	0.0069	61.2033

Figura 1.19 – Espaço celular com a Amazônia dividida em células de 25 x 25 km^2; o atributo visualizado é umidade média nos três meses mais secos do ano (cortesia: Ana Paula Dutra de Aguiar).

1.8 Do universo formal para o universo estrutural

A passagem do universo formal (geo-campos, geo-objetos e redes) para o universo estrutural não é unívoca. Para cada tipo de entidade do modelo formal, há diferentes possibilidades de uso de estruturas de dados, a saber:

- *Geo-objetos*: como as fronteiras são elementos essenciais, são usualmente armazenados em estruturas poligonais, com as opções *polígonos sem topologia* ou *topologia arco-nó-polígono*.
- *Redes:* como a topologia é parte essencial, as redes devem ser armazenadas como um grafo orientado.
- *Geo-campos numéricos*: podem ser armazenados como amostras 2,5D, malhas triangulares ou grades regulares.
- *Geo-campos temáticos:* admitem o armazenamento como estruturas vetoriais (polígonos) ou matriciais (matrizes temáticas).

Os diferentes compromissos de armazenamento para as entidades do modelo formal são discutidos a seguir. Note-se que um espaço celular (discutido na seção 1.7.7) pode guardar uma combinação arbitrária de geo-campos numéricos e temáticos.

1.8.1 Estruturas de dados para geo-objetos

A escolha entre estruturas topológicas ou não-topológicas para geo-objetos em bancos de dados geográficos depende também do suporte oferecido pelo SGBD. Nos SIG cujas estruturas de dados geométricas são manuseadas fora do SGBD (como o SPRING e o Arc/Info), é comum a escolha da topologia arco-nó-polígono. No caso dos bancos de dados geográficos, a maneira mais simples de armazenar geo-objetos é guardando cada um deles separadamente, o que implica em estruturas não-topológicas. Esta forma de trabalho foi sancionada pelo consórcio Open GIS e é suportada pelos diferentes SGBDs (Oracle, PostgreSQL, mySQL). No entanto, várias aplicações requerem o uso da topologia arco-nó-polígono, e alguns SGBDs com suporte espacial já estão incluindo esta opção, com o Oracle Spatial (Ravada, 2003).

1.8.2 Estruturas de dados para geo-campos temáticos

Geo-campos temáticos admitem tanto a representação matricial quanto a vetorial. Para a produção de cartas e em operações onde se requer maior precisão, a representação vetorial é mais adequada. As operações de álgebra de mapas são mais facilmente realizadas no formato matricial. No entanto, para um mesmo grau de precisão, o espaço de armazenamento requerido por uma representação matricial é substancialmente maior. Isto é ilustrado na Figura 1.20.

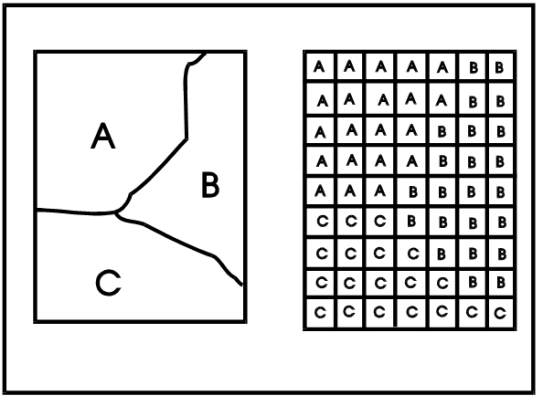


Figura 1.20 – Geo-campo temático em estruturas vetorial e matricial.

A Tabela 1.3 apresenta uma comparação entre as vantagens e desvantagens de armazenamento matricial e vetorial para geo-campos temáticos. Esta comparação leva em conta os vários aspectos: relacionamentos espaciais, análise, armazenamento. Nesta tabela, o formato mais vantajoso para cada caso é apresentado em destaque.

O armazenamento de geo-campos temáticos em estruturas vetoriais é uma herança da cartografia, onde limites entre classes temáticas eram desenhados com precisão em mapas. No entanto, sabemos que estes limites são imprecisos, na grande maioria dos casos. Assim, como nos ensina Peter Burrough, as estruturas matriciais são mais adequadas:

> *"Os limites desenhados em mapas temáticos (como solo, vegetação, ou geologia) raramente são precisos e desenha-los como linhas finas muitas vezes não representa adequadamente seu caráter. Assim, talvez não nos devamos preocupar tanto com localizações exatas e representações gráficas elegantes. Se pudermos aceitar que limites precisos entre padrões de vegetação e solo raramente ocorrem, nós estaríamos livres dos problemas de erros topológicos associados como superposição e interseção de mapas"(Burrough, 1986).*

Tabela 1.3 – Comparação entre estruturas vetoriais e matriciais para mapas temáticos

Aspecto	Vetorial	Matricial
Armazenamento	Por coordenadas (mais eficiente)	Requer mais espaço de armazenamento
Algoritmos	Problemas com erros geométricos	Processsamento mais rápido e eficiente.
Escalas de trabalho	Adequado tanto a grandes quanto a pequenas escalas	Mais adequado para pequenas escalas (1:25.000 e menores)
Análise, Simulação e Modelagem	Representação indireta de fenômenos contínuos Álgebra de mapas é limitada	Representa melhor fenômenos com variação contínua no espaço Simulação e modelagem mais fáceis

1.8.3 Estruturas de dados para geo-campos numéricos

Para geo-campos numéricos, a escolha básica se dá entre malhas triangulares e grades regulares. As demais estruturas de dados (amostras 2,5D e isolinhas) são formatos intermediários, utilizados para entrada ou saída de dados, mas não adequadas para análise.

As malhas triangulares são normalmente melhores para representar a variação do terreno, pois capturam a complexidade do relevo sem a necessidade de grande quantidade de dados redundantes. As grades regulares têm grande redundância em terrenos uniformes e dificuldade de adaptação a relevos de natureza distinta no mesmo mapa, por causa da grade de amostragem fixa.

Para o caso de variáveis geofísicas e para operações como visualização 3D, as grades regulares são preferíveis, principalmente pela maior facilidade de manuseio computacional. A Tabela 1.4 resume as principais vantagens e desvantagens de grades regulares e malhas triangulares.

Tabela 1.4 – Estruturas para geo-campos numéricos

	Malha triangular	*Grade regular*
Vantagens	Melhor representação de relevo complexo Incorporação de restrições como linhas de crista	Facilita manuseio e conversão Adequada para dados não-altimétricos
Problemas	Complexidade de manuseio	Representação relevo complexo Cálculo de declividade

1.8.4 Representações computacionais de atributos de objetos

Entende-se por atributo qualquer informação descritiva (nomes, números, tabelas e textos) relacionada com um único objeto, elemento, entidade gráfica ou um conjunto deles, que caracteriza um dado fenômeno geográfico. Nos bancos de dados geográficos, os atributos de objetos geográficos são armazenados em relações convencionais. As representações geométricas destes objetos podem ser armazenadas na mesma tabela que os atributos ou em tabelas separadas, mas ligadas por identificadores únicos. Estes aspectos são discutidos em maior detalhe nos capítulos 3, 5, e 8 deste livro.

1.9 Universo de implementação

No universo de implementação, são tomadas as decisões concretas de programação e que podem admitir número muito grande de variações. Estas decisões podem levar em conta as aplicações às quais o sistema é voltado, a disponibilidade de algoritmos para tratamento de dados geográficos e o desempenho do hardware. Neste livro, aspectos do universo de implementação são tratados em diferentes capítulos:

- Os algoritmos de geometria computacional para problemas como ponto-em-polígono, simplificação de linhas e intersecção de linhas e polígonos são tratados no Capítulo 2.

- Os problemas de indexação espacial, que representam um componente determinante no desempenho total do sistema, são abordados no Capítulo 6.
- As questões de processamento e otimização de consultas espaciais são discutidas no Capítulo 7.
- Uma discussão detalhada dos SGBDs com suporte espacial é apresentada no Capítulo 8.
- Os Capítulos 12 a 14 apresentam a biblioteca TerraLib, um ambiente para construção de aplicativos geográficos.

1.10 Leituras suplementares

Este capítulo apresentou uma visão geral dos diferentes aspectos envolvidos com a representação computacional dos dados geográficos, que é o grande objetivo dos bancos de dados geográficos. Para uma visão geral de geoinformação sob o ponto de vista da Ciência da Computação, a referência mais atualizada é Worboys e Duckham (2004). O livro de Druck et al (2004) apresenta uma discussão sobre as questões de análise espacial de dados geográficos. Sobre o tema de bancos de dados geográficos, os livros de Rigaux et al (2002) e Shekar e Chawla (2002) são leituras complementares a este livro. A coletânea editada por Sellis et al (2003) apresenta um conjunto de artigos excelentes sobre os problemas emergentes de bancos de dados espaço-temporais. ■

Referências

BERNERS-LEE, T.; HENDLER, J.; LASSILA, O. The Semantic Web. **Scientific American**, v. May, 2001.

BIRKIN, M.; CLARKE, G.; CLARKE, M. P.; WILSON, A. **Intelligent GIS : Location Decisions e Strategic Planning.** New York: John Wiley, 1996.

BONDY, J. A.; MURTY, U. S. R. **Graph Theory with Applications.** London: The Macmillan Press LTD, 1977.

BÖNISCH, S.; ASSAD, M. L.; CÂMARA, G.; MONTEIRO, A. M. Representação e Propagação de Incertezas em Dados de Solos: I - Atributos Numéricos. **Revista Brasileira de Ciência do Solo**, v. 28, n.1, p. 33-47, 2004.

BURROUGH, P. **Principles of Geographical Information Systems for Land Resources Assessment**. Oxford, England, Oxford University Press, 1986.

CALDEIRA, T. **Cidade de Muros: Crime, Segregação e Cidadania em São Paulo (City of Walls: Crime, Segregation e Citizenship in Sao Paulo).** São Paulo: Edusp, 2000.

CÂMARA, G. **Modelos, Linguagens e Arquiteturas para Bancos de Dados Geográficos.**São José dos Campos, SP: Instituto Nacional de Pesquisas Espaciais (INPE), 1995.Ph.D., 1995.

CÂMARA, G.; SOUZA, R.; FREITAS, U.; GARRIDO, J. SPRING: Integrating Remote Sensing e GIS with Object-Oriented Data Modelling. **Computers e Graphics**, v. 15, n.6, p. 13-22, 1996.

CÂMARA, G. S., R.C.M.; MONTEIRO, A.M.V.; PAIVA, J.A.C; GARRIDO, J. Handling Complexity in GIS Interface Design. In: I Brazilian Workshop on Geoinformatics. SBC, Campinas, SP, 1999.

CASTELLS, M. **A Sociedade em Rede.** São Paulo: Paz e Terra, 1999.

CHEN, P. S. S. The Entity-Relationship Model: Towards a Unified View of Data. **ACM Transactions on Database Systems**, v. 1, n.1, p. 9-36, 1976.

COUCLELIS, H. From Cellular Automata to Urban Models: New Principles for Model Development e Implementation. **Environment e Planning B: Planning e Design**, v. 24, p. 165-174, 1997.

DAVIS, C.; BORGES, K.; LAENDER, A. OMT-G: An Object-Oriented Data Model for Geographic Applications. **GeoInformatica**, v. 3, n.1, 2002.

DRUCK, S.; CARVALHO, M. S.; CÂMARA, G.; MONTEIRO, A. M. V. **Análise Espacial de Dados Geográficos.** Brasília: EMBRAPA (ISBN 85-7383-260-6), 2004.

EGENHOFER, M. Spatial SQL: A Query e Presentation Language. **IEEE Transactions on Knowledge e Data Engineering**, v. 6, n.1, p. 86-95, 1994.

EGENHOFER, M.; FRANZOSA, R. Point-Set Topological Spatial Relations. **International Journal of Geographical Information Systems**, v. 5, n.2, p. 161-174, 1991.

ESRI, 2000, Modelling Our World : The ESRI Guide to Geodatabase Design, Redlands, CA.

FONSECA, F.; DAVIS, C.; CAMARA, G. Bridging Ontologies e Conceptual Schemas in Geographic Applications Development. **Geoinformatica**, v. 7, n.4, p. 355-378, 2003.

GINZBURG, C. **Olhos de Madeira: Nove Reflexões sobre a Distância.** São Paulo: Companhia das Letras, 2001.

GODIN, L. **GIS in Telecommunications.** Redlands, CA: ESRI Press, 2001.

GOMES, J.; VELHO, L. Abstraction Paradigms for Computer Graphics. **The Visual Computer**, v. 11, n.5, p. 227-239, 1995.

GROSS, J.; YELLEN, J. **Graph Theory e Its Applications.** Boca Raton, FL: CRC Press, 1998.

GRUBER, T. R. Toward Principles for the Design of Ontologies Used for Knowledge Sharing. **Int. Journal of Human-Computer Studies**, v. 43, p. 907-928, 1995.

HOHL, P. **GIS Data Conversion: Strategies, Techniques, e Management.** Clifton Park, NY: OnWorld Press, 1998.

KRAAK, M.-J.; BROWN, A., eds., 2001, **Web Cartography**. London, Taylor e Francis.

KUHN, W.; FRANK, A., 1991. A Formalization of Metaphors e Image-Schemas in User Interfaces. In: MARK, D.; FRANK, A., eds., **Cognitive e Linguistic Aspects of Geographic Space**: Dordrecht, Kluwer Academic Publishers, p. 419-434.

LI, Z.; ZHU, Q.; GOLD, C. **Digital Terrain Modeling: Principles e Methodology.** London: Taylor e Francis, 2004.

MACEACHREN, A. M. **How Maps Work : Representation, Visualization, e Design.** New York: Guilford Press, 2004.

MASSEY, D. S.; DENTON, N. A. **American Apartheid: Segregation e the Making of the Underclass.** Cambridge: Harvard University Press, 1993.

MATHER, P. M. **Computer Processing of Remotely-Sensed Images : An Introduction (3rd ed).** New York: John Wiley, 2004.

MONMONIER, M. **Mapping It Out: Expository Cartography for the Humanities e Social Sciences.** Chicago: University of Chicago Press, 1993.

NOY, N. F.; SINTEK, M.; DECKER, S.; CRUBEZY, M.; FERGERSON, R. W.; MUSEN, M. A. Creating Semantic Web Contents with Protege-2000. **IEEE Intelligent Systems**, v. 16, n.2, p. 60-71, 2001.

OGC, 1998, The OpenGIS Specification Model: The Coverage Type e Its Subtypes, Wayland, MA, Open Geospatial Consortium.

PARDINI, R.; SOUZA, S.; BRAGANETO, R.; METZGER, J.-P. The role of forest structure, fragment size e corridors in maintaining small mammal abundance e diversity in an Atlantic forest landscape. **Biological Conservation**, v. 124, p. 253-266, 2005.

RAVADA, S. **Topology Management in Oracle Spatial 10g.** Dagstuhl Seminar on Computational Cartography e Spatial Modelling, 2003. http://www.dagstuhl.de/03401/Materials/.

RICHARDS, J.; EGENHOFER, M. A Comparison of Two Direct-Manipulation GIS User Interfaces for Map Overlay. **Geographical Systems**, v. 2, n.4, p. 267-290, 1995.

RIGAUX, P.; SCHOLL, M.; VOISARD, A. **Spatial Databases with Application to GIS.** San Francisco: Morgan Kaufman, 2002.

RUMBAUGH, J.; BLAHA, M.; PREMERLANI, W.; EDDY, F.; LORENSEN, W. **Object-Oriented Modeling e Design.** Englewood Cliffs, NJ: Prentice-Hall, 1991.

SANTOS, M. **Espaço e Método.** São Paulo: Nobel, 1985.

SEARLE, J. R. Rationality e Realism, What is at Stake? **Dædalus**, v. 122, n.4, 1993.

SEARLE, J. R. **Mind, Language e Society.** New York: Basic Books, 1998.

SELLIS, T.; FRANK, A. U.; GRUMBACH, S.; GUTING, R. H.; KOUBARAKIS, M., eds., 2003, Spatio-Temporal Databases: The Chorochronos Approach: Berlin, Springer.

SHEKHAR, S.; CHAWLA, S. **Spatial Databases: A Tour.** New York: Prentice-Hall, 2002.

SMITH, B., 2003. Ontology e Information Systems. In: ZALTA, E. N., ed., **The Stanford Encyclopedia of Philosophy. The Metaphysics Research Lab, Center for the Study of Language e Information.**: Stanford, Stanford University.

SMITH, B.; MARK, D. Ontology e Geographic Kinds. In: International Symposium on Spatial Data Handling. Vancouver, Canada, 1998. p. 308-320.

SPOSATI, A. **Mapa de Exclusão/Inclusão Social de São Paulo.** São Paulo: EDUC, 1996.

STEVENS, S. S. On the theory of scales of measurement. **Science**, v. 103, n.2684, p. 677-680, 1946.

TOMLIN, C. D. **Geographic Information Systems e Cartographic Modeling.** Englewood Cliffs, NJ: Prentice-Hall, 1990.

TORRES, H. Segregação residencial e políticas públicas: São Paulo na década de 1990 (Spatial segregation e public policies: São Paulo in the 1990s). **Revista Brasileira de Ciências Sociais (Brazilian Social Sciences Journal)**, v. 54, p. 41-56, 2004.

TUFTE, E. **The Visual Display of Quantitative Information.** Chesire, CT: Graphics Press, 1983.

WHITE, M. J. The measurement of spatial segregation. **American Journal of Sociology**, v. 88, p. 1008-1018, 1983.

WORBOYS, M. F.; DUCKHAM, M. **GIS - A Computing Perspective (2nd edition).** Boca Raton: CRC Press, 2004.

2 *Algoritmos geométricos e relacionamentos topológicos*

Clodoveu A. Davis Jr.
Gilberto Ribeiro de Queiroz

2.1 Introdução

Este capítulo apresenta uma introdução às principais técnicas e algoritmos utilizados na implementação das diversas funções de um sistema de gerência de bancos de dados espaciais (SGBDE), em especial as operações sobre representações vetoriais (pontos, linhas e polígonos), que estão subjacentes a situações típicas, tais como:

- Seleção por apontamento, em que um usuário seleciona um determinado objeto através da interface gráfica;
- Determinação do relacionamento espacial entre dois objetos, tanto para consultas quanto para o estabelecimento de restrições de integridade espaciais no banco de dados;
- Criação de mapas de distância (buffer zones) e solução de problemas de proximidade;
- Sobreposição e aritmética de polígonos para operações de análise espacial.

Essas operações são alvo de estudo de uma área da Ciência da Computação conhecida como Geometria Computacional (Preparata e Shamos, 1985), que procura desenvolver e analisar algoritmos e estruturas de dados para resolver problemas geométricos diversos. Neste particular, tem um ponto importante de contato com a área de projeto e análise de algoritmos, uma vez que também procura caracterizar a dificuldade de problemas específicos, determinando a eficiência computacional dos algoritmos e usando técnicas de análise de

complexidade assintótica (Knuth, 1973). Existe também uma preocupação em desenvolver soluções para problemas clássicos de geometria, construindo estruturas mais apropriadas para a representação geométrica robusta no ambiente computacional, que tem limitações conhecidas quanto à precisão numérica e a capacidade de armazenamento de dados (Schneider, 1997).

2.2 Definições

Em um SGBDE, cada objeto vetorial é codificado usando um ou mais pares de coordenadas, o que permite determinar sua localização. Para entender melhor a maneira como os SGBDE tratam a informação vetorial, são relacionadas a seguir algumas definições fundamentais (Davis Jr., 1997). Como na maioria dos SGBDE, as definições consideram apenas duas dimensões.

- *Ponto:* um *ponto* é um par ordenado *(x, y)* de coordenadas espaciais.
- *Reta e segmento de reta:* Sejam p_1 e p_2 dois pontos distintos no plano. A combinação linear $\alpha.p_1 + (1-\alpha)p_2$, onde α é qualquer número real, é uma *reta* no plano. Quando $0 \leq \alpha \leq 1$, se tem um *segmento de reta* no plano, que tem p_1 e p_2 como *pontos extremos*.
- A definição de reta e segmento é estritamente geométrica, e nos interessa uma definição mais aplicada. Assim, partimos para o conceito de linha poligonal, que é composta por uma seqüência de segmentos de reta. O mais comum, no entanto, é definir a linha poligonal através da seqüência dos pontos extremos de seus segmentos, ou seja, seus vértices.
- *Linha poligonal:* Sejam $v_0, v_1, \ldots, v_{n-1}$ n pontos no plano. Sejam $s_0 = \overline{v_0 v_1}, s_1 = \overline{v_1 v_2}, \ldots, s_{n-2} = \overline{v_{n-2} v_{n-1}}$ uma seqüência de n - 1 segmentos, conectando estes pontos. Estes segmentos formam uma *poligonal* L se, e somente se, (1) a interseção de segmentos consecutivos é apenas o ponto extremo compartilhado por eles (i.e., $s_i \cap s_{i+1} = v_{i+1}$), (2) segmentos não consecutivos não se interceptam (i.e., $s_i \cap s_j = \varnothing$ para todo *i, j* tais que $j \neq i+1$), e (3) $v_0 \neq v_{n-1}$, ou seja, a poligonal não é fechada.

Observe, na definição da linha poligonal, a exclusão da possibilidade de auto-interseção. Os segmentos que compõem a poligonal só se tocam nos vértices. Formalmente, poligonais que não obedecem a este critério são chamadas *poligonais complexas*. Estas poligonais podem criar dificuldades na definição da topologia e em operações como a criação de buffers (vide Seção 2.8).

- *Polígono:* Um polígono é a região do plano limitada por uma linha poligonal fechada.

A definição acima implica que, apenas invertendo a condição (3) da definição de linha poligonal, temos um polígono. Assim, também aqui não é permitida a interseção de segmentos fora dos vértices, e os polígonos onde isto ocorre são denominados *polígonos complexos*. Os mesmos comentários que foram feitos para poligonais valem para os polígonos. Observe-se também que o polígono divide o plano em duas regiões: o interior, que convencionalmente inclui a fronteira (a poligonal fechada) e o exterior.

Estas três entidades geométricas básicas podem ser definidas em uma linguagem de programação usando tipos abstratos de dados. Essa definição inclui tipos abstratos para retângulos e para segmentos, que são bastante úteis nos testes preliminares de alguns algoritmos geométricos. Não foi definido um tipo abstrato específico para polígonos, uma vez que correspondem a poligonais em que o primeiro e o último vértices coincidem. Para as poligonais, foi incluída no tipo uma variável `Retângulo`, para armazenar os limites do objeto segundo cada eixo[1].

```
estrutura Ponto
início
   inteiro x;
   inteiro y;
fim;
estrutura Segmento
início
   Ponto p1;
```

[1] Este retângulo é usualmente denominado *retângulo envolvente mínimo* (REM), e é o menor retângulo com lados paralelos aos eixos que contém o objeto em questão.

```
   Ponto p2;
fim;
estrutura Retângulo
início
   inteiro x1;
   inteiro y1;
   inteiro x2;
   inteiro y2;
fim;
estrutura Poligonal
início
   inteiro numPontos;
   Retângulo retânguloEnvolventeMínimo;
   Ponto[] vertice;
fim;
```

Programa 2.1 - Tipos abstratos de dados para Ponto, Retângulo e Poligonal.

2.3 Algoritmos básicos

Diversos problemas de geometria computacional utilizam resultados básicos de problemas mais simples em sua solução. Alguns destes resultados básicos vêm da análise geométrica do mais simples dos polígonos, e o único que sempre é plano: o triângulo.

2.3.1 Área de um triângulo

A determinação da área de um triângulo é uma das operações mais básicas empregadas por outros algoritmos. Ela é calculada como a metade da área de um paralelogramo (Figura 2.1) (Figueiredo e Carvalho, 1991). O produto vetorial dos vetores A e B determina a área (S) do paralelogramo com os lados A e B e, portanto, a área do triângulo ABC (que corresponde à metade do paralelogramo) pode ser computada a partir da seguinte equação:

$$S = \frac{1}{2}\begin{vmatrix} x_a & y_a & 1 \\ x_b & y_b & 1 \\ x_c & y_c & 1 \end{vmatrix} = \frac{1}{2}(x_a y_b - y_a x_b + y_a x_c - x_a y_c + x_b y_c - y_b x_c) \qquad (2.1)$$

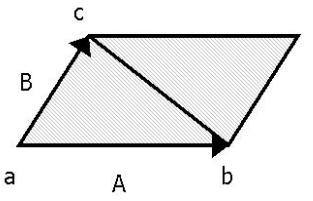


Figura 2.1 - Área do triângulo abc.

A Equação 2.1 fornece outra informação muito útil para os algoritmos de geometria computacional: a orientação dos três pontos que formam o triângulo. Caso a área seja negativa, os pontos a, b e c encontram-se no sentido horário; se positiva, os pontos encontram-se no sentido anti-horário; e se for zero, indica que os três pontos são colineares (estão alinhados).

2.3.2 Coordenadas baricêntricas

Para determinar se um determinado ponto pertence ou não a um triângulo, utiliza-se um método baseado em *coordenadas baricêntricas* (Figueiredo e Carvalho, 1991). De acordo com esse método, cada ponto p do plano pode ser escrito na forma $p = \lambda_1 p_1 + \lambda_2 p_2 + \lambda_3 p_3$, onde λ_1, λ_2 *e* λ_3 são números reais e $\lambda_1 + \lambda_2 + \lambda_3 = 1$. Os coeficientes λ_1, λ_2 e λ_3 são denominados *coordenadas baricêntricas* de p em relação a p_1, p_2 e p_3.

Os valores de λ_1, λ_2 e λ_3 podem ser obtidos usando a regra de Cramer, e expressos em termos de áreas de triângulos cujos vértices são p, p_1, p_2 e p_3. Temos, portanto:

$$\lambda_1 = \frac{S(pp_2p_3)}{S(p_1p_2p_3)},\ \lambda_2 = \frac{S(p_1pp_3)}{S(p_1p_2p_3)} \text{ e } \lambda_3 = \frac{S(p_1p_2p)}{S(p_1p_2p_3)}$$

A análise do sinal das coordenadas baricêntricas indica a região do plano em que se encontra p, em relação ao triângulo $p_1p_2p_3$ (Figura 2.2). Observe-se que, para isso, as áreas devem ser orientadas, ou seja, com sinal.

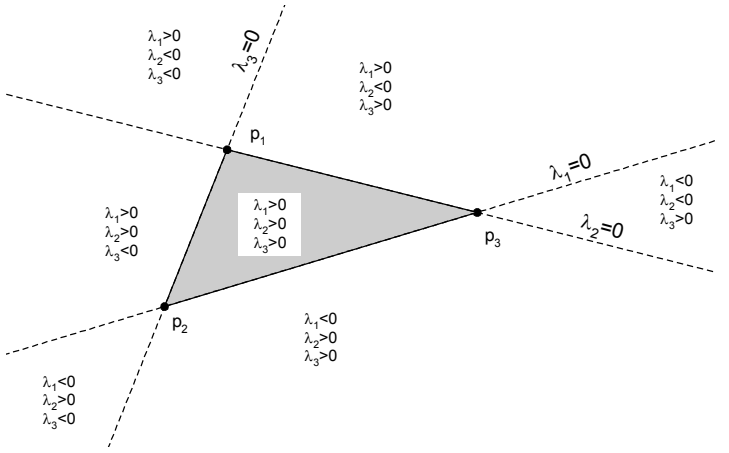


Figura 2.2 - Sinais das coordenadas baricêntricas.

2.3.3 Interseção de retângulos

Diversos algoritmos de geometria computacional se beneficiam de estratégias de implementação que procuram evitar, tanto quanto possível, o uso de procedimentos computacionalmente caros. Assim, os problemas são geralmente resolvidos em duas etapas: uma em que a representação geométrica dos objetos envolvidos é simplificada, e outra, executada apenas se necessário, em que a representação completa é empregada.

O uso do retângulo envolvente mínimo (REM) é uma dessas estratégias. O REM é o menor retângulo com lados paralelos aos eixos coordenados que contém a geometria do objeto. Por exemplo, antes de executar o algoritmo de determinação da interseção entre dois polígonos, comparamos seus REM diretamente. Caso os REM tenham alguma interseção, é *possível* que os polígonos se interceptem, e portanto o algoritmo completo precisa ser executado (Figura 2.3a); caso contrário, é certo que não existe interseção entre os objetos, e portanto a solução é o conjunto vazio (Figura 2.3b).

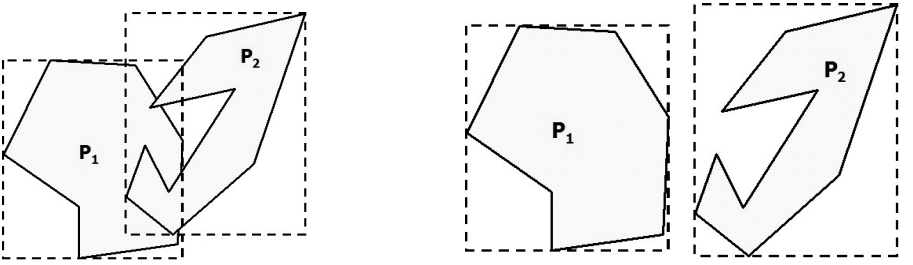


Figura 2.3 – (a) Interseção dos REMs (b) REMs disjuntos.

O programa 2.2 ilustra este teste.

```
função interseçãoRetângulos(Ponto A, Ponto B, Ponto C, Ponto
   D): booleano
início
   Ponto P, Ponto Q, Ponto P1, Ponto Q1;
   P.x = min(A.x, B.x);
   P.y = min(A.y, B.y);
   Q.x = max(A.x, B.x);
   Q.y = max(A.y, B.y);
   P1.x = min(C.x, D.x);
   P1.y = min(C.y, D.y);
   Q1.x = max(C.x, D.x);
   Q1.y = max(C.y, D.y);
   retorne ((Q.x >= P1.x) e (Q1.x >= P.x) e
               (Q.y >= P1.y) e (Q1.y >= P.y));
fim.
```

Programa 2.2 – Interseção de retângulos envolventes mínimos.

2.3.4 Interseção de dois segmentos de reta

Dados dois segmentos a e b, formados pelos pontos p_1p_2 e p_3p_4 (Figura 2.4), respectivamente, deseja-se verificar se eles se interceptam. A solução consiste em testar se os pontos p_1 e p_2 estão de lados opostos do segmento formado por p_3p_4 e também se p_3 e p_4 estão de lados opostos do segmento formado por p_1p_2. Este problema se conecta com o problema da área de triângulo, pois, determinar se p_3 está do lado oposto de p_4 em relação ao segmento p_1p_2, consiste em avaliar o sinal da área dos triângulos formados por $p_1p_2p_3$ e $p_1p_2p_4$. Se os sinais forem contrários, significa que os pontos

estão de lados opostos. Se o mesmo for verdadeiro para os triângulos $p_3p_4p_1$ e $p_3p_4p_2$, então, com certeza podemos afirmar que as retas que passam pelos segmentos se interceptam em algum ponto, embora não se possa afirmar ainda que os segmentos têm interseção.

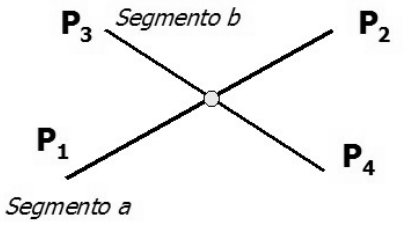


Figura 2.4 – Segmentos que se interceptam.

Em Saalfeld (1987) é discutida uma forma de determinar o ponto de interseção entre dois segmentos baseada na representação paramétrica dos segmentos[2]. Dados dois segmentos formados pelos pontos p_1p_2 e p_3p_4, respectivamente, e com $p_1 = (x_1, y_1)$, $p_2 = (x_2, y_2)$, $p_3 = (x_3, y_3)$ e $p_4 = (x_4, y_4)$, o ponto de interseção entre eles é dado por:

$$p_1 + u(p_2 - p_1) = p_3 + v(p_4 - p_3) \tag{2.2}$$

Esta igualdade dá origem a um sistema com duas equações e duas incógnitas (u e v):

$$\begin{cases} x_{intersecao} = x_1 + u(x_2 - x_1) \text{ ou } x_{intersecao} = x_3 + v(x_4 - x_3) \\ y_{intersecao} = y_1 + u(y_2 - y_1) \text{ ou } y_{intersecao} = y_3 + v(y_4 - y_3) \end{cases} \tag{2.3}$$

Desenvolvendo o sistema temos:

$$u = \frac{(x_4 - x_3)(y_1 - y_3) - (y_4 - y_3)(x_1 - x_3)}{(y_4 - y_3)(x_2 - x_1) - (x_4 - x_3)(y_2 - y_1)} \tag{2.4}$$

$$v = \frac{(x_2 - x_1)(y_1 - y_3) - (y_2 - y_1)(x_1 - x_3)}{(y_4 - y_3)(x_2 - x_1) - (x_4 - x_3)(y_2 - y_1)} \tag{2.5}$$

Calculados os parâmetros u e v, podemos determinar o ponto de interseção:

---

[2] A equação paramétrica para um segmento de coordenadas p_1 e p_2 é dada por: $p = p_1 + u(p_2 - p_1)$, onde se $0 < u < 1$ define um ponto localizado entre p_1 e p_2.

$$\begin{cases} x_{interseccao} = x_1 + u(x_2 - x_1) \text{ ou } x_{interseccao} = x_3 + v(x_4 - x_3) \\ y_{interseccao} = y_1 + u(y_2 - y_1) \text{ ou } y_{interseccao} = y_3 + v(y_4 - y_3) \end{cases} \quad (2.6)$$

As expressões de *u* e *v*, respectivamente 2.4 e 2.5, possuem interpretações importantes. Os denominadores são os mesmos, e, portanto, numa implementação computacional eles deverão ser calculados uma única vez. Se o denominador for zero, as duas linhas são paralelas. Se além do denominador os numeradores de ambos os parâmetros também forem zero, então as duas linhas são coincidentes. Na verdade, as equações paramétricas aplicam-se a linhas e, portanto, só haverá interseção entre os dois segmentos em um ponto localizado sobre ambos, o que significa valores de u e v ambos no intervalo [0,1].

2.3.5 Área de polígonos

A área de um polígono pode ser calculada em tempo linear com relação ao número de vértices, usando um somatório simples, baseado na soma de áreas de triângulos formados entre cada par de vértices consecutivos e a origem do sistema de coordenadas (O'Rourke, 1998):

$$A(P) = \frac{1}{2} \times \sum_{i=0}^{i=n-1} (x_i + x_{i+1}) \times (y_{i+1} - y_i) \quad (2.7)$$

Observe que na expressão acima (Equação 2.7), quando se tem $i = n - 1$, é necessário ter $x_n = x_0$ e $y_n = y_0$. Como no caso dos triângulos, o sinal da área obtida de acordo com esta fórmula indica a orientação dos vértices do polígono. Caso o resultado seja positivo, os vértices estão ordenados no sentido anti-horário, e caso seja negativo os vértices encontram-se no sentido horário.

2.3.6 Centróide de um polígono

O *centro de gravidade* ou *centro de massa*, mais conhecido como *centróide* de um polígono pode ser obtido a partir da sua divisão em triângulos, calculando em seguida a média ponderada dos centros de gravidade dos triângulos usando suas áreas como peso. O centro de gravidade de cada triângulo é simplesmente a média das coordenadas de seus vértices, ou seja, para um triângulo ABC:

$$x_G = \frac{x_A + x_B + x_C}{3} \text{ e } y_G = \frac{y_A + y_B + y_C}{3}$$

Embora este processo seja relativamente simples, pressupõe-se a implementação de um algoritmo de triangulação de polígonos. Os centróides dos triângulos são combinados usando um processo de média ponderada pela área. Assim, o centróide de um polígono formado por dois triângulos T_1 e T_2, cujos centróides são, respectivamente, (x_{G1}, y_{G1}) e (x_{G2}, y_{G2}) é o ponto (x_G, y_G), onde

$$x_G = \frac{x_{G1}S(T_1) + x_{G2}S(T_2)}{S(T_1) + S(T_2)} \; y_G = \frac{y_{G1}S(T_1) + y_{G2}S(T_2)}{S(T_1) + S(T_2)}$$

e o centróide do polígono pode ser determinado de maneira incremental, adicionando um triângulo e seu centróide por vez e calculando as coordenadas do centróide do conjunto.

No entanto, existe uma solução mais simples e independente da triangulação, e que leva em conta triângulos com áreas positivas e negativas, como no cálculo da área do polígono. O mesmo processo de média ponderada pela área pode ser usado, considerando todos os triângulos formados entre um ponto fixo, por exemplo (0, 0), e cada par de vértices sucessivos, (v_i, v_{i+1}).

Assim, temos que:

$$\begin{aligned} A(P) &= \frac{1}{2}\sum_{i=0}^{n-1}(x_i y_{i+1} - y_i x_{i+1}) \\ x_C &= \frac{\sum_{i=0}^{n-1}(x_{i+1} + x_i) \times (x_i y_{i+1} - y_i x_{i+1})}{3A(P)} \\ y_C &= \frac{\sum_{i=0}^{n-1}(y_{i+1} + y_i) \times (x_i y_{i+1} - y_i x_{i+1})}{3A(P)} \end{aligned} \tag{2.8}$$

O resultado pode ser facilmente implementado em um algoritmo com complexidade $O(n)$, que naturalmente pode fornecer ao mesmo tempo a

área do polígono. Mas apesar da simplicidade do processo, não existe garantia de que o centróide será um ponto *pertencente* ao polígono. Caso seja necessário encontrar um ponto interno a um polígono simples dado, pode-se utilizar o seguinte processo, que busca precisamente identificar rapidamente uma diagonal do polígono (O'Rourke, 1998):

- identificar um vértice convexo v_i (por exemplo, o vértice inferior mais à direita)
- para cada outro vértice v_j do polígono verificar:
- se v_j estiver dentro do triângulo $v_{i-1}v_iv_{i+1}$, então calcular a distância v_iv_j
- armazenar v_j em q se esta distância for um novo mínimo
- ao final do processo, se algum ponto interior a $v_{i-1}v_iv_{i+1}$ for encontrado, então o ponto médio do segmento qv_i é interior ao polígono; senão, então o ponto médio do segmento $v_{i-1}v_{i+1}$ (ou mesmo o centróide do triângulo $v_{i-1}v_iv_{i+1}$) é interior ao polígono.

Outras definições de centróide consideram que o mesmo se situa "aproximadamente no centro do polígono" (Laurini e Thompson, 1992). O centróide pode ser determinado por diversos processos, como o centro do retângulo envolvente mínimo, o centro de um círculo inscrito ou circunscrito ao polígono. Uma forma freqüentemente usada para determinar um centróide consiste em simplesmente obter a média das coordenadas x e y dos vértices (Figura 2.5).

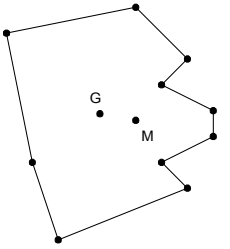


Figura 2.5 – Centróides calculados pela média (M) e como centro de gravidade (G).

2.4 Ponto em polígono

Uma das operações mais comuns em um SIG é determinar se um ponto está no interior de um polígono. Um dos algoritmos mais populares para solução deste problema é o teste do número de cruzamentos entre os segmentos que formam a fronteira do polígono e uma semi-reta (chamada de *raio*), que parte do ponto testado em qualquer direção (Haines, 1994) (Taylor, 1994). Se o número de cruzamentos for par, o ponto encontra-se fora do polígono; se for ímpar, encontra-se dentro. A Figura 2.6 ilustra a idéia desse teste. Conforme pode ser observado, o raio que parte do ponto *Q*, que está dentro do polígono, cruza os segmentos da fronteira um número ímpar de vezes (3 vezes).

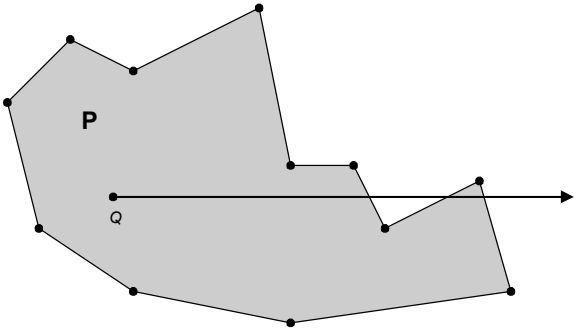


Figura 2.6 – Ponto em polígono.

Apesar da aparente simplicidade desse algoritmo, a sua implementação deve considerar alguns casos particulares (casos degenerados), como:

- a semi-reta passa por uma aresta do polígono (Figura 2.7a);
- a semi-reta passa por um vértice do polígono (Figura 2.7b);
- o ponto *Q* está sobre a fronteira do polígono (Figura 2.7c);
- o ponto *Q* coincide com um vértice do polígono (Figura 2.7d).

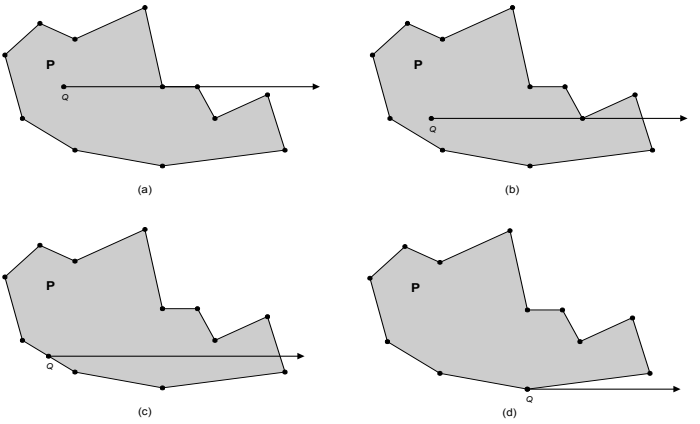


Figura 2.7 – Ponto em polígono : casos degenerados.

Para estes casos, a solução está em adotar um critério para a contagem de interseções de modo que:

- se a reta passa por um vértice, a interseção deve ser considerada apenas se for o vértice com maior ordenada do segmento, e ignorada caso contrário;
- se a reta passa por um segmento do contorno do polígono, nenhuma interseção deve ser considerada;
- se o ponto Q pertence a um segmento do contorno (exceto pontos extremos), considerar como uma interseção.

O caso em que Q coincide com um vértice pode ser tratado pelo primeiro critério. O terceiro critério faz com que todos os pontos da fronteira sejam considerados como pertencentes ao polígono.

Esse algoritmo possui complexidade linear em relação ao número de vértices do polígono. Para uma análise mais aprofundada do problema, o leitor é convidado a ler os trabalhos de Huang e Shih (1997) e Haines (1994).

2.5 Simplificação de poligonais

Muitas entidades do mundo real podem ser modeladas como linhas ou, mais genericamente, poligonais[3]. Essas entidades são freqüentes em bases de dados geográficas, onde correspondem tipicamente a cerca de 80% do volume de dados vetoriais (McMaster e Shea, 1992). Por isso, o problema de simplificação de linhas é particularmente importante, sendo estudado intensivamente desde os anos 60, quando ocorreram as primeiras experiências com o uso de instrumentos de transcrição de mapas para o computador, como a mesa digitalizadora.

No processo de digitalização de linhas, freqüentemente são introduzidos vértices em excesso, vértices que, se descartados, não provocariam uma alteração visual perceptível na poligonal. Assim, um primeiro objetivo para algoritmos de simplificação de linhas é "limpar" (significativamente, o verbo utilizado em inglês é *weed*, "capinar") a poligonal de pontos claramente desnecessários, do ponto de vista de sua visualização (Weibel, 1995), mantendo a qualidade de sua aparência gráfica (Peucker, 1975) (Beard, 1991).

Outro objetivo é o de gerar uma nova versão da linha, mais adequada para a representação do mesmo fenômeno geográfico em outra escala, menor que a original. Neste caso, está sendo obtida uma *generalização* da linha (McMaster, 1992). Em uma extensão deste enfoque, existe o interesse em organizar os vértices da poligonal de tal forma que seja possível produzir, dinamicamente, versões generalizadas adequadas para uma escala definida no momento da visualização (van Oosterom, 1993) (van Oosterom e Schenkelaars, 1995), conseguindo portanto gerar múltiplas representações geométricas para o mesmo fenômeno sem introduzir dados redundantes. No entanto, a utilização de métodos e algoritmos desenvolvidos originalmente apenas pensando na redução do número de vértices da linha podem não ser adequados para alcançar o objetivo de generalização (Laurini e Thompson, 1992), em geral por não

[3] Deste ponto em diante, será utilizado o termo *poligonal*, em lugar de simplesmente *linha*, para evitar confusão com a definição geométrica da linha reta (infinita).

conseguirem uma boa representação geométrica[4], e portanto devem ser analisados cuidadosamente quanto a este aspecto.

Assim, o problema de simplificação de linhas consiste em obter uma representação *mais grosseira* (formada por *menos vértices*, e portanto *mais compacta*) de uma poligonal a partir de uma representação mais refinada, atendendo a alguma restrição de aproximação entre as duas representações. Essa restrição pode ser definida de várias maneiras (Li e Openshaw, 1992), mas é em geral alguma medida da proximidade geométrica entre as poligonais, tais como o máximo deslocamento perpendicular permitido (Figura 2.8a) ou o mínimo deslocamento angular permitido (Figura 2.8b). Na Figura 2.8a, o vértice 2 será mantido, uma vez que a distância entre ele e a reta que passa pelos vértices 1 e 3 é superior à permitida. Na Figura 2.8b, o vértice 3 será eliminado, uma vez que o ângulo $3\hat{2}4$ é menor que o mínimo tolerável. Uma alternativa mais rara é a área entre as poligonais (Figura 2.8c), onde se estabelece um limite para ao deslocamento de área.

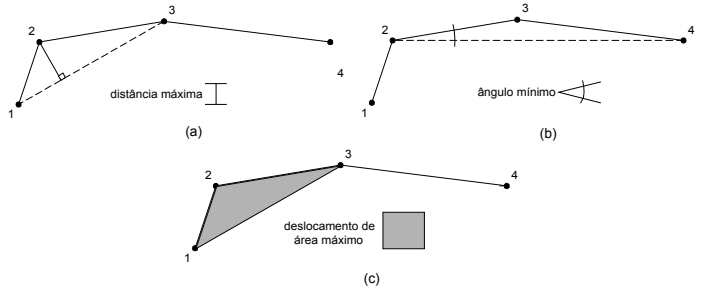


Figura 2.8 – Medidas de proximidade para simplificação de linhas.

Dentre todas as medidas possíveis, a mais utilizada é a distância perpendicular. O conceito de banda de tolerância, apoiado no cálculo de distâncias perpendiculares, é utilizado em grande parte dos algoritmos de simplificação que serão apresentados a seguir. Um problema eventualmente abordado na literatura é a escolha do parâmetro de tolerância (ε), e sua correlação com a escala da representação simplificada.

[4] Para auxiliar na manutenção do aspecto natural da poligonal, existem enfoques que integram algoritmos de simplificação com algoritmos de suavização.

Um critério interessante para a determinação da tolerância é o que usa o tamanho do menor objeto visível em uma determinada escala (Li e Openshaw, 1992). Este tamanho pode ser dado em termos de uma distância medida no espaço de coordenadas do mapa plotado, ou seja, em milímetros do papel, independente da escala utilizada. Assim, é definida uma correspondência linear entre a escala e a tolerância adotada.

Grande parte dos algoritmos de simplificação de poligonais necessita realizar de maneira eficiente cálculos de distância entre um ponto dado e uma reta definida por outros dois pontos. A maneira mais interessante de calcular essa distância é utilizar o produto vetorial, conforme apresentado na Seção 2.3.1, para determinar a área *S* do triângulo formado por um ponto *A* e uma reta definida por outros dois (*B* e *C*), de acordo com a equação 2.1. Assim, a distância do ponto *A* à reta definida pelos pontos *B* e *C* pode ser calculada como:

$$d = \frac{|S|}{dist(B,C)}$$

onde *dist(B, C)* é a distância euclidiana entre os pontos B e C.

Embora existam muitos algoritmos de simplificação de linhas, envolvendo variados critérios de aproximação e estratégias de processamento, um deles se destaca pela ampla aceitação. Foi proposto em 1973 por Douglas e Peucker (1973), e é reconhecidamente o melhor em termos de preservação das características da poligonal original (Marino, 1979) (McMaster, 1987).

```
Procedimento Douglas-Peucker(linha, numvert, tol)
   Procedimento DP(a, f, tol)
      início
            se ((f - a) == 1) então retorne;
               maxd = 0;
            maxp = 0;
            para i = a+1 até f-1 faça
            início
            d = distância(linha[i], linha[a], linha[f]);
            se d > maxd então
            início
                  maxd = d;
                  maxp = i;
            fim se;
            fim para;
            se maxd > tol então
            início
            vértice maxp selecionado;
```

```
                DP(a, maxp, tol);
                DP(maxp, f, tol);
            fim
            senão retorne;
        fim;
início
   vértice l selecionado;
   vértice numvert selecionado;
   DP(l, numvert, tol);
fim.
```

Programa 2.3 – Algoritmo Douglas-Peucker

O algoritmo é recursivo, e a cada passo processa o intervalo de pontos contido entre um vértice inicial (chamado de *âncora*) e um vértice final (denominado *flutuante*). É estabelecido um *corredor* de largura igual ao dobro da tolerância, formando duas faixas paralelas ao segmento entre o âncora e o flutuante (Figura 2.9b), como no algoritmo de Lang. A seguir, são calculadas as distâncias de todos os pontos intermediários ao segmento básico, ou seja, contidos entre o âncora e o flutuante. Caso nenhuma das distâncias calculadas ultrapasse a tolerância, ou seja, nenhum vértice fica fora do corredor, então todos os vértices intermediários são descartados. Caso alguma distância seja maior que a tolerância, o vértice mais distante é preservado, e o algoritmo é reiniciado em duas partes: entre o âncora e o vértice mais distante (novo flutuante), e entre o vértice mais distante (novo âncora) e o flutuante. De acordo com este processo, os pontos tidos como críticos para a geometria da linha, a cada passo, são mantidos, enquanto os demais são descartados.

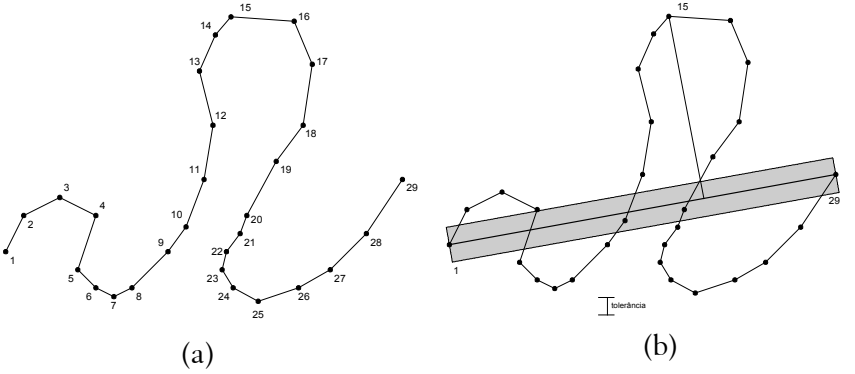


Figura 2.9 – Linha original, 29 vértices (a) e Douglas-Peucker, primeiro passo: seleção do vértice 15 (b).

Para a análise deste algoritmo e dos próximos será utilizada a poligonal da Figura 2.9a, com 29 vértices. As figuras seguintes ilustram melhor o comportamento do algoritmo Douglas-Peucker. Inicialmente, são calculadas as distâncias dos vértices 2 a 28 até a reta definida pelos vértices 1 e 29. O vértice mais distante nesta primeira iteração é o 15, a uma distância muito superior à tolerância (Figura 2.9b). Assim, o vértice 15 é selecionado e o procedimento é chamado recursivamente duas vezes, entre os vértices 1 e 15 e entre os vértices 15 e 29. Continuando pela primeira chamada, o vértice mais distante da reta entre 1 e 15 é o 9, também a uma distância superior à tolerância, e portanto é selecionado (Figura 2.10a). Duas novas chamadas recursivas são feitas, e agora estão empilhados os intervalos 1-9, 9-15 e 15-29. No intervalo 1-9, temos também que preservar o vértice 3, e portanto ficamos na pilha com os intervalos 1-3, 3-9, 9-15 e 15-29 (Figura 2.10b). Analisando agora o intervalo 1-3, verificamos que o vértice 2 pode ser dispensado (Figura 2.11a). Ao final, são preservados os vértices 1, 3, 4, 6, 9, 15, 16, 17, 22, 24, 27 e 29, ou seja, 41% do número original de vértices (Figura 2.11b).

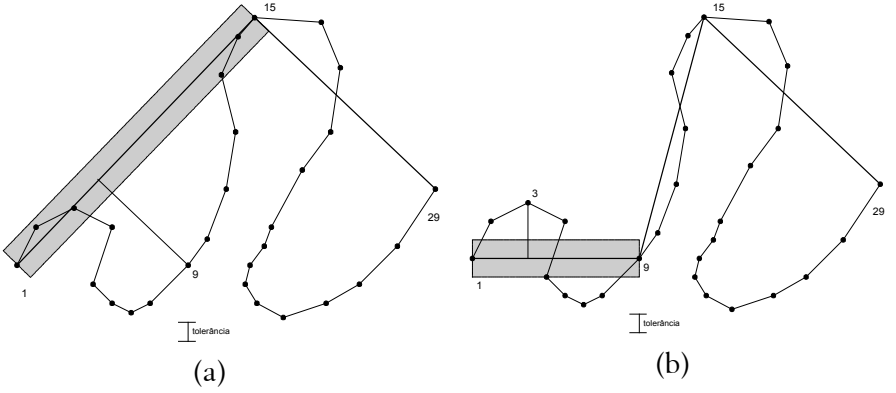


Figura 2.10 – Douglas-Peucker, segundo passo: seleção do vértice 9 (a) e Douglas-Peucker, terceiro passo: seleção do vértice 3 (b).

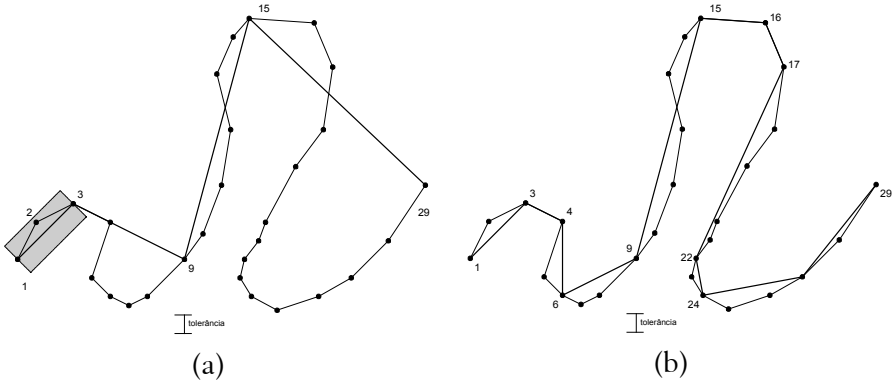


Figura 2.11 – Douglas-Peucker, passo 4: eliminação do vértice 2 (a) e Douglas-Peucker, final (b).

O resultado deste algoritmo é aclamado pela literatura como sendo o que mais respeita as características (ou, como no título do artigo de Douglas e Peucker, a "caricatura") da linha cartográfica (Marino, 1979). Assim, este algoritmo veio a ser a escolha dos desenvolvedores de software comercial na implementação de funções de simplificação de linhas para processamento pós-digitalização (Li e Openshaw, 1992), ou seja, para limpeza de vértices desnecessários. O uso do algoritmo Douglas-Peucker em generalização, no entanto, é comprometido pelo seu comportamento em situações de generalização mais radical, ou seja, com tolerâncias maiores. Conforme a situação, o algoritmo pode ser levado a escolher vértices que terminam por deixar a linha com uma aparência pouco natural, com tendência a apresentar "picos" (como no exemplo da Figura 2.11, entre os vértices 17, 24 e 29), com ângulos agudos e mudanças bruscas de direção.

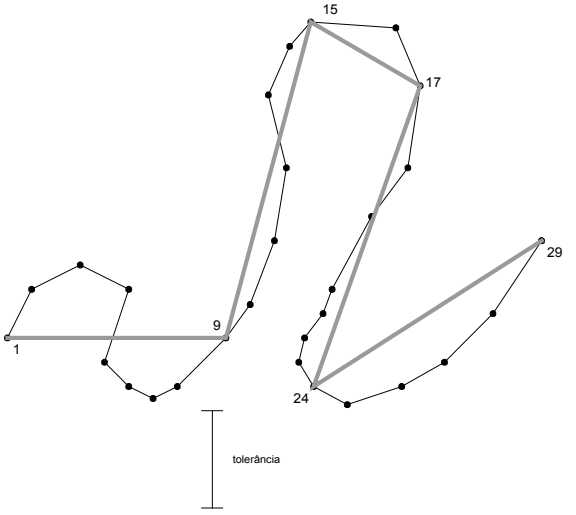


Figura 2.12 – Douglas-Peucker, simplificação radical.

Se a mesma linha da Figura 2.9a for processada novamente com uma tolerância, por exemplo, quatro vezes maior que a apresentada, seriam preservados apenas os vértices 1, 9, 15, 17, 24 e 29, todos pertencentes à solução anterior (Figura 2.12). Portanto, o algoritmo de Douglas-Peucker é hierárquico, pois os pontos são sempre selecionados na mesma ordem, e a tolerância serve para determinar até que ponto o processamento deve ser realizado.

Se a tolerância for igual a zero, todos os vértices serão eventualmente selecionados. O armazenamento das subdivisões nos permite representar a hierarquia dos vértices em uma árvore binária (van Oosterom, 1993). Em cada nó desta árvore é representado um vértice selecionado, e é armazenado o valor da distância calculado por ocasião da seleção, que corresponde ao valor `maxd` do algoritmo (Programa 2.3). Tendo sido estabelecido um valor de tolerância, basta caminhar na árvore em preordem para determinar quais vértices serão selecionados. Quando um nó interno contiver um valor de distância inferior à tolerância, o vértice correspondente e todos os descendentes poderão ser eliminados, não sendo necessário continuar com o caminhamento. Observe-se, no entanto, que a árvore binária pode ser bastante desbalanceada, e dificilmente será completa, o que virá a dificultar o seu armazenamento no banco de dados.

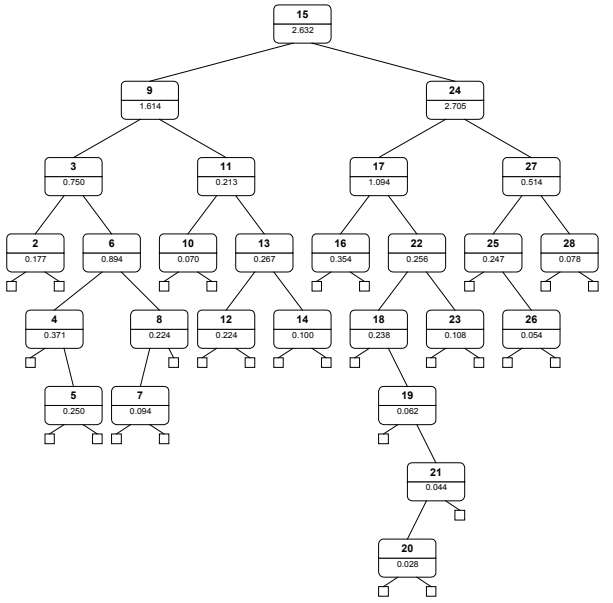


Figura 2.13 – Árvore binária formada a partir do exemplo da Figura 2.9 a.

2.6 Interseção de conjuntos de segmentos

O problema de se determinar os pontos de interseção entre um conjunto de segmentos é um dos mais importantes no caso de um SIG que trabalha com representação vetorial. Isso porque operações como união, interseção, diferença e as operações que avaliam o relacionamento topológico necessitam determinar esses pontos como uma das primeiras etapas de seus processamentos, sendo esta a de maior consumo de processamento. No que segue, são apresentados algoritmos para resolução deste problema.

2.6.1 Plane sweep

Shamos e Hoey (1976) apresentam um dos primeiros trabalhos discutindo o problema de interseção entre objetos com base na análise de complexidade. Eles fornecem um algoritmo para um problema similar que é determinar se, em um conjunto de n segmentos, há pelo menos um par que se intercepte.

A idéia para solução desse problema vem da análise de intervalos em uma dimensão. Considere-se que, em vez de n segmentos, tenha-se n intervalos entre números reais, do tipo $[x_L, x_R]$, onde $x_L \leq x_R$. Uma solução exaustiva seria analisar todos os n^2 pares de intervalos existentes, comparando-os sempre dois a dois, e interrompendo o processamento assim que a primeira interseção fosse detectada.

No entanto, uma maneira mais eficiente de resolver o problema é construir uma lista ordenada dos valores extremos dos intervalos, tomando o cuidado de identificá-los como sendo *L* ou *R*, de acordo com sua situação no intervalo. Assim, não haverá interseção alguma entre os intervalos se e somente se a lista ordenada contiver uma seqüência alternada de *Ls* e *Rs*: *L R L R ... L R L R*. Em qualquer outra situação, pode-se afirmar que existe superposição entre algum par de intervalos. Esta solução tem complexidade computacional da ordem de $O(n \log n)$, uma vez que é dominada pela ordenação dos valores extremos.

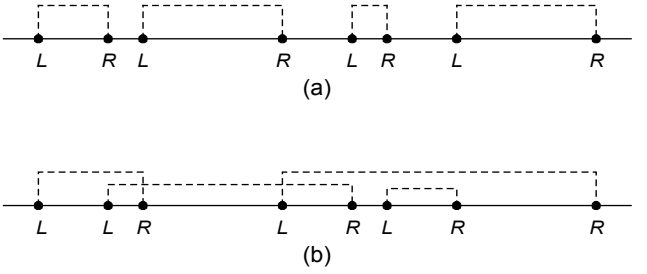


Figura 2.14 – Verificação de interseção em intervalos na reta

Em duas dimensões, o problema torna-se um pouco mais complicado, já que não existe maneira de produzir uma ordenação adequada para segmentos no plano. A técnica empregada é clássica na geometria computacional, e é denominada de *varredura do plano* (*plane sweep*). Esta técnica faz uso de duas estruturas de dados básicas, uma para registrar a situação da linha de varredura (*sweep line status*), e a outra que registra eventos ocorridos durante a varredura (*event-point schedule*).

A idéia consiste em deslocar uma reta vertical pelo conjunto de segmentos, buscando identificar inversões na ordem em que esta reta encontra dois segmentos quaisquer. Para implementar esta idéia, é

necessário definir uma nova relação de comparação, da seguinte forma: considere-se dois segmentos s_1 e s_2 no plano, sendo que s_1 não intercepta s_2. Diz-se que s_1 é *comparável* a s_2 se, para alguma abscissa x, existe uma linha vertical que intercepta tanto s_1 quanto s_2. Assim, diz-se que s_1 está *acima de* s_2 *em* x se, naquela abscissa, a interseção da reta com s_1 está acima da interseção da reta com s_2. Esta relação é denotada como $s_1 >_x s_2$. Na ז, temos as seguintes relações: $s_3 >_v s_2$; $s_4 >_v s_3$; $s_4 >_v s_2$; $s_4 >_w s_2$; $s_4 >_w s_3$; $s_2 >_w s_3$.

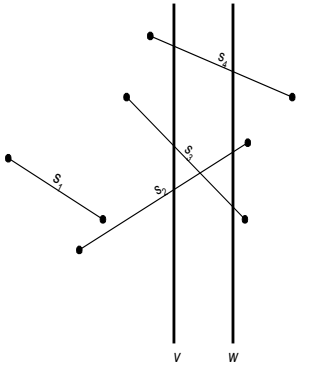


Figura 2.15 – Relação de ordenação entre segmentos.

Com esta relação é construída uma ordenação total dos segmentos, que muda à medida em que a linha é deslocada da esquerda para a direita. Nesse processo de varredura do plano, três coisas podem ocorrer:

- o ponto extremo à esquerda de um segmento é encontrado; o segmento é, portanto, inserido na ordenação;
- o ponto extremo à direita de um segmento é encontrado; o segmento é, portanto, retirado da ordenação;
- um ponto de interseção entre dois segmentos s_1 e s_2 foi encontrado; portanto, s_1 e s_2 trocam de posição na ordenação.

Observe-se que, para que s_1 e s_2 possam trocar de posição, é necessário que exista algum x para o qual s_1 e s_2 são consecutivos na ordenação. O algoritmo usa este fato, testando apenas elementos consecutivos, à medida em que novos eventos vão sendo detectados conforme descrito acima.

Portanto, é necessário operar duas estruturas de dados no processo. A primeira (*sweep line status*) é a responsável por manter a ordenação das interseções dos segmentos com a linha de varredura, e é usualmente implementada como um dicionário ou como uma árvore *red-black* (Cormen et al, 1990). As operações que o *sweep line status* deve suportar são inserção (insere, complexidade $O(\log n)$), exclusão (exclui, também $O(\log n)$), e duas funções para determinar qual segmento está imediatamente acima e imediatamente abaixo de um segmento dado na ordenação (acima e abaixo, $O(1)$). A segunda estrutura de dados (*event-point schedule*) é responsável por manter a seqüência das abscissas que serão analisadas pela linha de varredura, e é implementada como uma fila de prioridades. Deve suportar as clássicas operações de inclusão (insere), retirada do elemento de mais alta prioridade (min) e uma função que testa a presença de um determinado elemento na estrutura (membro), todas com complexidade $O(\log n)$.

Inicialmente, as abscissas dos pontos extremos dos segmentos são ordenadas e inseridas no *event-point schedule*. Em seguida, as abscissas são retiradas a partir da menor, e são realizadas as seguintes operações:

- Se a abscissa corresponder a um ponto extremo à esquerda de algum segmento, inserir o segmento no *sweep line status*. Verificar se existem interseções entre este segmento e os segmentos que estão imediatamente acima e abaixo dele na linha de varredura. Caso exista interseção, a abscissa do ponto de interseção deve ser calculada e inserida no *event-point schedule*, caso já não pertença a ele.
- Se for um ponto extremo à direita, excluir o segmento do *sweep line status*. Verificar se existem interseções entre os segmentos que estão imediatamente acima e abaixo dele na linha de varredura. Caso exista interseção (que estará necessariamente à direita do ponto extremo), a abscissa do ponto de interseção deve ser calculada e inserida no *event-point schedule*, caso já não pertença a ele.
- Se for um ponto de interseção entre dois segmentos, trocar a posição destes segmentos no *sweep line status*. Informar a existência de um ponto de interseção e suas coordenadas.

O algoritmo possui complexidade sub-ótima, $O(n \log n + k \log n)$, onde k é o número de interseções. Um dos motivos para que ele não atinja o limite inferior de $n \log n + k$ é que os pontos de interseção são reportados na ordem x, que é a ordem na qual eles são inseridos na fila de eventos. A complexidade desse algoritmo depende não só do número de segmentos de entrada, mas também do número de interseções reportadas. Esse algoritmo pertence a uma classe conhecida como algoritmos sensíveis à saída, sendo apresentado originalmente por Bentley e Ottmann (1979).

2.6.2 Algoritmos de interseção por partição do espaço

Andrews et al (1994) e Andrews e Snoeyink (1995) apresentam uma comparação entre métodos advindos da Geometria Computacional, e métodos desenvolvidos pela comunidade GIS (métodos pragmáticos) para resolver esse problema. Os algoritmos testados por eles foram agrupados em duas categorias: algoritmos por partição espacial e algoritmos por ordenação espacial.

Nos algoritmos da primeira classe, o espaço é subdivido em regiões, e os segmentos são atribuídos às regiões interceptadas por cada um deles. As interseções são computadas entre os segmentos de cada região, normalmente empregando um algoritmo de força bruta. A idéia é a aplicação de heurísticas que realizem filtros, diminuindo o número de segmentos a serem testados.

Os algoritmos agrupados na segunda classe são os baseados em estratégias de Geometria Computacional, onde a preocupação é com o desenvolvimento de algoritmos onde a análise de complexidade de pior caso possua uma boa complexidade. O algoritmo do *plane sweep* pode ser classificado nesta categoria.

Os testes realizados mostram que embora os algoritmos da primeira classe não garantam uma eficiência no pior caso como os da Geometria Computacional, eles acabam tirando proveito da característica dos dados de um SIG: segmentos curtos, espaçados, com poucas interseções por segmento e uniformemente distribuídos no plano. Dessa forma, eles acabam sendo mais eficientes (velozes) do que os da segunda classe.

Outro trabalho que realiza testes semelhantes é apresentado por Pullar (1990), onde é mostrado que uma técnica baseada no *Fixed Grid*, também pertencente à categoria dos algoritmos por partição, é bastante competitiva em relação aos algoritmos baseados no *plane sweep*, apesar da complexidade de pior caso ser bem maior, $O(n^2)$.

Nos trabalhos de Akman et al (1989), Franklin et al (1988) e Franklin et al (1989) é apresentado um algoritmo baseado no *fixed grid*. Dados dois conjuntos de segmentos, um vermelho e outro azul, os segmentos da primeira linha são cobertos por uma grade retangular fixa (*fixed grid*). Cada segmento vermelho é associado às células da grade por onde ele passa. Os pontos de interseção são determinados procurando para cada segmento azul a lista de células por onde ele passa e então utilizando um algoritmo de força bruta esses pontos são determinados.

Um dos pontos chaves desse algoritmo é a determinação da resolução da grade. Ela pode ser determinada a partir da média do comprimento dos segmentos. Franklin et al (1988) mostram que utilizando-se de parâmetros estatísticos, como a média do comprimento dos segmentos, a grade se adapta bem aos dados de entrada.

2.7 União, interseção e diferença de polígonos

Operações sobre polígonos são de fundamental importância em SIG. Através da detecção e processamento da união, interseção e diferença de polígonos, diversos tipos de operações, conhecidas como em conjunto como *polygon overlay*, são viabilizadas. São operações fundamentais para análise espacial, usadas em situações em que é necessário combinar ou comparar dados colocados em camadas distintas. Por exemplo, considere-se uma consulta como "identificar fazendas em que mais de 30% da área é de latossolo roxo". Para executar esta análise, é necessário combinar uma camada de objetos poligonais (os limites de propriedades rurais) com outra (o mapa de tipos de solo), para obter uma nova camada, de cujo conteúdo podem ser selecionados diretamente os objetos que atendem ao critério de análise colocado.

Algumas vezes, o *polygon overlay* é definido como uma operação topológica, ou seja, que é executada sobre dados organizados em uma estrutura de dados topológica. As funções de processamento de polígonos

que serão descritas a seguir são utilizadas em sistemas não topológicos, ou em situações em que o processamento é feito de maneira isolada, como na criação e uso de *buffers* (vide Seção 2.8).

Para realizar operações sobre polígonos, é interessante aplicar um passo preliminar de detecção rápida da possibilidade interseção entre os polígonos. Assim, se não for possível que dois polígonos P e Q tenham interseção, então podemos concluir diretamente que $P \cup Q = \{P, Q\}$, $P \cap Q = \varnothing$, $P - Q = P$ e $Q - P = Q$. Uma maneira simples de testar se dois polígonos têm ou não interseção é usar inicialmente o teste de interseção dos retângulos envolventes mínimos (Seção 2.3.3).

No caso geral, operações de união, interseção ou diferença entre dois polígonos simples podem gerar diversos polígonos como resultado. Mais ainda, os polígonos resultantes poderão conter buracos. A Figura 2.16 contém exemplos de produção de múltiplos polígonos e de polígonos com buracos em operações de interseção, união e diferença.

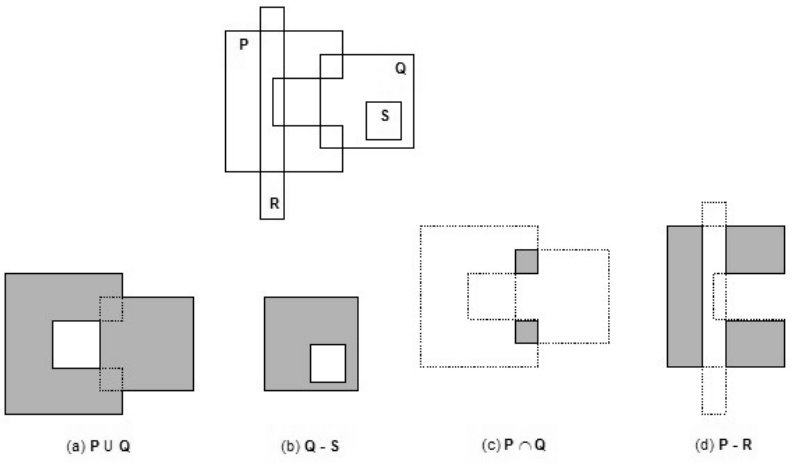


Figura 2.16 – Operações sobre polígonos produzindo buracos e múltiplos polígonos.

Apresentaremos aqui um método proposto por Margalit e Knott (1989). Esse algoritmo é sensível à orientação dos polígonos, e exige que os vértices de ilhas sejam codificados em um sentido (por exemplo, anti-

horário) e os vértices de buracos sejam dispostos no sentido inverso (horário). Isto coincide com a convenção usada para calcular a área de polígonos, conforme apresentado na Seção 2.3.5.

Tabela 2.1 – Orientação dos polígonos de acordo com a operação

Polígonos		**Operações**			
P	***Q***	$P \cap Q$	$P \cup Q$	$P - Q$	$Q - P$
ilha	**ilha**	*manter*	*manter*	*inverter*	*inverter*
ilha	**buraco**	*inverter*	*inverter*	*manter*	*manter*
buraco	**ilha**	*inverter*	*inverter*	*manter*	*manter*
buraco	**buraco**	*manter*	*manter*	*inverter*	*inverter*

O algoritmo tem seis passos, que serão descritos a seguir.

1. *Normalizar a orientação* dos polígonos de entrada *P* e *Q*, e inverter a orientação de *Q* dependendo do tipo de operação e da natureza (ilha ou buraco) dos dois polígonos de entrada, de acordo com a Tabela 2.1.
2. *Classificar os vértices,* verificando se cada um está *dentro*, *fora* ou *na fronteira* do outro polígono, usando o teste de ponto em polígono (Seção 2.4). Inserir os vértices assim classificados em duas listas circulares, *PL* e *QL*, onde aparecerão em seqüência, de modo a definir as arestas por adjacência.
3. *Encontrar as interseções* entre arestas dos dois polígonos, usando o teste de interseção de *n* segmentos (Seção 2.6). Inserir os pontos de interseção na posição apropriada em *PL* e *QL*, classificando-os como *na fronteira*. A partir deste ponto, teremos um conjunto de *fragmentos de arestas* em lugar das arestas originais. É necessário cuidar do caso especial de interseção ao longo de uma aresta comum, ou parte dela. Neste caso, ambos os pontos extremos da aresta devem ser classificados como *na fronteira* e inseridos nas listas.

4. *Classificar os* fragmentos de arestas (definidos pelos pares de vértices) formados em PL e QL com relação ao outro polígono, entre interior, exterior ou na fronteira. Não é necessário realizar novamente o teste de ponto em polígono. Uma aresta pode ser considerada interior ao outro polígono caso pelo menos um de seus vértices esteja classificado como dentro. Da mesma forma, uma aresta pode ser classificada como exterior ao outro polígono caso pelo menos um de seus vértices esteja classificado como fora. Se ambos os vértices estiverem classificados como na fronteira, então é necessário verificar a situação de um ponto interno ao segmento (por exemplo, seu ponto médio). Se este ponto estiver fora do outro polígono, então a aresta é classificada como exterior. Se o ponto estiver dentro do outro polígono, então a aresta é classificada como interior. Se o ponto estiver na fronteira, a aresta é classificada como fronteira.
5. Arestas na fronteira constituem um caso degenerado, que requer tratamento especial. Se existe um fragmento de aresta na fronteira de *P*, então necessariamente existe também um na fronteira de *Q*. Estes fragmentos podem estar orientados na mesma direção ou em direções opostas. A implementação pode decidir o que fazer nestes casos, ou seja, se interseções com dimensão de segmento ou de ponto serão ou não retornadas. Se as interseções como segmento forem retornadas, serão formadas por um ciclo de duas arestas sobrepostas, cada uma em uma direção. Interseção em um ponto será retornada como um ciclo de duas arestas, cada uma em uma direção, ligando dois vértices sobrepostos. Desta forma preserva-se a topologia do resultado (sempre cadeia fechada de segmentos), mas em SIG é mais interessante detectar estes casos e retornar objetos da dimensão adequada (no caso, ponto)[5].
6. *Selecionar e organizar as arestas* para formar os polígonos de resultado. Este processo de seleção é baseado na combinação das

[5] Para uma análise mais completa, inclusive com as combinações de hipóteses nos casos de ilhas e buracos, vide (Margalit e Knott, 1989).

duas listas em uma, denominada *RL*, usando apenas as arestas que interessam para a operação, conforme definido na Tabela 2.2.

7. *Construir os polígonos de resultado,* selecionando uma aresta e, com base em seu ponto final, procurar em *RL* sua continuação, até fechar o polígono. Repetir o processo, eliminando de *RL* a cada passo as arestas utilizadas, até que *RL* fique vazia.

Os polígonos resultantes manterão a orientação adotada para ilhas e buracos.

Tabela 2.2 – Tipos de arestas para seleção de acordo com o tipo de operação e os tipos de polígonos de entrada

Polígonos		**Operações**							
		$P \cap Q$		$P \cup Q$		$P - Q$		$Q - P$	
P	***Q***	*P*	*Q*	*P*	*Q*	*P*	*Q*	*P*	*Q*
ilha	**buraco**	interior	interior	exterior	exterior	exterior	interior	interior	exterior
ilha	**buraco**	exterior	interior	interior	exterior	interior	interior	exterior	exterior
buraco	**ilha**	interior	exterior	exterior	interior	exterior	exterior	interior	interior
buraco	**buraco**	exterior	exterior	interior	interior	interior	exterior	exterior	interior

2.8 Mapas de distância (buffer zones)

Outra operação importante para um GIS é a construção de mapas de distância ou *buffer zones*, que são áreas construídas ao redor de objetos mantendo uma certa distância. A Figura 2.17 ilustra a idéia dessas operações para pontos, linhas e polígonos respectivamente.

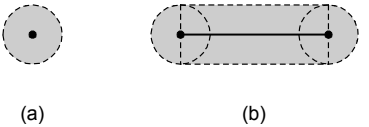


Figura 2.17 – *Buffers* elementares ao redor de ponto (a) e segmento (b).

A determinação do *buffer* ao redor de um ponto é feita de forma direta, como uma circunferência de raio *d* (Figura 2.17a). O *buffer* ao redor de uma linha é formada pela união de *buffers* elementares (Figura 2.17b) definidos para cada segmento da linha. Esses *buffers* elementares são formados a partir de semicircunferências traçadas nas extremidades dos segmentos (uma em cada extremidade). Utilizando o algoritmo de união (Seção 2.7) podemos combinar esses *buffers* até formar o resultado final da linha (Figura 2.18a).

O *buffer* de polígonos (Figura 2.18b) é semelhante ao de linha, com a diferença de que é possível gerar *buffers* negativos (voltados para o interior do polígono).

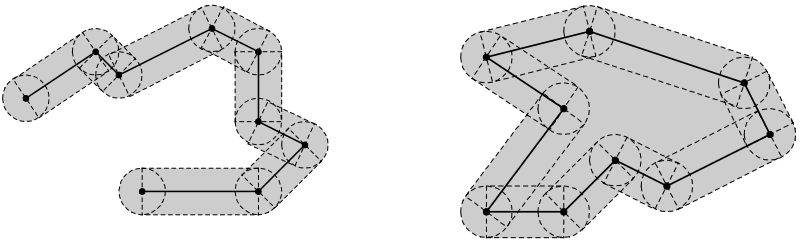


Figura 2.18 – *Buffer* ao redor de linha (a) e polígono (b).

2.9 Relacionamentos topológicos

A grande importância da caracterização dos relacionamentos topológicos entre estruturas vetoriais é poder atribuir um contexto semântico aos algoritmos geométricos. Para especificá-los, inicialmente definiremos as geometrias vetoriais como elementos do $\Re^2$, considerado como espaço topológico. Assim, um *ponto* é simplesmente um elemento de $\Re^2$. Uma *linha* L é um conjunto de pontos conectados. Uma *ilha* ou *linha circular* é uma linha em que o ponto inicial é igual ao ponto final. A fronteira de L, denotada por δL, é o conjunto dos pontos inicial e final, caso L não seja uma ilha, ou o conjunto vazio, em caso contrário. O interior de L, denotado por L^o, é composto pelos demais pontos. Uma *região* A é um conjunto de pontos com um interior conectado, denotado por A^o, uma fronteira conectada, denotada por δA, e um único exterior conectado, denotado por A^-. Assim, as regiões consideradas não tem "buracos".

Os relacionamentos topológicos podem ser definidos com base em um modelo, chamado *matriz de 4-interseções* (ver a Figura 2.19), que considera oito relações topológicas binárias, representando a interseção entre a fronteira e o interior de duas geometrias (Egenhofer e Franzosa, 1995).

Para definir relacionamentos topológicos entre geometrias com estruturas mais complexas, como regiões com ilhas e separações, é necessário estender a matriz de 4-Interseções para também considerar o exterior de uma geometria (Egenhofer e Herring, 1991). O novo modelo, chamado de *matriz de 9-Interseções* (ver Figura 2.20), considera então o resultado da interseção entre as fronteiras, interiores e exteriores de duas geometrias. Maiores detalhes sobre relações topológicas entre regiões com ilhas podem ser encontrado em (Egenhofer et al., 1994).

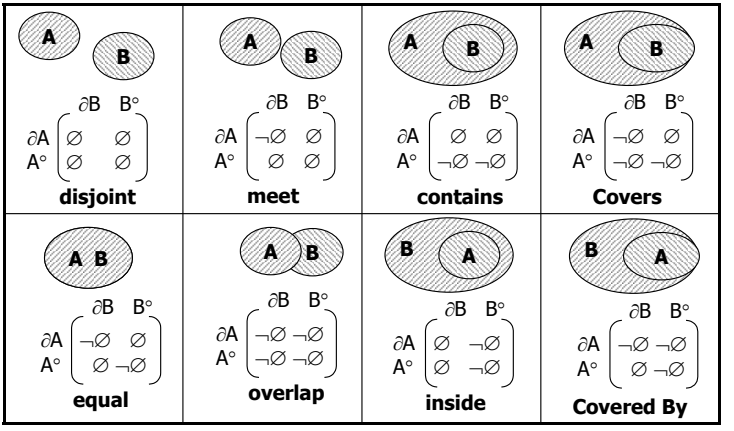


Figura 2.19 – Matriz de 4-Interseções para relações entre duas regiões.
Fonte: (Egenhofer et al., 1994).

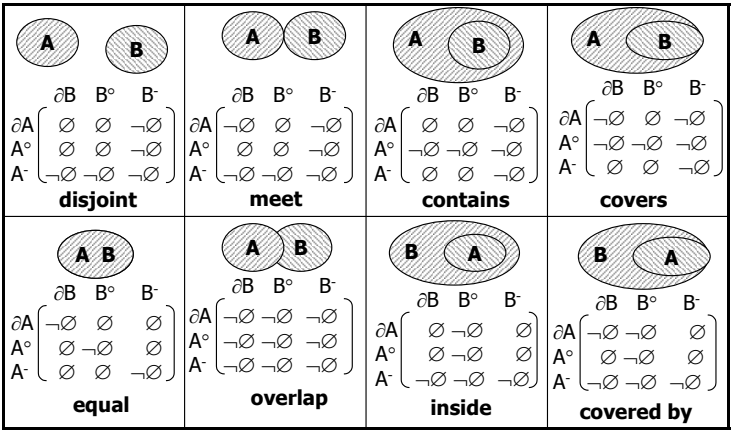


Figura 2.20 – Matriz de 9-Interseções para relações entre duas regiões. Fonte: (Egenhofer e Herring, 1991).

Nos modelos citados acima, os resultados das intersecções são avaliados considerando os valores vazio ou não-vazio. Há várias situações em que é necessário considerar as dimensões das interseções não vazias. Por exemplo, certo estado X só considera um outro estado Y como vizinho se eles têm pelo menos uma aresta em comum. Neste caso, para encontrar os vizinhos do estado X, não basta saber quais estados "tocam" ou são "adjacentes" a ele, mas sim se o resultado da interseção entre eles é uma aresta.

Para acomodar estas situações, novos modelos foram definidos, levando em consideração as dimensões dos resultados das interseções não vazias, como o *modelo para relações topológicas binárias detalhadas* (Egenhofer, 1993), baseado na matriz de 4-interseções, e a matriz de 9-Interseções estendida dimensionalmente (DE-9IM), baseada na matriz de 9-interseções (Paiva, 1998).

Clementini et al (1993) estenderam a abordagem da matriz de 4-interseções de forma a incluir a informação da dimensão da interseção. No espaço bidimensional, a dimensão da interseção pode ser vazia, um ponto, uma linha ou uma região. Este modelo contempla assim um conjunto de 52 relacionamentos topológicos, o que não é conveniente do ponto de vista do usuário. Para equacionar este problema, os relacionamentos topológicos foram agrupados em cinco mais gerais -

touch, *in*, *cross*, *overlap*, *disjoint* - que são sobrecarregados, ou seja, que podem ser usados indistintamente para ponto, linha e região. Estes relacionamentos são definidos da seguinte forma:

touch: aplica-se a pares de geometrias dos tipos região/região, linha/linha, linha/região, ponto/região e ponto/linha:

$$\langle \lambda_1, touch, \lambda_2 \rangle \Leftrightarrow (\lambda_1^o \cap \lambda_2^o = \emptyset) \wedge$$

$$((\partial\lambda_1 \cap \lambda_2^o \neq \emptyset) \vee (\lambda_1^o \cap \partial\lambda_2 \neq \emptyset) \vee (\partial\lambda_1 \cap \partial\lambda_2 \neq \emptyset))$$

in: aplica-se a pares de geometrias com qualquer combinação de tipos:

$$\langle \lambda_1, in, \lambda_2 \rangle \Leftrightarrow (\lambda_1^o \cap \lambda_2^o \neq \emptyset) \wedge (\lambda_1^o \cap \lambda_2^- = \emptyset) \wedge (\partial\lambda_1 \cap \lambda_2^- = \emptyset)$$

cross: aplica-se a pares de geometrias dos tipos linha/linha e linha/região. No caso de linha/região, temos:

$$\langle L, cross, R \rangle \Leftrightarrow (L^o \cap R^o \neq \emptyset) \wedge (L^o \cap R^- \neq \emptyset)$$

No caso de linha/linha, temos:

$$\langle L_1, cross, L_2 \rangle \Leftrightarrow \dim(L_1^o \cap L_2^o) = 0$$

overlap: aplica-se a pares de geometrias dos tipos região/região e linha/linha. No caso de região/região, temos:

$$\langle A_1, overlap, A_2 \rangle \Leftrightarrow (A_1^o \cap A_2^o \neq \emptyset) \wedge (A_1^o \cap A_2^- \neq \emptyset)$$

$$\wedge (A_1^- \cap A_2^o \neq \emptyset)$$

No caso de linha/linha, temos:

$$\langle L_1, overlap, L_2 \rangle \Leftrightarrow (\dim(L_1^o \cap L_2^o) = 1) \wedge (L_1^o \cap L_2^- \neq \emptyset)$$

$$\wedge (L_1^- \cap L_2^o \neq \emptyset)$$

disjoint: aplica-se a pares de geometrias com qualquer combinação de tipos:

$$\langle \lambda_1, disjoint, \lambda_2 \rangle \Leftrightarrow (\lambda_1^o \cap \lambda_2^o = \emptyset) \wedge (\partial\lambda_1 \cap \lambda_2^o = \emptyset) \wedge$$

$$(\lambda_1^o \cap \partial\lambda_2 = \emptyset) \wedge (\partial\lambda_1 \cap \partial\lambda_2 = \emptyset)$$

2.10 Determinação do relacionamento topológico

Nesta seção apresentaremos um algoritmo simples para a determinação dos relacionamentos topológicos entre dois polígonos (*A* e *B*), segundo a matriz de 9-Interseções (descrita na seção anterior). Ele utiliza uma combinação dos algoritmos geométricos apresentados anteriormente para determinar as interseções entre interior, fronteira e exterior dos dois polígonos. O algoritmo possui seis etapas, que estão descritas a seguir e são ilustradas na Figura 2.21.

1. Avaliar o relacionamento entre os REM dos polígonos A e B. Nessa avaliação podemos empregar as estratégias apresentadas por Clementini et al (1994) que consiste basicamente no estabelecimento de um mapeamento entre os relacionamentos topológicos dos REMs e das geometrias exatas. No caso, por exemplo, de dois polígonos *A* e *B* que estejam sendo testados para ver se *A* contém *B*, podemos rapidamente descartar essa possibilidade caso o REM de *A* não contenha o REM de *B*. Caso contrário vamos à próxima etapa;
2. Determinar os pontos de interseção entre os dois polígonos. Isso pode ser feito utilizando algum dos algoritmos apresentados na Seção 2.6. Esta etapa nos informa se há ou não interseção entre as fronteiras dos objetos.
3. Se não houve interseção na etapa anterior (etapa 2), então devemos testar qualquer ponto do polígono A, num teste de ponto em polígono (Seção 2.4), com o polígono B, para determinar a localização de A em relação a B. Este teste precisa ser feito também com um ponto de B em relação a A:
    a. Se o ponto do polígono A estiver dentro de B então A encontra-se dentro de B (relacionamento *inside*).
    b. Caso contrário, se o ponto de B estiver dentro de A então A contém B (relacionamento *contains*).
    c. Caso contrário, os polígonos são disjuntos (relacionamento *disjoint*).
4. Se houve interseção na etapa 2, devemos realizar a fragmentação da fronteira de A, em relação aos pontos de interseção.

5. Depois, verificamos a localização de cada um dos fragmentos em relação ao polígono B. Podemos utilizar o teste de ponto em polígono, tomando um ponto do fragmento que não esteja na extremidade.
6. Com base na localização dos fragmentos, as interseções entre fronteiras, interiores e exteriores podem ser inferidas:
    a. Se houve fragmentos dentro e fora do polígono B então os dois polígonos se sobrepõe (relacionamento *overlaps*);
    b. Se houve fragmentos somente dentro e na fronteira do polígono B, então o polígono A é coberto pelo polígono B (*covered by*).
    c. Se houve fragmentos somente fora e na fronteira do polígono B, temos que decidir se os polígonos se tocam ou se A cobre B. Isso pode ser feito fragmentando a fronteira do polígono B, como na etapa 4 e testando a localização dos fragmentos de B em relação a A (etapa 5). Se houver fragmentos dentro de A, então A cobre B (relacionamento *covers*) senão A toca B (relacionamento *touches*).
    d. Se todos os fragmentos encontram-se na fronteira de B, então os polígonos são iguais (relacionamento *equals*).

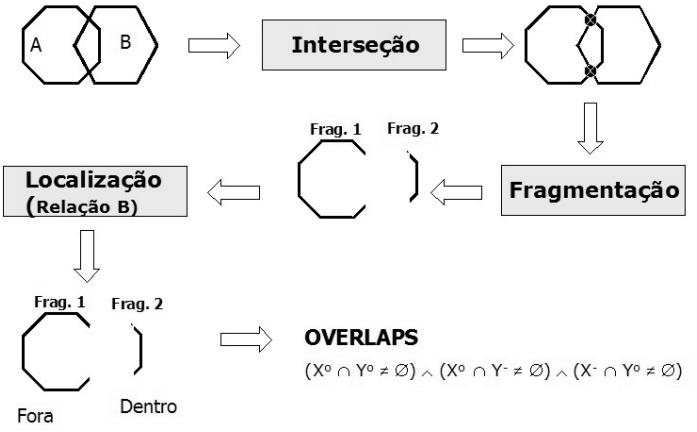


Figura 2.21 – Determinação do relacionamento topológico.

Referências

AKMAN, V.; FRANKLIN, W. R.; KANKANHALLI, M.; NARAYANASWAMI, C. Geometric computing and uniform grid technique. **Computer-Aided Design**, v. 21, n. 7, p. 410-420, set. 1989.

ANDREWS, D. S.; SNOEYINK, J. Geometry in GIS is not combinatorial: segment intersection for polygon overlay. In: Annual Symposium on Computational Geometry, 11., 1995, Vancouver. **Proceedings.** Canada: British Columbia, 1995, p. 424-425.

ANDREWS, D. S.; SNOEYINK, J.; BORITZ, J. CHAN, T.; DENHAM, G.; HARRISON, J.; ZHU, C. Further comparison of algorithms for geometric intersection problems. In: International Symposium on Spatial Data Handling, 6., 1994, Edinburgh. **Proceedings.** Edinburgh: Taylor e Francis, 1994, p. 709-724.

BEARD, K.; Theory of the cartographic line revisited: implications for automated generalization. In: **Cartographica**. 1991. v. 28, cap. 4, p. 32-58.

BENTLEY, J. L.; OTTMANN, T. A. Algorithms for reporting and counting geometric intersections. **IEEE Transactions on Computers**, v. C-28, n. 9, p. 643-647, set. 1979.

CLEMENTINI, E.; DI FELICE, P.; VAN OOSTEROM, P., 1993. A Small Set of Formal Topological Relationships Suitable for End-User Interaction. In: ABEL, D.; OOI, B. C., eds., **SSD '93: Lecture Notes in Computer Science, v. 692**: New York, NY, Springer-Verlag, p. 277-295.

CLEMENTINI, E.; SHARMA, J.; EGENHOFER, M. Modelling topological spatial relations: strategies for query processing. **Computers & Graphics**, v. 18, n. 6, p. 815-822, 1994.

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L. **Introduction to algorithms**. E.U.A.: McGraw-Hill, 1990.

DAVIS Jr., C.; Uso de vetores em GIS. **Fator GIS**, v. 21, n. 4, p. 22-23, 1997.

DOUGLAS, D. H.; PEUCKER, T. K. Algorithms for the reduction of the number of points required to represent a line or its caricature. **The Canadian Cartographer**, v. 10, n. 2, p. 112-122, 1973.

EGENHOFER, M. A Model for Detailed Binary Topological Relationships. **Geomatica**, v. 47, p. 261-273, 1993.

EGENHOFER, M.; P. DI FELICE; CLEMENTINI, E. Topological Relations between Regions with Holes. **International Journal of Geographical Information Systems**, v. 8, n.2, p. 129-144, 1994.

EGENHOFER, M.; FRANZOSA, R. On the Equivalence of Topological Relations. **International Journal of Geographical Information Systems**, v. 9, n.2, p. 133-152, 1995.

EGENHOFER, M.; HERRING, J. Categorizing Binary Topological Relationships Between Regions, Lines, and Points in Geographic Databases. Orono, ME: Department of Surveying Engineering, University of Maine, 1991.

FIGUEIREDO, L. H.; CARVALHO, P. C. P. **Introdução à geometria computacional**. Rio de Janeiro: IMPA, 1991.

FRANKLIN, W. R.; CHANDRASEKHAR, N.; Kankanhalli, M.; Seshan, M.; Akman, V. Efficiency of uniform grids for intersection detection on serial and parallel machines. In: Computer Graphics International, maio, 1988, Geneva, Switzerland. **Proceedings.** Berlin Heidelberg: Springer-Verlag, 1988, p. 51-62.

FRANKLIN, W. R.; CHANDRASEKHAR, N.; KANKANHALLI, M.; SUN, D.; ZHOU, M.; WU, P. YF Uniform grids: a technique for intersection detection on serial and parallel machines. In: Auto Carto 9, Baltimore, Mariland. **Proceedings.** Baltimore, Maryland: American Congress on Surveying and Mapping, 1989, p. 100-109.

HAINES, E. Point in polygon strategies. In: HECKBERT, P. S. (Org.). **Graphics gems IV**. Boston, E.U.A.: Academic Press, 1994. p. 24-46.

HUANG, C.; SHIH, T. On the complexity of point-in-polygon algorithms. **Computers & Geosciences**. v. 23, n. 1, p. 109-118, 1997.

KNUTH, D. E. **The art of computer programming, vol. 1: fundamental algorithms**. Boston, Massachusetts: Addison-Wesley, 1973.

LAURINI, R.; THOMPSON, D. **Fundamentals of spatial information systems**. : Academic Press, 1992.

LI, Z.; OPENSHAW, S. Algorithms for automated line generalization based on a natural principle of objective generalization. **International Journal of Geographic Information Systems**, v. 6, n. 5, p. 373-389, 1992.

MARGALIT, A.; KNOTT, G. D. An algorithm for computing the union, intersection or difference of two polygons. **Computers & Graphics**, v. 13, n. 2, p. 167-183, 1989.

MARINO, J. S.; Identification of characteristic points along naturally occurring lines: an empirical study. **The Canadian Cartographer**, v. 16, n. , p. 70-80, 1979.

MCMASTER, R. B.; SHEA, K. S. Generalization in digital cartography. **Association of American Geographers**, 1992.

O'ROURKE, J.. **Computacional geometry in C**. Cambridge: Cambridge University Press, 1998.

PAIVA, J. A. C. Topological Equivalence and Similarity in Multi-Representation Geographic Database. University of Maine,1998.

PEUCKER, T. K.; A theory of the cartographic line. In: **International yearbook of cartography**. 1975. cap. 16, p. 134-143.

PREPARATA , F. P.; SHAMOS , M. I. **Computational geometry an introduction**. New York: Springer-Verlag, 1985.

PULLAR, D. Comparative study of algorithms for reporting geometrical intersections. In: International Symposium on Spatial Data Handling, 4., 1990, Zurich. Proceedings… Edinburgh: Taylor and Francis, 1990, p. 66-76.

SAALFELD, A. It doesn't make me nearly as CROSS - some advantages of the point-vector representation of line segments in automated cartography. **International Journal of Geographical Information Systems**, v. 1, n. 4, p. 379-386, 1987.

SCHNEIDER, M. **Spatial data types for database systems**. Berlin Hidelberg: Springer-Verlag, 1997.

SHAMOS, M. I.; HOEY, D. Geometric intersection problems. In: Annual IEEE Symposium on Foundations of Computer Science, 17., oct. 1976, Houston, Texas. **Proceedings.** New York: IEEE, 1976, p. 208-215.

TAYLOR, G. E.; Point in Polygon Test. **Survey Review**, v. 32, n. 254, p. 479-484, 1994.

VAN OOSTEROM, P.; **Reactive data structures for geographic information systems**. : Oxford University Press, 1993.

VAN OOSTEROM, P.; SCHENKELAARS, V. The development of an interactive multi-scale GIS. **International Journal of Geographical Information Systems**, v. 9, n. 5, p. 489-507, 1995.

WEIBEL, R.; Map generalization in the context of digital systems. **Cartography and Geographic Information Systems**, v. 22, n. 4, p. 3-10, 1995.

3 *Modelagem conceitual de dados geográficos*

Karla A. V. Borges
Clodoveu A. Davis Jr.
Alberto H. F. Laender

3.1 Introdução

Este capítulo apresenta recursos para a modelagem de dados geográficos, apoiados principalmente no modelo OMT-G. Inicialmente, resume um pouco do histórico dos modelos de dados geográficos e discute os níveis de abstração usuais para aplicações geográficas. Em seguida, descreve o modelo OMT-G, apresenta classes de restrições de integridade espaciais, e introduz um algoritmo de mapeamento de esquemas OMT-G para esquemas físicos, considerando o padrão OpenGIS para representação de objetos. Por fim, apresenta um exemplo de modelagem e tece algumas considerações finais.

Um modelo de dados é um conjunto de conceitos que podem ser usados para descrever a estrutura e as operações em um banco de dados (Elmasri e Navathe, 2004). O modelo busca sistematizar o entendimento que é desenvolvido a respeito de objetos e fenômenos que serão representados em um sistema informatizado. Os objetos e fenômenos reais, no entanto, são complexos demais para permitir uma representação completa, considerando os recursos à disposição dos sistemas gerenciadores de bancos de dados (SGBD) atuais. Desta forma, é necessário construir uma abstração dos objetos e fenômenos do mundo real, de modo a obter uma forma de representação conveniente, embora simplificada, que seja adequada às finalidades das aplicações do banco de dados.

A *abstração* de conceitos e entidades existentes no mundo real é uma parte importante da criação de sistemas de informação. O sucesso de

qualquer implementação em computador de um sistema de informação é dependente da qualidade da transposição de entidades do mundo real e suas interações para um banco de dados informatizado. A abstração funciona como uma ferramenta que nos ajuda a compreender o sistema, dividindo-o em componentes separados. Cada um desses componentes pode ser visualizado em diferentes níveis de complexidade e detalhe, de acordo com a necessidade de compreensão e representação das diversas entidades de interesse do sistema de informação e suas interações.

Os primeiros modelos de dados para as aplicações geográficas eram voltados para as estruturas internas dos SIG. O usuário era forçado a adequar os fenômenos espaciais às estruturas disponíveis no SIG a ser utilizado. Conseqüentemente, o processo de modelagem não oferecia mecanismos para a representação da realidade de forma mais próxima ao modelo mental do usuário. Ficava evidente que a modelagem de aplicações geográficas necessitava de modelos mais adequados, capazes de capturar a semântica dos dados geográficos, oferecendo mecanismos de abstração mais elevados e independência de implementação. Apesar de toda a expressividade oferecida pelas técnicas tradicionais de modelagem, dificuldades surgem devido ao fato de que os dados geográficos possuem aspectos peculiares, particularmente com respeito à codificação da localização espacial e do tempo de observação, bem como em relação ao registro de fatores externos, como sua precisão de obtenção. A modelagem do mundo real é uma atividade complexa porque envolve a *discretização* do espaço como parte do processo de abstração, visando obter representações adequadas aos fenômenos geográficos.

Os fatores envolvidos nesse processo de discretização do espaço são inúmeros. Entre eles citamos:

- **Transcrição da informação geográfica em unidades lógicas de dados** – Para Frank e Goodchild (1990), o esquema de uma aplicação geográfica é uma representação limitada da realidade, tendo em vista a natureza finita e discreta da representação nos computadores. Por maior que seja o nível de abstração utilizado, a realidade é modelada através de conceitos geométricos (Frank, 1992) e, para que esses conceitos sejam implementados em computadores, eles precisam ser formalizados, sendo necessário um maior número de conceitos

abstratos para descrever os dados, e um maior número de operações apropriadas, que podem ser definidas de modo independente da implementação (Mark e Frank, 1990).

- **Forma como as pessoas percebem o espaço –** O aspecto cognitivo na percepção espacial é um dos aspectos que faz com que a modelagem de dados geográficos seja diferente da modelagem tradicional. Dependendo do observador, de sua experiência e de sua necessidade específica, uma mesma entidade geográfica pode ser percebida de diversas formas. Uma escola, por exemplo, poderá ser representada usando um ponto (posicionado de forma aproximada), como uma área (do terreno que ocupa), ou como um conjunto de edificações, dependendo do observador e do que ele pretende obter com essa representação. A escala de representação exige que a mesma entidade geográfica possa ser representada por diferentes formas geométricas, com detalhamento variável. O uso de múltiplas representações para a mesma entidade pode ocorrer simultaneamente, usando-se várias formas geométricas para uma mesma entidade geográfica, ou poderá ser exclusiva, fazendo com que uma representação seja válida para visualização em determinadas circunstâncias, como, por exemplo, uma determinada faixa de escalas.

- **Natureza diversificada dos dados geográficos –** Além de geometria, localização no espaço, informações associadas e características temporais, os dados geográficos ainda podem prover de origens distintas. Dados ambientais, por exemplo, são derivados de dados como topografia, clima e tempo, propriedades do solo, propriedades geológicas, cobertura da terra, uso da terra, e hidrografia. Alguns desses fenômenos, como elevação e propriedades do solo, variam continuamente sobre o espaço (visão de *campos*). Outros, como montanhas e bacias hidrográficas, podem ser individualizados (visão de *objetos*). Alguns podem estar em ambas as categorias, dependendo do nível de detalhe considerado.

- **Existência de relações espaciais (topológicas, métricas, de ordem e *fuzzy*) –** Essas relações são abstrações que nos ajudam a compreender como no mundo real os objetos se relacionam uns com os outros (Mark e Frank, 1990).

3.2 Modelos de dados geográficos

Modelos de dados semânticos e orientados a objetos, tais como ER (Chen, 1976), OMT (Rumbaugh et al., 1991), IFO (Abiteboul e Hull, 1987), UML (Rational Software Corporation, 1997) e outros, têm sido largamente utilizados para a modelagem de aplicações geográficas. Apesar da grande expressividade desses modelos, eles apresentam limitações para a adequada modelagem de aplicações geográficas, já que não possuem primitivas apropriadas para a representação de dados espaciais.

Modelos de dados para aplicações geográficas têm necessidades adicionais, tanto com relação à abstração de conceitos e entidades, quanto ao tipo de entidades representáveis e seu inter-relacionamento. Diversas propostas existem atualmente, principalmente focalizadas em estender os modelos criados para aplicações convencionais, como GeoOOA (Kösters et al., 1997), MODUL-R (Bédard et al., 1996), GMOD (Oliveira et al., 1997), IFO para aplicações geográficas (Worboys et al., 1990), GISER (Shekhar et al., 1997), OMT-G (Borges et al., 2001), GeoFrame (Lisboa Filho, 1997), MADS (Parent et al., 1999). Todos esses modelos procuram refletir melhor as necessidades de aplicações geográficas. A escolha de um deles pode ser feita observando as necessidades de modelagem quanto à abstração de conceitos geográficos, ao atendimento de requisitos usuais para modelos de dados (como clareza e facilidade de uso) (Borges et al., 2001), e à possibilidade de mapeamento dos esquemas produzidos para a implementação em SGBD espaciais, o que inclui a necessária identificação de restrições de integridade espaciais (Borges et al., 2002) (Davis Jr. et al., 2005).

3.3 Níveis de abstração de dados geográficos

Modelos de dados são classificados de acordo com o nível de abstração empregado. Para aplicações geográficas, são considerados quatro níveis distintos de abstração:

Nível do mundo real – Contém os fenômenos geográficos reais a representar, como rios, ruas e cobertura vegetal.

Nível de representação conceitual – Oferece um conjunto de conceitos formais com os quais as entidades geográficas podem ser modeladas da forma como são percebidas pelo usuário, em um alto nível de abstração. Neste nível são definidas as classes básicas, contínuas ou discretas, que serão criadas no banco de dados. Essas classes estão associadas a classes de representação espacial, que variam de acordo com o grau de percepção que o usuário tem sobre o assunto. Essa preocupação não aparece com freqüência nas metodologias tradicionais de modelagem de dados, uma vez que as aplicações convencionais raramente precisam lidar com aspectos relativos à representação espacial (única ou múltipla) de objetos.

Nível de apresentação – Oferece ferramentas com as quais se pode especificar os diferentes aspectos visuais que as entidades geográficas têm de assumir ao longo de seu uso em aplicações.

Nível de implementação – Define padrões, formas de armazenamento e estruturas de dados para implementar cada tipo de representação, os relacionamentos entre elas e as necessárias funções e métodos.

O modelo OMT-G, descrito a seguir, atua nos níveis de representação conceitual e apresentação. No nível de implementação, situam-se as linguagens de definição de dados associadas a SGBD espaciais. Apresentaremos mais adiante um algoritmo de mapeamento entre esquemas OMT-G e estruturas físicas, definidas pelo padrão OpenGIS, conforme implementadas no SGBD Oracle Spatial.

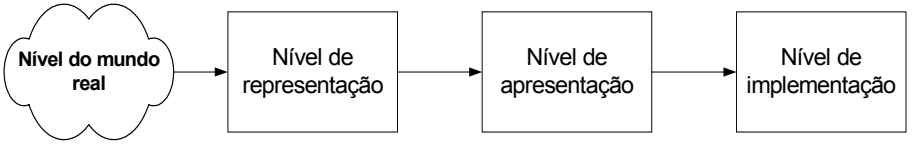


Figura 3.1 – Níveis de abstração de aplicações geográficas. Fonte: adaptado de (Borges et al., 2001).

3.4 Modelo de dados OMT-G

3.4.1 Visão geral do modelo

O modelo OMT-G parte das primitivas definidas para o diagrama de classes da *Unified Modeling Language* (UML) (Rational Software Corporation, 1997), introduzindo primitivas geográficas com o objetivo de aumentar a capacidade de representação semântica daquele modelo e, portanto reduzindo a distância entre o modelo mental do espaço a ser modelado e o modelo de representação usual. Portanto, o modelo OMT-G provê primitivas para modelar a geometria e a topologia dos dados geográficos, oferecendo suporte a estruturas topológicas "todo-parte", estruturas de rede, múltiplas representações de objetos e relacionamentos espaciais. Além disso, o modelo permite a especificação de atributos alfanuméricos e métodos associados para cada classe. Os principais pontos do modelo são sua expressividade gráfica e sua capacidade de codificação, uma vez que anotações textuais são substituídas pelo desenho de relacionamentos explícitos, que denotam a dinâmica da interação entre os diversos objetos espaciais e não espaciais.

O modelo OMT-G é baseado em três conceitos principais: *classes*, *relacionamentos* e *restrições de integridade espaciais*. Classes e relacionamentos definem as primitivas básicas usadas para criar esquemas estáticos de aplicação. OMT-G propõe o uso de três diferentes diagramas no processo de desenvolvimento de uma aplicação geográfica. O primeiro e mais usual é o *diagrama de classes*, no qual todas as classes são especificadas junto com suas representações e relacionamentos. A partir do diagrama de classes é possível derivar um conjunto de restrições de integridade espaciais, que deve ser observado na implementação. Quando o diagrama de classes especifica múltiplas representações ou a derivação de uma classe a partir de outra, é necessário desenvolver um *diagrama de transformação*. Nele todo o processo de transformação pode ser especificado, permitindo a identificação dos métodos necessários para a implementação. Finalmente, para especificar as alternativas de visualização que cada representação pode assumir, é necessário desenvolver um *diagrama de apresentação*. As primitivas para cada um desses diagramas são detalhadas nas próximas seções.

A identificação de restrições de integridade espacial é uma atividade importante no projeto de uma aplicação, e consiste na identificação de condições que precisam ser garantidas para que o banco de dados esteja sempre íntegro. Os principais tipos de restrições de integridade, que ocorrem freqüentemente na modelagem de banco de dados convencionais, são restrições de *domínio*, de *chave*, de *integridade referencial* e de *integridade semântica* (Elmasri e Navathe, 2004). Cockcroft (1997) estende essa classificação com o objetivo de abranger as peculiaridades dos dados espaciais, incluindo restrições *topológicas*, *semânticas* e *definidas pelo usuário*. Restrições de integridade topológicas consideram as propriedades geométricas e as relações espaciais dos objetos. Existem vários estudos teóricos dos princípios que formalmente definem os relacionamentos espaciais (Egenhofer e Franzosa, 1991). Esses princípios podem ser aplicados entre entidades para prover a base do controle de integridade. As restrições de integridade semânticas dizem respeito ao *significado* implícito às feições geográficas; um exemplo desta restrição é uma regra que impede que edifícios sejam interceptados por trechos de logradouro. As restrições de integridade definidas pelo usuário permitem manter a consistência do banco de dados atuando como "regras de negócio". Um exemplo do uso desta restrição é na localização de postos de gasolina, os quais, por razão legal, precisam estar a pelo menos 200 metros de distância de qualquer escola. Restrições definidas pelo usuário podem ser armazenadas e garantidas por um repositório ativo.

As primitivas dos diagramas de classe, transformação e apresentação são apresentadas a seguir.

3.4.2 Diagrama de classes

No OMT-G o diagrama de classes é usado para descrever a estrutura e o conteúdo de um banco de dados geográfico. Ele contem elementos específicos da estrutura de um banco de dados, em especial classes de objetos e seus relacionamentos. O diagrama de classes contem apenas regras e descrições que definem conceitualmente como os dados serão estruturados, incluindo a informação do tipo de representação que será adotada para cada classe. Por esta razão, o diagrama de classe é o produto fundamental do nível de representação conceitual (Figura 3.1). A seguir

estão descritas as primitivas do modelo OMT-G que são usadas para criar o diagrama de classes para as aplicações geográficas.

Classes

As classes definidas pelo modelo OMT-G representam os três grandes grupos de dados (contínuos, discretos e não-espaciais) que podem ser encontrados nas aplicações geográficas, proporcionando assim, uma visão integrada do espaço modelado. Suas classes podem ser *georreferenciadas* ou *convencionais*.

A distinção entre classes convencionais e georreferenciadas permite que aplicações diferentes compartilhem dados não espaciais, desta forma facilitando o desenvolvimento de aplicações integradas e a reutilização de dados. A *classe georreferenciada* descreve um conjunto de objetos que possuem representação espacial e estão associados a regiões da superfície da terra (Câmara, 1995), representando a visão de campos e de objetos. A *classe Convencional* descreve um conjunto de objetos com propriedades, comportamento, relacionamentos, e semântica semelhantes, e que possuem alguma relação com os objetos espaciais, mas que não possuem propriedades geométricas.

As classes georreferenciadas são especializadas em classes do tipo *geo-campo* e *geo-objeto*. Classes *geo-campo* representam objetos e fenômenos distribuídos continuamente no espaço, correspondendo a variáveis como tipo de solo, relevo e geologia (Câmara, 1995). Classes *geo-objeto* representam objetos geográficos particulares, individualizáveis, associados a elementos do mundo real, como edifícios, rios e árvores. As classes covencionais são simbolizadas exatamente como na UML. As classes georreferenciadas são simbolizadas no modelo OMT-G de forma semelhante (Figura 3.2a), incluindo no canto superior esquerdo um retângulo que é usado para indicar a forma geométrica da representação. Em ambos os casos, símbolos simplificados podem ser usados. Os objetos podem ou não ter atributos não espaciais associados, listados na seção central da representação completa. Métodos ou operações são especificados na seção inferior do retângulo.

O modelo OMT-G apresenta um conjunto fixo de alternativas de representação geométrica, usando uma simbologia que distingue geo-objetos e geo-campos (Figura 3.3 e Figura 3.4).

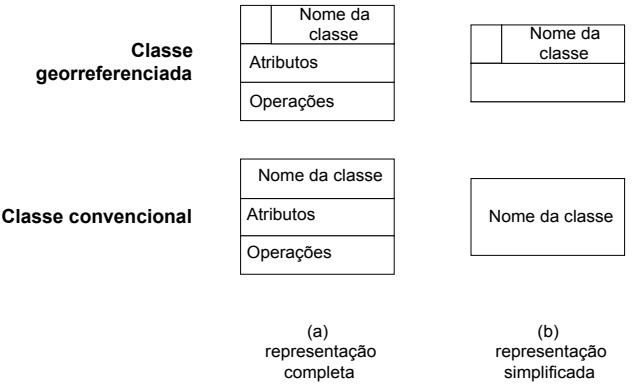


Figura 3.2 – Notação gráfica para as classes do modelo OMT-G.

O modelo OMT-G define cinco classes descendentes de geo-campo: isolinhas, subdivisão planar, tesselação, amostragem e malha triangular (*triangulated irregular network*, TIN) (Figura 3.3), e duas classes descendentes de geo-objeto: geo-objeto com geometria e geo-objeto com geometria e topologia (Figura 3.4).

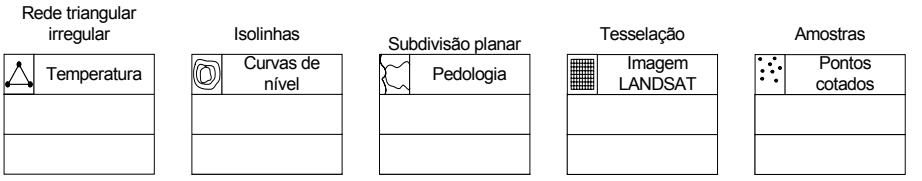


Figura 3.3 – Geo-campos.

A classe geo-objeto com geometria representa objetos que possuem apenas propriedades geométricas, e é especializada em classes: *Ponto*, *Linha* e *Polígono*. Como exemplo citamos, respectivamente, árvore, meio-fio e edificação (Figura 3.4). A classe geo-objeto com geometria e topologia representa objetos que possuem, além das propriedades geométricas, propriedades de conectividade topológica, sendo especificamente voltadas para a representação de estruturas em rede, tais como sistemas de abastecimento de água ou fornecimento de energia

elétrica. Essas propriedades estão presentes em classes descendentes que representam nós e arcos, da forma usualmente adotada na teoria dos grafos. Os arcos podem ser unidirecionais, como em redes de esgoto, ou bidirecionais, como em redes de telecomunicações. Assim, as especializações previstas são denominadas *nó de rede*, *arco unidirecional* e *arco bidirecional*. Os segmentos orientados traduzem o sentido do fluxo da rede, se unidirecional ou bidirecional, dando mais semântica à representação. O foco do modelo OMT-G com respeito a redes não está concentrado na implementação do relacionamento entre seus elementos, mas sim na semântica da conexão entre elementos de rede, que é um fator relevante para o estabelecimento de regras que garantam a integridade do banco de dados. Nas aplicações de rede os relacionamentos do tipo *conectividade* e *adjacência* são fundamentais. Alguns SIG oferecem suporte ao armazenamento desses tipos de relacionamentos.

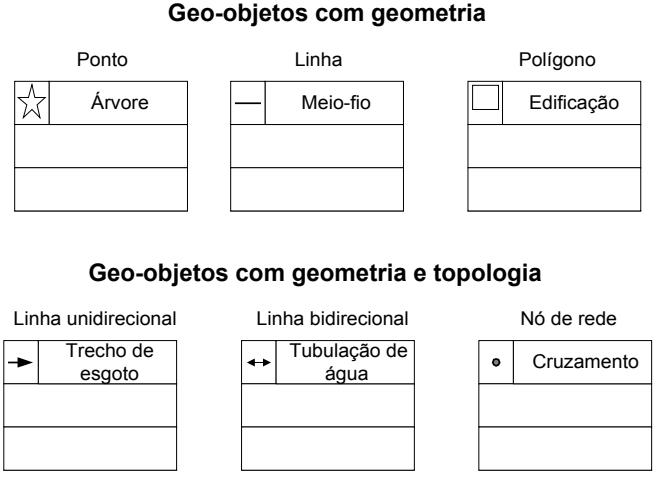


Figura 3.4 – Geo-objetos.

Relacionamentos

Um problema existente na maioria dos modelos de dados é o fato deles ignorarem a possibilidade de modelagem dos relacionamentos entre fenômenos do mundo real (Oliveira et al., 1997). Considerando a importância das relações espaciais e não espaciais na compreensão do espaço modelado, o modelo OMT-G representa três tipos de relacionamentos entre suas classes: associações simples, relacionamentos topológicos em rede e relacionamentos espaciais. A discriminação de tais relacionamentos tem o objetivo de definir explicitamente o tipo de interação que ocorre entre as classes.

Associações simples representam relacionamentos estruturais entre objetos de classes diferentes, convencionais ou georreferenciadas. *Relacionamentos espaciais* representam relações topológicas, métricas, de ordem e *fuzzy*. Algumas relações podem ser derivadas automaticamente, a partir da forma geométrica do objeto, no momento da entrada de dados ou da execução de alguma análise espacial. Relacionamentos topológicos são um exemplo dessa possibilidade. Outras relações no entanto, precisam ser especificadas explicitamente pelo usuário, para permitir que o sistema armazene e mantenha atualizada aquela informação. Estas relações são chamadas de *explícitas* (Peuquet, 1984).

No modelo OMT-G, associações simples são indicadas por linhas contínuas, enquanto relacionamentos espaciais são indicados por linhas pontilhadas (Figura 3.5a/b). Isso torna fácil a distinção visual entre relacionamentos baseados em atributos alfanuméricos e baseados na localização e forma geométrica dos objetos. O nome do relacionamento é anotado sobre a linha, e uma seta usada para deixar clara a direção de leitura (por exemplo, na Figura 3.5b, lê-se "lote contém edificação").

Os relacionamentos de rede são relacionamentos entre objetos que estão conectados uns com os outros. Relacionamentos de rede são indicados por duas linhas pontilhadas paralelas, entre as quais o nome do relacionamento é anotado (Figura 3.5c). Os relacionamentos são em geral especificados entre uma classe de nós e uma classe de arcos, mas estruturas de redes sem nós podem ser definidas, especificando um relacionamento recursivo sobre uma classe de arcos (Figura 3.5d).

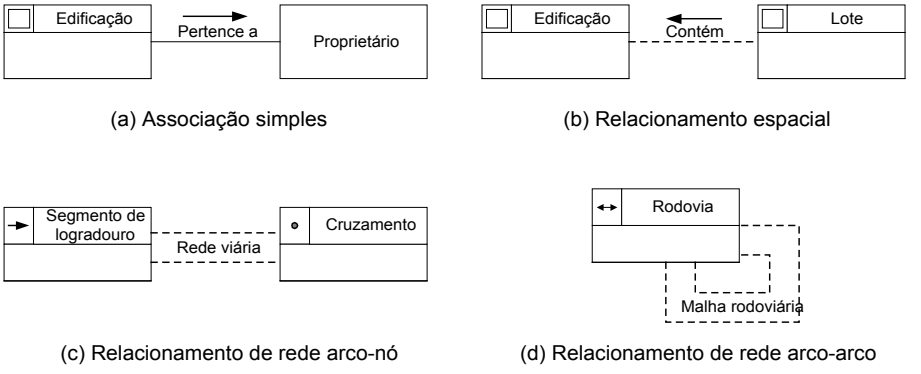


Figura 3.5 – Relacionamentos.

Com base em trabalhos anteriores (Câmara, 1995) (Egenhofer e Franzosa, 1991) (Egenhofer e Herring, 1990), o modelo OMT-G considera um conjunto de relacionamentos espaciais entre classes georreferenciadas. Em (Clementini et al., 1993), um conjunto mínimo de relacionamentos espaciais é identificado, compreendendo somente cinco relacionamentos espaciais, a partir dos quais todos os outros podem ser especificados: *toca*, *em*, *cruza*, *sobrepõe* e *disjunto*. Relacionamentos definidos com base nas matrizes de 4 interseções (Egenhofer e Franzosa, 1991) e de 9 interseções (Egenhofer, 1993) têm sido adotados de forma crescente pelos SIG e SGBD espaciais comerciais. Entretanto, consideramos que, eventualmente, um conjunto maior de relacionamentos é necessário devido a fatores culturais ou semânticos que são familiares para os usuários, incluindo relacionamentos de significado "difuso", tais como *perto de*, ou *ao norte de* (Goyal, 2000).

Alguns relacionamentos só são possíveis entre determinadas classes, pois são dependentes da representação geométrica. Por exemplo, o relacionamento *contém* pressupõe que uma das classes envolvidas seja um polígono. Neste aspecto, as aplicações tradicionais diferem das geográficas, onde as associações entre classes convencionais podem ser feitas livremente, sendo independente de fatores como comportamento geométrico. O conjunto de conceitos que o usuário tem sobre cada objeto do mundo real sugere uma determinada representação porque existe uma interdependência entre a representação, o tipo de interpretação e a finalidade que será dada a cada entidade geográfica. No modelo OMT-G

isto é considerado para que sejam estabelecidas as relações que envolvem classes georreferenciadas.

Cardinalidade

Os relacionamentos são caracterizados por sua cardinalidade. A cardinalidade representa o número de instâncias de uma classe que podem estar associadas a instâncias da outra classe. A notação de cardinalidade adotada pelo modelo OMT-G (Figura 3.6) é a mesma usada na UML (Rational Software Corporation, 1997).

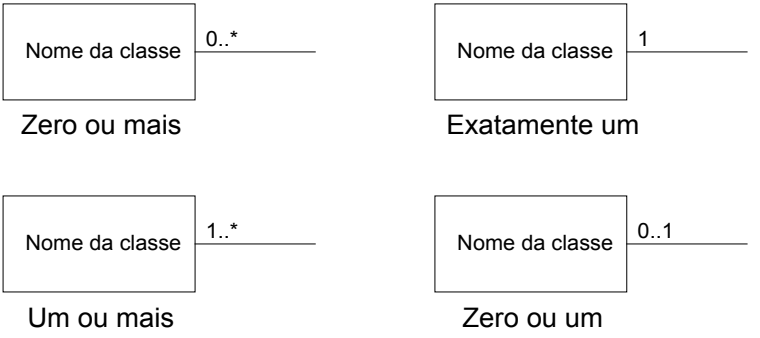


Figura 3.6 – Cardinalidade.

Generalização e especialização

Generalização é o processo de definição de classes mais genéricas (superclasses) a partir de classes com características semelhantes (subclasses) (Elmasri e Navathe, 2004) (Laender e Flynn, 1994). A especialização é o processo inverso, no qual classes mais específicas são detalhadas a partir de classes genéricas, adicionando novas propriedades na forma de atributos. Cada subclasse herda atributos, operações e associações da superclasse.

No modelo OMT-G, as abstrações de generalização e especialização se aplicam tanto a classes georreferenciadas quanto a classes convencionais, seguindo as definições e a notação propostas na UML, em que um triângulo conecta a superclasse a suas subclasses. (Figura 3.7). Cada generalização pode ter um *discriminador* associado, que indica qual

propriedade ou característica está sendo abstraída pelo relacionamento de generalização.

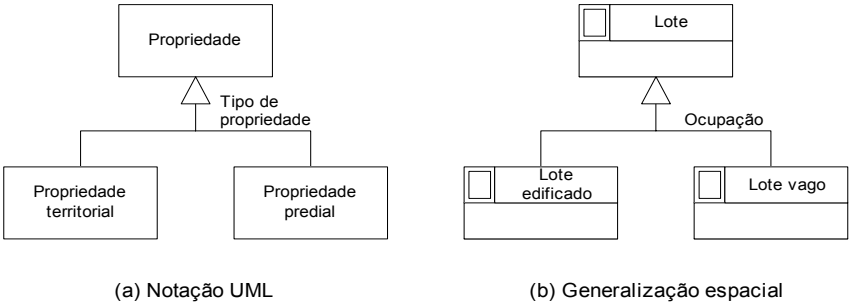


Figura 3.7 - Generalização/especialização.

Uma generalização (espacial ou não) pode ser especificada como *total* ou *parcial* (Laender e Flynn, 1994; Rational Software Corporation, 1997). Uma generalização é total quando a união de todas as instâncias das subclasses equivale ao conjunto completo de instâncias da superclasse. A UML representa a totalidade através do uso dos elementos de restrição predefinidos como *completo* e *incompleto*, mas no modelo OMT-G foi adotada a notação introduzida em (Laender e Flynn, 1994), na qual um ponto é colocado no ápice do triângulo para denotar a totalidade (Figura 3.8). Além disso, o modelo OMT-G também adota a notação OMT (Rumbaugh et al., 1991) para os elementos de restrição predefinidos como *disjunto* e *sobreposto* da UML, ou seja, em uma generalização disjunta o triângulo é deixado em branco e em uma generalização sobreposta o triângulo é preenchido. Portanto, a combinação de disjunção e totalidade gera quatro tipos de restrições aplicáveis a generalização/especialização. A Figura 3.8 apresenta exemplos de cada combinação.

Agregação

A agregação é uma forma especial de associação entre objetos, onde se considera que um deles é formado a partir de outros. A notação gráfica usada no modelo OMT-G segue a empregada na UML (Figura 3.9). Uma agregação pode ocorrer entre classes convencionais, entre classes

georreferenciadas ou entre uma classe convencional e uma classe georreferenciada (Figura 3.10). Quando a agregação ocorre entre classes georreferenciadas, é necessário usar a *agregação espacial*.

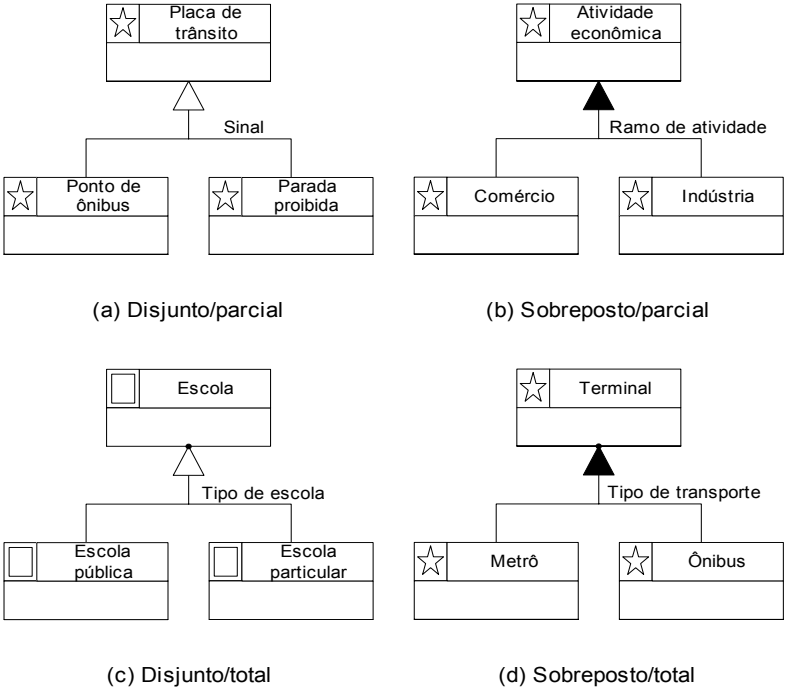


Figura 3.8 – Exemplos de generalização espacial.

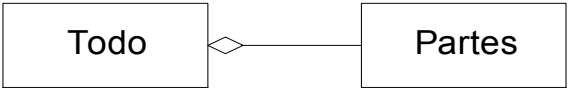


Figura 3.9 – Agregação na notação UML.

Figura 3.10 – Agregação entre uma classe convencional e uma georreferenciada.

A *agregação espacial* é um caso especial de agregação na qual são explicitados relacionamentos topológicos "todo-parte" (Abrantes e Carapuça, 1994) (Kösters et al., 1997). A utilização desse tipo de agregação impõe restrições de integridade espacial no que diz respeito à existência do objeto agregado e dos sub-objetos. Além de o modelo ganhar mais clareza e expressividade, a observação dessas regras contribui para a manutenção da integridade semântica do banco de dados geográfico. Muitos erros no processo de entrada de dados podem ser evitados, se procedimentos baseados nessas restrições forem implementados.

A agregação espacial indica que a geometria de cada parte deve estar contida na geometria do todo. Não é permitida a superposição entre geometria das partes, a geometria do todo deve ser totalmente coberta pela geometria das partes, configurando assim, uma *partição do plano* ou *subdivisão planar* (Davis Jr., 2000) (Preparata e Shamos, 1985). A notação para essa primitiva é apresentada na Figura 3.11, onde mostra uma situação em que quadras são compostas de lotes, ou seja, as quadras são geometricamente equivalentes à união dos lotes contidos nelas.

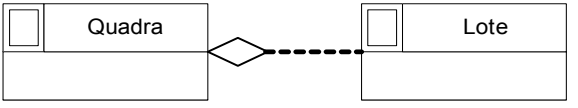


Figura 3.11 – Agregação espacial ("todo-parte").

Generalização conceitual

A *generalização*[1], no sentido cartográfico, pode ser definida como uma série de transformações que são realizadas sobre a representação da informação espacial, cujo objetivo é melhorar a legibilidade e aumentar a facilidade de compreensão dos dados por parte do usuário do mapa. Por exemplo, um objeto do mundo real pode ser diversas representações espaciais, de acordo com a escala de visualização. Uma cidade pode ser

[1] Não se deve confundir a generalização cartográfica com a generalização utilizada como um tipo de abstração usado nos modelos de dados semânticos e orientados a objetos ELMASRI, R.; NAVATHE, S. **Fundamentals of Database Systems.** Pearson Education, 2004..

representada em um mapa de escala pequena por um ponto, e como um polígono em um mapa de escala maior (Davis Jr. e Laender, 1999). Neste sentido, o termo *representação* é usado no sentido de representação da forma geométrica do objeto geográfico.

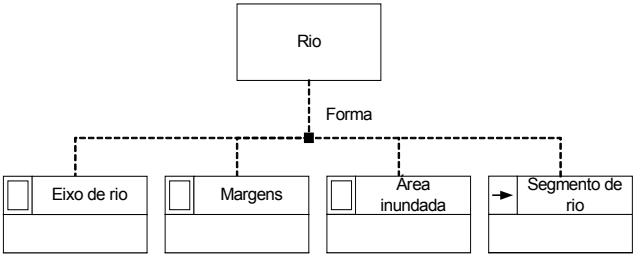
Definir se a representação deve ser simples ou mais elaborada depende da percepção que o usuário tem do objeto correspondente no mundo real, e como essa representação afeta os relacionamentos espaciais que podem ser estabelecidos com outros objetos modelados. Considerando a necessidade de tais relacionamentos, pode haver a demanda para mais de uma representação para um dado objeto. Isso acontece, por exemplo, quando a informação geográfica precisa ser compartilhada entre diversas aplicações em um ambiente corporativo (ou cooperativo).

Portanto, no desenvolvimento de aplicações geográficas, existem situações em que duas ou mais representações para um objeto do mundo real precisam coexistir. Isso significa que, dependendo da visão do usuário, é necessário ter formas geométricas distintas para representar o mesmo objeto geográfico, com a mesma resolução e ao mesmo tempo. Além disso, é freqüente a necessidade de se representar o mesmo objeto com graus variáveis de resolução e detalhamento, configurando representações adequadas para diferentes faixas de escalas.

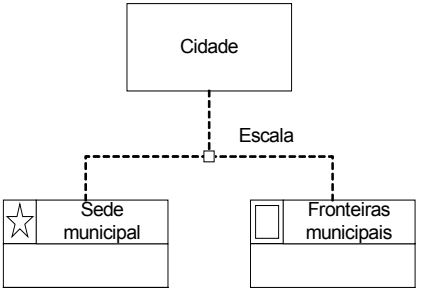
A primitiva de *generalização conceitual* foi incluída no modelo OMT-G para registrar a necessidade de representações diferentes para um mesmo objeto. Nesse tipo de relacionamento, a superclasse não tem uma representação específica, já que poderá ser percebida de maneiras diferentes, conforme especificado nas subclasses. Essas são representadas por formas geométricas distintas, podendo herdar os atributos alfanuméricos da superclasse e ainda possuir atributos próprios. O objetivo é permitir a especificação de relacionamentos independentes envolvendo cada alternativa de representação considerada.

A generalização conceitual pode ocorrer em duas variações: de acordo com a *forma geométrica* (Figura 3.12a) ou de acordo com a *escala* (Figura 3.12b). A v*ariação de acordo com a forma* é utilizada para registrar a existência de múltiplas representações para uma classe, independente de escala. A descrição geométrica da superclasse é deduzida a partir do uso

das subclasses. Por exemplo, um rio pode ser percebido como um espaço entre suas margens, como um polígono de água ou como um fluxo (linha direcionada), formando a rede hidrográfica (Figura 3.12a). *A variação de acordo com a escala* é usada na representação de diferentes aspectos geométricos de uma classe, cada aspecto corresponde a uma faixa de escalas. Uma cidade pode ser representada por suas fronteiras políticas (um polígono) em uma escala maior, e por um símbolo (um ponto) em uma escala menor (Figura 3.12b).



(a) Variação de acordo com a forma (sobreposto)



(b) Variação de acordo com a escala (disjunto)

Figura 3.12 - Generalização conceitual.

Uma estrutura como a apresentada na Figura 3.12 é rara em esquemas de aplicações geográficas, porque as decisões quanto à modelagem de são freqüentemente (e erroneamente) tomadas já pensando na apresentação final, conforme exigido pela aplicação que está sendo modelada. Ou seja, o esquema é muitas vezes concebido visando

um tipo específico de visualização, antecipando uma exigência da aplicação. Esta tendência acaba por inibir usos que exijam representações alternativas, ou aplicações que compartilhem dados geográficos (Davis Jr., 2000).

3.4.3 Diagrama de transformação

O diagrama de transformação, proposto para o modelo OMT-G em (Davis Jr. e Laender, 1999), adota uma notação semelhante à proposta na UML para os diagramas de estados e de atividades (Rational Software Corporation, 1997), e é usado para especificar transformações entre classes. Como tanto a origem quanto o resultado das transformações são sempre as representações de cada classe, o diagrama de transformação também está no nível conceitual de representação. Observe que o diagrama de transformação não pretende descrever aspectos dinâmicos da aplicação, como a interface com o usuário e a execução de consultas, restringindo-se à manipulação de representações.

Os diagramas de transformação são baseados nas primitivas de classe, conforme definidas para os diagramas de classes. As classes que estão envolvidas em algum tipo de transformação são conectadas por meio de linhas contínuas, com setas que indicam a direção da transformação. Os operadores de transformação (TR) envolvidos e seus parâmetros, quando houver, são indicados por meio de texto sobre a linha que indica a transformação.

No diagrama de transformação, pode-se indicar se o resultado da transformação precisa ou não ser materializado. Classes resultantes muito simples, ou que são passos intermediários em uma transformação mais complexa, freqüentemente não precisam ser materializadas, e podem ser armazenadas apenas temporariamente. Tais classes temporárias são indicadas usando linhas tracejadas em seu contorno. As classes que são resultantes de alguma transformação e que precisam ser materializadas (devido à complexidade do processo ou às necessidades específicas da aplicação) são denotadas com linhas contínuas, exatamente como no diagrama de classes.

As transformações indicadas no diagrama de classes podem relacionar qualquer número de classes originais, bem como qualquer número de classes resultantes, dependendo da natureza da operação de

transformação. Cadeias de transformações também podem ser definidas, permitindo, dessa forma, a especificação de processos complexos de análise espacial.

Um operador de transformação adequado para o diagrama de transformação pode ser basicamente qualquer algoritmo que manipula e modifica a representação de um objeto. Algumas operações podem ser melhor caracterizadas como operações TR quando existe apenas uma classe de origem e uma classe resultante, e a classe resultante é ou (1) de natureza diferente da classe original (ou seja, pertence a uma classe georreferenciada diferente), ou (2) menos detalhada que a classe original, mantendo a natureza da representação (Davis Jr. e Laender, 1999).

A especificação de transformações no diagrama de transformação é em geral exigida quando as primitivas de generalização conceitual e de agregação espacial são usadas no diagrama de classes. Essas duas primitivas são indicativas da possibilidade de produzir uma representação a partir de outras.

Um estudo das possíveis transformações entre representações de geo-objetos e geo-campos pode ser visto em (Davis Jr., 2000) (Davis Jr. e Laender, 1999). Os operadores aplicados para cada transformação são baseados em algoritmos definidos nas áreas de geometria computacional, generalização cartográfica e análise espacial. A Seção 3.7 traz um exemplo do uso de diagramas de transformação.

3.4.4 Diagrama de apresentação

O diagrama de apresentação para o modelo OMT-G pertence ao nível de apresentação. Em contraste com o conceito de representação, o termo apresentação é usado no sentido de determinar o aspecto visual ou gráfico (envolvendo parâmetros como cor, tipo de linha, espessura da linha e padrão de hachura), de geo-objetos e geo-campos, no papel ou na tela do computador.

No diagrama de apresentação estão reunidos os requisitos definidos pelo usuário quanto às alternativas de apresentação e saída para cada objeto geográfico. Essas alternativas podem incluir apresentações criadas especificamente para visualização em tela, para impressão na forma de

mapas ou cartas, para interpretação visual em um processo de análise, e outras.

Cada apresentação é definida a partir de uma representação contida no diagrama de classes ou no diagrama de transformação do nível de representação. Operações de transformação para apresentação (TA) são especificadas, permitindo obter o aspecto visual desejado a partir da simples forma geométrica, definida para a representação. Observe-se que a operação TA não modifica a alternativa de representação definida previamente, nem muda o detalhamento definido no nível de representação. Se isso for necessário, uma nova representação tem de ser criada a partir de uma representação existente, usando as ferramentas de especificação de múltiplas representações (como a primitiva de generalização conceitual) e registrando essa demanda nos diagramas de classes e de transformação.

O diagrama de apresentação necessita de apenas três primitivas. A primeira é a própria primitiva de classes, definida para os diagramas de classes e de transformação. A segunda é usada para indicar a operação TA, de maneira semelhante à usada para denotar as transformações no diagrama de transformação. É composta de uma linha tracejada simples, com uma seta que indica o sentido da operação, sobre a qual é especificado o operador a ser usado. No processo de especificação dessa expressão de transformação, quaisquer características geométricas ou atributos alfanuméricos que foram definidos no nível de representação para a classe podem ser usadas como parâmetros. As linhas indicando operações TA são tracejadas para distingui-las visualmente das operações TR, especificadas no diagrama de transformação com linhas contínuas. A terceira primitiva serve para especificar uma apresentação, e contém duas seções. A seção superior indica o nome da classe, o nome da apresentação, e a aplicação na qual é usada. Nessa seção pode-se especificar uma faixa de escalas onde a apresentação será usada. A segunda é dividida em duas partes: à esquerda, um pictograma indica o aspecto visual dos objetos após a transformação e à direita são lançadas especificações mais precisas quanto aos atributos gráficos, incluindo cor da linha, tipo e espessura de linha, padrão de preenchimento, cor de preenchimento, e nome do símbolo (Figura 3.13). A especificação dos atributos gráficos pode ser feita já considerando a codificação de símbolos

usada pelo sistema de informação geográfica subjacente. Pode existir qualquer número de pictogramas na seção esquerda da primitiva de especificação de apresentações, cada qual associada a um valor ou faixa de valores obtidos a partir das características de cada objeto. Nesse caso, a seção da direita deve detalhar os atributos gráficos de cada apresentação gerada. Atributos comuns podem ser especificados apenas uma vez, enquanto atributos variáveis são especificados como listas de valores individuais. Como no caso do diagrama de transformação, os resultados das transformações (ou seja, as apresentações) são indicados com linhas tracejadas quando não precisam ser materializados no banco de dados e com linhas contínuas no caso contrário.

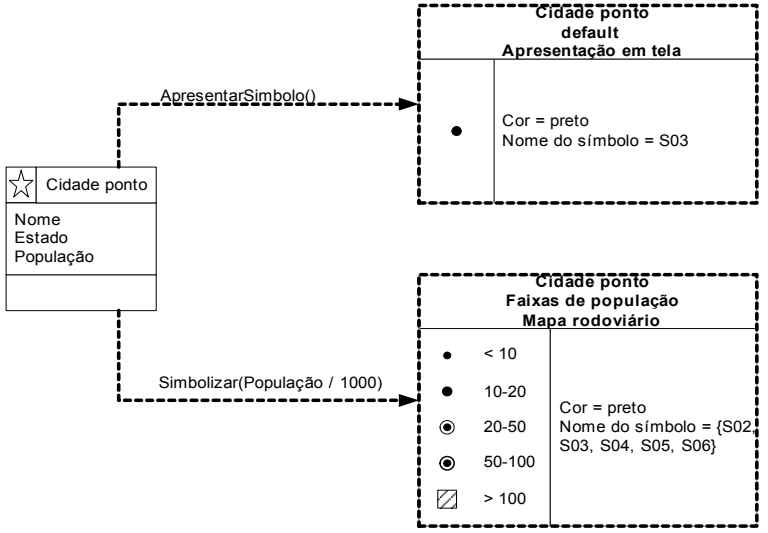


Figura 3.13 – Diagrama de apresentação para a classe cidade ponto.

Cada classe georreferenciada especificada no diagrama de classes precisa ter pelo menos uma apresentação correspondente especificada no diagrama de apresentação. Caso exista mais de uma apresentação para uma dada representação, uma delas deve ser identificada como a *default*. Alternativamente, cada usuário ou aplicação pode eleger sua apresentação *default*.

As operações TA mais comuns envolvem a simples definição de atributos gráficos. No entanto, outros operadores mais sofisticados, muitos dos quais derivados de operações da cartografia temática (classificação, simbolização, exagero, deslocamento, destaque) também podem ser empregados. Uma descrição detalhada dos operadores TA pode ser encontrada em (Davis Jr., 2000) (Davis Jr. e Laender, 1999). A Seção 3.7 exemplifica o uso deste diagrama.

3.5 Restrições de integridade espaciais

No modelo OMT-G, existem diversas restrições de integridade que são implícitas às primitivas do modelo ou que podem ser deduzidas a partir da análise dos diagramas. Assim, restrições de integridade topológica são definidas através de regras para geo-campos (Seção 3.5.1), relacionamentos espaciais (Seção 3.5.2), relacionamentos em rede (Seção 3.5.3) e para agregação espacial (Seção 3.5.4). Da mesma forma, restrições de integridade semântica são definidas através de regras associadas a relacionamentos espaciais. Já as restrições de integridade definidas pelo usuário podem ser modeladas como métodos associados a cada classe. Não estão incluídas aqui restrições de integridade referentes às formas geométricas vetoriais básicas (pontos, linhas e polígonos), fundamentais em SIG e SGBD espaciais, pois consideramos que são inerentes à sua implementação em qualquer produto.

Listamos a seguir as restrições de integridade inerentes às demais primitivas e conceitos do modelo OMT-G, baseadas em trabalhos anteriores (Borges et al., 1999) (Davis Jr. et al., 2001, 2005).

3.5.1 Restrições de integridade para geo-campos

As restrições de integridade R1 a R5 são decorrentes do conceito de geo-campo e de da semântica inerente a cada uma das representações suportada pelo modelo OMT-G.

R1 – Restrição de Preenchimento do Plano. Seja *C* um geo-campo e seja *P* um ponto tal que $P \subset F$. Então o valor *V(P)* = *f(P, C)*, *i.e.*, o valor de *C* em *P*, pode ser univocamente determinado.

R2 – Isolinhas. Seja *C* um geo-campo. Sejam $v_0, v_1, \ldots, v_n$ $n+1$ pontos no plano. Sejam $a_0 = \overline{v_0 v_1}, a_1 = \overline{v_1 v_2}, \ldots, a_{n-1} = \overline{v_{n-1} v_n}$ n segmentos,

conectando os pontos. Esses segmentos formam uma *isolinha* L se, e somente se, (1) a interseção dos segmentos adjacentes em L ocorre apenas no ponto extremo compartilhado pelos segmentos (*i. e.*, $a_i \cap a_{i+1} = v_{i+1}$), (2) segmentos não adjacentes não se interceptam (ou seja, $a_i \cap a_j = \varnothing$ para todo *i, j* tais que $j \neq i+1$), e (3) o valor de C em cada ponto P tal que $P \in a_i$, $0 \leq i \leq n-1$, é constante.

R3 – Tesselação. Seja C um geo-campo. Seja $T = \{t_0, t_1, t_2, ..., t_n\}$ um conjunto de células de forma regular que cobrem C. T é uma *tesselação* de C se, e somente se, para qualquer ponto $P \subset F$, existe exatamente uma célula correspondente $t_i \in T$ e, para cada célula t_i, o valor de C é determinado.

R4 – Subdivisão Planar. Seja C um geo-campo. Seja $A = \{A_0, A_1, A_2, ..., A_n\}$ um conjunto de polígonos tais que $A_i \subset F$ para todo i, sendo $0 \leq i \leq n-1$. A forma uma *subdivisão planar* que representa C se, e somente se, para qualquer ponto $P \subset F$ existir exatamente um polígono A_i correspondente, $A_i \in A$, para o qual o valor de C é determinado (ou seja, os polígonos não se sobrepõem e cobrem C completamente).

R5 – Malha Triangular. Seja C um geo-campo. Seja $T = \{T_0, T_1, T_2, ..., T_n\}$ um conjunto de triângulos tais que $T_i \subset F$ para todo i, sendo $0 \leq i \leq n-1$. T forma uma *malha triangular* que representa C se, e somente se, para qualquer ponto $P \subset F$, existir exatamente um triângulo T_i correspondente, $T_i \in T$, e o valor de C é determinado em todos os vértices de T_i.

3.5.2 Restrições de integridade referentes a relacionamentos topológicos

Restrições referentes a relacionamentos espaciais foram originalmente propostas para o modelo OMT-G baseadas em (Clementini et al., 1993), conforme apresentado em (Borges et al., 2002). Uma descrição detalhada destes relacionamentos está apresentada na Seção 2.9 deste livro.

3.5.3 Restrições de integridade para estruturas em rede

Estruturas em rede, ou seja, formadas por arcos e nós (unidirecionados ou bidirecionados) estão sujeitas às restrições usuais impostas a grafos, enquanto estruturas de dados. Como o modelo OMT-G considera

também o caso de redes formadas apenas por arcos, são apresentados a seguir duas restrições de integridade correspondentes a esses casos.

R6– Redes arco-nó. Seja G = {N, A} uma estrutura de rede, composta de um conjunto de nós N = {n_0, n_1, ..., n_p} e um conjunto de arcos A = {a_0, a_1, ..., a_q}. Membros de N e membros de A são relacionados de acordo com as seguintes restrições: (1) para cada nó $n_i \in N$ deve existir pelo menos um arco $a_k \in A$; (2) para cada arco $a_k \in A$ devem existir exatamente dois nós $n_i, n_j \in N$.

R7 – Redes arco-arco. Seja G = {A} uma estrutura de rede, composta de um conjunto de arcos A = {a_0, a_1, ..., a_q}. A seguinte restrição se aplica: Cada arco $a_k \in A$ deve estar relacionado a pelo menos um outro arco $a_i \in A$, sendo $k \neq i$.

3.5.4 Restrições de integridade referentes à agregação espacial

A restrição a seguir é necessária para garantir a correta semântica de relacionamentos todo-parte no banco de dados.

R8 – Agregação espacial. Seja $P = \{P_0, P_1, \ldots, P_n\}$ um conjunto de geo-objetos. *P* forma outro objeto, *W*, por agregação espacial se, e somente se (1) $P_i \cap W = P_i$ para todo *i* tal que $0 \leq i \leq n$, e (2) $\left(W \cap \bigcup_{i=0}^{n} P_i\right) = W$, e ainda (3) (($P_i$ *toca* P_j) $\vee$ (P_i *disjunto* P_j)) = VERDADEIRO para todo *i*, *j* tais que $i \neq j$.

3.6 Mapeamento para esquemas de implementação

Apresentamos a seguir uma proposta de mapeamento de esquemas OMT-G no nível de representação conceitual para esquemas de implementação. Em seguida, faremos algumas considerações sobre alternativas de estruturação física para corresponder a classes georreferenciadas.

3.6.1 Mapeamento de esquemas conceituais OMT-G para esquemas de implementação

Na fase de mapeamento, é necessário o conhecimento de qual SGBD será usado na aplicação. No caso deste capítulo, a fim de simplificar a explicação, consideramos por enquanto um SGBD espacial objeto-relacional genérico, em que os dados alfanuméricos e geográficos estão codificados num mesmo registro, e os dados geográficos são codificados de acordo com as especificações do OpenGIS Consortium (1999). Como veremos na próxima seção, é possível optar entre algumas organizações físicas diferentes; esta opção pode ser feita após a conclusão do mapeamento, ou em uma etapa posterior de *tuning* do banco de dados.

Inicialmente, faremos um mapeamento das classes de objetos presentes no diagrama de classes do OMT-G para estruturas objeto-relacionais adequadas. Em seguida, cuidaremos da escolha de estruturas de dados para a implementação das alternativas de representação previstas no modelo OMT-G. Por fim, faremos o mapeamento dos relacionamentos necessários. Observe que relacionamentos espaciais em geral não precisam ser materializados no esquema de implementação, uma vez que a associação entre os objetos envolvidos pode ser feita por meio de algoritmos geométricos (vide Capítulo 2 deste livro).

A Tabela 3.1 é uma adaptação da tabela de correspondência entre os modelos ER e relacional apresentada em (Elmasri e Navathe, 2004), e resume uma correspondência básica entre os construtores dos modelos OMT-G e objeto-relacional.

Tabela 3.1 - Mapeamento entre primitivas OMT-G e objeto-relacionais

Modelo OMT-G	Modelo Objeto-relacional
Classe Georreferenciada	Relação "entidade" com representação geométrica associada (vide Seção 3.6.2); se do tipo geo-campo, restrições de integridade referentes à representação adotada (R1 a R5)
Classe Convencional	Relação "entidade"

Associação simples com cardinalidade 1:1 ou 1: N	Par chave estrangeira-chave primária
Associação simples com cardinalidade N : M	Relação "relacionamento" e dois pares chave estrangeira-chave primária
Relacionamento espacial topológico	Restrição de integridade relativa ao tipo de relacionamento espacial (R6 a R12)
Relacionamento em rede arco-nó	Dois pares chave estrangeira-chave primária entre a relação arco e a relação nó (nó anterior e nó posterior); restrição de integridade espacial adequada (R13)
Relacionamento em rede arco-arco	Dois pares chave estrangeira-chave primária em auto-relacionamento sobre a relação arco; restrição de integridade espacial adequada (R14)
Agregação	Par chave estrangeira-chave primária entre a classe "parte" e a classe "todo"
Agregação espacial	Restrição de integridade relativa a agregação espacial (R15)
Generalização / especialização	Restrições de integridade entre subclasses e superclasse (Elmasri e Navathe, 2004 Cap. 7)
Atributo simples	Atributo simples (coluna)
Atributo composto	Conjunto de atributos simples componentes
Atributo multivalorado	Relação e chave estrangeira
Atributo-chave	Chave primária (ou candidata)
Métodos ou operações	*Triggers* ou programas associados

Detalhamos a seguir os quatro principais passos do mapeamento de esquemas conceituais OMT-G para esquemas de implementação, nos quais a correspondência expressa na Tabela 3.1 é empregada.

Passo 1: Mapeamento de classes georreferenciadas e convencionais. Para cada classe convencional presente no diagrama, criar uma tabela, sendo que cada atributo alfanumérico da classe é transformado em uma coluna da tabela. Escolher um dos atributos-chave para ser a chave primária da tabela; caso nenhum atributo atenda aos requisitos de não-duplicidade e inexistência de valores nulos, um novo atributo precisa ser criado para essa finalidade.

O mesmo procedimento se aplica a classes georreferenciadas, decidindo-se adicionalmente a alternativa de representação segundo os tipos geométricos disponíveis no banco de dados escolhido. A Tabela 3.2 apresenta uma correspondência entre os tipos geométricos básicos do modelo OMT-G e os propostos pelo Consórcio OpenGIS (1999). Naturalmente, as representações de geo-campos exigem mais do que apenas a codificação geométrica: atributos devem ser incluídos de modo a armazenar o valor do geo-campo associado a cada elemento da representação.

Tabela 3.2 – Tipos Geométricos

Representação OMT-G		*Representação OpenGIS (Simple Features Specification)*
Geo-objeto	Ponto	Point
Geo-objeto	Linha	LineString
Geo-objeto	Polígono	Polygon
Geo-objeto	Nó de rede	Point
Geo-objeto	Arco unidirecionado	LineString
Geo-objeto	Arco bidirecionado	LineString
Geo-campo	Amostras	Point
Geo-campo	Isolinhas	LineString e/ou Polygon
Geo-campo	Subdivisão planar	Polygon
Geo-campo	Triangulação	Point (vértices) e Polygon (triângulos)
Geo-campo	Tesselação	GeoRaster, campo binário longo

Observe-se que tesselações no OMT-G podem corresponder a dois tipos de representação física sutilmente diferentes: imagens digitais e grades regulares. Assim, caso a representação conceitual seja uma tesselação, pode-se optar entre uma representação matricial própria do SGBD (como a GeoRaster do Oracle™ Spatial) ou um campo binário longo, contendo dados binários em um determinado formato de imagem ou grade. Em ambos os casos, é necessário ter recursos para recuperar o valor de uma determinada célula individualmente.

Passo 2: Mapeamento das associações simples. Para cada relacionamento por associação simples entre classes, de cardinalidade 1:1, escolher uma das classes e incluir nela a chave primária da outra, no papel de chave estrangeira. Para associações de cardinalidade 1:N, incluir na tabela correspondente à classe do lado N, como chave estrangeira, a

chave primária da tabela correspondente à classe do lado 1. No caso de associações de cardinalidade N:M, criar uma tabela intermediária, contendo as chaves primárias de ambas as tabelas envolvidas, no papel de chaves estrangeiras de suas respectivas tabelas, e formando, juntas, a chave primária da nova tabela. O tratamento de associações simples independe da existência ou não de representação geométrica na tabela. Tratar desta forma também os relacionamentos de agregação convencionais.

Passo 3: Mapeamento de relacionamentos espaciais. Na maioria dos casos, relacionamentos espaciais explicitados em diagramas de classe OMT-G (incluindo agregações espaciais) não são materializados no esquema físico. Por outro lado, constituem declarações do relacionamento esperado entre instâncias das classes envolvidas, e freqüentemente denotam restrições de integridade espaciais.

Assim, o mapeamento ideal de relacionamentos espaciais não causa alterações diretamente nas tabelas construídas até este passo, mas requer a implementação de controles dinâmicos (*triggers*) ou estáticos (verificações *offline* de consistência).

Passo 4: Mapeamento de generalizações e especializações. Em esquemas OMT-G, tanto a superclasse quanto as subclasses recebem, se forem georreferenciadas, o mesmo tipo de representação geométrica. Assim, o mapeamento de generalizações e especializações é o mesmo para classes convencionais e georreferenciadas, e pode ainda ser estendido para generalizações conceituais. Subclasses especializadas constituem subconjuntos das instâncias das superclasses, contendo eventualmente atributos próprios. Nesses casos é conveniente que as subclasses sejam tabelas distintas por motivos de gerenciamento da informação geográfica e de visualização. Apesar de estarmos considerando o uso de um banco de dados objeto-relacional deve-se ter em mente que a visualização e a facilidade de manipulação das tabelas deve sempre nortear o modelo lógico e físico de um banco de dados geográfico. O mapeamento pode ser feito de acordo com uma das seguintes opções:

Opção 1. Criar uma tabela para a superclasse, contendo todos os seus atributos e sua chave primária. Criar uma tabela para cada subclasse,

usando a mesma chave primária da superclasse, e também estabelecendo-a como chave estrangeira em relação à tabela correspondente à superclasse. Neste caso, a representação geográfica deverá ficar nas subclasses. Esta abordagem é conveniente para subclasses que necessitam sempre serem visualizadas de forma distinta como por exemplo, com simbologia diferente ou tipo de traço diferente. Também é conveniente para que visualização seja automática não precisando depender de nenhum comando específico para que isto aconteça. Quando as subclasses herdam todos os atributos da superclasse e não possuem atributos específicos, ou quando recebem alguma numeração seqüencial essa opção deve ser usada. Um exemplo do uso de numeração seqüencial é o caso dos nós da rede de esgoto, onde cada nó recebe uma numeração de cadastro independente do seu tipo.

Opção 2. Criar uma tabela para cada subclasse, contendo todos os seus atributos e também todos os atributos herdados da superclasse, inclusive a chave primária. Não criar tabela para a superclasse. Essa abordagem é conveniente para subclasses que contenham atributos próprios e visualização distinta.

Opção 3. Criar uma única tabela contendo todos os atributos da superclasse, inclusive a chave primária, e todos os atributos de cada subclasse. Acrescentar dois atributos (discriminador), um para indicar o tipo da subclasse e outro para indicar a qual subclasse pertence cada linha da tabela. Apesar desta opção ser usada em projetos físicos de banco de dados relacionais, ela não é adequada a aplicações geográficas por requerer outros tipos de controle para acesso e visualização correta dos dados.

A alternativa 1 é mais conveniente para especialização/generalização total e disjunta, quando as subclasses possuírem alguma identificação única gerenciada pela superclasse. Já a alternativa 2 é mais conveniente para especialização/generalização total e disjunta, onde as subclasses possuem atributos próprios. No caso de sobreposição, a alternativa 1 é mais conveniente, uma vez que os atributos em comum ficam na tabela da superclasse. Normalmente, em casos de sobreposição, existe um conjunto de atributos que são comuns.

3.6.2 Alternativas de estruturação de tabelas

Para efetivar adequadamente um mapeamento entre um esquema lógico e um esquema físico dentro das definições do OpenGIS Consortium, é necessário discutir algumas alternativas de implementação. O *Simple Features Specifications* (SFS) do OGC (1999) se restringe à codificação da forma geométrica dos objetos, incluindo as coordenadas geográficas de seus vértices e a definição do sistema de coordenadas. Não faz parte das especificações do OGC a organização física das tabelas, sendo deixada para o projetista a tarefa de decidir qual é a melhor alternativa para receber os componentes espaciais e alfanuméricos de cada classe de objetos constante do esquema conceitual.

Dependendo do volume relativo de dados e da intensidade do uso, o projetista pode optar por deixar a representação geométrica integrada ou separada dos atributos convencionais. Vamos aqui assumir que a representação geométrica possa ser armazenada em uma coluna de uma tabela, através de um mecanismo objeto-relacional, uma extensão especial, ou mesmo um campo binário longo. Com isso, configuram-se três alternativas:

1. Armazenamento de todas as representações geométricas de todos os objetos de todas as classes em uma única tabela, relacionando esta tabela por meio de uma chave estrangeira com diversas outras tabelas, cada qual contendo os atributos alfanuméricos de uma classe específica (Figura 3.14).
2. Armazenamento da representação geométrica em uma coluna de uma tabela, relacionada com outra tabela contendo os atributos alfanuméricos da classe de objetos através de uma chave estrangeira (Figura 3.15);
3. Armazenamento da representação geométrica e dos atributos alfanuméricos de uma classe de objetos como colunas da mesma tabela (Figura 3.16);

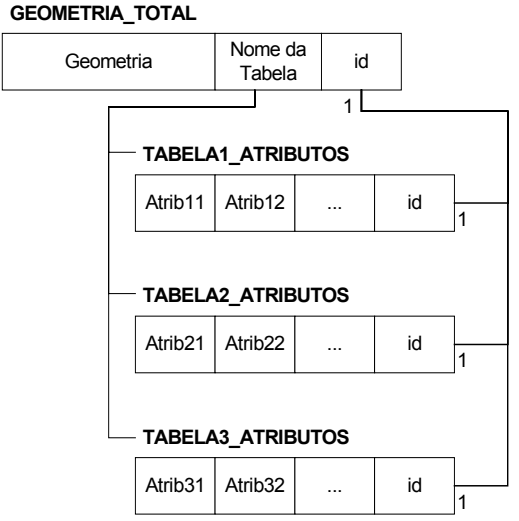


Figura 3.14 – Alternativa 1: geometrias concentradas em uma única tabela (Fonte: (Davis Jr. e Oliveira, 2002)).

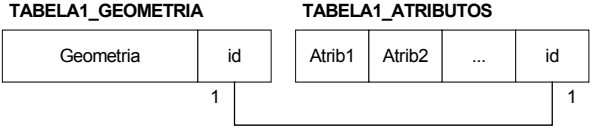


Figura 3.15 – Alternativa 2: um par de tabelas para cada classe georreferenciada (Fonte: (Davis Jr. e Oliveira, 2002)).

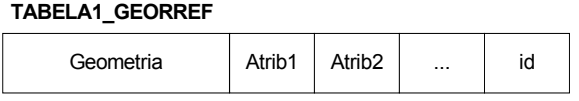


Figura 3.16 – Alternativa 3: geometria e atributos na mesma tabela (Fonte: (Davis Jr. e Oliveira, 2002)).

A alternativa 1 tende a introduzir um desequilíbrio no SGBD, fazendo com que todas as consultas e operações envolvendo dados geométricos passem pela única tabela que os armazena. Em um banco de dados dotado de um volume razoavelmente grande de dados geográficos, essa tabela pode rapidamente se tornar um gargalo para todo o sistema.

Por outro lado, pode-se imaginar que a indexação espacial e as operações topológicas entre classes de objetos sejam eventualmente beneficiadas pela integração das representações geométricas em uma única tabela. É também possível imaginar vantagens quanto ao acesso às tabelas de atributos alfanuméricos, que se tornam menos volumosas pela separação das representações geométricas. Esse esquema foi adotado por alguns SIG no passado, com relativo sucesso.

A alternativa 2 destaca-se por sua flexibilidade, apesar de exigir a navegação entre tabelas ou a realização de operações de junção para que se possa resgatar a estrutura completa de cada objeto geográfico. A separação dos atributos alfanuméricos em uma tabela independente facilita a integração com aplicações convencionais. A implementação da restrição de integridade referencial entre as duas tabelas é, no entanto, indispensável – o que pode se constituir em um problema para a implementação de aplicações exclusivamente alfanuméricas e que pretendam operar sobre esses dados.

A terceira alternativa é a que mais se assemelha à concepção de objetos geográficos adotada pelo modelo OMT-G. Cada tupla de cada tabela passa a corresponder, aproximadamente, a uma instância de um objeto, sendo que a tabela contém todas as instâncias de uma determinada classe. Com isso, não são necessárias junções para acessar dados geométricos e atributos, o que pode beneficiar aplicações de análise espacial ou mapeamento temático, em particular aquelas que não exigem muitos dados alfanuméricos. Esta alternativa corresponde ao mapeamento mais simples de se executar e à opção mais conservadora quanto ao desempenho, considerando uma igual incidência de operações alfanuméricas e espaciais.

Observe-se que esquemas baseados na primeira alternativa podem ser facilmente mapeados para a segunda, dividindo a grande tabela de dados geométricos em várias (o que pode ser feito usando o mecanismo de visões). Também podem ser mapeados para a terceira, pela realização de uma junção após a separação da tabela geométrica em várias, o que também pode ser feito usando visões. O mesmo raciocínio pode ser empregado para implementar um mapeamento entre a alternativa 2 e a alternativa 3, ou vice-versa.

Uma alternativa adicional consiste na implementação de uma terceira tabela para viabilizar um relacionamento *n:m* entre representações geométricas e atributos alfanuméricos, dando ao usuário a possibilidade de combinar esses aspectos de acordo com a sua necessidade (Figura 3.17). Essa terceira tabela deve manter duas chaves estrangeiras, uma chave da tabela de representações geométricas com outra da tabela que contém os atributos, que juntas compõem sua chave primária. Com isso é possível, por exemplo, manter simultaneamente uma representação cartográfica e uma representação esquemática de uma rede de distribuição de energia. A situação oposta (várias tuplas de atributos relacionadas a uma única representação geométrica) também é útil. Existem diversas aplicações que armazenam séries temporais de dados, por exemplo na área de meteorologia: cada tupla alfanumérica conteria dados meteorológicos obtidos em uma localidade em um determinado dia e horário, a tupla geométrica traria a localização da estação meteorológica, e a chave composta seria formada pelo identificador da estação meteorológica e pela data e hora da medição.

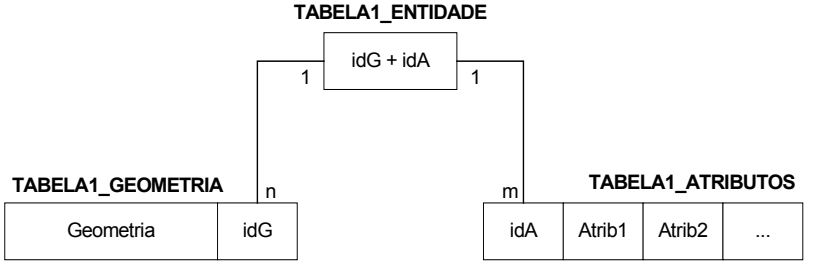


Figura 3.17 – Alternativa 4: múltiplas representações e/ou múltiplos conjuntos de atributos.

No entanto, a opção de manter mais de uma geometria em uma mesma tabela pode ser implementada usando os bancos de dados espaciais atuais. Nesse caso, seria utilizada em tabelas que contenham sempre um ou mais tipos de representação, como por exemplo a frente principal de um lote e o contorno do lote. É importante que, em tabelas desse tipo, as instâncias disponham sempre das representações

consideradas no esquema conceitual, viabilizando a implementação das restrições de de integridade espaciais.

3.7 Discussão de um exemplo

Para exemplificar o uso das principais primitivas do modelo OMT-G, apresentamos nesta Seção um exemplo de modelagem, apresentado originalmente em (Davis Jr., 2000). A aplicação proposta combina aspectos de interesse em três diferentes contextos:

- cadastro técnico municipal (CTM), em que os usuários estão interessados na estruturação da ocupação do solo urbano em quadras, lotes e vias públicas;
- gerenciamento de transportes e trânsito, em que o interesse está na estruturação do sistema viário;
- mapeamento em escala regional, em que os usuários se interessam apenas pelos principais aspectos de ocupação do território e acessos, em especial a malha rodoviária.

Alguns objetos do ambiente urbano são necessários nos três contextos, como por exemplo o sistema viário. No entanto, cada um deles percebe esses objetos de uma maneira diferente, gerando a necessidade de mais de uma representação.

Para os usuários da área de cadastro técnico municipal, os principais objetos são as quadras e lotes da cidade. Particularmente no caso dos lotes, adotamos três diferentes alternativas para representação. A primeira e mais simples delas é utilizando pontos. No caso de Belo Horizonte, esta forma de representação foi adotada no início da construção do banco de dados geográfico para o cadastro, a partir de uma metodologia que envolvia uma rápida referência visual à planta cadastral convencional, transformada em imagem e colocada no *background* (Davis Jr., 1993). Essa representação é suficiente para que se possa localizar cada lote, porém não permite que se verifique topologicamente as relações de vizinhança e de inclusão em uma quadra. A segunda alternativa de representação consiste em traçar apenas a testada do lote, usando uma poligonal. Esta alternativa é quase tão simples quanto a primeira quanto ao esforço de conversão de dados, porém permite que se realize alguns

tipos adicionais de análise, como a de vizinhança, e fornece um dado geométrico, que é a largura do lote no segmento frontal. Por fim, a terceira alternativa de representação usa polígonos que definem todas as fronteiras entre o lote e seus vizinhos. É a representação ideal, pois permite verificar todas as confrontações e ainda fornecer parâmetros geométricos básicos, como a área do lote. No entanto, é a mais custosa em termos de conversão de dados. A convivência das três formas de representação de lotes em um mesmo banco de dados se justifica do ponto de vista da formação incremental dos componentes do cadastro urbano. O relacionamento dos lotes e quadras com o sistema de endereçamento da cidade, através da malha viária, é também desejável, para que seja possível simplificar a tarefa de localizá-los em campo e também para facilitar a comunicação com os proprietários e outros cidadãos.

Para que se possa trabalhar com transportes e trânsito, é fundamental poder contar com todas as informações relevantes quanto à malha viária, incluindo a localização de cada logradouro e cada cruzamento entre logradouros. Na presente aplicação, a malha viária básica está representada por uma rede, em que arcos bidirecionais representam os segmentos de logradouro entre cruzamentos, que por sua vez constituem os nós. Observe-se que, por simplicidade, optou-se por não modelar a malha de circulação viária, composta por arcos unidirecionais e que atendem a todas as restrições da legislação de trânsito quanto ao sentido de fluxo. Cada trecho de logradouro recebe uma classificação de acordo com o *Plano de Classificação Viária*, um componente do Plano Diretor do município que define vias de ligação regional, arteriais, coletoras e locais. Essa classificação depende basicamente do volume de tráfego e da função primária de cada trecho de logradouro. Observe-se que a classificação é um atributo do trecho e não do logradouro inteiro, pois existem situações em que parte do logradouro recebe tráfego intenso e parte tem características de via local. Considerando apenas as vias mais importantes para a circulação, concebe-se uma nova rede, esta adequada para o planejamento da circulação de veículos entre regiões da cidade.

Por fim, os responsáveis pelo mapeamento regional estão interessados em obter os limites da área urbanizada da cidade, usualmente denominados "mancha urbana", além das ligações da cidade a outras por

meio de rodovias e outros meios de transporte de cargas. As ferrovias foram deixadas de fora do problema por simplicidade, mas as rodovias que atravessam a cidade fazem parte da malha viária concebida para a aplicação de transportes e trânsito. Além disso, considerando as escalas em que se pretende construir o mapeamento regional, é interessante poder contar com (1) uma representação simplificada do polígono que compõe os limites entre a cidade e suas vizinhas, e (2) uma representação da cidade como ponto, para a geração de mapas temáticos sobre transportes rodoviários.

Considerando as necessidades descritas, foi construído o diagrama de classes para a aplicação (Figura 3.18). No diagrama, a primitiva de generalização conceitual do modelo OMT-G foi usada duas vezes, uma para a classe *Município*, que pode ter representações pontuais ou poligonais, em subdivisão planar (esta em dois diferentes níveis de detalhamento), e outra para a classe *Lote CTM*, que podem ser representados usando pontos, linhas ou polígonos. Existe também uma primitiva de agregação, que indica que as instâncias da classe *Quadra CTM* serão criadas pela agregação de instâncias de *Lote CTM*.

A classe *Rodovia* está relacionada à classe de vias principais, assumindo a regra de que todos os trechos de rodovia são classificados como vias de ligação regional. A classe *Via principal*, por sua vez, é um subconjunto da classe *Trecho*, pois nem todo trecho de logradouro pertence à malha viária principal. Com isso, nem todos os nós de cruzamento constituem interseções na malha viária principal. O diagrama de classes indica, assim, que existe uma superposição parcial entre a malha de logradouros e a malha viária principal, mas não determina a forma de estruturação do banco de dados geográfico quanto a esse aspecto.

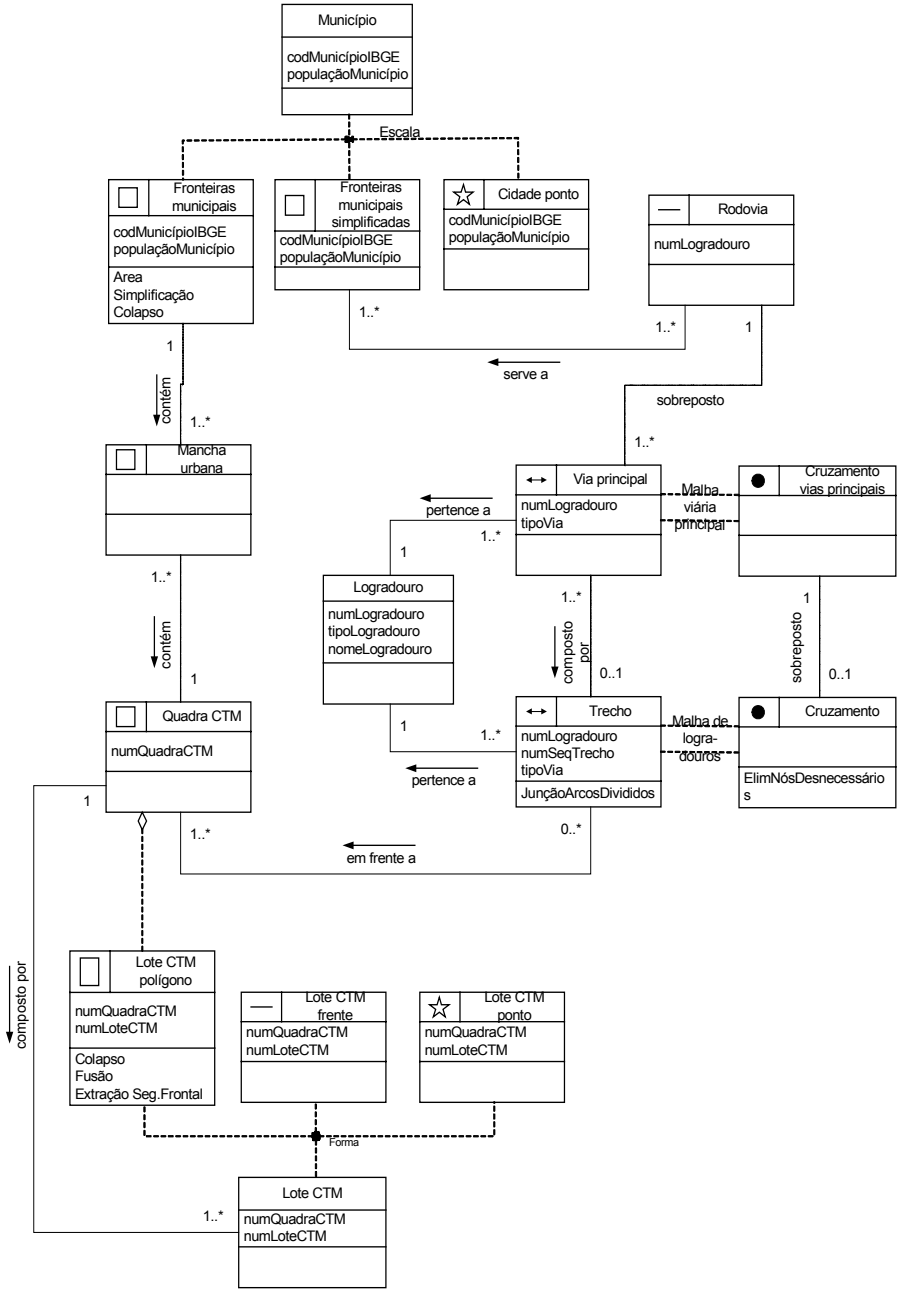


Figura 3.18 – Diagrama de classes.

O diagrama de transformação correspondente ao diagrama de classes da Figura 3.18 pode ser criado em blocos, cada qual correspondendo a um grupo de transformações demandado pela aplicação.

O primeiro desses blocos diz respeito à agregação de *Lote CTM* para formar *Quadra CTM*, e à relação `contém` entre *Quadra CTM* e *Mancha Urbana* (Figura 3.19). No caso da agregação, optou-se por usar o operador de generalização cartográfica denominado *Fusão*, com tolerância de espaçamento igual a zero. Isso faz com que o seu comportamento seja idêntico ao de um operador de união de polígonos, conforme definido na área de geometria computacional. A opção pela implementação do operador *Fusão* pode ser feita considerando que seu uso já é necessário na aplicação, para a transformação que leva à criação da mancha urbana, desta vez considerando uma tolerância de 15 metros. Esse valor de tolerância faz com que desapareçam da mancha urbana todas as ruas cuja largura seja inferior a 30 metros, caso da maioria das vias locais e mesmo algumas avenidas de menor porte. Apenas logradouros mais largos permanecerão visíveis após a aplicação do operador. A Figura 3.20 apresenta um exemplo de aplicação desse operador a um grupo de quadras.

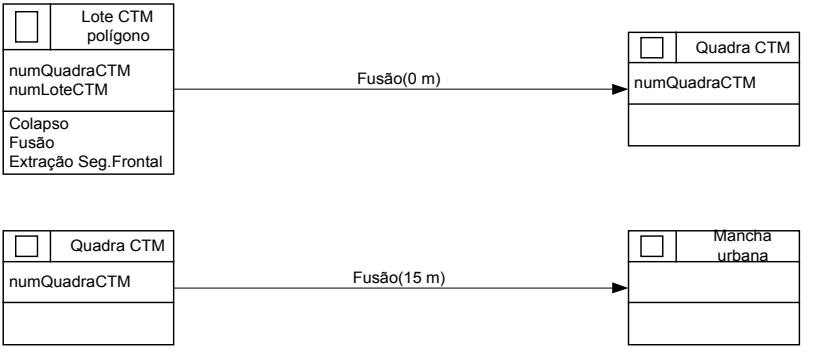


Figura 3.19 – Diagrama de transformação – 1º. bloco.

O segundo bloco de transformações dinâmicas refere-se à criação da malha viária principal a partir da malha de logradouros (Figura 3.21). Como a malha de logradouros é mais detalhada e precisa ser mantida

atualizada para benefício das aplicações de transportes e cadastro, seus arcos e nós são considerados representações primárias, a partir das quais as representações secundárias correspondentes à malha viária principal são criadas. O processo consiste em, inicialmente, selecionar as instâncias de *Trecho* classificadas como vias de ligação regional ou arteriais, ou seja, componentes do sistema viário principal, e em duplicar toda a classe *Cruzamento*. Em seguida, os nós que são desnecessários para a malha viária principal são eliminados, gerando a classe *Cruzamento vias principais*, e juntando os segmentos de arcos onde os nós foram eliminados, produzindo a *classe Via principal*. A eliminação é feita sempre que um nó for encontrado com exatamente zero ou dois arcos conectados: no primeiro caso, o nó não cumpre função alguma na rede, e no segundo ele não configura mais um cruzamento entre vias.

O terceiro bloco de transformações corresponde à generalização conceitual sobre as classes *Município* e *Lote CTM* (Figura 3.22). No primeiro caso, admite-se que a classe *Fronteiras municipais* seja a mais genérica e a mais detalhada, e portanto a partir dela pode-se produzir a classe *Fronteiras municipais simplificadas*, usando um operador de simplificação de polígonos com tolerância de 10 metros, e a classe *Município ponto*, usando o operador *colapso*. A tolerância foi escolhida a partir da definição da escala de trabalho para o mapeamento regional pretendido, no caso equivalente a 1:50.000. A tolerância corresponde a metade da largura de uma linha traçada com pena 0,4mm. Esse valor foi escolhido considerando a dedução do tamanho do menor objeto visível (*smallest visible object*, ou SVO) (Li e Openshaw, 1992). Naturalmente, o tamanho do SVO é expresso nas unidades de medida da tela ou do mapa, e portanto pode ser traduzido em medidas reais através da aplicação do fator de escala.

Como no caso da classe *Fronteiras municipais*, a classe *Lote CTM* polígono foi escolhida como representação primária. A partir dela pode-se gerar a classe *Lote CTM ponto*, usando o operador *Colapso*, e a classe *Lote CTM frente*, usando um operador especial que determina qual é o segmento frontal de cada lote poligonal – no caso, segmentos frontais são aqueles que não são compartilhados pelos lotes adjacentes. Observe-se que as representações produzidas pelo operador *colapso* não precisam ser materializadas, uma vez que seu processamento é bastante rápido.

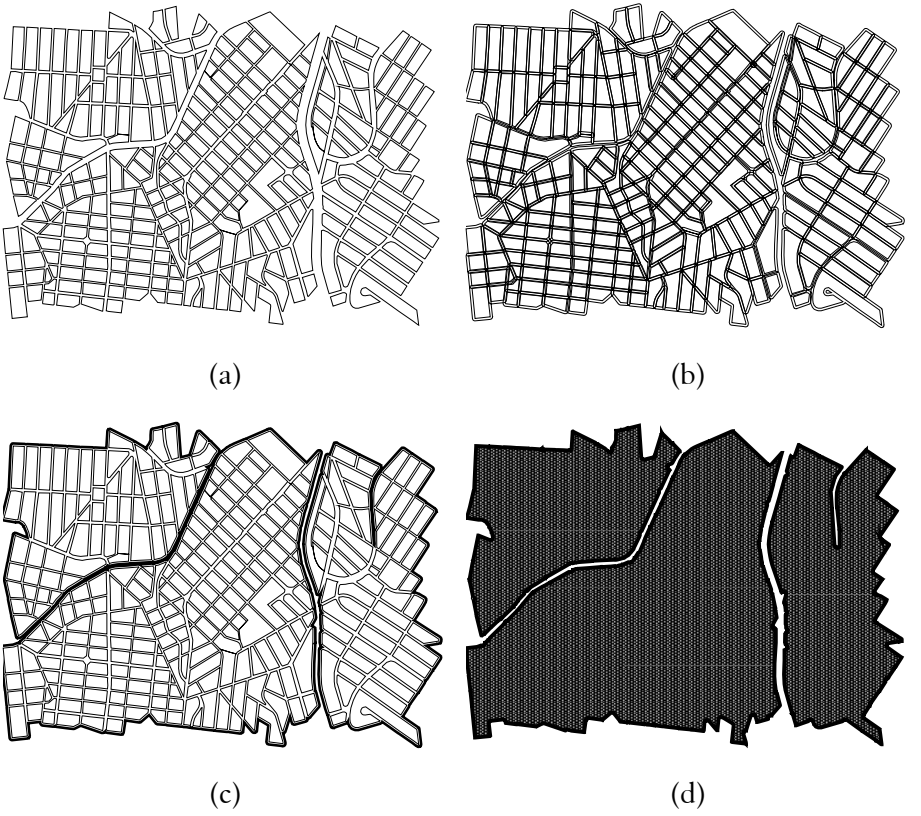


Figura 3.20 – Fusão.

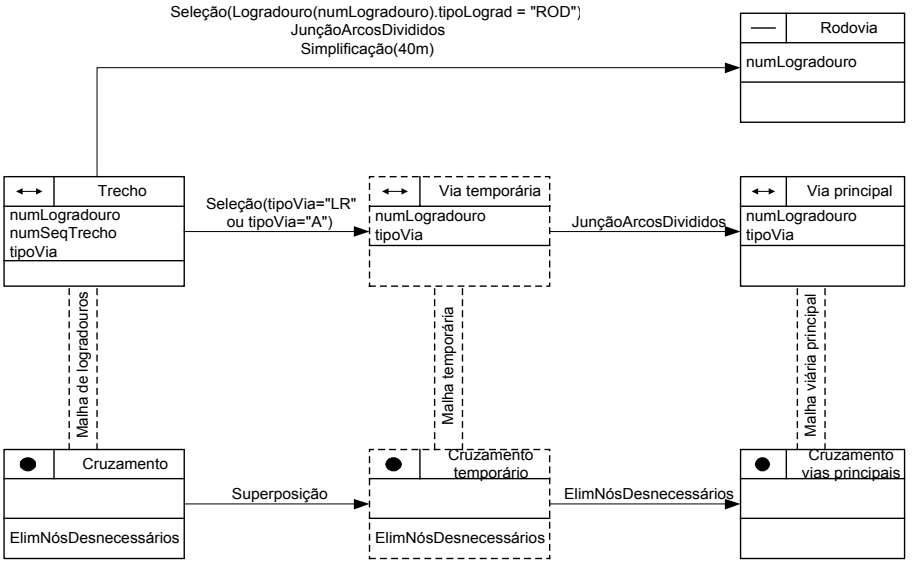


Figura 3.21 – Diagrama de transformação – 2º. bloco.

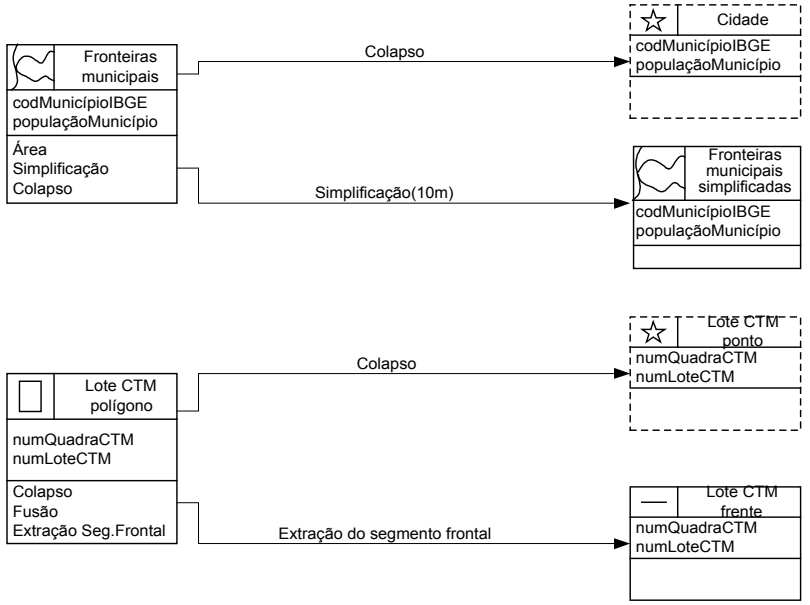


Figura 3.22 – Diagrama de transformação – 3º. bloco.

Cada uma das classes que compõem os diagramas de classes e de transformação precisam ter pelo menos uma apresentação definida no diagrama de apresentação. Além dessa apresentação *default*, cada classe pode ter um número indeterminado de apresentações alternativas, de acordo com as necessidades da aplicação. É possível ter, por exemplo, uma apresentação voltada para visualização em tela e outra voltada para a saída plotada em um mapa.

As classes *Fronteiras municipais* e *Cidade ponto* são associadas a duas apresentações cada, uma para visualização em tela e outra para usos específicos (Figura 3.23). As apresentações em tela estabelecem um limiar de escala, de modo que em escalas até 1:25.000 as fronteiras entre municípios serão visualizadas, entre 1:25.000 e 1:50.000 serão apresentadas as fronteiras simplificadas, e em escalas menores, a partir de 1:50.000, serão apresentados os símbolos.

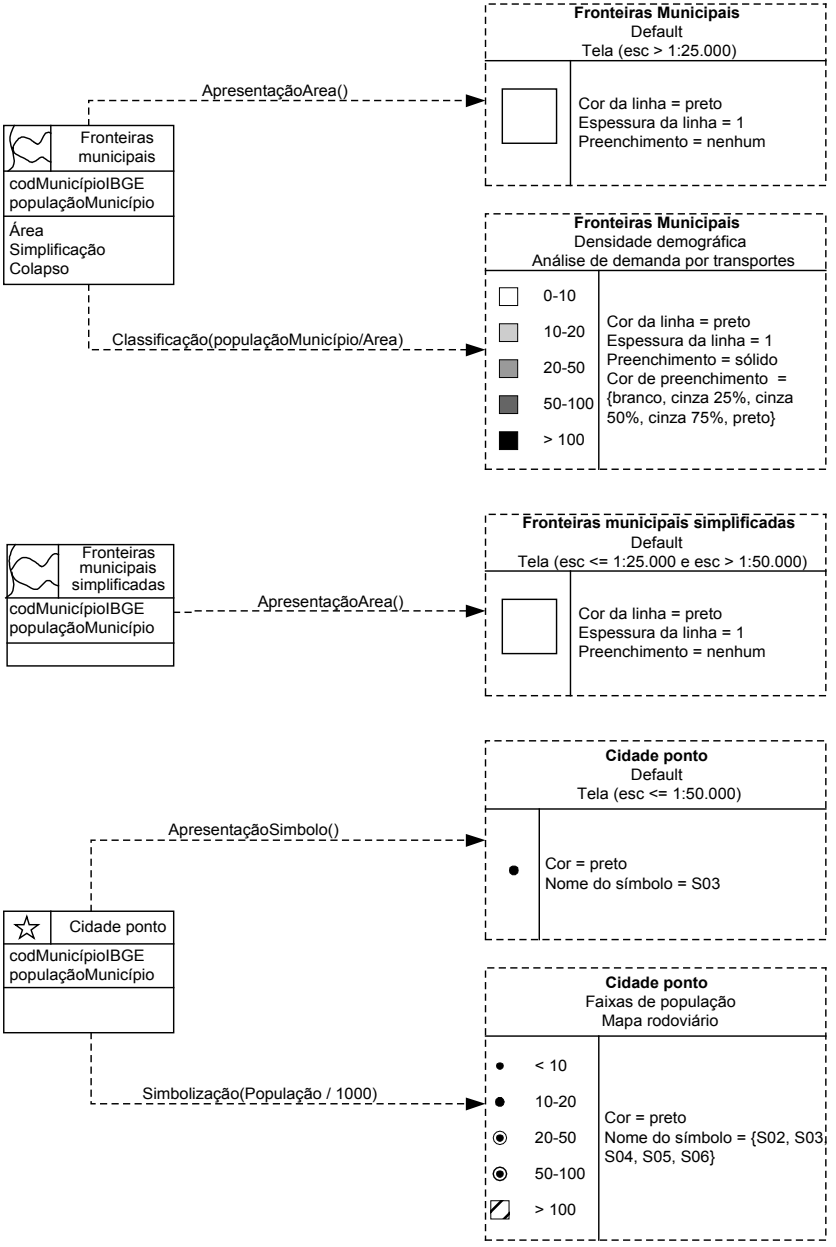


Figura 3.23 – Diagrama de apresentação- 1º. bloco.

Para a classe Rodovia, foi definida apenas uma apresentação, em que estradas de terra são distinguidas visualmente de estradas asfaltadas. Também a classe Mancha urbana conta com apenas uma apresentação, que procura distinguir levemente a área urbanizada da área rural (Figura 3.24).

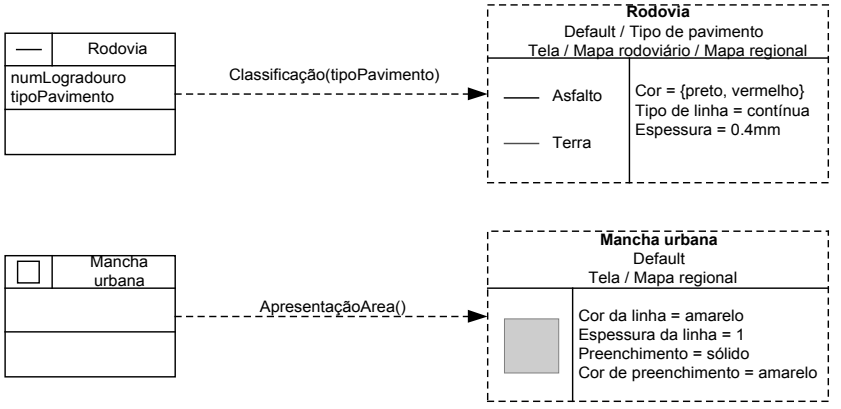


Figura 3.24 – Diagrama de apresentação – 2º. bloco.

Em seguida, as classes que compõem a malha de vias principais têm sua apresentação definida. A classe *Via principal* conta com duas apresentações, sendo que na primeira as vias de ligação regional e arteriais, mais importantes na hierarquia de classificação viária, são diferenciadas usando a espessura da linha, e na segunda apenas as vias de ligação regional são destacadas. A apresentação correspondente à classe *Cruzamento vias principais* usa um símbolo muito pequeno, o que faz com que suas instâncias efetivamente desapareçam na tela. O usuário poderá perceber os cruzamentos de vias visualmente, sem a necessidade de um símbolo mais evidente, o que traria apenas poluição visual (Figura 3.25).

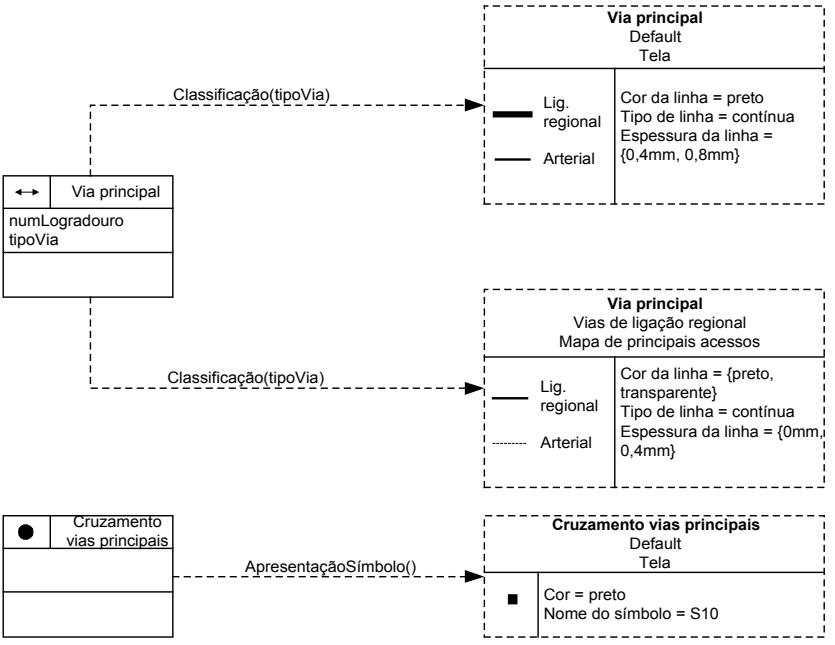


Figura 3.25 – Diagrama de apresentação -3º. bloco.

Para a classe *Trecho* são definidas duas variações de apresentação. A primeira define uma classificação com base na hierarquização do sistema viário. A classe *Cruzamento*, que com *Trecho* compõe a malha de logradouros, é apresentada usando um símbolo circular simples, porém visível. Em ambos os casos, a visualização só é permitida em escalas superiores a 1:5000, pois fora dessa faixa a densidade de elementos na tela seria excessivamente alta (Figura 3.26).

Por fim, todas as demais classes anteriormente definidas recebem uma apresentação correspondente. Pelo menos uma apresentação tem que estar definida para cada classe, e na Figura 3.27 isso foi feito para as classes *Quadra CTM*, *Lote CTM polígono*, *Lote CTM frente* e *Lote CTM ponto*. Observe-se a definição da simbologia para a classe *Lote CTM frente*, em que é utilizado um recurso comum em SIG e cartografia: o lançamento de símbolos ao longo de linhas. No caso, foi inserido um símbolo no início da linha, para estabelecer um marco visual que a separe da frente do lote vizinho. O intervalo entre símbolos foi

especificado usando um valor intencionalmente muito alto, para garantir que o símbolo não venha a ser usado mais de uma vez na frente do mesmo lote. É importante destacar que a especificação dos parâmetros de cada apresentação pode ser baseada nos recursos conhecidos do SIG onde a aplicação será implementada.

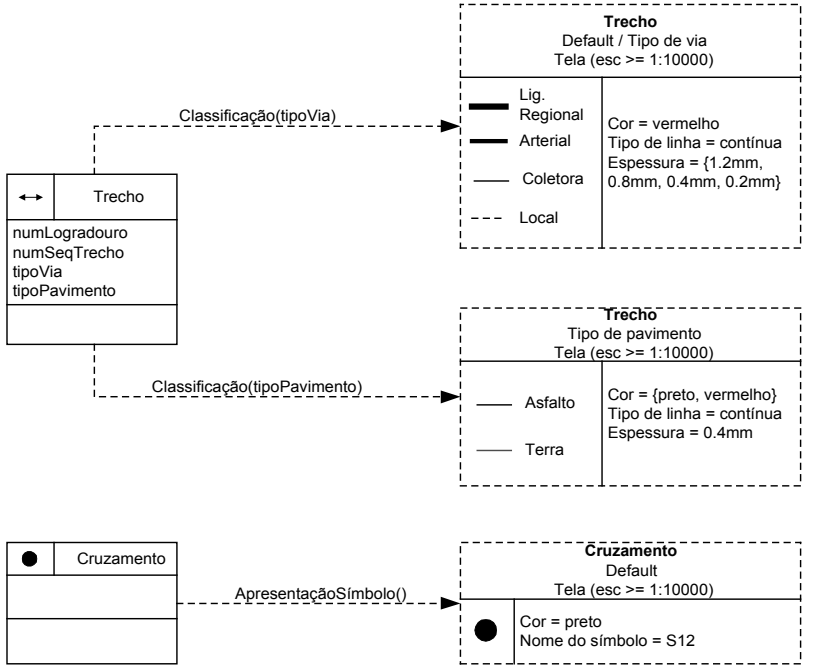


Figura 3.26 – Diagrama de apresentação – 4º. bloco.

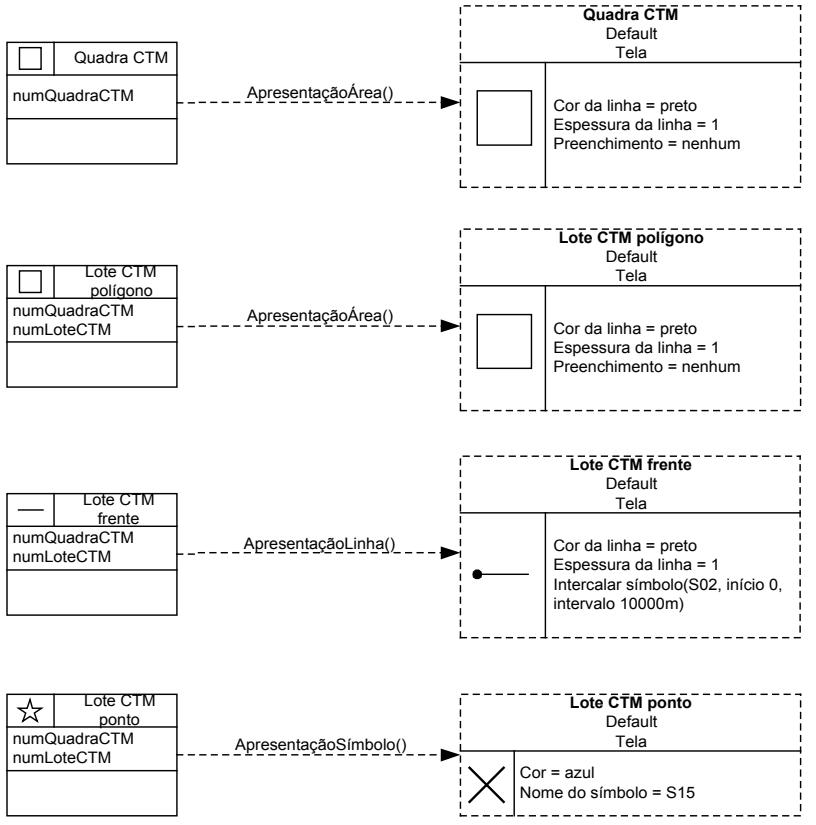


Figura 3.27 – Diagrama de apresentação – 5º. bloco.

3.8 Leituras suplementares

Neste capitulo, apresentamos o OMT-G, um modelo de dados orientado a objetos para modelagem de aplicações geográficas, e técnicas para transformar esquemas OMT-G em esquemas de implementação, supondo um SGBD objeto-relacional compatível com o padrão OGC. O modelo OMT-G oferece primitivas para modelar a geometria e topologia dos dados geográficos. Devido ao uso de pictogramas representando a geometria dos objetos, o esquema resultante é mais compacto, intuitivo e de fácil compreensão. Além do mais a combinação de diagramas de classes, transformação, apresentação faz com que a distância entre a modelagem conceitual e a implementação de aplicações geográficas seja

reduzida, permitindo uma definição mais precisa dos objetos requisitados, suas operações, seus parâmetros de visualização.

Aos leitores interessados em um maior aprofundamento neste tema, recomendamos uma revisão das referências mais citadas ao longo do capítulo. Para obter uma visão mais detalhada de operações de transformação, veja (Davis Jr., 2000) (Davis Jr. e Laender, 1999). Exemplos adicionais de modelagem usando OMT-G podem ser encontrados em numerosos trabalhos, dentre os quais (Bertini e Cézar Neto, 2004) (Davis Jr. et al., 2003) (Martins Netto, 2003) (Preto, 1999) Souza et al., 2004) (Voll, 2002). Comparações entre modelos de dados para aplicações geográficas podem ser encontrados em (Borges, 1997) (Borges et al., 2001) (Lisboa Filho, 1997). Lembramos ainda aos leitores que o modelo OMT-G foi inicialmente chamado de GeoOMT (Borges, 1997), e que existem algumas publicações que se referem a ele por este nome.

O uso de ontologias no projeto e construção de sistemas de informação geográficos, algo que não foi abordado neste capítulo, mas que parece estar um passo adiante das atuais técnicas de modelagem conceitual, é apresentado e discutido em (Fonseca, 2001) (Fonseca et al., 2000) (Fonseca et al., 2002). Uma discussão a respeito da conexão que existe entre modelagem conceitual e ontologias pode ser encontrada em (Fonseca et al., 2002).

Referências

ABITEBOUL, S.; HULL, R. IFO: a formal semantic database model. **ACM Transactions on Database Systems**, v. 12, n.4, p. 525-565, 1987.

ABRANTES, G.; CARAPUÇA, R. Explicit representation of data that depend on topological relationships and control over data consistency. In: Fifth European Conference and Exhibition on Geographical Information Systems - EGIS/MARI'94. 1994. p.

BÉDARD, Y.; CARON, C.; MAAMAR, Z.; MOULIN, B.; VALLIÈRE, D. Adapting data models for the design of spatio-temporal databases. **Computers, Environment and Urban Systems**, v. 20, n.1, p. 19-41, 1996.

BERTINI, G. C.; CÉZAR NETO, J. Uma modelagem orientada a objeto para o Mapa Urbano Básico de Belo Horizonte. **Informática Pública**, v. 6, n.1, p. 33-51, 2004.

BORGES, K. A. V. **Modelagem de dados geográficos - uma extensão do modelo OMT para aplicações geográficas.**Belo Horizonte: Fundação João Pinheiro, 1997.Dissertação de mestrado, Escola de Governo, 1997.

BORGES, K. A. V.; DAVIS JR., C. A.; LAENDER, A. H. F. OMT-G: an object-oriented data model for geographic applications. **GeoInformatica**, v. 5, n.3, p. 221-260, 2001.

BORGES, K. A. V.; DAVIS JR., C. A.; LAENDER, A. H. F., 2002. Integrity constraints in spatial databases. In: DOORN, J. H.; RIVERO, L. C., eds., **Database Integrity: Challenges and Solutions**: Hershey (PA), Idea Group Publishing, p. 144-171.

BORGES, K. A. V.; LAENDER, A. H. F.; DAVIS JR., C. A. Spatial data integrity constraints in object oriented geographic data modeling. In: 7th International Symposium on Advances in Geographic Information Systems (ACM GIS'99). Kansas City, 1999. p. 1-6.

CÂMARA, G. **Modelos, linguagens e arquiteturas para bancos de dados geográficos.**São José dos Campos: INPE, 1995.Tese de doutorado, 1995.

CHEN, P. The entity-relationship model - toward a unified view of data. **ACM Transactions on Database Systems**, v. 1, n.1, p. 9-36, 1976.

CLEMENTINI, E.; DIFELICE, P.; VAN OOSTEROM, P. A small set of formal topological relationships suitable for end-user interaction. In: 3rd Symposium on Spatial Database Systems. 1993. p. 277-295.

COCKROFT, S. A taxonomy of spatial data integrity constraints. **GeoInformatica**, v. 1, n.4, p. 327-343, 1997.

DAVIS JR., C. A. Address base creation using raster-vector integration. In: URISA Annual Conference. Atlanta (GA), 1993. p. 45-54.

DAVIS JR., C. A. **Múltiplas representações em sistemas de informação geográficos.**Belo Horizonte (MG): UFMG, 2000.Departamento de Ciência da Computação, 2000.

DAVIS JR., C. A.; BORGES, K. A. V.; LAENDER, A. H. F. Restrições de integridade em bancos de dados geográficos. In: III Workshop Brasileiro de GeoInformática (GeoInfo 2001). Rio de Janeiro (RJ), 2001. p. 63-70.

DAVIS JR., C. A.; BORGES, K. A. V.; LAENDER, A. H. F., 2005. Deriving spatial integrity constraints from geographic application schemas. In: RIVERO, L. C.; DOORN, J. H.; FERRAGINE, V. E., eds., **Encyclopedia of Database Technologies and Applications**: Hershey (PA), Idea Group Publishing.

DAVIS JR., C. A.; FONSECA, F.; BORGES, K. A. V. A flexible addressing system for approximate urban geocoding. In: V Simpósio Brasileiro de GeoInformática (GeoInfo 2003). Campos do Jordão (SP), 2003. p. em CD-ROM.

DAVIS JR., C. A.; LAENDER, A. H. F. Multiple representations in GIS: materialization through geometric, map generalization, and spatial analysis operations. In: 7th International Symposium on Advances in Geographic Information Systems (ACM GIS'99). Kansas City, 1999. p. 60-65.

DAVIS JR., C. A.; OLIVEIRA, P. A. SIG interoperável e distribuído para administrações municipais de grande porte. **Informática Pública**, v. 4, n.1, p. 121-141, 2002.

EGENHOFER, M. A Model for Detailed Binary Topological Relationships. **Geomatica**, v. 47, n.3 & 4, p. 261-273, 1993.

EGENHOFER, M. J.; FRANZOSA, R. D. Point-set topological spatial relations. **International Journal of Geographic Information Systems**, v. 5, n.2, p. 161-174, 1991.

EGENHOFER, M. J.; HERRING, J. A mathematical framework for the definition of topological relationships. In: 4th International Symposium on Spatial Data Handling. 1990. p. 803-813.

ELMASRI, R.; NAVATHE, S. **Fundamentals of Database Systems.** Pearson Education, 2004.

FONSECA, F. **Ontology-Driven Geographic Information Systems.**Orono: University of Maine, 2001.Ph.D. Thesis, Dept of Spatial Information Science and Engineering, 2001.

FONSECA, F.; EGENHOFER, M.; DAVIS, C.; BORGES, K. Ontologies and Knowledge Sharing in Urban GIS. **Computer, Environment and Urban Systems**, v. 24, n.3, p. 232-251, 2000.

FONSECA, F.; EGENHOFER, M.; DAVIS, C.; CÂMARA, G. Semantic Granularity in Ontology-Driven Geographic Information Systems. **AMAI Annals of Mathematics and Artificial Intelligence - Special Issue on Spatial and Temporal Granularity**, v. 36, n.1-2, p. 121-151, 2002.

FRANK, A. U. Spatial concepts, geometric data models, and geometric data structures. **Computers & Geosciences**, v. 18, n.4, p. 409-417, 1992.

FRANK, A. U.; GOODCHILD, M. F., 1990, Two perspectives on geographical data modeling, Santa Barbara (CA), National Center for Geographic Information and Analysis (NCGIA).

GOYAL, R. K. **Similarity assessment for caardinal directions between extended spatial objects.** Orono, Maine: University of Maine, 2000.PhD Thesis, 2000.

KÖSTERS, G.; PAGEL, B.; SIX, H. GIS application development with GeoOOA. **International Journal of Geographic Information Science**, v. 11, n.4, p. 307-335, 1997.

LAENDER, A. H. F.; FLYNN, D. J., 1994. A semantic comparison of modelling capabilities of the ER and NIAM models. In: ELMASRI, R.; KOURAMAJIAN, V.; THALHEIM, B., eds., **Entity-Relationship Approach - ER'93**, Springer-Verlag, p. 242-256.

LI, Z.; OPENSHAW, S. Algorithms for automated line generalization based on a natural principle of objective generalization. **International Journal of Geographic Information Systems**, v. 6, n.5, p. 373-389, 1992.

LISBOA FILHO, J., 1997, Modelos de dados conceituais para sistemas de informação geográfica, Porto Alegre, UFRGS.

MARK, D. M.; FRANK, A. U., 1990, Language issues for geographical information systems, National Center for Geographic Information and Analysis (NCGIA).

MARTINS NETTO, V. **Proposta de esquema conceitual para um banco de dados de limpeza urbana no município de Belo Horizonte.**Belo Horizonte: PRODABEL, 2003.Monografia de especialização, Curso de Especialização em Informática Pública, 2003.

OLIVEIRA, J. L.; PIRES, F.; MEDEIROS, C. M. B. An environment for modeling and design of geographic applications. **GeoInformatica**, v. 1, n.1, p. 29-58, 1997.

OPEN GIS CONSORTIUM, 1999, OpenGIS Simple Features Specification for SQL - Revision 1.1.

PARENT, C.; SPACCAPIETRA, S.; ZIMANYI, E. Spatio-temporal conceptual models: data structures + space + time. In: 7th International Symposium on Advances in Geographic Information Systems (ACM GIS'99). Kansas City, 1999. p. 26-33.

PEUQUET, D. J. A conceptual framework and comparison of spatial data models. **Cartographica**, v. 21, p. 66-113, 1984.

PREPARATA, F. P.; SHAMOS, M. I. **Computational geometry: an introduction.** New York: Springer-Verlag, 1985.

PRETO, A. G. **MetaSIG: ambiente de metadados para aplicações de sistemas de informações geográficos.**Rio de Janeiro (RJ): Instituto Militar de Engenharia, 1999.Dissertação de mestrado, 1999.

RATIONAL SOFTWARE CORPORATION, 1997, The Unified Modeling Language: notation guide, version 1.1.

RUMBAUGH, J.; BLAHA, M.; PREMERLANI, W.; EDDY, F.; LORENSEN, W. **Object-oriented Modeling and Design.** Prentice-Hall, 1991.

SHEKHAR, S.; COYLE, M.; GOYAL, B.; LIU, D.; SARKAR, S. Data models in geographic information systems. **Communications of the ACM**, v. 40, n.4, p. 103-111, 1997.

SOUZA, L. A.; DELBONI, T.; BORGES, K. A. V.; DAVIS JR., C. A.; LAENDER, A. H. F. Locus: um localizador espacial urbano. In: VI Simpósio Brasileiro de GeoInformática (GeoInfo 2004). Sociedade Brasileira de Computação (SBC), Campos do Jordão (SP), 2004. p. 467-478.

VOLL, V. L. **Utilização de SIG na análise de aspectos sociais do garimpo de diamantes em Coromandel, MG.**Belo Horizonte: UFMG, 2002.Monografia de especialização, Curso de Especialização em Geoprocessamento, 2002.

WORBOYS, M. F.; HEARNSHAW, H. M.; MAGUIRE, D. J. Object-oriented data modelling for spatial databases. **International Journal of Geographic Information Systems**, v. 4, n.4, p. 369-383, 1990.

4 *Modelos espaço-temporais*

Taciana de Lemos Dias
Gilberto Câmara
Clodoveu A. Davis Jr.

4.1 Introdução

Este capítulo apresenta recentes estudos de *modelos espaço-temporais*, correspondentes as iniciativas para modelar o comportamento de objetos em sua trajetória espaço-temporal, visando sua representação em um sistema de informação.

A maioria das aplicações de tecnologia de geoinformação utiliza representações estáticas de fenômenos espaciais. Isto se deve ao fato de que a principal abstração utilizada em Sistemas de Informação Geográficas (SIG) é o mapa. No entanto, um significativo conjunto de fenômenos espaciais, como o cadastro urbano, uso e ocupação da terra, fluxos hidrológico e poluição são inerentemente dinâmicos e as representações estáticas comumente utilizadas não os capturam de forma adequada. Deste modo, um dos grandes desafios da geoinformação é o desenvolvimento de *modelos espaço-temporais*, que sejam capazes de representar adequadamente fenômenos que variam tanto no espaço como no tempo.

Modelos espaço-temporais reúnem dois aspectos distintos: a escolha de conceitos adequados do espaço e do tempo e a construção de representações computacionais apropriadas correspondentes a esses conceitos. No caso das representações espaciais estáticas, os Capítulos 1 e 3 apresentam as diversas alternativas existentes, juntamente com detalhes de implementação e de modelagem de aplicações. Neste capítulo, damos

ênfase à representação temporal e à construção de modelos semânticos apropriados ao tratamento de mudanças espaço-temporais.

A visão do espaço (do grego *choros*) e do tempo (*chronos*) é uma experiência subjetiva do ser humano. O espaço e o tempo se misturam ao se descrever uma realidade (Kavouras, 2001). Podemos modelar a superfície da terra usando geo-objetos, correspondentes a parcelas do solo, ou usando geo-campos, indicando a variação espacial da vegetação da mesma área. Geo-objetos podem ser estáticos, como uma montanha; mudar de lugar, como o traçado de uma linha férrea, ou se movimentar, como um carro (Frank, 1997). Conforme a semântica associada ao geo-objeto, suas características espaciais (incluindo forma geométrica e localização) e não espaciais (atributos alfanuméricos) podem sofrer alterações ao longo do tempo.

Para produzir uma representação do mundo real com o objetivo de elaborar um sistema de informação espaço-temporal muitas questões precisam ser investigadas e respondidas (Cheylan, 2001). Essas questões envolvem a visão de mundo inerente ao sistema, as regras aplicáveis, o comportamento dos objetos ao longo do tempo, a interpretação da variação do tempo, a natureza das mudanças e a influência dos processos de medida. Por este motivo, o uso de ontologias para modelagem espaço-temporal é um dos principais temas de pesquisa nessa área atualmente (Worboys e Duckhan, 2004) (Grenon e Smith, 2003). Os conceitos envolvidos não são óbvios e têm se mostrado de difícil formalização (Smith e Mark, 1998) (Frank, 2003) (Fonseca et al., 2002) (Fonseca, et al. 2003). Em particular quando lidam com aspectos espaciais e temporais simultaneamente, ontologias buscam capturar as propriedades dos objetos e os conceitos que determinam sob que condições eles são criados ou deixam de existir, e quais mudanças podem ocorrer em suas características (Grenon e Smith, 2003). Uma ontologia deve incluir categorias de espaço e tempo, além dos conceitos ligados ao histórico dos objetos e às intenções de mudança. Ontologias espaço-temporais representam um conjunto de conceitos que lidam com a natureza do espaço, do tempo e das interações espaço-temporais (Peuquet, 2001).

Ainda não existe um consenso sobre as técnicas de modelagem de dados espaço-temporais, ou mesmo sobre extensões das técnicas de modelagem de dados geográficos atualmente existentes para refletir as necessidades de aplicações que envolvam simultaneamente tempo e espaço. Optamos por apresentar alternativas de representação específicas, deixando a cargo do leitor a escolha do(s) modelo(s) que melhor se ajusta(m) às suas necessidades.

4.2 Representação do tempo

Uma representação temporal considera os aspectos de *ordem*, *variação e granularidade* (Edelweiss e Oliveira, 1994) que serão analisados a seguir.

4.2.1 Ordem

Quanto à ordem, o tempo pode ser consecutivo e linearmente ordenado, ramificado ou circular (Worboys e Duckhan, 2004). O tempo linearmente ordenado possui uma ordenação entre quaisquer dois pontos: se t e t' são dois pontos diferentes no tempo, e "<" é o operador de ordem de precedência temporal, apenas uma das expressões é verdadeira: (1) $t < t'$ ou (2) $t' < t$ (Figura 4.1).

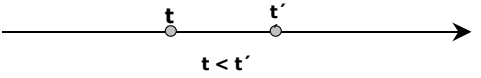
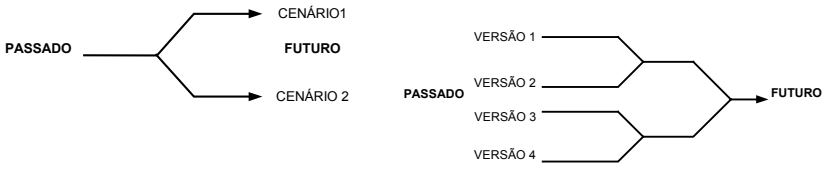


Figura 4.1 – Tempo consecutivo e linearmente ordenado .

O tempo ramificado implica na possibilidade de existência de diferentes histórias futuras ou passadas. Podem existir várias versões do passado correspondentes a uma situação do presente, ou existir vários cenários para uma situação futura (Figura 4.2).



(a) (b)

Figura 4.2 – Tempo ramificado (Fonte: adaptado de Worboys e Duckhan , 2004).

O tempo ramificado no futuro possui diferentes sucessores (Figura 4.2 a), e ramificado no passado possui diferentes antecessores (Figura 4.2 b). A maioria das representações da realidade utiliza um passado linear e um futuro ramificado (Edelweiss e Oliveira, 1994).

Eventos recorrentes são representados pelo tempo *circular*. Neste caso, a periodicidade de sua ocorrência faz com que sempre se volte à mesma referência de tempo. Um exemplo é o ciclo anual de produção de mudas de plantas como mostra a Figura 4.3.

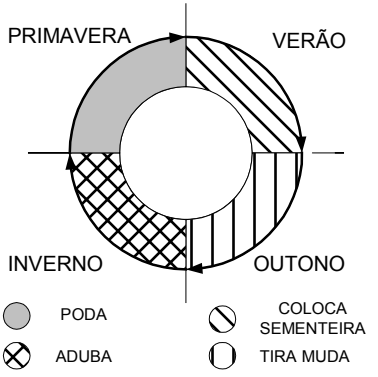


Figura 4.3 – Tempo circular.

4.2.2 Variação

Quanto à variação, o tempo pode ser *contínuo* ou *discreto*. Entendemos, em geral, que o tempo é contínuo por natureza. Para sua representação computacional, é necessário utilizar uma representação discreta do tempo, na qual a variação temporal corresponde a uma linha de tempo,

composta por uma seqüência de *chronons* consecutivos e com idêntica duração. Um *chronon* é um intervalo temporal que não pode ser decomposto. Ele é considerado a menor unidade de duração do tempo de um sistema. A duração de tempo pode ser fixa, como uma hora, ou variável, como um mês (Figura 4.4).

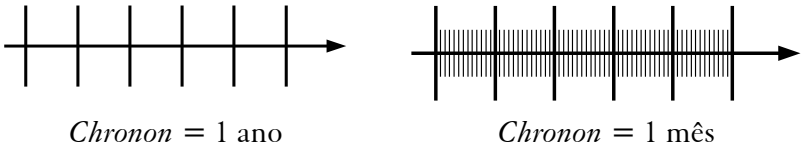


Figura 4.4 – Diferentes granularidades temporais.

Um *eixo temporal* é uma seqüência de pontos consecutivos com tempo de variação discreto, linear e finito. A variação do tempo discreto é classificada como *ponto-a-ponto*, *escada* e *função* (Renolen, 1997). A *variação ponto-a-ponto* considera valores válidos do tempo somente nos pontos temporais definidos. Na variação *escada*, o valor válido do tempo ocorre desde o momento de sua definição até o momento em que outro valor é definido. A variação do tempo discreto pode ser determinada por uma função de interpolação para determinar valores em pontos onde não se tem valor, configurando a *variação por função*. Esses tipos de variações são mostradas na Figura 4.5.

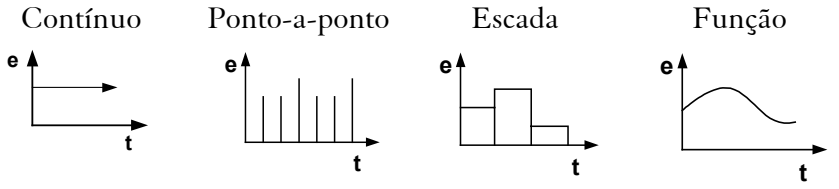


Figura 4.5 – Variação do tempo contínuo e discreto.

4.2.3 Granularidade

A *granularidade temporal* é um parâmetro que corresponde à duração de um *chronon*. Pode-se considerar, simultaneamente, diferentes granularidades (ano, mês, dia e minuto), para possibilitar uma melhor

representação da realidade. Pesquisas de opinião, por exemplo, podem ser realizadas anualmente ou mensalmente (Edelweiss e Oliveira, 1994), porém esperamos obter uma representação mais fiel ao fenômeno real com a granularidade mensal.

Os elementos primitivos de representação da granularidade do tempo são *instante*, *intervalo* e *elemento temporal*. A granularidade depende da variação do tempo considerada. Para o tempo continuamente variável, um *instante* é um ponto no tempo cuja duração é infinitesimal, sendo que entre dois pontos no tempo sempre existirá outro ponto. Se a variação de tempo for discreta, um instante é representado por um *chronon* no eixo temporal. Um *intervalo* é um subconjunto de pontos do eixo temporal equivalente ao tempo decorrido entre dois pontos. Considerando o tempo discreto, o intervalo é representado por um conjunto finito de *chronons* consecutivos; no caso de tempo contínuo, existem infinitos instantes de tempo em um intervalo. Um *elemento temporal* é a união finita de intervalos de tempo, produzindo um novo elemento temporal para as operações de conjunto de união, interseção e complemento (Langran, 1993).

O tempo pode ainda ser *absoluto* ou *relativo*. O tempo absoluto está associado a um fato com granularidade definida. O tempo é considerado relativo quando se refere à validade de outro fato. A definição do tempo pode também ser *explícita* ou *implícita*. Ela é explicitada através de um rótulo temporal (*timestamp*) associado a cada valor de atributo de um objeto. A definição de tempo é necessária para utilização de uma linguagem de lógica temporal implícita, como no caso do tempo relativo (Edelweiss e Oliveira, 1994).

4.3 Dimensão temporal

A dimensão temporal determina as representações de tempo num banco de dados. Essas dimensões auxiliam na definição da composição histórica do geo-objeto. Uma análise de dados espaço-temporais requer uma distinção entre o momento em que o evento ocorreu, conforme a representação adotada (*tempo de validade*), e o momento em que essa ocorrência foi registrada no banco de dados (*tempo de transação*), que

indica a partir de quando a informação correspondente ao evento se tornou disponível para o usuário. Por motivos diferentes, é importante, por exemplo, saber quando dados que indicam a possibilidade de um desastre foram inseridos no banco de dados e quando eles foram coletados ou identificados em campo. As informações temporais, nesse caso, permitem analisar se dados que indicam um possível desastre foram identificadas e/ou registrados no banco de dados em tempo de apoiar o processo de tomada de decisão.

4.3.1 Tempo em bancos de dados

Os SGBD podem ser classificados, (Tabela 4.1), de acordo com seu suporte à dimensão temporal (Worboys e Duckhan, 2004) (Snodgrass, 1992). Eles podem ser *estáticos* (*instantâneos*), *de tempo de validade* (*históricos*), *de tempo de transação* (*rollback*) e *bitemporais* (Snodgrass, 1990) (Al-Taha e Barrera, 1994) (Edelweiss e Oliveira, 1994).

Um *SGBD estático* é também denominado *banco de dados de instantâneos*, por não suportar nenhuma dimensão temporal. Neste caso, o SGBD suporta um único estado, o mais recente, e não é possível realizar comparações entre estados. Um *SGBD de tempo de validade* registra fatos de acordo com o tempo de ocorrência do evento, e possibilita uma recuperação de histórico. Não é possível recuperar o instante em que os dados foram inseridos no banco de dados. *Um SGBD de tempo de transação* registra o instante da inserção de dados no banco, possibilitando uma recuperação de dados para desfazer uma transação (*rollback*). No entanto, não realiza o registro de quando o fato ocorreu. *Um SGBD bitemporal* registra tanto o tempo de validade quanto o de transação, sem qualquer interação entre eles, não sendo possível uma "atualização do passado". É possível acessar o histórico das transações realizadas e da ocorrência dos eventos.

O tempo pode ser associado a cada valor no banco de dados. O tempo de validade pode ser representado em um banco de dados temporal por um ponto no tempo e com variação por escada ou intervalo de tempo. O tempo de transação é representado por um *chronon* único.

Tabela 4.1 – Classificação de SGBD de acordo com a dimensão temporal (Fonte: adaptado de Snodgrass, 1992).

	Sem tempo de Transação	**Com tempo de Transação**
Sem tempo de Validade	Banco de dados estático	Banco de dados por tempo de transação
Com tempo de Validade	Banco de dados por tempo de validade	Banco de dados bitemporal

Worboys (1998) propôs um modelo conceitual para objetos espaço-temporais denominado *complexo espaço-bitemporal*. Este complexo é resultante da composição de classes primitivas de geo-objetos com os componentes temporais dos geo-objetos. O componente temporal é dado por uma estrutura, denominada *elemento bitemporal* (BTE), cuja semântica representa simultaneamente o geo-objeto de acordo com o tempo de validade do evento e com o tempo de transação do banco de dados (Figura 4.6). Um BTE é a união de um conjunto finito de produtos cartesianos (I_D x I_E) de intervalos de tempo de banco de dados (I_D) e de *chronons* de evento (I_E), assumindo que o domínio temporal contém elementos de $-\infty$ a $+\infty$, representando desde o passado indefinido a um futuro incerto (Worboys, 1994).

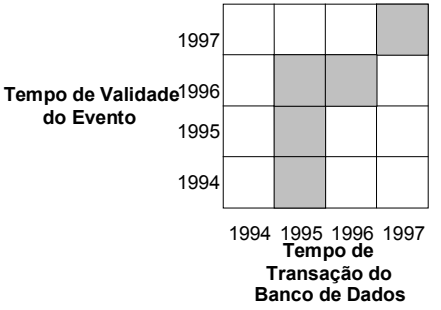


Figura 4.6 – Elemento Bitemporal (Fonte: adaptado de Worboys e Duckham, 2004).

Por esse modelo, geo-objetos são decompostos em representações geométricas espaciais primitivas, como pontos e linhas, associadas a um

elemento bitemporal. Assim, um complexo espaço-bitemporal pode ser representado através de duas dimensões espaciais e duas dimensões temporais, como mostra a Figura 4.7. Na Figura 4.7, em 1995 todos os elementos primitivos espaciais pertenciam ao banco de dados e eram eventos válidos. O ponto P2 era valido em 1994 e foi registrado em 1995. Em 1994 teve o registro do ponto P3 e das linhas L2 e L3 no banco de dados, e estes eram um evento válido em 1994 e 1995. Foi registrado no banco de dados em 1995 o ponto P2, válido em 1994 e 1995.

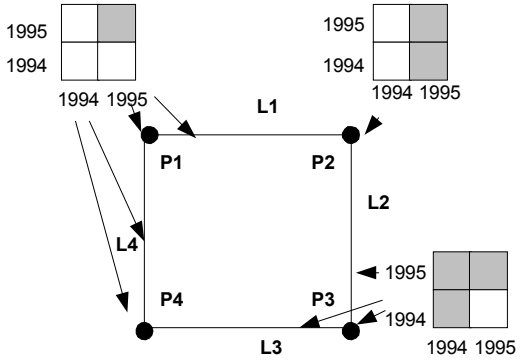


Figura 4.7 – Complexo espaço-bitemporal (Fonte: adaptado de Worboys e Duckham, 2004).

Um complexo espaço-bitemporal possibilita as operações *Lifetime*, *Max-S-Project e Min-S-Project*. A operação *Lifetime* projeta um objeto espaço-temporal em um período temporal ou bitemporal no qual ele possua existência. A operação *Max-S-Project* projeta o objeto espaço-temporal sobre toda a sua extensão espacial ou sobre todas as partes do objeto que já possuíram existência em algum intervalo de tempo. A *Min-S-Project* projeta o objeto espaço-temporal em sua maior extensão comum, que corresponde às partes que possuíram existência em todo o tempo de vida do objeto (Worboys, 1998).

As consultas temporais utilizam lógica temporal para recuperar os valores (tempo de transação e/ou de validade). Nas consultas sobre um elemento temporal, o espaço é visto como um conjunto de identificadores, correspondendo a pesquisas em banco de dados

temporais. Muitas consultas demandam recuperações históricas complexas. A seleção pode ser feita sobre dados, espaço, valores temporais ou ambos. O grande desafio dos pesquisadores na área de recuperação de informação espaço-temporal é explorar a consistência e completeza das propostas existentes e buscar soluções em relação à integridade espaço-temporal, consultas de multi-versões ou de versões históricas longas e complexas (Cheylan, 2001).

4.3.2 Relacionamentos espaço-temporais

A representação do tempo e do espaço implica na combinação de relacionamentos temporais e espaciais em uma estrutura integrada (Claramunt e Juang, 2000). Os relacionamentos entre intervalos temporais utilizam operadores booleanos de igualdade e desigualdade de instantes, de maneira semelhante à definição das matrizes de 4 e 9 interseções usadas em relacionamentos espaciais (Egenhofer e Franzoza, 1991). As relações entre intervalos de tempo implicam na definição de um operador de precedência, associado a um conjunto de operadores típicos da teoria de conjuntos, tais como *união, interseção, inclusão* e *igualdade*. Allen (1983) definiu sete relações (Figura 4.8): *before* (antes de), *meets* (toca), *during* (durante), *finishes* (finaliza junto com), *equal* (igual a), *overlaps* (sobrepõe) e *starts* (inicializa junto com). Claramunt e Juang (2000) apresentaram 56 possíveis relacionamentos espaço-temporais através da combinação desses sete relacionamentos temporais e dos 8 relacionamentos espaciais derivados da matriz de 4 interseções.

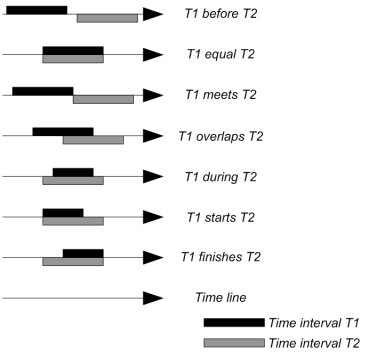


Figura 4.8 – Predicados temporais (Fonte: adaptado de Allen, 1993).

4.4 Identidade, vida e evolução de geo-objetos

4.4.1 Identidade de geo-objetos

A *identidade* é uma característica imutável de um geo-objeto. Ela é fundamental para a representação espaço-temporal, pois possibilita a distinção entre geo-objetos independentemente de sua estrutura, valores e atributos, incluindo aí atributos chave. Por exemplo, a identidade de uma pessoa não muda se o número de sua carteira de motorista for alterado. Note a diferença entre o conceito de identidade e o de identificador, ou de chave primária, usual em bancos de dados, que é empregado no sentido de "atributo cujo valor não se repete". Consideramos aqui que o valor de qualquer atributo pode mudar, dentro das regras semânticas que levaram à sua definição durante a modelagem; por outro lado, a mudança da identidade de um objeto só ocorre em situações particulares, em que se pode afirmar que a *natureza* do objeto mudou o suficiente para que consideremos seu novo estado como sendo um objeto inteiramente diferente. Por exemplo, quando um lote urbano é dividido em dois, uma das partes pode reter a identidade do lote original (no caso, apenas um dos seus atributos mudou: a forma geométrica). A outra parte constitui um lote novo, e que portanto recebe uma nova identidade.

A existência de um geo-objeto está associada à manutenção da sua identidade ao longo do tempo. As premissas das propostas para se implementar mudanças baseadas em identidade de geo-objetos são os critérios de *imutabilidade*, *reusabilidade* e *singularidade da identidade*. O registro das mudanças que ocorrem em geo-objetos estão fundamentadas pelos conceitos de orientação por objetos e de bancos de dados temporais. Essas propostas se baseiam na ordem temporal de estados de identidade e o vínculo temporal com os predecessores ou com o geo-objeto original (Al-Taha e Barrera, 1994).

4.4.2 Vida e evolução de geo-objetos

Segundo Frank et al. (2001) as mudanças de geo-objetos podem ser de *vida*, *genealogia* e *movimento*.

A noção de *vida* corresponde ao conjunto das mudanças de características de um geo-objeto durante sua existência, caracterizada pela sua identidade. Um lote, por exemplo, pode ter alterados o seu proprietário ou sua zona de uso do solo, sem que se torne um novo lote. Essas mudanças, portanto, fazem parte do registro de eventos que ocorreram ao longo da vida do lote.

A *genealogia* corresponde a um *link* temporal para o gerenciamento de sucessivas versões temporais de um objeto, como no caso em que se usa um rótulo temporal (*timestamp*). É similar a uma árvore genealógica familiar, no sentido em que um geo-objeto pode dar origem a outro, permanecendo ligado a ele como seu predecessor.

O *movimento* contempla mudanças de expansão ou contração, deformação e localização, como as que ocorrem na ampliação ou redução de subdivisões administrativas, no deslocamento simultâneo ao derretimento de um *iceberg*, ou no caso de objetos que se deslocam constantemente, como veículos.

Cheylan (2001) propôs quatro classificações para as mudanças espaciais elementares. A primeira classificação é a dos geo-objetos*permanentes*. Eles recebem esse nome por que permanecem inalteradas a sua forma e localização e podem ser alteradas as suas características não-espaciais. A segunda classificação é a dos *geo-objetos variáveis*. Seu tamanho e topologia podem variar sem gerar novos geo-objetos. Neste caso, são mantidas a forma e a identidade dos geo-objetos e são geradas múltiplas versões da mesma unidade espacial no tempo. A terceira classificação é a dos *geo-objetos modificáveis. Sua modificação ocorre* através da recomposição de um determinado espaço, adotando operações de partição dinâmica (divisão e junção). Os objetos podem mudar a forma, topologia e gerar novos objetos. Isso ocorre com os lotes urbanos e seus limites dentro de uma mesma quadra (Figura 4.9).

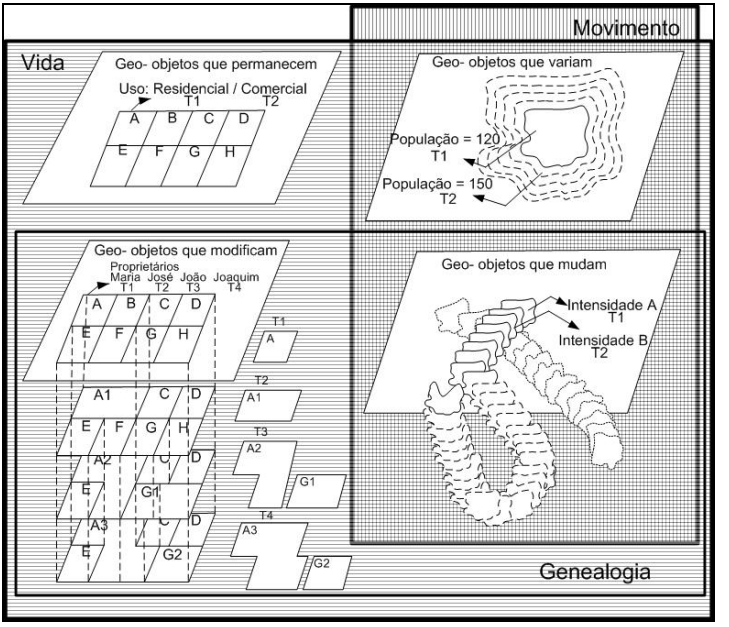


Figura 4.9 – Situações de mudanças espaciais no tempo (Fonte: adaptada de Cheylan, 2001).

A quarta classificação é a dos geo-objetos *que mudam* ou dinâmicos. Eles admitem mudanças na forma, topologia, atributo, localização e podem gerar novos objetos.

O conceito de vida é válido para todas as situações de mudanças, mas suficiente somente para o tipo de mudança de geo-objetos permanentes. A noção de movimento afeta os geo-objetos que variam e mudam, mas é suficiente somente para os que variam. A noção genealógica afeta os geo-objetos que modificam e mudam, mas é suficiente somente para os que modificam (Cheylan, 2001).

Algumas regras de mudanças durante a vida ou existência de um geo-objeto, são determinadas pelas operações que definem como ele adquire, muda ou perde a identidade ao longo do tempo (Figura 4.10). Essas operações são denominadas *construtores temporais*, e podem ocorrer em um geo-objeto ou entre geo-objetos diferentes (Medak, 1999).

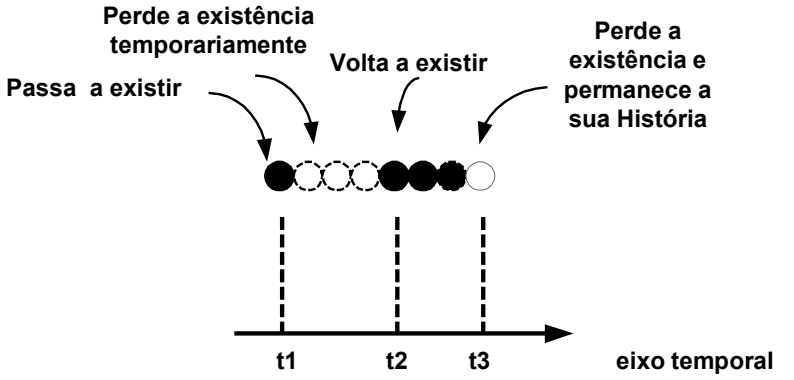


Figura 4.10 – Linha da vida - Lifeline (Fonte: adaptada de Medak, 2001).

Uma mudança pode ser *incremental* ou *contínua*. Em uma escala microscópica, todo geo-objeto sofre mudanças constantemente. Mas, em uma escala macroscópica, é importante uma distinção entre duas situações conceituais: (a) geo-objetos cujas propriedades podem ser consideradas estáveis, durante um certo tempo, até que ocorre uma mudança instantânea (por exemplo, quando um lote muda de proprietário); (b) geo-objetos cujas propriedades mudam constantemente, porém as representações são apenas capazes de capturar instantâneos temporais (*snapshots*) (por exemplo, no registro da temperatura do ar em uma estação meteorológica). No primeiro caso, considera-se que o objeto é *estável entre mudanças* e, no segundo, o objeto está *sujeito a mudanças contínuas*.

Quanto à sua existência diante da ocorrência de mudanças, os geo-objetos podem ser classificados como *continuants* e *ocurrents*. Objetos *continuant* são capazes de permanecer com a mesma identidade e de existir por um longo tempo, como um lote, um planeta ou um animal. Por outro lado, objetos *ocurrent* acontecem em um determinado tempo e têm curta duração, como, por exemplo, um *evento* da picada de um mosquito, uma *ação* de remembramento de lotes, um *processo* de deslizamento de terra ou a *atividade* de cortar uma árvore (Worboys, 2005).

Geo-objetos associados a partições definidas por convenções humanas, como no caso de informações sobre a realidade de um sistema de cadastro

urbano, demandam consultas complexas, do tipo "selecionar os lotes públicos que foram criados pela junção de lotes particulares e que perderam parte de sua área para formar logradouro(s)". Por esta razão, alguns pesquisadores têm dedicado especial atenção à elaboração de construtores capazes de modelar e gerenciar relacionamentos temporais entre objetos, considerando sua identidade. (Al-Taha e Barrera, 1994) (Hornsby e Engenhofer, 1997) (Hornsby e Engenhofer, 1998) (Hornsby e Engenhofer, 2000).

A evolução de geo-objetos somente pode ser recuperada pela identidade de um geo-objeto ou sua localização espacial. A identidade de um geo-objeto tem sido apresentada como uma parte da semântica associada aos processos de mudanças espaço-temporais. Apresentamos, a seguir, as operações propostas considerando a identidade de geo-objetos.

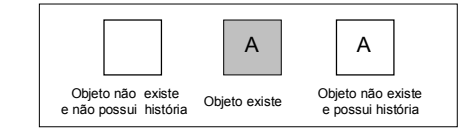
Pode-se recuperar toda a *evolução* histórica de um geo-objeto através de sua identidade, desde que haja uma conexão genealógica entre objetos. Alternativamente, pode-se determinar as mudanças ocorridas em um determinado espaço conhecendo-se apenas sua localização. É fundamental poder determinar, em qualquer situação, que mudanças permitem que o geo-objeto permaneça o mesmo (i.e., mantenha a sua identidade apesar das mudanças) e que mudanças fazem com que o objeto evolua para outro(s), deixando de existir com aquela identidade.

4.4.3 Operações de mudança baseadas na identidade

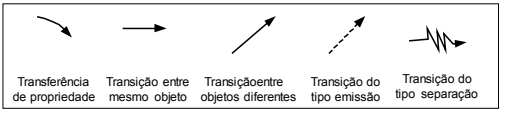
Para demonstrar as possíveis mudanças de estado relativas à identidade de objetos, Hornsby e Egenhofer (1997, 1998, 2000) desenvolveram uma linguagem visual pictórica, denominada CDL (*Change Description Language*). A CDL permite descrever cenários através da seqüência de transições da identidade de objetos ao longo do tempo. Trata-se de um modelo qualitativo, baseado na seqüência temporal dos eventos, e que possui extensões para representar relacionamentos espaciais, associações entre geo-objetos e propriedades de geo-objetos.

A Figura 4.11. apresenta algumas primitivas da CDL. Existem primitivas que indicam que um geo-objeto pode nunca ter existido, existe no tempo atual ou existiu em algum tempo anterior, mas não possui

existência no tempo atual. O geo-objeto pode possuir um vinculo temporal com a sua história ou não. A primitiva de *transferência de propriedade* corresponde à cópia de propriedades e à transição de estados em mesmo geo-objeto ou entre geo-objetos diferentes. A transmissão do tipo *emissão* corresponde à substituição de um geo-objeto por outro do mesmo tipo, mantendo as propriedades, e a transição do tipo *separação* subentende um cenário de divisão da geometria.



(a)



(b)

Figura 4.11 – a) Primitivas de estados da identidade de geo-objetos e b)Primitivas de tipos de transição (Fonte: adaptada de Hornsby e Egenhofer (1997, 2000)).

A CDL é usada aqui para facilitar o entendimento das semelhanças e diferenças existentes entre as diversas propostas de operações de mudança baseadas na identidade descritas na literatura (Clifford e Croker, 1988) (Chu et al., 1992) (Al-Taha e Barrera, 1994) (Hornsby e Egenhofer, 1997, 1998 e 2000) (Medak, 1999 e 2001) (Al-Taha, 2001).

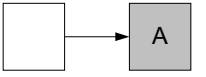
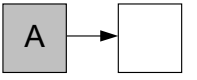
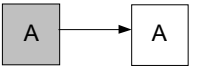
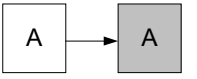
As propriedades de um geo-objeto podem ser transmitidas para outro geo-objeto já existente ou criado especificamente para recebê-las. No último caso, o geo-objeto origem é em geral destruído, porém mantido como predecessor do novo geo-objeto. Um geo-objeto pode dar origem a um novo geo-objeto ou ser substituído por outro objeto. São criados *links* temporais entre os geo-objetos origem e os seus sucessores.

As primeiras operações de identidade definidas foram *create* e *destroy*. A operação *create* cria o vínculo temporal (predecessores) com o(s) objeto(s) origem, sendo que somente um geo-objeto que nunca existiu

pode não possuir predecessores. A operação *destroy* elimina o geo-objeto e a sua história.

Clifford e Croker (1988) acrescentaram as operações *kill* e *reincarnate*. A operação *kill* suspende a existência de um geo-objeto temporariamente, sem destruir a sua identidade, e a operação *reincarnate* "ressuscita" um geo-objeto suspenso. Essas operações são apresentadas na Tabela 4.2, com os nomes das operações por autor e a sua representação em CDL.

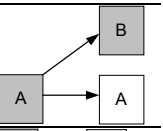
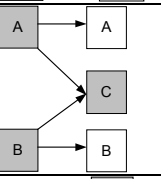
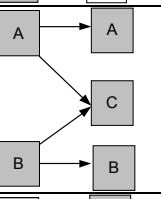
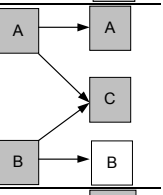
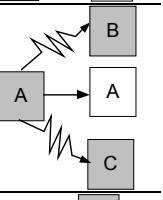
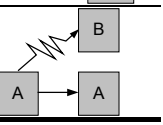
Tabela 4.2 – Operadores básicos de identidade de geo-objetos

Al-taha e Barrera (1994)	*Hornsby e Engenhofer (1997)*	*Medak (1999)*	*CDL (2000)*
create	*create*	*create*	
destroy	*destroy*	*remove*	
kill	*eliminate*	*destroy*	
reincarnate	*reincarnate*	*resume*	

Chu et al. (1992) propuseram as operações *evolution*, *fission e fusion* (Tabela 4.3). A operação *evolution* substitui a identidade atual por uma nova identidade. A *fission* cria novas identidades a partir de um geo-objeto existente com o sentido de subdivisão. A *fusion* cria uma nova identidade a partir de outras, com o sentido de fusão. Essas operações destroem todos os geo-objetos origem.

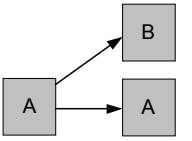
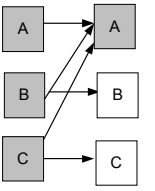
Hornsby e Engenhofer (1997 e 1998) incluíram as operações *generate*, *mix* e *splinter* (Tabela 4.3). As operações *generate* e *mix* possuem a semântica da operação *fusion*, porém na operação generate todos os geo-objetos origem são preservados. Na operação *mix*, nem todos são destruídos. A operação *splinter* possui a mesma semântica da *fission*, porém mantém a existência do geo-objeto origem.

Tabela 4.3 – Operadores de identidade de geo-objetos

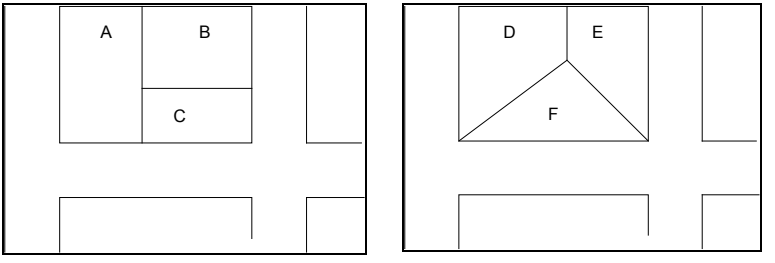
Al-taha e Barrera (1994)	*Hornsby e Engenhofer (1997)*	*Medak (1999)*	*CDL (2000)*
evolve	*metamorphose*	*evolve*	
fuse	*merge*	*fusion*	
	generate		
	mix		
fission	*divide*	*fission*	
	splinter		

Al-Taha e Barrera (1994) acrescentaram ainda as operações s*pawn e identify* (Tabela 4.4). A operação *spawn* gera novas identidades a partir de um geo-objeto existente, e só difere da operação *evolution* porque mantém o geo-objeto origem. A operação *identify* funde um conjunto de geo-objetos em um dos geo-objetos do conjunto, destruindo os demais (Al-Taha, 2001).

Tabela 4.4 – Operadores de identidade Spawn e Identify

Al-taha e Barrera (1994)	*Hornsby e Engenhofer (1997)*	*Medak (1999)*	*CDL (2000)*
spawn	*spawn*		
identify			

Medak (1999) formalizou uma proposta de 14 operadores de identidade que contemplam as propostas de Hornsby e Engenhofer (1997, 1998) e de Al-Taha e Barrera (1994), acrescentando a operação *restructure*. A operação *restructure* (Figura 4.12) foi criada para processos de reestruturação que envolvem vários objetos. São criados novos objetos através de um processo de redivisão interna de um espaço. Essa operação não mantém a ligação de cada objeto origem com o seu sucessor. Um exemplo dessa operação é uma redivisão de lotes em uma quadra.



(a) (b)

Figura 4.12 – Unificação dos lotes A, B, C originais (a) através da operação *reestruct* gerando os Lotes D, F, E.

Medak (1999) concluiu que as operações básicas para a mudança de um simples geo-objeto são as operações relacionadas a sua existência

(Tabela 4.2): *create, resume, suspend* e *destroy*. Pois, com essas operações pode-se obter as outras operações propostas.

As propostas de operação baseada na identidade de geo-objetos evoluíram muito em termos da identidade do geo-objeto, permitindo que (a) um geo-objeto seja suspenso para ser recuperado no futuro, no sentido de retormar a existência do geo-objeto; (b) existam vínculos temporais entre identidades de geo-objetos para composição da história do geo-objeto; (c) as operações propostas sejam obtidas através da composição de um conjunto de operações básicas. No entanto, mesmo tendo sido desenvolvidas considerando objetos espaciais, essas propostas não são detalhadas o bastante para considerar o efeito das operações sobre as propriedades espaciais e não espaciais dos objetos e os seus relacionamentos.

4.5 Exemplo de modelo espaço-temporal: cadastro urbano

Um sistema de informação cadastral é geralmente, uma ferramenta de responsabilidade pública. Ele é essencial para a administração das cidades e para os planejamentos urbanos e sociais, envolvendo áreas responsáveis pelas informações legais, sociais, administrativas, econômicas e de meio ambiente (Al-Taha, 2001).

Apresentamos, a seguir, um exemplo baseado na realidade do Cadastro Técnico Municipal (CTM) do município de Belo Horizonte. Esse exemplo refere-se a um projeto de implantação de uma linha de metrô de superfície, hoje existente, sendo que as datas apresentadas são fictícias. Os geo-objetos considerados são: *linha de metrô* (para representar a área de um segmento da linha de metrô), *lote, quadra, logradouro* (representando trecho de logradouro) e *área remanescente*. Áreas remanescentes, para o CTM, são sobras de áreas oriundas de um processo de desapropriação. Elas são remanescentes por não possuírem área suficiente e nem as características necessárias, determinadas pela lei de regulamentação do solo, para serem definidas como um lote urbano. Adotamos também o *polígono* como representação espacial e o *ano* para a representação da granularidade temporal.

A Figura 4.13 mostra o impacto da mudança na realidade da cidade. Essa mudança pode ser observada através da representação vetorial dos geo-objetos antes da implantação da linha de metrô (Figura 4.13 a) e da imagem após a sua implantação (Figura 4.13 b). A imagem mostra que a área interna aos limites da linha de metrô passou por grandes mudanças: foram demolidos trechos de logradouros e edificações; deixaram de existir lotes, quadras e logradouros; e, surgiram a área do metrô e a linha de metrô.

(a) (b)

Figura 4.13 – Área afetada pela linha de metrô antes (a) e depois da sua implantação (b) (Fonte: cadastro CTM da Prodabel).

4.5.1 Elemento temporal

Demonstramos a seguir a importância de ter o elemento bitemporal associado à representação espacial para informações urbanas. Foi registrado no banco de dados do CTM, em 2/5/1996, que a previsão de implantação da linha de metrô seria em 20/3/1998. Depois, em 1/10/1998, registrou-se que a linha de metrô foi implantada em 3/9/1998. Mas, em 7/12/1998 descobriu-se que o dado de implantação estava incorreto, pois a implantação tinha ocorrido em 3/5/1998 (Tabela 4.5).

Tabela 4.5 – Tempo bitemporal.

Tempo do BD (t_D)	Tempo de Evento (t_E)	Significado
2/5/1996	20/3/1998	Implantação prevista
1/10/1998	3/9/1998	Implantação
7/12/1998	3/5/1998	Correção Implantação

Para o planejamento urbano é importante saber qual era o tempo previsto para implantação da linha de metrô e quando ele foi realmente implantado (tempo de evento). E, além disso, é extremamente relevante identificar o período em que a informação de implantação do projeto esteve errada no banco de dados (1/10/1998 a 7/12/1998), tempo de transação. Pois, durante esse período, a informação disponível para apoiar à tomada de decisão estava incorreta. O elemento bitemporal possibilita essa distinção, porém ainda é objeto de pesquisa a possibilidade da alteração do tempo de transação para corrigir a história passada.

4.5.2 Mudanças espaço-temporais

Na Figura 4.14 ilustramos algumas mudanças no parcelamento do solo, durante o período de 1994 a 2000. Em 1995, ocorreu uma ação de remembramento dos lotes LA e LB, que originou (criou) o lote LD. Um remembramento pode ocorrer por uma solicitação do proprietário quando adquire lotes adjacentes. Neste caso, é realizada uma operação de junção da geometria. Em 1996, criou-se um projeto para implantação de uma linha de metrô de superfície nessa área. A linha de metrô (LT1) do projeto foi implantada (criada) em 1998. E, ao mesmo tempo, os geo-objetos que deram origem a essa linha de metrô perderam a sua existência. Esses geo-objetos foram: o trecho de logradouro R1, o lote LE e parte dos lotes LC e LF. Também, devido a mesma ação, surgiram as áreas remanescentes A1 e A2 com a área restante do logradouro R1 e do lote LF. Alem disso, foi alterada a área e geometria do lote LC. A seguir, em 2000, uma outra ação de remembramento do lote LD com a área A1 foi realizada. Essa ação deu origem ao lote LG.

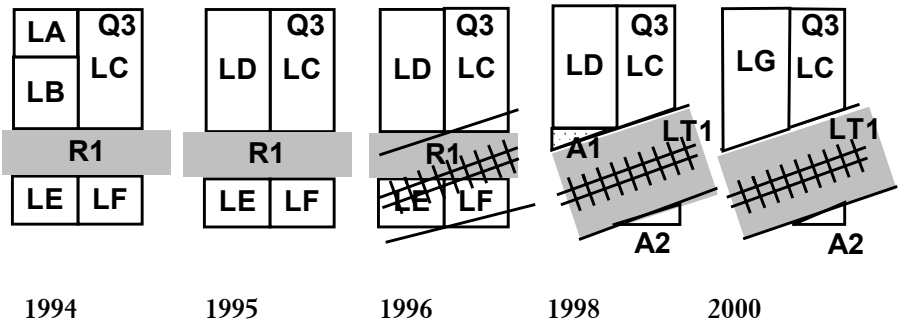


Figura 4.14 – Situações de mudanças espaciais no tempo pela implantação da linha de metrô.

A Figura 4.15 apresenta a CDL do exemplo com as respectivas operações realizadas e os seus objetos resultantes. Alguns objetos resultantes são intermediários (como o LH) por isso não aparecem na Figura 4.14. Eles são gerados para determinar uma melhor composição da história desses geo-objetos.

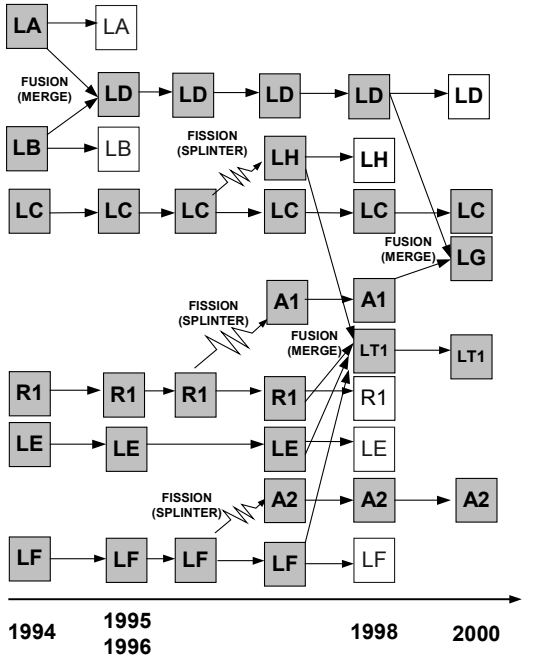


Figura 4.15 – CDL das operações baseadas na identidade realizadas para implantação da linha de metrô.

Sobre a situação apresentada na Figura 4.14, podemos imaginar as seguintes consultas relacionadas à vida e evolução dos geo-objetos:

1. Quais eram as configurações espaciais dos lotes da quadra Q3 em 1994?
2. Quais eram as configurações espaciais dos lotes da quadra Q3 utilizadas na cobrança do IPTU entre 1996 e 2000?
3. Quais foram todas as configurações espaciais do lote LC no período de 1994 até hoje?
4. Quais foram os lotes que ficaram sem água das 8:00 da manhã do dia 12 de janeiro até às 20:00 do dia 15 de janeiro de 1998?
5. Quais foram cronologicamente os proprietários do lote LD?
6. Quais foram às sucessivas taxas anuais de IPTU cobradas do lote LC?

7. Quais foram os imóveis que foram desapropriados entre 1998 e 2000?
8. Quais são as quadras que possuem lotes com áreas alteradas pela implantação da linha de metrô LT1?
9. Quando a pavimentação do logradouro R1 correspondeu a um percentual maior que 40%?
10. Quais os períodos de tempo em que o lote LD não obteve o beneficio de iluminação pública no ano de 1998?
11. Quais ações geraram mudanças na quadra Q3 entre 1994 e 2000?
12. Quais foram os lotes que deixaram de existir no período de 1996 a 2000?
13. Quais lotes foram criados entre 1994 e 1995?
14. Quais geo-objetos deram origem a linha de metrô?
15. Como essas ações foram realizadas? Quais foram os objetos origem dessa ação e os objetos resultantes? Como as alterações foram realizadas nos objetos? Quais foram as operações utilizadas para gerar a alteração? Qual a lógica utilizada pela operação?

Essas consultas buscam recuperar o estado de um geo-objeto, a História de uma região espacial específica ou de geo-objetos e de um elemento temporal. Como também, informações sobre as ações que promoveram as mudanças.

Esse exemplo mostra que ainda são necessários avanços nas pesquisas para capturar todos os aspectos da dinâmica de geo-objetos espaço-temporais. Uma ontologia de mudança deve responder a perguntas como (1) *que mudanças ocorreram em cada objeto*? (2) *quando essas mudanças ocorreram*?, (3) c*omo as mudanças foram geradas*?, (4) *quais regras determinaram essas mudanças*? e (5) *quais foram as causas da mudança*? Criar modelos para suportar ontologias de mudanças é essencial para que se possam obter informações dessas mudanças e dos geo-objetos envolvidos (Dias et al., 2004).

4.6 Exemplo de modelo espaço-temporal: queimadas na Amazônia

No Brasil, a queima de biomassa vegetal constitui uma prática de manejo, principalmente, para a criação de gado e a expansão da fronteira agrícola. As queimadas estão amplamente inseridas no processo produtivo da Amazônia e do Cerrado brasileiro e são fatores que impulsionam a expansão agropecuária nestas regiões. As queimadas ocorrem todo ano durante a estação seca, sendo no final deste período em que ocorre a maior incidência.

A ocorrência de queimadas acarreta inúmeros impactos ambientais, por isso a modelagem de um banco de dados geográficos que contemple o aspecto temporal de questões relacionadas a desmatamentos e queimadas é de grande importância para a análise e combate a estes problemas. Essa seção apresenta como exemplo, uma proposta de modelagem de um banco de dados espaço-temporal para dados de queimadas. A descrição completa desse trabalho pode ser encontrada em Correia et al. (2004).

A modelagem espaço-temporal das queimadas foi feita considerando-se os focos de calor como eventos isolados que podem ou não ser conectados espacial e temporalmente, formando assim queimadas com variação espaço-temporal. Os focos de calor são obtidos através de imagens de sensoriamento remoto. Para tanto, cada *pixel* da imagem onde foram identificados os focos de calor, é representado por um ponto com uma região contígua ao seu redor representando o tamanho nominal dos *pixels* da imagem (Figura 4.16). Com isso, é possível acompanhar a evolução dos focos de calor e de queimadas. Ao se realizar uma consulta, pode-se tanto analisar focos de calor quanto queimadas.

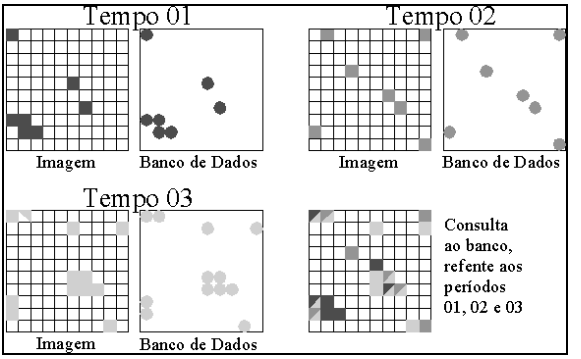


Figura 4.16 – Modelagem espaço-temporal de focos de calor e queimadas.

Além dos dados de focos de calor, o banco de dados foi projetado de forma a armazenar dados de uma base cartográfica, dados de legislação, cadastro de imóveis e dados de fiscalização.

O processo de desmatamento é representado por polígonos que representam talhões de áreas desmatadas e que, normalmente, evoluem ao longo do tempo em um processo incremental. Para inclusão do processo de desmatamento foi definido um conjunto de operações sobre as geometrias de entrada do sistema, para possibilitar a construção dos relacionamentos topológicos e temporais entre os geo-objetos, usando a abordagem descrita em Dias et al. (2004) (junção, separação e destruição).

Todos os dados foram incorporados a um SIG de arquitetura em camadas, ou seja, onde os dados descritivos e espaciais são guardados em tabelas relacionais armazenadas em um SGBD objeto-relacional. Sobre esse modelo lógico foi possível implementar consultas espaço-temporais de interesse como : "*Quais são as áreas desmatadas no período de 01-06-2004 a 09-06-2004 e posteriormente queimadas em 17-06-2004*?" ou "*As queimadas agrícolas foram mais intensas do que as queimadas florestais no período de 01-06-2004 a 09-06-2004 no município 9*?". A Tabela 4.6 mostra os passos necessários para a resposta à primeira pergunta usando comandos SQL – Structured Query Language.

Tabela 4.6 – Operações espaciais para obtenção das áreas desmatadas entre 01 a 09-06-2004 e posteriormente queimadas em 17-06-2004

Etapas	Comando SQL
Seleção dos polígonos das áreas desmatadas entre 01-06-2004 e 09-06-2004	`Select object_id, spatial_data into temporary desmatamento from te_polygons_desmatamento where geom_id=(select geom_id from temp_desmatamento where initial_time >= '2004-06-01' and initial_time <= '2004-06-09');`
Seleção dos pontos de queimadas em 17-06-2004	`Select object_id, spatial_data into temporary queimadas from te_points_focos where object_id=(select object_id from att_focos where initial_time = '2004-06-17');`
Seleção das áreas desmatadas e depois queimadas	`Select t1.object_id, t2.object_id from queimadas t1, desmatamento t2 where within(t1.spatial_data, t2.spatial_data);`

Essas consultas espaço-temporais preliminares forneceram os resultados esperados, sinalizando para uma modelagem adequada, o que possibilitou a integração de dados históricos de desmatamentos e queimadas. Mas faz-se necessário a ampliação das consultas espaço-temporais, com o objetivo de avaliar, mais detalhadamente, o modelo lógico implementado nesse exemplo.

4.7 Leituras suplementares

Como visto neste capítulo, o uso de modelos espaço-temporais é necessário para uma grande variedade de aplicações de geoinformação. Devido à complexidade envolvida na representação combinada de espaço e tempo, estes modelos ainda estão em desenvolvimento, e trata-se de uma área de pesquisa extremamente ativa. Os principais temas de pesquisa incluem: *ontologias de mudança de geo-objetos, modelos espaciais dinâmicos, e modelos para objetos móveis*.

A questão de ontologias de mudança é abordada nos artigos de Frank (2003), Grenon e Smith (2003), Worboys e Hornsby (2004), Worboys

(2005) e Dias et al (2004). O tema específico de identidade de geo-objetos é discutido em Medak (1999) e Hornsby e Egenhofer (2000). A questão de ações e atividades é discutida em Kuhn (2001). A questão de modelagem dinâmica espacial, não discutida em detalhe neste capítulo, é apresentada em Burrough (1998) e Couclelis (1985). Exemplos de aplicações são apresentados em Almeida et al. (2003) e Pedrosa et al. (2002). Os problemas de modelagem de objetos móveis são discutidos no capítulo 5 do livro e revisados no artigo de Guting et al. (2003).

Referências

ALLEN, J. F. Maintaining Knowledge about Temporal Intervals. **ACM Communications**, 26, pp. 832-843, 1983.

ALMEIDA, C. M. D.; MONTEIRO, A. M. V.; CAMARA, G.; SOARES-FILHO, B. S.; CERQUEIRA, G. C.; PENNACHIN, C. L.; BATTY, M. Empiricism and Stochastics in Cellular Automaton Modeling of Urban Land Use Dynamics. **Computers, Environment and Urban Systems**, v. 27, n.5, p. 481-509, 2003.

Al-TAHA, K., 2001. Identities through time. In: FRANK, A. U.; RAPER, J. e CHEYLAN, J. eds., **Life and Motion of Socio-Economic Units**, London: Taylor & Francis.

AL-TAHA, K; BARRERA, R., 1994. Identities through time. In: International Workshop on Requirements for Integrated Geographic Information Systems, New Orleans, Lousiana, pp. 1-12.

BURROUGH, P., 1998. Dynamic Modelling and Geocomputation. In: LONGLEY, P.; BROOKS, S.; MCDONNELL, R.; MACMILLAN, B., eds., **Geocomputation: A Primer**: New York, John Wiley.

CHEYLAN, J., 2001. Time, actuality, novelty and history, In: FRANK, A. U.; RAPER, J. e CHEYLAN, J. eds., **Life and Motion of Socio-Economic Units**, London: Taylor & Francis.

CHU, W. W.; IEONG, I. T.; TAIRA, R. K.; BREANT, C. M., 1992. A temporal evolutionary object-oriented data model for medical image management. Proceedings of the Fifth Annual IEEE 5th Symposium on Computer-Based Medical Systems, Durham, North Carolina.

CLARAMUNT, C.; JIANG, B. A representation of relationships in temporal spaces. In: ATKINSON, P. e MARTIN, D. eds., **Innovations in GIS VII: GeoComputation**, Taylor and Francis, pp. 41-53, 2000.

CLIFFORD, J.; CROKER, A. Objects in Time. **Database Engineering**, v.7, n.4, 189-196. 1988.

CORREIA, A. H.; PIROMAL, R. A. S.; QUEIROZ, G. R.; de SOUZA, R. C. M. Modelagem de um Banco De Dados Espaço Temporal para Desmatamentos e Queimadas. **Anais XII SBSR**, Goiânia, Brasil, 16-21 abril 2005, INPE, p. 2619-2627.

COUCLELIS, H. Cellular Worlds: A Framework for Modeling Micro-Macro Dynamics. **Environment and Planning A**, v. 17, p. 585-596, 1985.

DIAS, T.; CAMARA, G.; FONSECA, F.; DAVIS, C. Bottom-Up Development of Process-Based Ontologies. In: **GIScience 2004**, 2004,College Park, MA. AAG.

EDELWEISS, N.; OLIVEIRA, J. P. M , 1994. Modelagem de Aspectos Temporais de Sistemas de Informação. Recife, UFPE-DI. P.

EGENHOFER, M. J.; FRANZOSA, R. Point-Set Topological Spatial Relations. **International Journal of Geographical Information Systems**, v. 5, n.2, p. 161-174, 1991.

FONSECA, F.; DAVIS, C.; CAMARA, G. Bridging Ontologies and Conceptual Schemas in Geographic Applications Development. **Geoinformatica**, v. 7, n.4, p. 355-378, 2003.

FONSECA, F.; EGENHOFER, M.; AGOURIS, P.; CÂMARA, G. Using Ontologies for Integrated Geographic Information Systems. **Transactions in GIS**, v. 6, n.3, p. 231-257, 2002.

FRANK, A. U., 1997. Spatial ontology: A geographical information point of view. In STOCK, O. ed., **Spatial and Temporal Reasoning**, pages 135–153. Kluwer Academic Publishers, Dordrecht.

FRANK, A. U., 2003, Ontology for Spatio-temporal Databases. In: KOUBARAKIS, M.; SELLIS, T., eds., **Spatio-Temporal Databases: The Chorochronos Approach: Lecture Notes in Computer Science**: Berlin Heidelberg New York, Springer-Verlag, p. 9-78.

FRANK, A. U., RAPER, J., CHEYLAN, J. **Life and Motion of Socio-Economic Units**. ESF Series, London, Taylor & Francis, p. 353. 2001.

GRENON, P.; SMITH, B. SNAP and SPAN: Towards Dynamic Spatial Ontology, **Spatial Cognition and Computation**, v.4, n.1, p.69–104. 2003.

GÜTING, R. H.; BOHLEN, M. H.; ERWIG, M.; JENSEN, C. S.; LORENTZOS, N.; NARDELLI, E.; SCHNEIDER, M.; VIQUEIRA, J. R. R., 2003. Spatio-temporal Models and Languages: An Approach Based on Data Types. In: KOUBARAKIS, M., ed., **Spatio-Temporal Databases**: Berlin, Springer.

HORNSBY, K.; EGENHOFER, M. J., 1997. Qualitative Representation of Change. In: FRANK, A. U. e HIRTLE eds. **Spatial Information Theory - A Theoretical Basis for GIS (International Conference COSIT'97)**, Springer-Verlag, Berlin-Heidelberg, 15-33.

HORNSBY, K.; EGENHOFER, M. J., 1998. Identity-Based Change Operations for Composite Objects. In: POIKER, T. K. e CHRISMAN, N. eds. **Proceedings of 8th Internal Symposium on Spatial Data Handling**, July 11-15, Vancouver.

HORNSBY, K.; EGENHOFER, M. J., Identity-Based Change: A Foundation for Spatio-Temporal Knowledge Representation. **International Journal of Geographical Information Science**. v.14, n.3, p.207-224. 2000.

KAVOURAS, M., 2001. Understanding and Modelling Spatial Change. In: FRANK A. RAPER J. e CHEYLAN J.P. eds.: Life and Motion of Socio-Economic Units, Chapter 4. London: Taylor & Francis, GISDATA Series 8.

KUHN, W. Ontologies in support of activities in geographical space. **International Journal of Geographical Information Science**, v. 15, n.7, p. 613-631, 2001.

LANGRAN. G. **Time in Geographic Information Systems**. Washington, DC, Taylor & Francis, 1993.

MEDAK, D. Lifestyles - A New Paradigm in Spatio-Temporal Databases. Ph.D. Thesis. Technical University of Vienna, 1999.

MEDAK, D., 2001. Lifestyles. In: FRANK, A. U.; RAPER, J. e CHEYLAN, J. eds., **Life and Motion of Socio-Economic Units**, London: Taylor & Francis, p.353.

PEDROSA, B.; CÂMARA, G.; FONSECA, F.; SOUZA, R. C. M. TerraML - A Cell-Based Modeling Language for an Open-Source GIS Library. In: **II International Conference on Geographical Information Science (GIScience 2002)**. Proceedings, AAG, 2002, Boulder, CO, 2002.

PEUQUET, D. J. Making Space for Time: Issues in Space-Time Data Representation. **GeoInformatica** v. 5, n. 1, p.11-32, 2001.

RENOLEN, A. Temporal Maps and Temporal Geographical Information Systems. Review of Research. Department of Surveying and Mapping (IKO) The Norwegian Institute of Technology. November, 1995 revised February 1997.

SMITH, B.; MARK, D. Ontology and Geographic Kinds. In: **International Symposium on Spatial Data Handling**. Vancouver, Canada, 1998. p. 308-320.

SNODGRASS, R. T., 1992. Temporal Databases. In: FRANK, A. U.; CAMPARI, I. e FORMENTINI, U. eds. **Theories and Methods of Spatio-Temporal Reasoning in Geographic Space**, Springer-Verlag, Heidelberg-Berlin. p. 22-64.

SNODGRASS, R. T. Temporal databases: Status and research directions. **SIGMOD RECORD**, v.19, n.4, p. 83–89, 1990.

WORBOYS, F. M. F. A unified model of spatial and temporal information, **Computer Journal**, v.37, n.1, p.26-34, 1994.

WORBOYS, F. M. F..; DUCKHAM, M. GIS: **A Computing Perspective**. CRC Press, Boca Raton, Florida, Taylor Francis Ltd. 1995, 376 p.

WORBOYS, M. F. Event-oriented approaches to geographic phenomena. accepted for publication in International Journal of Geographic Information Systems, 2005. Disponível em : http://www.spatial.maine.edu/~worboys/downloads.htm.

WORBOYS, M. F., 1998. A Generic Model for Spatio -Bitemporal Geographic Information. In: EGENHOFER, M. J., GOLLEDGE, E. R. G. eds. **Spatial and Temporal Reasoning in Geographics Information Systems**. New York, Oxford University Press: 25-39.

WORBOYS, M. F.; HORNSBY, K. From Objects to Events: GEM, the Geospatial Event Model. In: EGENHOFER, M. J.; FREKSA, C. e MILLER, H. J. eds. **Third International Conference, GIScience 2004**, 2004, Adelphi, MD, USA. Springer, p. 327-343.

5 *Arquiteturas e linguagens*

Karine Reis Ferreira
Marco Antonio Casanova
Gilberto Ribeiro de Queiroz
Olga Fradico de Oliveira

5.1 Introdução

Este capítulo inicialmente apresenta uma visão geral de sistemas de gerência de banco de dados (SGBDs) e resume os principais conceitos da linguagem SQL. Em seguida, aborda a evolução das arquiteturas de SIG. Por fim, apresenta os principais operadores espaciais e discute a definição de linguagens de consulta espacial.

Ao longo dos anos, as implementações de SIGs seguiram diferentes arquiteturas, distinguindo-se principalmente pela estratégia adotada para armazenar e recuperar dados espaciais. Mais recentemente, tais arquiteturas evoluíram para utilizar, cada vez mais, recursos de SGBDs.

Por seu lado, a pesquisa na área de Banco de Dados passou, já há algum tempo, a preocupar-se com o suporte a aplicações não convencionais (Schneider, 1997), incluindo as aplicações SIG. Uma aplicação é classificada como não convencional quando trabalha com outros tipos de dados, além dos tradicionais, como tipos de dados espaciais, temporais e espaço-temporais. Uma das vertentes de pesquisa tem sido exatamente a definição de linguagens de consulta para tratar tais tipos de dados.

5.2 Preliminares

5.2.1 Sistemas de gerência de banco de dados

Um sistema de gerência de banco de dados (SGBD) oferece serviços de armazenamento, consulta e atualização de bancos de dados. A Tabela 5.1 resume os requisitos mais importantes para SGBDs e a Tabela 5.2 lista as principais tecnologias desenvolvidas para atendê-los.

Tabela 5.1 – Principais requisitos para SGBDs

Requisito	*Definição*
Facilidade de uso	a modelagem do banco de dados deve refletir a realidade das aplicações, e o acesso aos dados deve ser feito de forma simples
Correção	os dados armazenados no banco de dados devem refletir um estado correto da realidade modelada
Facilidade de manutenção	alterações na forma de armazenamento dos dados devem afetar as aplicações o mínimo possível
Confiabilidade	atualizações não devem ser perdidas e não devem interferir umas com as outras
Segurança	o acesso aos dados deve ser controlado de acordo com os direitos definidos para cada aplicação ou usuário
Desempenho	o tempo de acesso aos dados deve ser compatível com a complexidade da consulta

Tabela 5.2 – Principais tecnologias para SGBDs

Requisito	Tecnologia
Facilidade de uso	linguagem de definição de dados e linguagem de consulta baseadas em modelo de dados de alto nível
Correção	restrições de integridade, *triggers* e assertivas
Facilidade de manutenção	especificação do banco de dados em níveis, isolando os detalhes de armazenamento das aplicações

Confiabilidade	transações atômicas implementadas através de mecanismos para controle de concorrência e mecanismos de recuperação em caso de falhas
Segurança	níveis de autorização e controle de acesso
Desempenho	otimização de consultas, baseada em métodos de acesso e de armazenamento eficientes, gerência eficaz do buffer pool e modelos de custo, entre outras tecnologias

O mercado para SGBDs concentra-se em duas tecnologias, SGBDs Relacionais (SGBD-R) e SGBDs Objeto-Relacionais (SGBD-OR), com uma pequena fatia para SGBDs Orientados-a-Objeto (SGBD-OO).

Os SGBD-R seguem o modelo relacional de dados, em que um banco de dados é organizado como uma coleção de relações, cada qual com atributos de um tipo específico. Nos sistemas comerciais atuais, os tipos incluem números inteiros, de ponto flutuante, cadeias de caracteres, datas e campos binários longos (BLOBs). Para esses tipos encontram-se disponíveis uma variedade de operações (exceto para o tipo BLOB), como operações aritméticas, de conversão, de manipulação textual e operações com data.

Os SGBD-R foram concebidos para atender as necessidades de aplicações manipulando grandes volumes de dados convencionais. De fato, tais sistemas não oferecem recursos para atender as necessidades de aplicações não convencionais. A mera simulação de tipos de dados não convencionais em um SGBD-R pode ter efeitos colaterais, como queda de desempenho, dificuldade de codificação e posterior manutenção da aplicação (Stonebraker, 1996).

Os SGBD-OR estendem o modelo relacional, entre outras características, com um sistema de tipos de dados rico e estendível, oferecendo operadores que podem ser utilizados na linguagem de consulta. Possibilitam ainda a extensão dos mecanismos de indexação sobre os novos tipos. Essas características reduzem os problemas ocorridos na simulação de tipos de dados pelos SGBD-R, tornando os SGBD-OR uma solução atrativa para aplicações não convencionais.

5.2.2 A linguagem SQL

A linguagem SQL (*Structured Query Language*) é adotada pela maioria dos SGBD-R e SGBD-OR comerciais. Desenvolvida inicialmente pela IBM na década de 70, a linguagem sofreu sucessivas extensões, culminando com a publicação do padrão conhecido por SQL:1999 (ver Tabela 5.3).

Tabela 5.3 – Breve histórico de SQL

Ano	Versão	Características
1974	SEQUEL	linguagem original, adotada no protótipo de mesmo nome, desenvolvido pela IBM
1976	SEQUEL 2	extensão de SEQUEL, adotada no *System R* da IBM
1986	SQL-86 (SQL1)	padrão publicado pela ANSI em 1986 ratificado pela ISO em 1987
1989	SQL-89	extensão do SQL-86
1992	SQL-92 (SQL2)	padrão publicado pela ANSI e pela ISO
1996	SQL-92 / PSM	extensão do SQL-92
2001	SQL:1999	padrão aprovado em 1999 pela ISO, resultado de 7 anos de trabalho, e publicado em maio de 2001

SQL é formada basicamente por duas sub-linguagens:

- Linguagem de definição de dados (SQL DDL): fornece comandos para definir e modificar esquemas de tabelas, remover tabelas, criar índices e definir restrições de integridade.
- Linguagem de manipulação de dados (SQL DML): fornece comandos para consultar, inserir, modificar e remover dados no banco de dados.

A Tabela 5.4 apresenta alguns comandos em SQL.

Tabela 5.4 – Comandos em SQL

Comando	*Descrição*	*Tipo*
select	Recupera dados de uma ou mais tabelas	DML
insert *update* *delete*	Servem para incluir, alterar e eliminar registros de uma tabela, respectivamente	DML
commit *roolback*	Responsáveis pelo controle de transações, permitem que o usuário desfaça (*rollback*) ou confirme (*commit*) alterações em tabelas	DML
create *alter* *drop*	Usados para definir, alterar e remover tabelas de um banco de dados	DDL

No contexto de SQL, usaremos os termos "tabela" em lugar de "relação", e "coluna" em lugar de "atributo", seguindo a prática mais comum. Os tipos de dados de SQL mais comumente utilizados são *char(n), int, float, date* e *time*. Estes representam, respectivamente, um conjunto de caracteres de tamanho definido, um número inteiro, um número real, uma data (composta por dia, mês e ano) e um instante de tempo (composto por horas, minutos e segundos).

Uma restrição especifica um critério de consistência para o banco de dados, podendo ser definida a nível de coluna ou a nível de tabela. A restrição de nulidade sobre uma coluna A de uma tabela T indica que nenhuma linha t de T pode ter o valor de A nulo.

Uma chave primária K (*primary key*) de uma tabela T é formada pelo nome de uma única coluna ou por uma lista de nomes de colunas de T. Indica que quaisquer duas linhas t e t' de T não podem ter valores idênticos de K, ou seja, $t[K] \neq t'[K]$.

Uma chave estrangeira F (*foreign key*) de uma tabela T para uma tabela U, com chave K, também é formada pelo nome de uma única coluna ou por uma lista de nome de colunas de T. Indica que, para cada linha t de T, deve haver uma linha u de U tal que $t[F] = u[K]$.

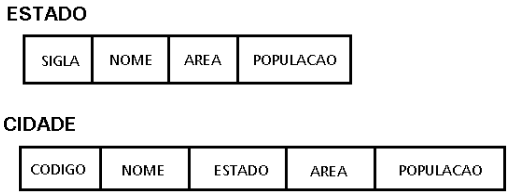


Figura 5.1 – Relações cidade e estado.

A seguir, apresentaremos alguns exemplos de SQL, baseados nas tabelas `cidade` e `estado`, ilustradas na Figura 5.1 e definidas como:

```
CREATE TABLE estado
(  sigla   CHAR(2)    NOT NULL,
   nome    CHAR(30),
   area    FLOAT,
   PRIMARY KEY(sigla) )

CREATE TABLE cidade
(  codigo  INT        NOT NULL,
   nome    CHAR(30)   NOT NULL,
   estado  CHAR(2),
   area    FLOAT,
   PRIMARY KEY(codigo),
   FOREIGN KEY(estado) REFERENCES estado(sigla) )
```

No exemplo anterior, `codigo` é a chave primária da tabela `cidade`, e `estado` de `cidade` atua como uma chave estrangeira referenciando a chave `sigla` da tabela `estado`. Assim, garantimos que somente os valores que ocorrem na coluna `sigla` de `estado` podem ser usadas como valores na coluna `estado` em `cidade`. Ainda, nessa tabela, as colunas `codigo` e `nome` não podem conter valores nulos.

Para alterar a definição de uma tabela, usamos o comando `ALTER TABLE`:

```
ALTER TABLE estado ADD região CHAR(20)
```

Para remover tanto a definição quanto os dados de uma tabela, usamos o comando `DROP TABLE`:

```
DROP TABLE cidade
```

Para remover apenas os dados, usamos o comando `DELETE`:

```
DELETE * FROM estado
```

Para inserir uma nova linha, usamos o comando `INSERT`:

```
INSERT INTO estado
VALUES ('MG', 'Minas Gerais', 10000000)
```

Para alterar dados já existentes, usamos o comando `UPDATE`:

```
UPDATE estado   SET area = 20000000
WHERE sigla = 'MG'
```

Para consultar os dados, usamos o comando `SELECT`, cuja sintaxe é resumida a seguir (ver a Tabela 5.5):

```
SELECT [distinct] {*, coluna [alias],
        expressões [alias], funções [alias], ...}
from {tabelas [alias],}
[where condição]
[group by colunas]
[having condição]
[order by colunas [asc | desc]]
```

onde os colchetes representam cláusulas opcionais, as chaves indicam que os elementos podem aparecer repetidas vezes e a barra vertical indica que as opções são mutuamente exclusivas (ou uma ou outra).

Tabela 5.5 – Sintaxe dos comandos

Opções	*Descrição*
`distinct`	Indica que as linhas duplicadas devem ser eliminadas do resultado da consulta.
*	Indica que todas as colunas de todas as tabelas da cláusula *from* devem ser incluídas nas linhas da resposta da consulta.
`coluna`	Coluna de uma tabela listada na cláusula *from* que deve ser incluída em cada linha da resposta da consulta.
`expressões`	Expressões aritméticas envolvendo uma ou mais colunas das tabelas listadas na cláusula *from*.
`funções`	Funções definas em SQL como, por exemplo, funções de agregação, que calculam estatísticas sobre colunas

	numéricas (*avg, min*, *max*, *count*, dentre outras).
`alias (select)`	Nome alternativo para uma coluna ou expressão, usado para melhorar a legibilidade do comando, ou para nomear diretamente uma coluna da resposta da consulta.
`tabelas`	Uma ou mais tabelas envolvidas na consulta.
`alias (from)`	Nome alternativo para uma tabela, usado para melhorar a legibilidade, ou para permitir o uso da mesma tabela mais de uma vez na cláusula *from*. Pode ser precedido por "*as*".
`condição (where)`	Especifica uma condição à qual as linhas das tabelas listadas na cláusula *from* devem satisfazer para gerar uma linha da resposta da consulta.
`group by colunas`	Indica que o resultado deve ser agrupado pelas colunas especificadas.
`condição (having)`	Limita os grupos a serem mostrados àqueles satisfazendo a condição.
`order by`	Ordenação que será aplicada ao resultado da consulta, podendo ser crescente (*asc*) ou decrescente (*desc*).

As consultas a seguir ilustram o uso do comando *select*:

Q1. Selecione todos os nomes dos estados.

```
SELECT nome
FROM estado
```

Q2. Selecione todos os nomes das cidades, sem repetição.

```
SELECT DISTINCT nome
FROM cidade
```

Q3. Recupere os nomes e as áreas das cidades que possuam a sigla de estado igual a "MG" e população maior que 50.000 habitantes.

```
SELECT nome, area
FROM cidade
WHERE estado = "MG" AND populacao > 50000
```

Q4. Retorne todas as cidades e o estado ao qual pertence.

```
SELECT C.nome, E.nome
```

```
FROM cidade AS C, estado AS E
WHERE C.estado = E.sigla
```

Q5. Calcule a média das áreas de todas as cidades.

```
SELECT AVG(area)
FROM cidade
```

Q6. Retorne o nome de cada estado e a soma das áreas de todas as cidades desse estado.

```
SELECT E.nome, SUM(C.area)
FROM estado AS E, cidade AS C
WHERE E.sigla = C.estado
GROUP BY E.nome
```

Além do comando *select*, os comandos *delete* e *update* podem usar a cláusula *where* como, por exemplo:

Q7. Remova as cidades cuja população é menor que 1000 habitantes.

```
DELETE FROM cidade
WHERE populacao < 1000
```

Q8. Altere a população do estado de Minas Gerais para 5000000.

```
UPDATE estado SET populacao = 500000
WHERE codigo = "MG"
```

5.3 Arquiteturas de SIGs

A partir desta seção, usaremos os conceitos introduzidos na Tabela 5.6 e definidos em detalhe ao longo das seções seguintes ou em outros capítulos deste texto.

Tabela 5.6 – Resumo dos principais conceitos relativos a bancos de dados espaciais

Conceito	Definição
geometria matricial	uma matriz de elementos, usualmente pertencentes a um sub-conjunto dos números reais $\Re$
geometria vetorial	um elemento do $\Re^2$, considerado como um espaço topológico.
	Pontos, linhas e regiões são particulares

	geometrias.
geometria	uma geometria vetorial ou matricial
atributo espacial	um atributo de um objeto cujo domínio seja um conjunto de geometrias.
objeto espacial	qualquer objeto com um atributo espacial. Os geo-objetos são uma classe particular de objetos espaciais.
componente espacial ou geometria de um objeto	valor de um atributo espacial de um objeto.
banco de dados espacial	um banco de dados armazenando, entre outros, objetos espaciais.
consulta espacial	uma consultas definida sobre um banco de dados espacial.

Existem basicamente duas principais formas de integração entre os SIGs e os SGBDs, que são a *arquitetura dual* e a *arquitetura integrada*.

A *arquitetura dual*, mostrada na Figura 5.2 armazena as componentes espaciais dos objetos separadamente. A componente convencional, ou alfanumérica, é armazenada em um SGBD relacional e a componente espacial é armazenada em arquivos com formato proprietário. Os principais problemas dessa arquitetura são:

- Dificuldade no controle e manipulação das componentes espaciais;
- Dificuldade em manter a integridade entre a componente espacial e a componente alfanumérica;
- Separação entre o processamento da parte convencional, realizado pelo SGBD, e o processamento da parte espacial, realizado pelo aplicativo utilizando os arquivos proprietários;
- Dificuldade de interoperabilidade, já que cada sistema trabalha com arquivos com formato proprietário.

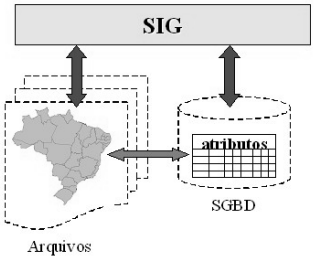


Figura 5.2 – Arquitetura Dual

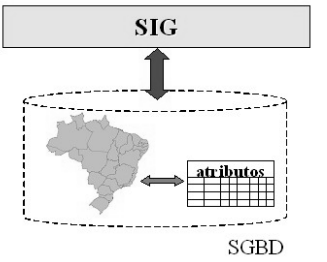


Figura 5.3 – Arquitetura Integrada

A *arquitetura integrada*, mostrada na Figura 5.3, consiste em armazenar todos os dados em um SGBD, ou seja, tanto a componente espacial quanto a alfanumérica. Sua principal vantagem é a utilização dos recursos de um SGBD para controle e manipulação de objetos espaciais, como gerência de transações, controle de integridade, concorrência e linguagens próprias de consulta. Sendo assim, a manutenção de integridade entre a componente espacial e alfanumérica é feita pelo SGBD.

Esta última arquitetura pode ainda ser subdividida em três outras: *baseada em campos longos, em extensões espaciais* e *combinada*.

A *arquitetura integrada baseada em campos longos* utiliza BLOBs para armazenar a componente espacial dos objetos. Como comentado anteriormente, um SGBD-R ou -OR trata um BLOB como uma cadeia de bits sem nenhuma semântica adicional. Portanto, esta arquitetura apresenta algumas desvantagens:

- um BLOB não possui semântica;
- um BLOB não possui métodos de acesso;
- SQL oferece apenas operadores elementares de cadeias para tratar BLOBs.

Portanto, ao codificar dados espaciais em BLOBs, esta arquitetura torna a sua semântica opaca para o SGBD. Ou seja, passa a ser responsabilidade do SIG implementar os operadores espaciais, capturando a semântica dos dados, e métodos de acesso que possam ser

úteis no processamento de consultas, embora seja bastante difícil incorporá-los ao sistema de forma eficiente.

A *arquitetura integrada com extensões espaciais* consiste em utilizar extensões espaciais desenvolvidas sobre um SGBD-OR. Esta arquitetura oferece algumas vantagens:

- permite definir tipos de dados espaciais, equipados com operadores específicos (operadores topológicos e métricos);
- permite definir métodos de acesso específicos para dados espaciais;

Exemplos desta arquitetura são o Oracle Spatial (Murray, 2003) e PostGIS , descritos em maior detalhe no capítulo 8 deste livro. Embora largamente baseados nas especificações do OpenGIS (OGC, 1996), estas implementações possuem variações relevantes entre os modelos de dados, semântica dos operadores espaciais e mecanismos de indexação.

As extensões espaciais utilizadas nas arquiteturas integradas possuem as seguintes características :

- fornecem tipos de dados espaciais (TDEs) em seu modelo de dados, e mecanismos para manipulá-los;
- estendem SQL para incluir operações sobre TDEs, transformando-a de fato em uma linguagem para consultas espaciais;
- adaptam outras funções de nível mais interno ao SGBD para manipular TDEs eficientemente, tais como métodos de armazenamento e acesso, e métodos de otimização de consultas.

Normalmente, essas extensões tratam somente objetos espaciais cuja componente espacial seja uma geometria vetorial, utilizando BLOBs para armazenar dados matriciais, com todos os problemas já citados.

De fato, no caso de aplicativos SIG que manipulam objetos com geometrias tanto matriciais quanto vetoriais, é possível a utilização de uma *arquitetura integrada combinada*, formada pela combinação das duas últimas. Ou seja, as geometrias vetoriais são armazenadas utilizando-se os recursos oferecidos pelas extensões e as geometrias matriciais são armazenadas em BLOBs. As funcionalidades para manipulação de geometrias matriciais são fornecidas por uma camada externa ao SGBD,

de modo a complementar os recursos ausentes, até o momento, nas extensões. Um exemplo desta arquitetura, a TerraLib (Câmara et al., 2000), será discutida em detalhe nos Capítulos 12, 13 e 14.

5.4 Operações espaciais

As consultas espaciais baseiam-se em relacionamentos espaciais de vários tipos: métricos, direcionais e topológicos. Por serem dois conceitos de natureza distinta, as operações sobre as componentes espaciais de geo-campos e geo-objetos também são diferentes. Abordaremos nesta seção apenas operações sobre as geometrias vetoriais de geo-objetos. Portanto, no que se segue, omitiremos o adjetivo "vetorial". As operações espaciais podem ser classificadas em (Rigaux et al., 2002):

Operação unária booleana: mapeia geometrias em valores booleanos. Como exemplos, temos: *Convex,* que testa se uma geometria é convexa; e *Connected,* que testa se uma geometria está conectada.

Operação unária escalar: mapeia geometrias em valores escalares. Como exemplos, temos: *Length,* que computa o comprimento ou perímetro de uma geometria; e *Área,* que computa a área de uma geometria.

Operação unária espacial: mapeia geometrias em geometrias. Como exemplos, temos: *Buffer,* que retorna uma nova geometria a partir de uma distância em torno de uma geometria específica; *ConvexHull*, que retorna uma geometria convexa a partir da geometria; MBR, que retorna o mínimo retângulo envolvente de uma geometria; e *Centroid,* que retorna o centróide de uma geometria.

Operação binária booleana: também chamada de *predicado espacial* ou *relacionamento espacial*, mapeia pares de geometrias em valores booleanos. Esta classe pode ser dividida em:

Relacionamento topológico: um relacionamento que não é alterado por transformações topológicas, como translação, rotação e mudança de escala. Como exemplos, temos: contém (*contains*), disjunto (*disjoint*), intercepta (*intersects*), cruza (*crosses*), como apresentado na seção 2.9.

Relacionamento direcional: um relacionamento que expressa uma noção de direção. Como exemplos, temos: acima de (*above*), ao norte de (*northOf*), dentre outras;

Relacionamento métrico: um relacionamento que expressa uma noção métrica. Por exemplo, o relacionamento que retorna Verdadeiro se duas geometrias estão a menos de uma determinada distância uma da outra.

Operação binária escalar: mapeia pares de geometrias em valores escalares. Por exemplo, *distance* computa a distância entre duas geometrias.

Operação binária espacial: mapeia pares de geometrias em geometrias. Como exemplos, temos as operações de conjunto, como interseção (*Intersection*), união (*Union*) e diferença (*Difference*).

Operação n-ária espacial: mapeia n-tuplas de geometrias em geometrias. Por exemplo, a operação *ConvexHull* pode pertencer a essa classe quando receber mais de uma geometria como parâmetro de entrada.

5.5 Linguagens de consulta espacial

Como SQL-89 não acomodava consultas espaciais, várias propostas surgiram na década de 90 para estender a linguagem, notadamente (Egenhofer, 1994) (OGIS, 1995). Segundo Frank e Mark (1991), as extensões devem considerar dois pontos básicos:

- embora seja possível estender SQL com operadores espaciais, a semântica destes operadores deve ser formalmente definida;
- embora seja possível estender SQL para incluir controle de apresentação de geometrias, aconselha-se projetar uma linguagem separada para lidar com esta questão.

Esta seção apresenta inicialmente uma breve classificação para consultas espaciais. Em seguida, discute algumas extensões para incluir suporte a consultas espaciais na linguagem SQL, incluindo o padrão publicado pela ISSO, chamado SQL/MM Spatial.

5.5.1 Tipos de consultas espaciais

A eficácia das estratégias de otimização de consultas espaciais depende fundamentalmente da complexidade e freqüência das consultas. Embora estes fator seja de difícil caracterização, esta seção sugere uma classificação das consultas espaciais bastante útil para o estudo do problema, seguindo (Brinkhoff et al., 1993).

Dentre os tipos de consultas, destacamos os seguintes:

seleção espacial: dado um conjunto de objetos espaciais D e um predicado de seleção espacial ρ sobre atributos espaciais dos objetos em D, determine todos os objetos em D cujas geometrias satisfazem ρ.

junção espacial: dados dois conjuntos de dados espaciais, D e D', e um predicado de seleção espacial θ, determine todos os pares $(d,d') \in D \times D'$ cujas geometrias satisfazem θ.

Identificamos ainda os seguintes casos particulares importantes de seleção espacial:

seleção por ponto: dado um ponto P e um conjunto de objetos espaciais D, determine todos os objetos em D cujas geometrias contêm P.

seleção por região: dada uma região R e um conjunto de objetos espaciais D, determine todos os objetos em D cujas geometrias estão contidos em R.

seleção por janela: dado um retângulo R com os lados paralelos aos eixos e um conjunto de objetos espaciais D, determine todos os objetos em D cujas geometrias estão contidos em R.

Um *predicado de seleção espacial* é uma expressão booleana B(x) com uma variável livre, x, varrendo geometrias, tal que B(x) envolve apenas operações espaciais. Semelhantemente, um *predicado de junção espacial* $J(x,y)$ é uma expressão booleana com duas variáveis livres, x e y, varrendo geometrias, tal que $J(x,y)$ envolve apenas operações espaciais. Um exemplo de seleção espacial seria:

S1. *Selecione as regiões da França adjacentes à região de Midi-Pirenées.*

A Figura 5.4 ilustra o resultado desta consulta, onde a região de Midi-Pirenées aparece em cinza escuro, e as regiões adjacente, em cinza claro.

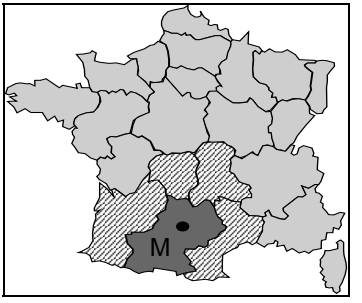


Figura 5.4 – Operação de seleção espacial.

Exemplos de junção espacial seriam:

J1. *Para cada estrada da Amazônia, selecione as reservas indígenas a menos de 5 km de uma estrada.*

J2. *Para as cidades do sertão cearense, selecione quais estão a menos de 10 km de algum açude com capacidade de mais de 50.000 m^3 de água*

5.5.2 SF-SQL

A SF-SQL, proposta pelo *OGC - Open Geoscience Consortium* , especifica um conjunto de tipos de geometrias vetoriais, operações topológicas e operações métricas. A proposta especifica ainda um esquema de tabelas para metadados das informações espaciais. Ela introduz o conceito de "tabela com feições" para representação dos dados geográficos. Nesta tabela, os atributos não espaciais são mapeados para colunas de tipos disponíveis na SQL-92, e os atributos espaciais para colunas cujo tipo de dados é baseado no conceito de "tipos de dados geométricos adicionais para SQL".

A representação dos atributos espaciais pode seguir dois modelos, chamados SQL-92 e SQL-92 com Tipos Geométricos. O primeiro modelo utiliza uma tabela para representar os atributos espaciais. O segundo utiliza tipos abstratos de dados específicos para geometrias, estendendo os tipos da SQL.

Hierarquia de tipos de geometrias da SF-SQL

A Figura 5.5 ilustra a hierarquia de tipos da SF-SQL. Este diagrama é o mesmo tanto para o modelo da SQL-92 quanto para o da SQL-92 com Tipos Geométricos.

Alguns tipos são abstratos como: *Curve*, *Surface*, *MultiSurface* e *MultiCurve*. Um tipo especial é a *GeometryCollection*, que pode ser composta por mais de um tipo de geometria (tipo heterogêneo). Os outros são tipos básicos, como *Point*, *LineString* e *Polygon*, que podem formar tipos de coleções homogêneas como *MultiPoint*, *MultiLineString* e *MultiPolygon*, respectivamente. Cada um destes tipos possui uma série de atributos, métodos e definições que são apresentadas na especificação.

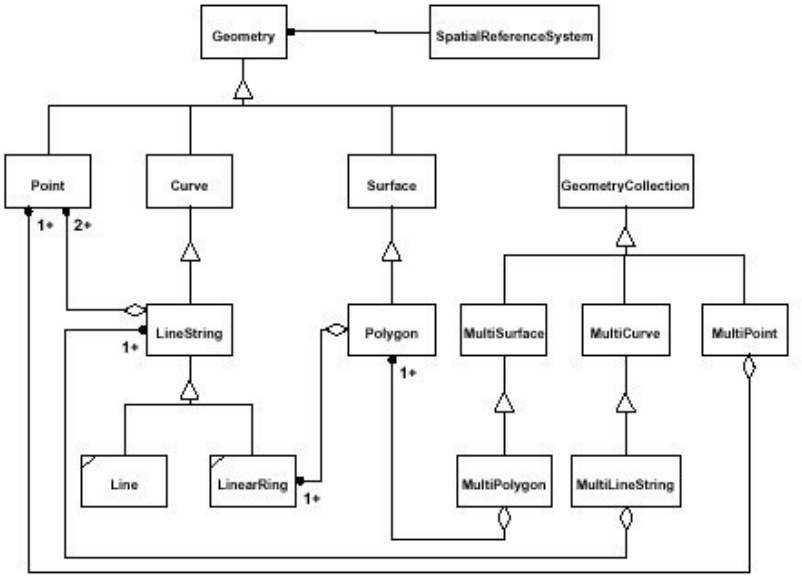


Figura 5.5 – Hierarquia de tipos de geometrias da SF-SQL .

Relacionamentos Topológicos

A SL-SQL basicamente adota os relacionamentos topológicos introduzidos na seção 2.9, definidos como métodos do tipo *Geometry*. Há ainda um outro relacionamento, chamado *relate*, que recebe a geometria a ser comparada e um segundo parâmetro que representa um padrão da matriz de interseção, na forma de uma cadeia de nove caracteres, representando um teste para interseção entre fronteira, interior e exterior. Ele retorna o inteiro 1 (TRUE) se as geometrias forem relacionadas espacialmente de acordo com os valores especificados na matriz. Considere, por exemplo, a seguinte consulta espacial:

Q. *Selecione os municípios que fazem fronteira com o município de Belo Horizonte.*

Esta consulta pode ser expressa em SF-SQL da seguinte forma:

```
SELECT M1.name
FROM Municipio M1,Municipio M2
WHERE Touch(M1.location,M2.location)=1
AND M2.Name ='Belo Horizonte'
```

Outros Operadores Espaciais

Além dos relacionamentos topológicos, SL-SQL inclui outros métodos para o tipo *Geometry*. Entre eles, estão:

distance(outraGeometria:Geometry):Double
retorna a distância entre as geometrias

buffer(distância:Double):Geometry
retorna uma geometria definida por um mapa de distância

convexHull():Geometry
retorna um polígono convexo com todos os pontos da geometria

intersection(outraGeometria:Geometry):Geometry
retorna a geometria resultante da interseção das geometrias

union(outraGeometria:Geometry):Geometry
retorna a geometria resultante da união de duas geometrias

difference(outraGeometria:Geometry):Geometry
retorna a geometria resultante da diferença entre as geometrias

Os tipos *Surface* e *MultiSurface* possuem ainda os seguintes métodos:

area ():double
área de uma região

centroid():point
um ponto representando o centróide da geometria

pointOnSurface():point
um ponto que esteja na superfície

Tabelas com Feições

A especificação da SF-SQL propõe o esquema de metadados mostrado na Figura 5.6 para representar conjuntos de geo-objetos em tabelas com atributos dos tipos de geometrias anteriormente descritos:

SPATIAL_REF_SYS, tabela armazenando dados sobre cada sistema de referenciamento espacial (SRS) utilizado no banco de dados.

GEOMETRY_COLUMNS, tabela de metadados para as colunas geométricas das tabelas com feições *(feature tables)*.

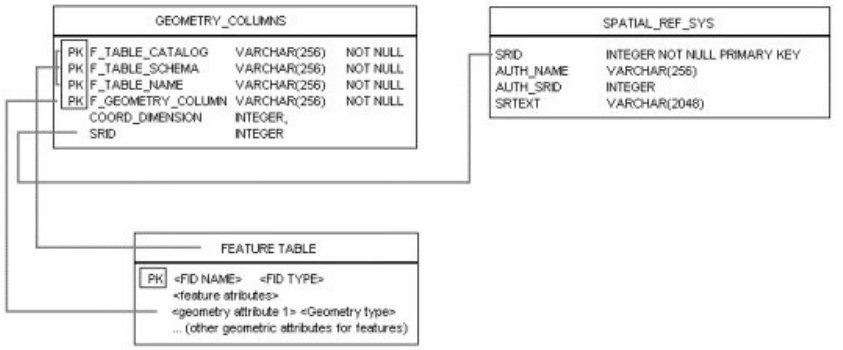


Figura 5.6 – Esquema de Metadados da SF-SQL .

Cada atributo do tipo *Geometry* (ou de seus sub-tipos), de cada tabela com feições, deve estar associado a um SRS, de tal forma que seja possível determinar o sistema de referenciamento espacial utilizado para representar cada geometria armazenada na tabela. Isso é essencial para que os métodos possam verificar a compatibilidade dos sistemas de referenciamento espacial utilizados pelas geometrias.

5.5.3 Spatial SQL

A linguagem Spatial SQL, desenvolvida por Egenhofer (Egenhofer, 1994), representa um outro exemplo de linguagem de consulta espacial baseada em SQL. Spatial SQL divide-se em duas sub-linguagens, uma para consulta e outra para apresentação de objetos espaciais, buscando aproximar-se da forma como os seres humanos conceitualizam o espaço geográfico. A sub-linguagem de consulta estende SQL com operadores e relacionamentos espaciais. Distingue-se de outras extensões por preservar os conceitos de SQL e por conseguir um tratamento de alto nível dos objetos espaciais.

Exemplos de operações disponíveis na Spatial SQL são:

- Operadores unários: fornecem, por exemplo, dados de limite, interior, comprimento, área e volume de objetos.
- Operadores binários: fornecem, por exemplo, distância e direção.
- Funções de agregação: operam sobre conjuntos de objetos, como as funções de mínimo e média.

Os relacionamentos topológicos são baseados na matriz de 9-interseções. (ver seção 2.9). Podemos citar como exemplo: *overlap* (sobreposição), *inside* (dentro) e *cover* (cobertura).

A sub-linguagem de apresentação, chamada *Graphical Presentation Language* (GPL), baseia-se em um trabalho anterior (Egenhofer e Frank, 1988) e especifica como o resultado de uma consulta pode ser visualizado e manipulado em um ambiente gráfico. GPL permite ao usuário definir as características do ambiente gráfico, fornecendo operações e relacionamentos espaciais. Possui ainda métodos para referenciar objetos, apresentados na tela, através de apontamento.

Uma seqüência de comandos escritos em Spatial SQL, adaptada de (Egenhofer, 1994), é mostrada abaixo. Os comandos exibem um mapa de *Grove Street* em *Orono,* com suas construções, limites de terrenos e rodovias. Para a visualização, utiliza-se verde para as residências, azul para os prédios comerciais e preto para os limites de terrenos. Utiliza-se também os nomes das ruas para identificá-las.

- Definição do ambiente gráfico:

```
SET  LEGEND
       COLOR       black
       PATTERN  dashed
FOR  SELECT  boundary (geometry)
       FROM     parcel;
SET  LEGEND
       COLOR    green, blue
FOR SELECT residence.geometry, commercial.geometry
       FROM     residence.type="Residential" AND
              commercial.type="Commercial";
```

- Identificação da janela de interesse e definição da visualização:

```
SET  WINDOW
       SELECT   geometry
       FROM     road
       WHERE    town.name = "Orono";

SET CONTEXT
FOR road.geometry,
              SELECT parcel.geometry, road.name,
                     building.geometry
              FROM   road, parcel, building;
```

- Exibição de um novo mapa e recuperação dos dados:

```
SET MODE new;
SELECT road.geometry
FROM   road, town
  WHERE   town.name = "Orono"  AND
        road.name = "Gove Street" AND
        road.geometry INSIDE town.geometry
```

Este exemplo ilustra vários pontos interessantes de Spatial SQL. Além de oferecer operações espaciais, Spatial SQL separa as etapas de consulta e visualização em etapas menores, permitindo que o usuário reuse um ambiente em diversas consultas. Porém, a linguagem possui uma sintaxe complexa, principalmente para os usuários finais de SIGs, além de supor a existência de uma interface gráfica capaz de apresentar e manipular dados em GPL.

5.5.4 SQL/MM Spatial

A especificação da última versão de SQL, designada pelo nome de SQL:1999 (Melton, 2002), divide-se em várias partes. A SQL/MM (MM para *MultiMedia*) compreende extensões para tratar de texto, dados espaciais e imagem estática ou em movimento.

A especificação da SQL/MM (ISO, 2000) (Melton e Eisenberg, 2001) (Stolze, 2003) também se desdobra em várias partes, bastante independentes entre si. A SQL/MM Spatial define tipos e métodos para tratar geometrias no $\Re^2$. SQL/MM também modela sistemas de referenciamento espacial (SRS). Revisões futuras tratarão de tipos de geometrias no $\Re^3$, ou mesmo em dimensões superiores.

A especificação da SQL/MM Spatial está alinhada com outros esforços de padronização para SIGs, notadamente o ISO Technical Committee, TC 211 (Geomatics) e o OGC - Open Geoscience Consortium. Em particular, a especificação segue a hierarquia de tipos geométricos definidos pelo OGC, discutida na seção 5.6.2.

A especificação do SQL/MM consistentemente usa o prefixo "ST" para os nomes de todas as tabelas, tipos, métodos e funções, para indicar *"Spatial and Temporal"*. De fato, a intenção original da especificação era adotar um modelo de dados espaço-temporal. Porém, durante o desenvolvimento do SQL/MM, decidiu-se que a componente temporal transcendia o escopo das aplicações espaciais e que, portanto, deveria ser incorporado ao SQL:1999 em separado, formando a sub-linguagem SQL/Temporal (ISO, 2001). Porém, esta parte do SQL:1999 foi abandonada, e a proposta de padrão, retirada.

A Figura 5.7 mostra a hierarquia de tipos geométricos de SQL/MM Spatial, adaptada da hierarquia definida pela OGC, onde os tipos sombreados não são instanciáveis. A hierarquia de tipos geométricos da SQL/MM Spatial difere, porém, da hierarquia da OGC em alguns pontos:

- adota ST_LineString em substituição a Line e LinearRing;
- oferece arcos circulares e regiões cujas fronteiras são arcos circulares;

- omite a indicação de que tipos são agregações de outros tipos. Voltando à Figura 5.7, não é óbvio uma geometria do tipo ST_MultiPoint seja uma agregação de geometrias do tipo ST_Point.

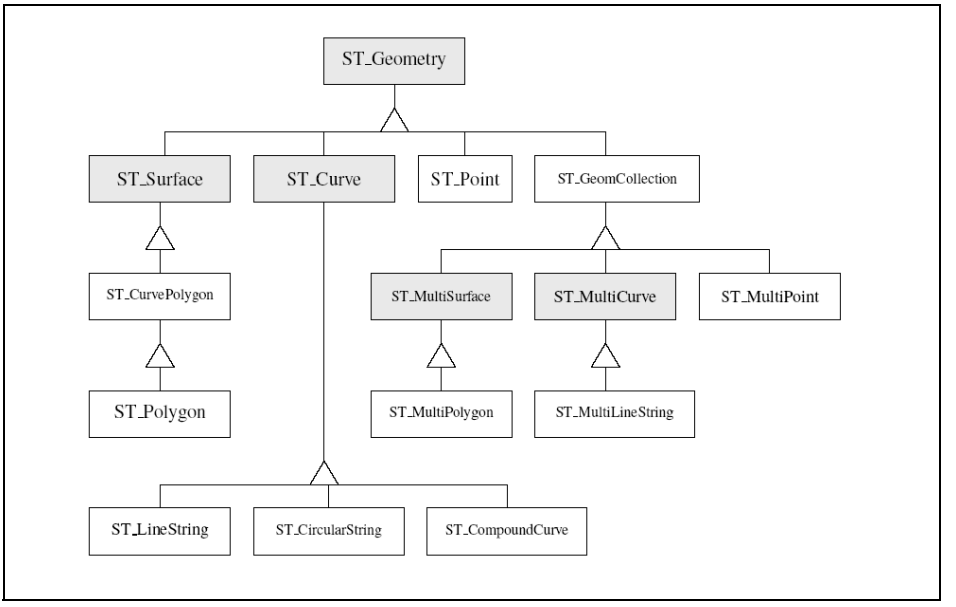


Figura 5.7 – Hierarquia de tipos do SQL/MM Spatial (Stolze, 2003).

A especificação do SQL/MM Spatial agrupa os métodos implementando as operações espaciais em quatro categorias (compare com a classificação introduzida na seção Seção 5.4.

- métodos convertendo geometrias para formatos de exportação e vice-versa;
- métodos computando propriedades métricas de geometrias;
- métodos implementando relacionamentos topológicos;
- métodos gerando geometrias a partir de outras.

Os tipos possuem ainda métodos que permitem extrair informações básicas sobre as suas instâncias, como as coordenadas de um ponto. Por exemplo, considere a seguinte tabela:

```
CREATE TABLE cidade (   nome VARCHAR(30),
                        populacao INTEGER,
                        localizacao ST_GEOMETRY )
```

A consulta abaixo retorna a área da Cidade de Belo Horizonte:

```
SELECT localizacao.area
FROM city
WHERE nome = 'Belo Horizonte'
```

A expressão `localizacao.area` retorna o valor do atributo `area` da instância do tipo estruturado *ST_Geometry* armazenada na coluna `localizacao` da linha da tabela `cidade` tal que a coluna `nome` possui valor `'Belo Horizonte'`.

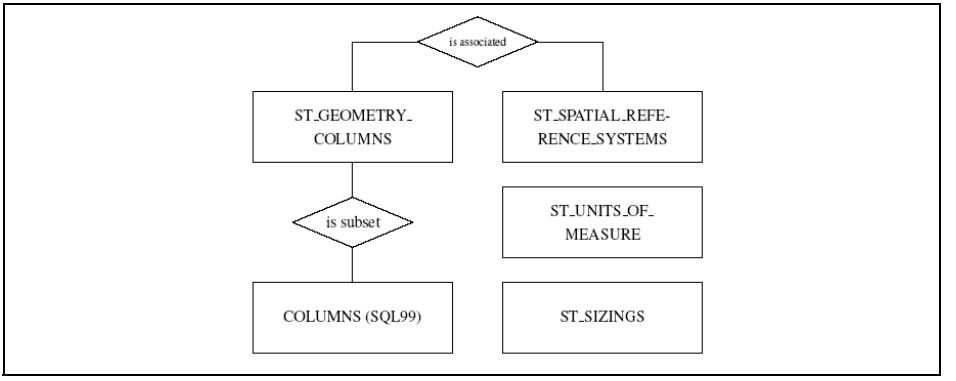


Figura 5.8 – Esquema de metadados da SQL/MM Spatial (Stolze, 2003).

A Parte 3 da especificação do SQL/MM Spatial define um esquema de metadados, semelhante ao esquema definido para o SF-SQL, ilustrado no diagrama entidade-relacionamento da Figura 5.8. O esquema de metadados compreende as seguintes tabelas:

- *ST_GEOMETRY_COLUMNS,* armazena metadados descrevendo as colunas geométricas das tabelas do banco de dados (idêntica à tabela *GEOMETRY_COLUMNS* da SF-SQL, definida pela OGC).
- *ST_SPATIAL_REFERENCE_SYSTEMS,* armazenando dados sobre cada sistema de referenciamento espacial (SRS) utilizado no banco de dados (idêntica à tabela *SPATIAL_REF_SYS* da SF-SQL).

- *ST_UNITS_OF_MEASURE,* armazena as unidades de medida utilizadas no banco de dados.
- *ST_SIZINGS,* semelhante à tabela *SIZINGS* do SQL99, define as meta-variáveis, e seus valores, específicas para a componente especial do banco de dados.

Por fim, a segunda versão do padrão da SQL/MM deverá incluir suporte à *Geography Markup Language (GML)*, definida pela OGC, e suporte a ângulos e direções.

5.6 Direções de pesquisa

5.6.1 Consultas espaço-temporais

Como o nome indica, *consultas espaço-temporais* tratam de dados convencionais, espaciais e temporais. Tais consultas recuperam dados a respeito do estado, da história e da evolução dos objetos ao longo do tempo. Para exemplificar os dados temporais, utilizaremos o monitoramento do desmatamento da Amazônia realizado periodicamente pelo PRODES (Sistema de Detecção de Desmatamentos) e mostrado em Correia et al. (2005).

Como exemplos dessa evolução temporal, temos áreas que em um determinado momento eram parte da floresta e algum tempo depois foram desmatadas. Essas áreas podem alterar seu tamanho ou sua forma em outras observações. Existe também a possibilidade de uma área desmatada ser replantada ou sofrer outros tipos de regeneração.

Um banco de dados espaço-temporal que modela os dados de desmatamento poderia conter várias imagens semelhantes, e os dados relativos a essas imagens. Aliando esses dados a informações, como reservas indígenas ou hidrografia, pode-se pensar em consultas que incluem a dimensão espacial como:

a) *Quais são as áreas de reservas indígenas próximas a áreas desmatadas?*

b) *Quais são as áreas de preservação de hidrografia que possuem desmatamentos?*

Quando incluímos a dimensão temporal e buscamos combiná-la com a dimensão espacial, as consultas se tornam mais complexas. Podemos pensar em consultas que tratem a evolução ao longo do tempo dos objetos envolvidos, como:

a) *Quais são as áreas desmatadas no período de 01-01-1990 a 31-12-1999 e posteriormente recuperadas?*

b) *Quais são as áreas desmatadas no último bimestre que ficam a menos de 50 Km de distância de estradas ou rios?*

Na primeira consulta nos deparamos com o desafio de definir as áreas que eram florestas e foram desmatadas na década de 90. Dentro dessas áreas, procuramos quais evoluíram e deixaram de ser áreas desmatadas. Na segunda consulta também nos deparamos com outro desafio, o de definir quais áreas foram desmatadas e como essas áreas se relacionam com estradas e rios próximos.

Essas duas consultas ilustram questões complexas de serem modeladas pelas linguagens disponíveis, e mostram que as linguagens de consulta espaço-temporais ainda têm muitos desafios a superar.

5.6.2 Álgebra para objetos móveis

Uma álgebra para *objetos móveis*, ou seja, objetos que se movem continuamente ao longo do tempo, é apresentada em Güting et al (2003). Essa álgebra foi implementada no ambiente SECONDO, um SGBD extensível e modular, que permite a utilização de diversas álgebras (Guting et al., 2004).

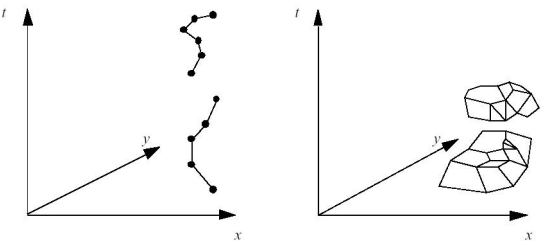


Figura 5.4 – Exemplos de representações de *mpoint* e *mregion* (Fonte: adaptada de Guting et al., 2004).

Dois tipos básicos de objetos móveis são (ver Figura 5.4):

- *moving point (mpoint)*: descreve um objeto cuja relevância está nas posições ocupadas no espaço ao longo do tempo. Um exemplo de um *mpoint* é o deslocamento de um veículo.
- *moving region (mregion):* descreve uma área que possui extensão e se modifica, por exemplo, crescendo ou se movimentando. Um exemplo pode ser o movimento de uma tempestade.

Operações podem ser definidas envolvendo tipos convencionais e os tipos básicos de objetos móveis. Por exemplo, temos:

- intersection: mpoint × mregion → mpoint
- retorna um mpoint formado pela parte de um mpoint que estiver dentro de uma mregion.
- trajectory: mpoint → line
- projeta um mpoint no plano, retornando uma line, definida como uma curva em espaço bidimensional.
- deftime: mpoint → periods

  retorna um intervalo de tempo, do tipo *periods*, definido por determinado *mpoint*.

Por fim, considere as seguintes tabelas:

```
flights (id:string,from:string,to:string,route:mpoint)
```

```
weather (id:string,kind:string,area:mregion)
```

Exemplos de consultas sobre estas tabelas são:

Q1. *Selecione todos os vôos partindo de Düsseldorf que percorrem mais de 5000 Km*

```
SELECT id
FROM flights
WHERE from = "DUS" AND
   length(trajectory(route)) > 5000
```

Q2. *Que horas o vôo BA488 atravessou a tempestade de neve de identificador S16?*

```
SELECT deftime(intersection(f.route,w.area))
FROM flights as f, weather as w
WHERE f.id = "BA488" AND w.id = "S16"
```

5.7 Leituras suplementares

Há inúmeros livros-texto sobre bancos de dados e a linguagem SQL. Recomenda-se, em especial, os textos sobre SQL:1999 (Melton e Simon, 2001) (Melton, 2002). O primeiro trata dos conceitos relacionais básicos da linguagem, enquanto que o segundo aborda as extensões objeto-relacionais, incluindo suporte a consultas espaciais.

Um pouco da história do desenvolvimento de bancos de dados espaciais pode ser avaliada através de referências básicas sobre a área, como Güting (1994) e Medeiros (1994). Recomenda-se também a leitura dos livros-texto de Rigaux e Voisard (2001) e Shekhar (2002).

Conforme discutido na seção 5.3, a arquitetura integrada com extensões espaciais mostra-se promissora. O Oracle Spatial (Murray, 2003), o DB2 Spatial Extender (Adler, 2001) e o PostGIS são exemplos desta arquitetura, que merecem um estudo detalhado. O documento descrevendo a arquitetura de referência do *Open Geoscience Consortium* (Percivall, 2003) apresenta uma abordagem para o problema de interoperabilidade em SIGs, assunto do Capítulo 10.

A definição de relacionamentos topológicos foi extensamente estudada, incluindo a matriz de 4-interseções e suas variantes (Egenhofer e Franzosa, 1995), a matriz de 9-Interseções (Egenhofer et al., 1994), a matriz de 9-Interseções estendida dimensionalmente (Paiva, 1998) e o trabalho de Clementini et al. (1993) adotado pela SF-SQL e SQL/MM Spatial.

Quanto a linguagens de consulta espacial, Stolze (2003) apresenta uma análise sucinta, porém criteriosa, do padrão SQL/MM Spatial.

Além dos tópicos abordados acima, há linguagens de consulta endereçando outros tipos de dados geográficos, ou ambientes com necessidades específicas. Como exemplo, podemos citar linguagens de consulta para redes georeferenciadas, como a rede de distribuição de água de uma cidade, onde a noção de espaço Euclidiano não é adequada

(Papadias et al., 2003). Há também toda uma linha de pesquisa sobre consultas dependentes de localização em ambientes para computação móvel (Ayse et al., 2001) (Zhang et al., 2003) (Manical et al., 2004).

Referências

ADLER, D.W. DB2 Spatial Extender - Spatial data within the RDBMS. 27th International Conference on Very Large Data Bases, 2001. Morgan Kaufmann Publishers Inc. p. 687-690

AYSE, Y.; DUNHAM, H. M.; KUMAR, V. M. Location dependent query processing. In: 2nd ACM international workshop on Data engineering for wireless and mobile access. Santa Barbara, CA, USA, 2001. p. 47-53.

BRINKHOFF, T.; HORN, H.; KRIEGEL, H.-P.; SCHNEIDER, H. A Storage and Access Architecture for Efficient Query Processing in Spatial Database Systems. In: Third International Symposium on Advances in Spatial Databases. **Lecture Notes In Computer Science 692**. Springer-Verlag London, UK, 1993. p. 357-376.

CÂMARA, G.; SOUZA, R.; PEDROSA, B.; VINHAS, L.; MONTEIRO, A. M.; PAIVA, J.; CARVALHO, M. T.; GATTASS, M. TerraLib: Technology in Support of GIS Innovation. In: II Brazilian Symposium on Geoinformatics, GeoInfo2000. São Paulo, 2000. p.

CLEMENTINI, E.; DI FELICE, P.; VAN OOSTEROM, P., 1993. A Small Set of Formal Topological Relationships Suitable for End-User Interaction. In: ABEL, D.; OOI, B. C., eds., **SSD '93: Lecture Notes in Computer Science, v. 692**: New York, NY, Springer-Verlag, p. 277-295.

CORREIA, A. H.; PIROMAL, R. A. S.; QUEIROZ, G. R.; SOUZA, R. C. M., 2005, Modelagem de um banco de dados espaço-temporal para desmatamento e queimadas, XII Simpósio Brasileiro de Sensoriamento Remoto, Goiânia.

EGENHOFER, M. Spatial SQL: A Query and Presentation Language. **IEEE Transactions on Knowledge and Data Engineering**, v. 6, n.1, p. 86-95, 1994.

EGENHOFER, M.; FRANK, A. Towards a Spatial Query Language: User Interface Considerations. In: 14th International Conference on Very Large Data Bases. Los Angeles, CA, 1988. p. 124-133.

EGENHOFER, M.; P. DI FELICE; CLEMENTINI, E. Topological Relations between Regions with Holes. **International Journal of Geographical Information Systems**, v. 8, n.2, p. 129-144, 1994.

EGENHOFER, M.; FRANZOSA, R. On the Equivalence of Topological Relations. **International Journal of Geographical Information Systems**, v. 9, n.2, p. 133-152, 1995.

FRANK, A.; MARK, D., 1991. Language Issues for GIS. In: MAGUIRE, D.; GOODCHILD, M.; RHIND, D., eds., **Geographical Information Systems, Volume 1: Principles**: London, Longman, p. 147-163.

GÜTING, R. An Introduction to Spatial Database Systems. **VLDB Journal**, v. 3, n.4, p. 357-399, 1994.

GÜTING, R. H.; BEHR, T.; ALMEIDA, V. T. D.; DING, Z.; HOFFMANN, F.; SPIEKERMANN, M., 2004, SECONDO: An Extensible DBMS Architecture and Prototype, Fernuniversität Hagen.

GÜTING, R. H.; BOHLEN, M. H.; ERWIG, M.; JENSEN, C. S.; LORENTZOS, N.; NARDELLI, E.; SCHNEIDER, M.; VIQUEIRA, J. R. R., 2003. Spatio-temporal Models and Languages: An Approach Based on Data Types. In: KOUBARAKIS et al. ed., **Spatio-Temporal Databases**: Berlin, Springer.

ISO, 2001, Information Technology – Database Languages – SQL – Part 7: Temporal (SQL/Foundation), International Organization for Standardization.

MANICAL, H.; CAMARGO, S. M.; CIFERRI, A. D. C.; CIFERRI, R. R. Processamento de Consultas Espaciais Baseado em Cache Semântico Dependente de Localização. In: VI Simpósio Brasileiro de Geoinformática – GeoInfo 2004. Campos de Jordão, SP, 2004.

MEDEIROS, C. B.; PIRES, F. Databases for GIS. **ACM SIGMOD Record**, v. 23, n.1, p. 107-115, 1994.

MELTON, J.; SIMON, A. **SQL: 1999 Understanding Relational Language Components**. Morgan Kaufmann, 2001.

MELTON, J. **Advanced SQL 1999: Understanding Object-Relational, and Other Advanced Features**. New York, NY, USA: Elsevier Science Inc., 2002.

MELTON, J.; EISENBERG, A. SQL Multimedia and Application Packages (SQL/MM). **SIGMOD Record**, 2001.

MURRAY, C., 2003, **Oracle® Spatial User's Guide and Reference 10g Release 1 (10.1)**, Redwood City, Oracle Corporation, p. 602.

OGC, ed., 1996, **The OpenGIS® Guide - Introduction to Interoperable Geoprocessing**. Boston, Open GIS Consortium, Inc.

OGIS, 1995, **OpenGIS® simple features specification for SQL revision 1.1.**

PAIVA, J. A. C. Topological Equivalence and Similarity in Multi-Representation Geographic Database. University of Maine,1998.

PAPADIAS, D.; ZHANG, J.; MAMOULIS, N.; Y., T. Query Processing in Spatial Network Databases. In: 29th Very Large Data Base Conference. Berlin, Germany, 2003.

PERCIVALL, G., 2003, **OpenGIS® Reference Model, Open Geoscience Consortium**.

RIGAUX, P.; SCHOLL, M.; VOISARD, A. **Spatial Databases with Application to GIS**. San Francisco: Morgan Kaufmann, 2002.

SCHNEIDER, M. **Spatial data types for database systems.** Berlin Hidelberg: Springer-Verlag, 1997.

SHEKHAR, S.; CHAWLA, S. **Spatial Databases: A Tour.** New York: Prentice-Hall, 2002.

STOLZE, K. SQL/MM Spatial: The Standard to Manage Spatial Data in Relational Database Systems. In: BTW 2003, Datenbanksysteme für Business, Technologie und Web. Leipzig, Alemanha, 2003. p. 247-264.

STONEBRAKER, M. **Object-relational DBMSs: the next great wave.** San Francisco: Morgan Kaufmann, 1996.

ZHANG, J.; ZHU, M.; PAPADIAS, D.; TAO, Y.; LEE, D. L. Location-based spatial queries. In: 2003 ACM SIGMOD International Conference on Management of Data. San Diego, CA, USA, 2003. p. 443-454.

6 *Métodos de acesso para dados espaciais*

Clodoveu A. Davis Jr.
Gilberto Ribeiro de Queiroz

6.1 Introdução

Este capítulo aborda alguns dos métodos de acesso espacial mais tradicionais, denominados *k-d trees, grid files, quad-trees* e *R-trees*.

Métodos de acesso espacial, ou índices espaciais, são estruturas de dados auxiliares, mas essenciais para o processamento eficiente de consultas espaciais. De fato, normalmente uma consulta espacial envolve apenas uma pequena parcela do banco de dados. Neste caso, percorrer todo o banco pode ser bastante ineficiente. Portanto, um plano de execução para a consulta tipicamente começa com uma fase de filtragem, que identifica quais objetos podem satisfazer a qualificação da consulta. Esta fase depende da existência de índices espaciais, em geral definidos sobre aproximações das geometrias dos objetos.

6.2 Preliminares sobre métodos de acesso

Uma forma comum de indexarmos um conjunto de dados é através do uso de árvores de pesquisa. As mais simples e conhecidas são as árvores binárias, como: AVL, Red Black e Splay Tree (Cormen et al., 1990), que são árvores balanceadas, ou seja, todos os caminhos desde a raiz até as folhas possuem o mesmo comprimento. Essas estruturas permitem que algumas operações, como a localização de um elemento, sejam executadas em tempo logarítmico – $O(\log n)$. A Figura 6.1 ilustra a representação de uma árvore binária balanceada, onde as cidades estão organizadas segundo a ordem alfabética de seus nomes. Esse tipo de estrutura é empregado para uso em memória principal.

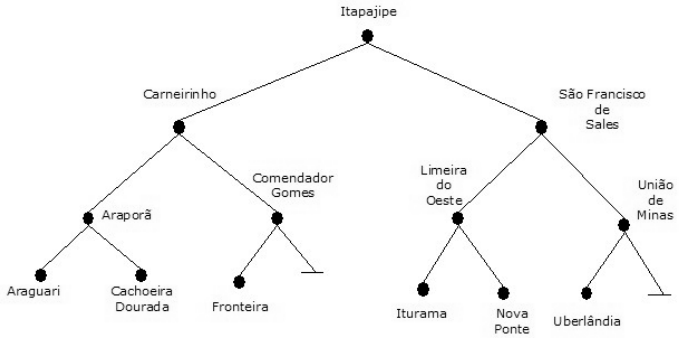


Figura 6.1 – Árvore binária balanceada.

No caso de grandes bancos de dados onde a memória principal pode não ser suficiente para armazenar todos os nós da árvore, é comum armazená-la em disco. Nesse caso utilizamos uma representação que procura minimizar o acesso a disco. A forma mais comum, e mais largamente empregada pelos sistemas comerciais atuais, é a representação do índice através de uma árvore B^+ (Comer, 1979). Conforme podemos observar na Figura 6.2, essa estrutura tenta minimizar o número de acessos a disco durante o processo de pesquisa agrupando várias chaves em um único nó, denominado de página.

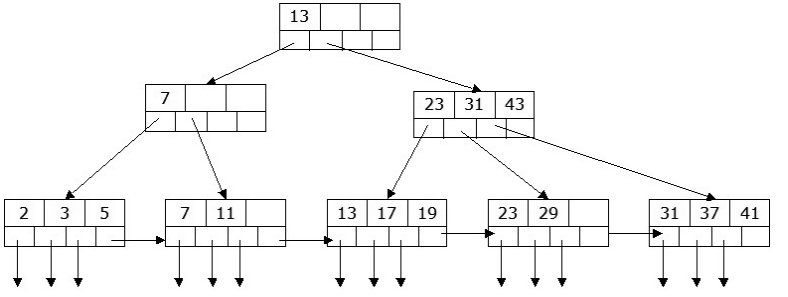


Figura 6.2 – Índice unidimensional B-Tree. Fonte: adaptada de Garcia-Molina et al., (2001).

Em comum, tanto a árvore binária quanto a B^+, são estruturas unidimensionais, isto é, elas pressupõem que a chave de pesquisa seja formada por apenas um atributo ou pela concatenação de vários atributos. Em sistemas que trabalham com informações

multidimensionais (mais de um atributo), como os sistemas de bancos de dados espaciais, estas estruturas não são diretamente aplicáveis.

Para esses sistemas existe uma classe de métodos conhecidos como *métodos de acesso multidimensionais*. No caso dos bancos de dados espaciais, estes métodos estão ligados ao processamento de consultas típicas como: consulta por apontamento (encontrar os objetos que contém um dado ponto), consultas por região (encontrar os objetos dentro de uma janela ou retângulo) e consultas com predicados topológicos (encontrar os objetos vizinhos de um determinado objeto).

Uma idéia fundamental dos índices espaciais é o uso de aproximações, isto é, a estrutura do índice trabalha com representações mais simples dos objetos, como o menor retângulo envolvente (REM) do objeto. Dessa forma, a maneira mais comum de processar as consultas é dividir o processamento em duas etapas: uma de filtragem e outra de refinamento (Figura 6.3). Na etapa de filtragem são usados métodos de acesso espaciais. O principal objetivo do uso destes métodos é o de reduzir e rapidamente selecionar os possíveis candidatos que satisfaçam a consulta. A redução do espaço de busca é muito importante, pois a etapa seguinte, a de refinamento, envolve a aplicação de algoritmos geométricos computacionalmente complexos e custosos e que são aplicados à geometria exata dos candidatos selecionados na etapa anterior.

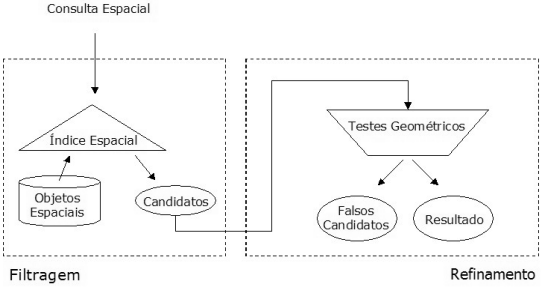


Figura 6.3 – Processamento de consultas espaciais. Fonte: adaptada de Gaede e Günther (1998).

6.3 *k*-d Tree

A *k*-d tree é uma árvore binária, ou seja, cada nó interno possui dois descendentes não importando a dimensionalidade *k* do espaço envolvido. Originalmente, esta árvore é apresentada em Bentley (1975), como uma alternativa a um problema mais genérico do que o geométrico dos bancos de dados espaciais: o de busca baseado em vários atributos.

Em um banco de dados tradicional, uma consulta como: "encontrar todos os empregados nascidos entre 1950 e 1955 que recebem entre R$3.000,00 e R$5.000,00", poderia ser vista como uma pesquisa em duas dimensões, considerando cada atributo, neste caso data de nascimento e salário, como uma das componentes do espaço (Figura 6.4a). No caso dos sistemas de bancos de dados espaciais, essa consulta equivale a encontrar todos os pontos no interior do retângulo definido pelos intervalos (3000:5000)x(1950:1955) (Figura 6.4b).

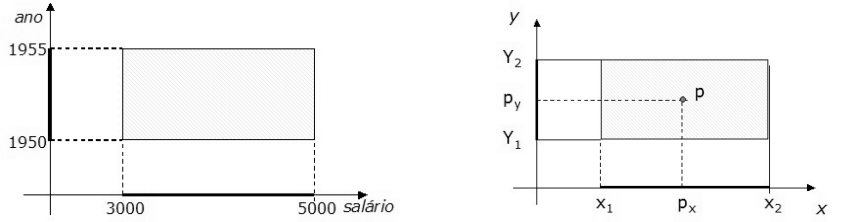


Figura 6.4 – Busca em dois atributos (a) Interpretação geométrica da busca (b).

Dada uma chave formada por *k* atributos (*k* dimensões), onde k_i é o valor da *i-ésima* componente da chave, a idéia básica da *k*-d tree é a seguinte:

- A cada nó da árvore associamos uma chave e um discriminador;
- O discriminador é um valor entre 0 e *k*-1 que representa a dimensão ao longo da qual o espaço será subdividido, ou seja, a componente k_i que será empregada para definição da ordem na árvore;
- Todos os nós do mesmo nível são associados ao mesmo valor de discriminador;
- O nó raiz está associado ao discriminador 0, aos dois descendentes diretos, discriminador 1, e assim por diante, até o k-ésimo nível que está associado ao discriminador *k*-1;

- A partir do nível $k+1$ o ciclo começa se ser repetido, sendo associado o discriminador 0 a este nível.
- A ordem imposta por essa árvore é tal que, dado um nó qualquer P de discriminador j, qualquer nó descendente (L) da sub-árvore esquerda possui um valor de chave cuja componente k_j é menor do que a componente k_j do nó P;
- Da mesma forma, qualquer nó descendente (R) da sub-árvore direita possui um valor de chave cuja componente k_j é maior do que a componente k_j do nó P.

Para o caso de pontos no espaço bidimensional, como o da Figura 6.5, a k-d tree ou 2-d tree que corresponde à decomposição do espaço ao longo das dimensões x e y, compara os valores da componente "x" nos níveis pares da árvore e da componente "y" nos níveis ímpares.

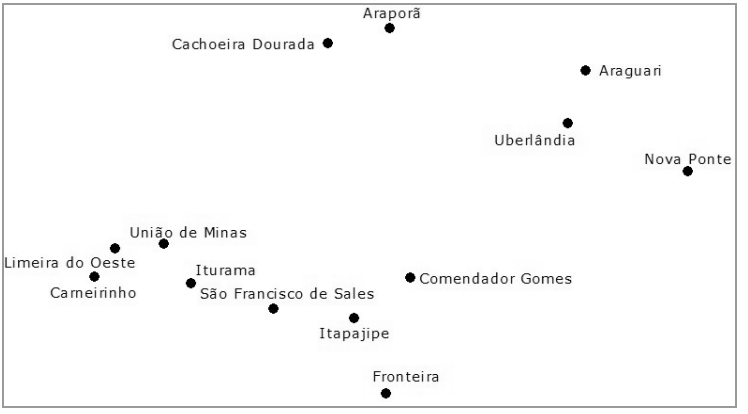


Figura 6.5 – Sedes de alguns municípios do Estado de Minas Gerais.

Assim, se adotarmos uma ordem arbitrária de entrada dos pontos correspondentes às sedes municipais, podemos construir a árvore da Figura 6.6 da seguinte forma:

- Considerando Araguari a primeira cidade a ser inserida na árvore, esta ocupará a raiz;
- Sendo Araporã, a próxima, como ela possui um valor de coordenada "x" menor do que a de Araguari, esta é inserida à esquerda. Aqui, como a raiz ocupa um nível par (0), comparamos

os valores da componente "*x*" das coordenadas das sedes municipais.

- Nova Ponte possui um valor "*x*" maior, e, portanto, é inserida à direita.
- No caso de Cachoeira Dourada, ela possui um valor de "*x*" menor do que Araguari (comparação em um nível par), e um valor de "y" menor do que Araporã (comparação no nível ímpar – 1), e por isso é inserida à esquerda desta.
- Esse processo é realizado até que todas as cidades estejam armazenadas na árvore.

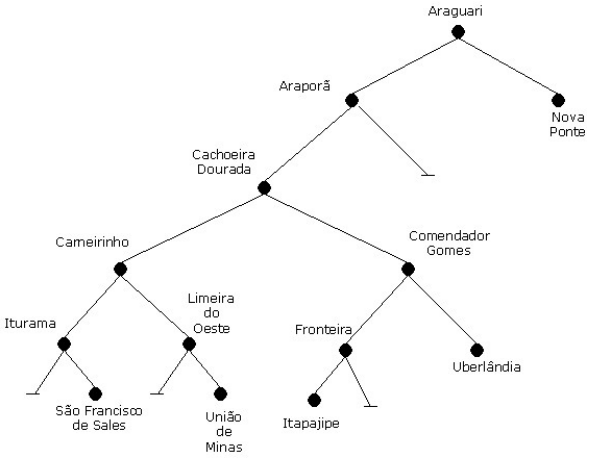


Figura 6.6 – Representação em forma de árvore para as sedes da Figura 6.5.

A partição do espaço representada pela *k*-d tree do exemplo acima pode ser vista na Figura 6.7.

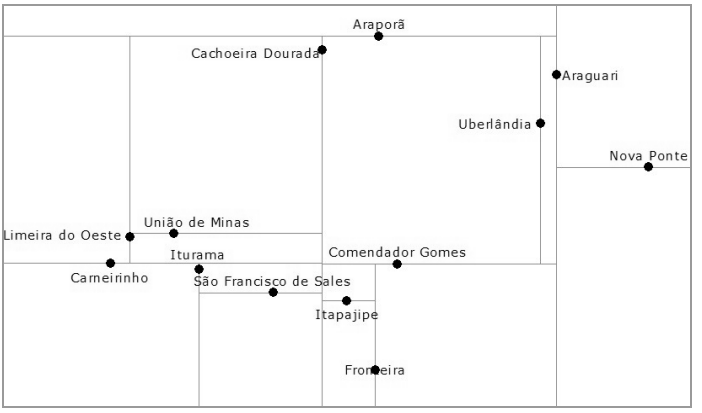


Figura 6.7 – Representação planar para as sedes da Figura 6.5.

Um dos problemas que podemos observar na *k*-d tree da Figura 6.6 é que ela depende da ordem em que os nós são inseridos. No pior caso, podemos acabar obtendo uma árvore completamente desbalanceada, que apresenta a mesma profundidade do número de elementos inseridos. Uma forma de contornar este problema é adotar um método que otimize a construção da árvore. Uma forma de construir a árvore mais balanceada é escolher o elemento que ocupa a posição média ao longo da componente em divisão. A Figura 6.8 ilustra uma árvore construída tomando-se o elemento central em relação à componente em cada nível da divisão.

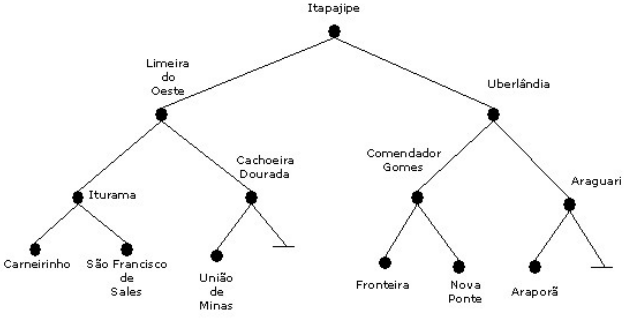


Figura 6.8 – Representação em árvore (balanceada) para as sedes.

Na Figura 6.9, podemos observar que Itapajipe possui o mesmo número de elementos do lado esquerdo e direito e por isso foi tomada

como o primeiro elemento a ser inserido na árvore da Figura 6.8. Limeira do Oeste pode ser tomada como elemento médio entre os elementos à esquerda de Itapajipe quando comparamos ao longo do eixo *y*.

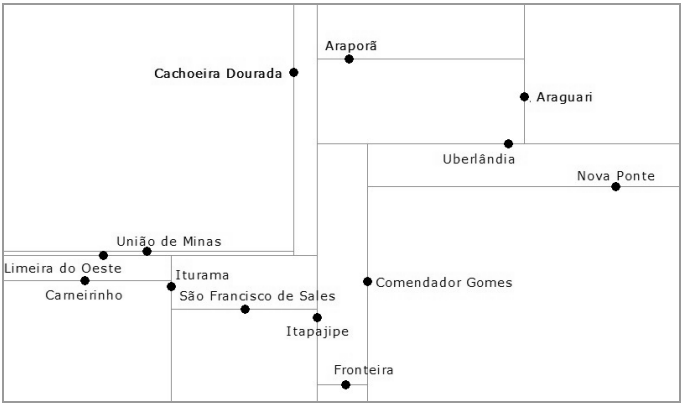


Figura 6.9 – Representação planarpara as sedes da Figura 6.5.

A pesquisa por intervalos na 2-d tree do exemplo anterior pode ser realizada da seguinte forma:

1. Seja um retângulo de coordenadas inferior esquerda (x_1:y_1) e superior direita (x_2:y_2) contendo o intervalo de pesquisa, partindo da raiz, temos que verificar:
2. Se o nó corrente encontra-se no intervalo ele é reportado;
3. Se ele for um nó em um nível par (caso do nó raiz):
    a. Se a coordenada "x_1" do retângulo de pesquisa for menor do que a coordenada "x" do nó, aplicamos recursivamente os passos a partir do passo 2, à sub-árvore esquerda;
    b. Se a coordenada "x_2" do retângulo de pesquisa for maior do que a coordenada "x", aplicamos recursivamente os passos a partir do passo 2, à sub-árvore direita;
4. Se ele for um nó em um nível ímpar:
    a. Se a coordenada "y_1" do retângulo de pesquisa for menor do que a coordenada "y" do nó, aplicamos recursivamente os passos a partir do passo 2, à sub-árvore esquerda;

b. Se a coordenada "y_2" do retângulo de pesquisa for maior do que a coordenada "y", aplicamos recursivamente os passos a partir do passo 2, à sub-árvore direita;

Na literatura, encontramos diversas variantes dessa estrutura. Friedman et al. (1977) apresentam uma variante em que os elementos (registros ou pontos) são armazenados apenas nos nós folha, sendo que os nós internos contém apenas os valores usados para particionar o espaço. Robinson (1981), combinando a B-Tree, adaptou esta estrutura para armazenamento em memória secundária, chamando esta nova estrutura de *k*dB-Tree.

6.4 Grid File

Uma das técnicas mais interessantes para facilitar a pesquisa em dados convencionais é o *hashing*. Este método, chamado também de "transformação de chave", consiste em criar uma série de receptáculos (chamados habitualmente de *buckets*), que receberão os identificadores dos itens que se quer pesquisar. Estes *buckets* são numerados seqüencialmente de 1 a *n*, e portanto sua implementação mais natural é sob a forma de arranjo (Figura 6.10). Cada identificador que chega, seja para ser inserido, seja para ser pesquisado, é transformado em um número de 1 a n, identificando o bucket correspondente a ele. Então, só se precisa inserí-lo no *bucket* (no caso de inserção) ou procurar um elemento com identificador igual que já esteja dentro do *bucket*. Quanto maior for o valor de *n*, menos itens existirão em média dentro de cada *bucket*, e o resultado da pesquisa será mais rápido. No entanto, a escolha do valor de *n* e da função de transformação ideal é problemática, pois não se deseja provocar uma distribuição irregular dos itens pelos *buckets*. Em geral, recomenda-se utilizar *n* primo, e uma função de transformação que (1) seja simples de ser computada, e (2) produza uma distribuição uniforme de saídas entre 1 e *n*.

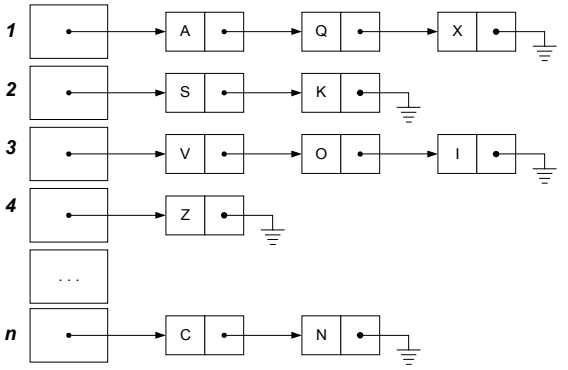


Figura 6.10 – *Hashing.*

O método de indexação chamado *grid file* (Nievergelt et al., 1984). estende o conceito de *hashing* para duas dimensões. Ou seja, em vez de *n buckets*, teremos uma grade regular que cobre todo o espaço de pesquisa. O método foi originalmente concebido para possibilitar pesquisas convencionais, como no caso do *hashing*, porém utilizando duas variáveis distintas. No caso geográfico, é natural imaginar quadrados sobre a superfície da Terra, que funcionam como os *buckets* do *hashing* (Figura 6–11). A partir de cada elemento da grade (que pode ser modelada na memória com o uso de uma matriz no lugar do vetor utilizado no *hashing*) são organizadas listas lineares contendo os identificadores dos objetos que estão localizados naquela área.

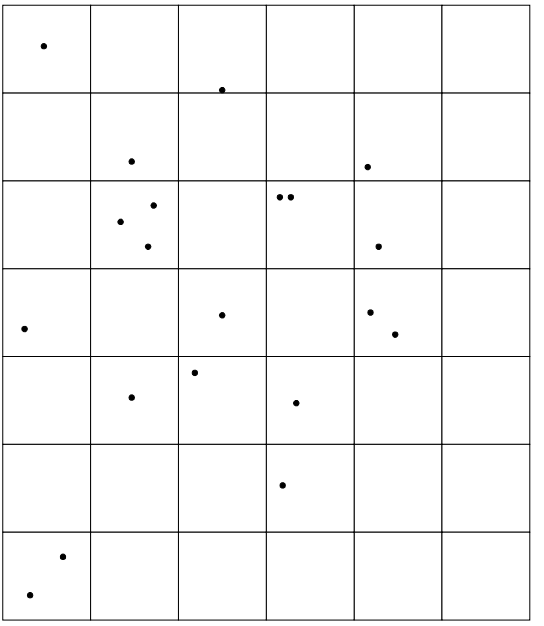


Figura 6.11 – *Grid File*.

Alguns tipos de pesquisas podem ser feitas com muita facilidade utilizando *grid files*. Por exemplo, deseja-se saber quais pontos estão a menos de M metros de um determinado ponto P. Para isto, basta calcular as coordenadas de dois pontos auxiliares, um deles subtraindo M de ambas as coordenadas, e no outro somando:

$$x_1 = x_P - M$$
$$y_1 = y_P - M$$
$$x_2 = x_P - M$$
$$y_2 = y_P - M$$

Depois, determina-se qual é a linha e a coluna dos quadrados que contêm os pontos x_1 e x_2. Todos os pontos procurados estarão nos quadrados compreendidos entre os limites expressos pelas linhas e colunas encontradas. Naturalmente, deve-se calcular a distância de cada ponto encontrado a P, para verificar o atendimento à restrição de distância, pois a pesquisa na realidade foi feita usando um quadrilátero, e não com um círculo.

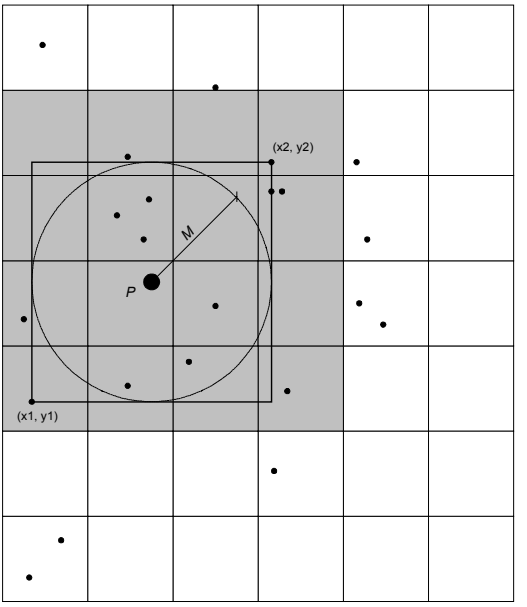


Figura 6.12 – Localização de pontos por proximidade.

A principal limitação deste método, superada pelos métodos que serão apresentados a seguir, é o fato de somente poder lidar com pontos, ou com objetos que caibam inteiramente em um quadrado. Não é um método adequado para lidar com linhas ou polígonos, mas é bastante eficiente e simples de implementar para o tratamento de dados puntuais.

6.5 Quad-tree

A quad-tree é um tipo de estrutura de dados organizada em árvore, em que cada "nó" ou "tronco" gera sempre (e exatamente) quatro "folhas", como mostrado na Figura 6.13 (Samet, 1984). O SIG, utilizando esta estrutura de dados para realizar a indexação espacial, considerará que cada nó corresponderá a uma região quadrada do espaço. Esta região será subdividida, novamente em quatro partes, gerando mais um nível na árvore, e assim sucessivamente, até que se chegue a ter um ou nenhum objeto geográfico dentro dos quadrados resultantes da subdivisão.

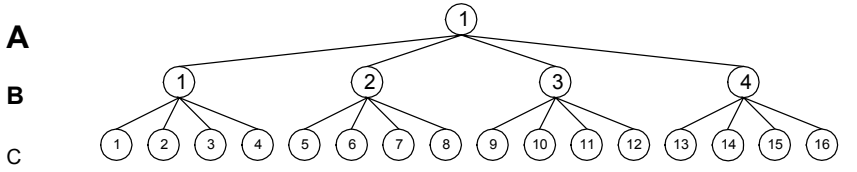


Figura 6.13 – Quad-tree.

Na Figura 6.13 existem três níveis (A, B e C) na quad-tree. A cada um deste níveis corresponderá um quadrado geográfico, conforme apresentado na Figura 6.14.

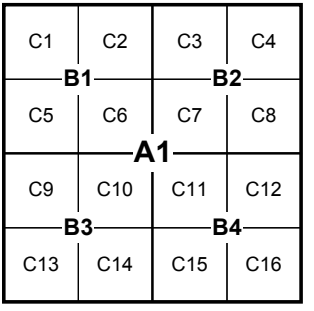


Figura 6.14 – Subdivisão do espaço por uma quad-tree.

Esta subdivisão poderá continuar indefinidamente, de forma recursiva: basta considerar um quadrado do nível C como sendo o quadrado A1, e reaplicar o processo. Observe-se que o resultado final será uma árvore que terá um número variável de níveis, de acordo com a região geográfica: regiões mais densas formarão mais níveis, e regiões menos densas terão menos níveis.

O processamento de consultas por região (janela ou retângulo) nessa estrutura inicia-se a partir da raiz, selecionando apenas as ramificações cujos quadrados estiverem em contato com o retângulo da região. Se nenhuma estiver, todas as ramificações abaixo daquele nó são descartadas. Se alguma estiver, toda a ramificação abaixo daquele ponto é selecionada, e sucessivamente testada, até que se alcance os objetos geográficos.

6.6 Tiling

Este processo é semelhante ao anterior, com a diferença de que as subdivisões não prosseguem indefinidamente.

O processo consiste em, conhecendo-se a priori os limites geográficos da base de dados, criar camadas de "tiles", ou ladrilhos, quadrados, de dimensões fixas. O número de camadas e a largura dos ladrilhos de cada camada são determinados pelo usuário, de modo a corresponder da melhor maneira possível à variedade de dimensões dos objetos geográficos que serão encontrados na base de dados. O número total de ladrilhos endereçáveis é limitado pelo sistema, para melhorar sua performance.

Cada objeto será associado a um ladrilho, que será o menor ladrilho que contiver inteiramente o objeto. Assim, a cada ladrilho, de cada camada, serão associados diversos objetos, formando uma lista. No momento do display, testa-se os limites da janela de visualização contra a lista de ladrilhos, selecionando aqueles que tenham pelo menos uma extremidade contida no retângulo. Os objetos associados a cada um destes ladrilhos serão, então, recuperados na base de dados para apresentação.

Com estes limites previamente calculados, evita-se percorrer freqüentemente uma estrutura de dados como a quad-tree, que geralmente tem que ser mantida em disco.

A Figura 6–15 ilustra a estrutura básica do *tiling*.

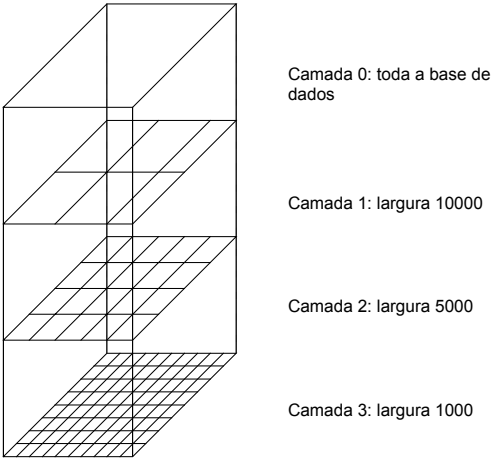


Figura 6.15 – Tiling.

6.7 R-tree

A *R-tree*, ou árvore-R, é uma estrutura de dados hierárquica derivada da árvore-B. As diferenças estão centradas na natureza das chaves: valores numéricos ou alfanuméricos simples, no caso das árvores-B, e pontos extremos de retângulos, no caso das árvores-R (Guttman, 1984).

O que a árvore-R busca organizar não é exatamente o contorno ou a forma gráfica do objeto, e sim o seu *retângulo envolvente mínimo* (*minimum bounding rectangle*, MBR). Este retângulo é formado a partir da observação dos limites geométricos mínimo e máximo do contorno do objeto, e é expresso pelas coordenadas dos seus pontos inferior esquerdo e superior direito (Figura 6.16). No caso de objetos puntuais, estes extremos vão coincidir com as coordenadas do próprio objeto (portanto formando um retângulo nulo), mas isto não compromete o método.

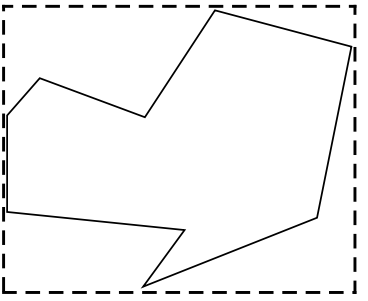


Figura 6.16 – Objeto poligonal e seu retângulo envolvente mínimo.

A árvore-R possui parâmetros para determinar a quantidade de chaves (retângulos) que poderão ocorrer em cada bloco de armazenamento, analogamente à árvore-B, em função do tamanho da página de armazenamento em disco.

As regras básicas para formação de uma árvore-R são semelhantes às da árvore-B. Todas as folhas aparecem sempre no mesmo nível da árvore. Nós internos contêm a delimitação de retângulos que englobam todos os retângulos dos nós nos níveis inferiores (Figura 6.17). Uma árvore-R de ordem (m, M) conterá entre m e M entradas em cada nó da árvore ($m \leq M / 2$), exceto a raiz, que terá pelo menos dois nós (a menos que a árvore só tenha uma entrada).

Um problema com as árvores-R é que a ordem de inserção dos objetos interfere na forma final da árvore, e portanto vai interferir também com o resultado das operações de subdivisão dos nós para manter o balanceamento. Existem algumas técnicas para tentar otimizar este comportamento da árvore-R, mas sempre com algum custo adicional em termos de processamento.

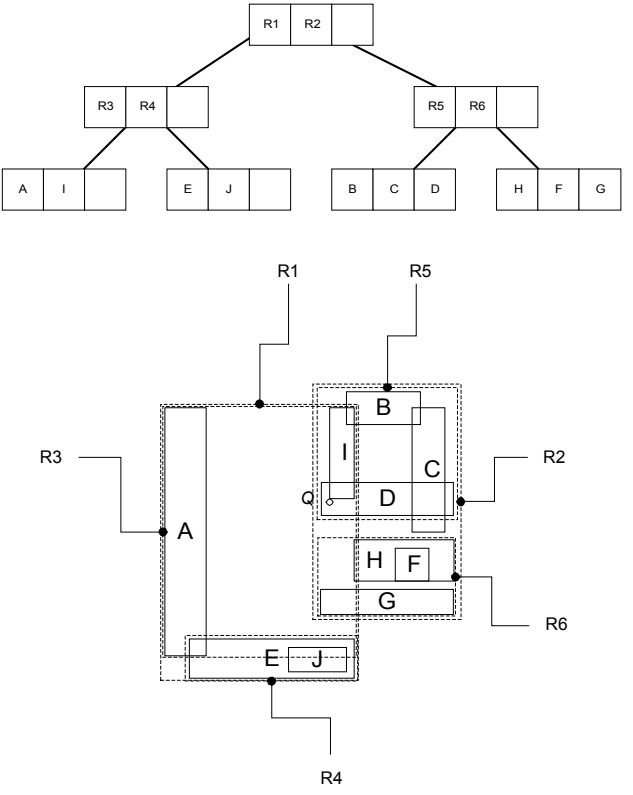


Figura 6.17 – Árvore-R e retângulos correspondentes.

Pesquisas na árvore-R são relativamente simples de serem executadas. O único problema é que um grande número de nós pode ter que ser examinado, pois um retângulo pode estar contido em vários outros, mesmo que o objeto esteja contido em apenas um nó folha. Por exemplo, na Figura 6.17 a identificação do retângulo que contém o ponto *Q* deverá varrer a árvore nível a nível. Inicialmente, inspecionamos a raiz e constatamos que *Q* pode estar tanto em *R1* quanto em *R2*, e portanto

devemos prosseguir a pesquisa em ambas as direções no segundo nível. Pesquisando os filhos de *R1*, verificamos que *Q* só pode estar contido em *R3*. Continuando a pesquisa, não chegamos a conclusão alguma, uma vez que *Q* está dentro do retângulo *D*, que tem apenas uma parte dentro de *R3*, e não é acessado por ele. Continuando a pesquisa a partir de *R2*, verificamos que *Q* somente poderia estar dentro de *R5*. Pesquisando R5, encontramos *D*, o único MBR que contém *Q* efetivamente. Observe-se, no entanto, que *Q* pode não estar dentro do *objeto* cujo MBR é *D*: esta constatação depende ainda de um teste adicional, que verificará se o ponto pertence ao polígono do objeto, agora sim levando em consideração a geometria deste.

Existem diversas variações das árvores-R, cada uma tentando aperfeiçoar um aspecto diferente. No entanto, muitas vezes estas variações introduzem desvantagens, ou uma maior complexidade de implementação, fazendo com que a árvore-R original acabe sendo a opção mais usual.

6.8 Leituras suplementares

Uma fonte importante de leitura complementar são os trabalhos de Gaede e Günther (1998) e, mais recentemente, Vitter (2001), que apresentam uma revisão detalhada da maioria dos índices existentes.

Há inúmeras variantes de R-trees, sendo a R*-tree a mais popular (Beckmann et al., 1990). Uma comparação entre os métodos de acesso espacial mais comum pode ser encontrada em (Kriegel et al., 1990).

Por fim, as TV-trees (Lin et al., 1994) e as X-trees (Berchtold et al., 1996) são exemplos de métodos de acesso para dados de dimensões superiores a dois.

Referências

BECKMANN, N.; KRIEGEL, H. P.; SCHNEIDER, R.; SEEGER, B. The R*-tree: An Efficient and Robust Access Method for Points and Rectangles. In: **ACM SIGMOD**, p. 322-331, 1990.

BENTLEY, J. L. Multidimensional binary search trees used for associative searching. **Communications of the ACM**, v. 18, n. 9, p. 509-517, 1975.

BERCHTOLD, S.; KEIM, D. A.; KREIGEL, H.-P. "The X-Tree: An Index Structure for High-Dimensional Data". In Proc. 22nd Int. Conf. on Very Large Databases (VLDB'96), 1996.

COMER, D. The ubiquitous B-Tree. **ACM Computing Surveys**, v. 11, n. 2, p. 121-137, 1979.

CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L. **Introduction to algorithms**. E.U.A.: McGraw-Hill, 1990.

FRIEDMAN, J. H.; BENTLEY, J. L.; FINKEL, R. A. An algorithm for finding best matches in logarithmic expected time. **ACM Transactions on Mathematical Software**, v. 3, n. 3, p. 209-226, 1977.

GAEDE, V.; GÜNTHER, O. Multidimensional access methods. **ACM Computing Surveys**, v. 30, p. 170-231, 1998.

GARCIA-MOLINA, H.; ULLMAN, J. D.; WIDOM, J. **Implementação de sistemas de bancos de Dados**. Rio de Janeiro: Campus, 2001.

GUTTMAN, A. R-Trees: A Dynamic Index Structure for Spatial Searching. In: Annual Meeting ACM SIGMOD. Boston, MA, 1984. p. 47-57.

KRIEGEL, H.-P.; SCHIWIETZ, M.; SCHNEIDER, R.; SEEGER, B. Performance comparison of point and spatial access methods. **Proceedings of the first symposium on Design and implementation of large spatial databases**. Santa Barbara, California, United States. p. 89 – 114, 1990 .

LIN, K.-I.; JAGADISH, H. V.; FALOUTSOS, C. "The TV-tree an index structure for highdimensional data". VLDB Journal: Very Large Data Bases, v.3, n.4, p.517–542, 1994.

NIEVERGELT, J.; HINTERBERGER, H.; SEVCIK, K. The GRID FILE: An Adaptable, Symmetric Multi-Key File Structure. **ACM Transactions on Database Systems (TODS)**, v. 9, n.1, p. 38-71, 1984.

ROBINSON, J. T. Physical storage structures: The K-D-B-tree: a search structure for large multidimensional dynamic indexes. In: 1981 ACM SIGMOD international conference on Management of data. ACM Press, Washington, 1981. p.

SAMET, H., 1984. The Quadtree and Related Hierarchical Data Structures. **ACM Computing Surveys**, v. 16, p. 187-260.

VITTER, J.S. External Memory Algorithms and Data Structures. ACM **Computing Surveys**, v.33, n. 2, p. 209-271, 2001.

7 *Processamento de consultas e gerência de transações*

Marco Antonio Casanova

7.1 Introdução

Este capítulo aborda dois aspectos da implementação de um sistema de gerência de bancos de dados geográficos: processamento de consultas espaciais e gerência de transações. O texto enfatiza as questões que necessitam soluções distintas daquelas empregadas em sistemas convencionais.

Um dos principais componentes de todo sistema de gerência de bancos de dados, o processador de consultas, recebe uma consulta e prepara um plano de execução, consistindo de operações de mais baixo nível, implementadas pelo subsistema de armazenamento. A parte do plano responsável pelo processamento dos objetos geográficos tipicamente começa com uma fase de filtragem, que identifica quais objetos podem satisfazer a qualificação da consulta. Esta fase pode utilizar índices espaciais, em geral definidos sobre aproximações das geometrias dos objetos. A fase de refinamento do plano recupera para memória principal a geometria exata dos objetos identificados na fase anterior, computando quais deles de fato satisfazem a qualificação da consulta. A fase final do plano aplica transformações aos objetos retornados na fase anterior, produzindo a resposta final da consulta.

Um segundo componente importante de todo sistema de gerência de banco de dados implementa a noção de transação atômica, tipicamente de curta duração. No caso de sistemas de informação geográfica, a interação com o banco de dados é comumente mais longa exigindo uma revisão dos mecanismos de controle de transações, principalmente a criação de

mecanismos para versionamento, atualização cooperativa e bloqueio parcial de objetos.

Referências para problemas específicos sobre processamento de consultas e gerência de transações encontram-se ao longo do texto e sugestões para leitura adicional, ao final do capítulo.

7.2 Estratégia para processamento centralizado de consultas

Um dos principais componentes de todo sistema de gerência de bancos de dados é o processador de consultas que, ao receber uma consulta Q, prepara um *plano de execução* para Q consistindo de operações de mais baixo nível, implementadas pelo subsistema de armazenamento.

O principal componente do processador de consultas, por sua vez, é um otimizador capaz de gerar planos alternativos para execução de consultas, estimar o custo de cada plano e escolher o de menor custo estimado. O otimizador incorpora heurísticas que limitam o espaço de busca, evitando uma explosão combinatorial de planos alternativos. Seguindo (Brinkhoff et al., 1993), a parte do plano responsável pelo processamento da componente espacial de uma consulta contém três fases (ver Figura 7.1):

Fase 1: Filtragem

Um banco de dados geográfico normalmente contém estruturas de dados auxiliares, chamadas de *índices espaciais*, *estruturas de indexação espacial* ou simplesmente *índices*, que facilitam a localização dos dados cuja componente espacial satisfaz à qualificação de uma consulta. Reservamos o termo *método de indexação espacial* para designar uma família de estruturas de indexação espacial. De fato, diversos métodos de indexação espacial foram discutidos no Capítulo 6.

O desempenho deste passo depende da distribuição espacial dos dados e da forma como a estrutura decompõe o espaço, sendo que em geral é baixo o fator de seletividade. Dada a complexidade da componente espacial e da semântica dos próprios operadores espaciais, em geral os índices trabalham apenas com aproximações das componentes espaciais (por exemplo, o retângulo mínimo envolvente). Assim, o uso de índices para resolver a parte espacial de uma consulta apenas filtra, dos conjuntos de dados armazenados no banco de dados, aqueles que certamente não satisfazem à qualificação da consulta, sendo necessário um segundo passo.

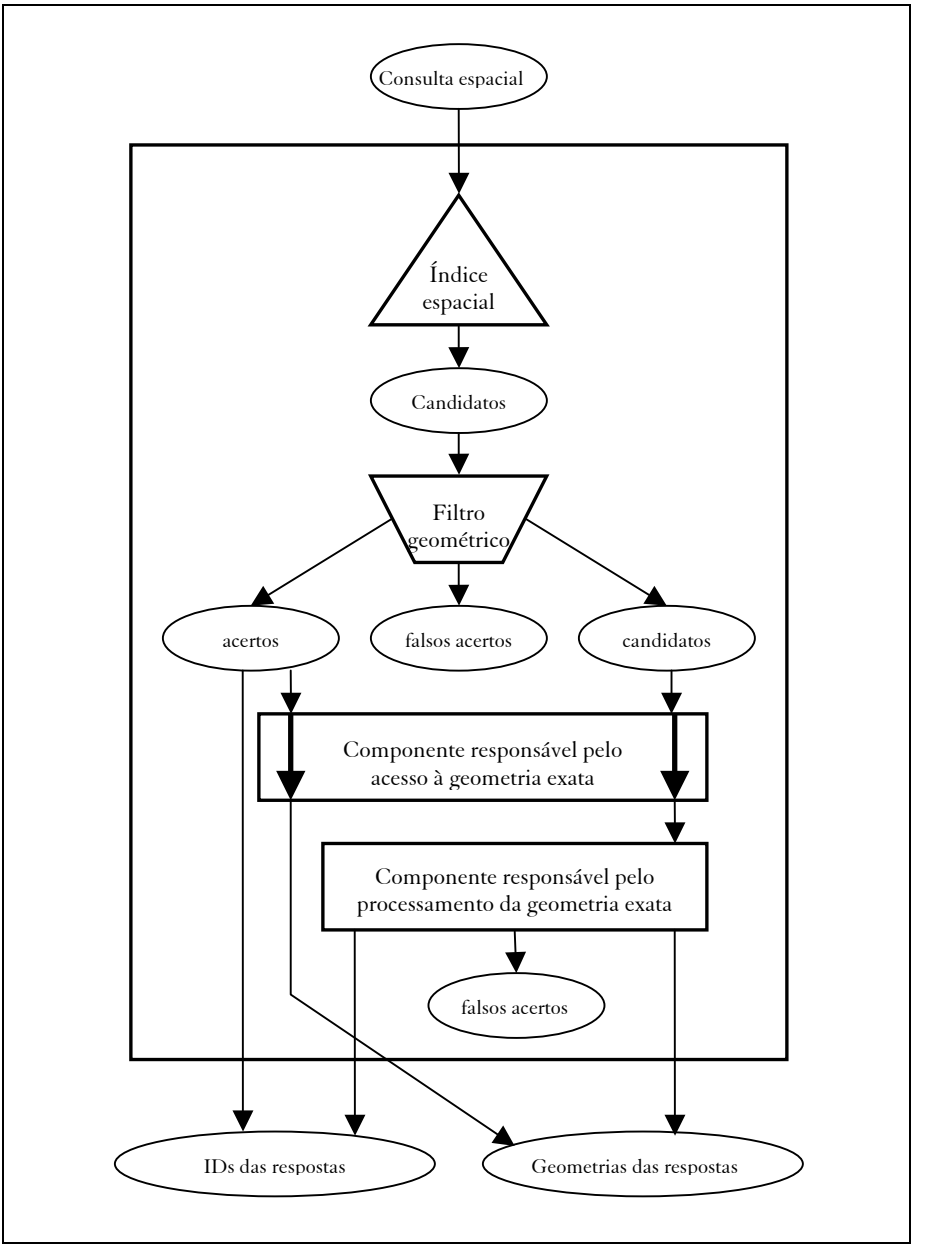


Figura 7.1 – Processamento de consultas espaciais (Kriegel et al., 1993).

Fase 2: Refinamento

Uma vez identificados os dados que se candidatam a satisfazer a qualificação da consulta, as geometrias exatas das componentes espaciais dos dados são trazidas para memória principal. É sobre estas geometrias que são então executados os operadores espaciais especificados na qualificação. As estruturas de armazenamento e indexação devem preferencialmente permitir que seja trazida para memória principal apenas a parte das componentes espaciais necessária à execução dos operadores.

O desempenho deste passo depende da quantidade de dados recuperados, da sua complexidade e do custo dos operadores a aplicar, que em geral é elevado.

Fase 3: Pós-Processamento

Depois de identificados os dados que satisfazem à qualificação da consulta, usualmente é necessário passá-los para as camadas superiores da arquitetura onde sofrerão processamento posterior ou serão visualizados. Assim, o sistema deve oferecer um mecanismo eficiente para passar a geometria das componentes espaciais dos dados, na sua totalidade ou em parte, para as camadas superiores.

Cabe ao subsistema de armazenamento implementar os *métodos de armazenamento*, que governam a utilização das páginas físicas em memória secundária, onde residem tanto as componentes espaciais dos dados quanto os próprios índices espaciais.

Este subsistema deve equacionar dois problemas:

espaço de armazenamento: a representação interna de uma componente espacial pode necessitar um espaço substancial, ocupando mais de uma página física;

contigüidade física: as páginas físicas de uma mesma componente espacial ou de componentes contíguas no espaço devem ser vizinhas em memória secundária, diminuindo o custo de acesso.

Uma estratégia para resolver estes problemas consiste em armazenar a geometria exata das componentes espaciais dos dados fora dos índices, digamos, em um segmento comum de páginas físicas. Esta forma de

armazenamento leva a índices mais compactos e não impõe limitações ao tamanho das componentes, resolvendo assim o problema de espaço de armazenamento. Porém, para endereçar o problema de contigüidade física, é necessário definir uma estratégia de gerência para o segmento de página físicas de tal forma que componentes vizinhas no espaço geográfico tenham as suas geometrias armazenadas em páginas próximas e, idealmente, que permita armazenar nas mesmas páginas diferentes tipos de geometrias.

7.3 Exemplos de otimização de consultas

Esta seção aborda, por meio de exemplos, o problema de otimização de consultas espaciais, à luz dos comentários feitos na Seção 7.2.

7.3.1 Exemplo de processamento de seleção por região

Considere um banco de dados geográfico sobre cidades brasileiras consistindo de apenas um conjunto de geo-objetos, CIDADES. Para cada cidade (do mundo real), há um objeto em CIDADES, cujos atributos convencionais correspondem aos dados convencionais da cidade e cuja localização é dada por um par ordenado indicando a latitude e longitude da cidade; ou seja, a localização destes geo-objetos é representada por um ponto em coordenadas geográficas. No que se segue, suporemos que estas coordenadas estão armazenadas junto com o próprio geo-objeto, embora as operações sobre elas possam se beneficiar de um índice espacial sobre CIDADES.

Considere agora a consulta Q, formulada informalmente como:

Q. *Selecione o nome e população de todas as cidades a menos de 50km da cidade de Belo Horizonte e com mais de 50 mil habitantes.*

Esta consulta seria definida através do comando:

```
SELECT d.nome, d.populacao
       FROM  d Cidade, c Cidade
       WHERE c.nome = "Belo Horizonte"
         and distance(d,c) < 50
         and d.populacao > 50.000
```

Esta consulta exemplifica uma seleção por região, definida por um círculo de raio de 50km, cujo centro está na localização geográfica da

cidade de Belo Horizonte. Há essencialmente três planos de execução mais plausíveis para esta consulta (entre outros possíveis), discutidos a seguir.

Considere inicialmente o plano de execução P_1:

P_{11}. Determine a posição b de Belo Horizonte;

P_{12}. Determine o conjunto C' de todas as cidades a 50km de b;

P_{13}. Determine o subconjunto C de C' de cidades com mais de 50 mil habitantes.

As subconsultas P_{11} e P_{13} não envolvem nenhum operador espacial e, portanto, são subconsultas convencionais de Q para este plano. Já a subconsulta P_{12} envolve o operador de distância e é a única subconsulta espacial de Q em P_1.

O processamento de P_1 começa com a execução da subconsulta convencional P_{11}, que retorna a localização da cidade de Belo Horizonte, o ponto b.

Suponha que o banco de dados inclua um índice espacial sobre CIDADES, que chamaremos de ID-CIDADES, invisível aos usuários, que facilite identificar todos os objetos de CIDADES cuja localização esteja a uma certa distância de um dado ponto. Suponha ainda que o índice espacial seja impreciso para este caso, no sentido de também retornar objetos que não estão à distância especificada do ponto. Esta imprecisão é típica de métodos de indexação espacial, e contrasta com os métodos de indexação tradicionais, que são precisos.

O processamento da subconsulta P_{12} segue em três passos:

P_{121}. Através de ID-CIDADES, determine que objetos em CIDADES podem estar a menos de 50km de b, criando o conjunto I.

P_{122}. Leia da memória secundária para a memória principal a representação interna dos objetos indicados por I, criando o conjunto C".

P_{123}. Determine que elementos de C" estão de fato a menos de 50km de b, criando o conjunto C'.

Os elementos de I são apontadores para a localização em memória secundária da representação interna de objetos em CIDADES. Além

disto, como supomos que o índice não é exato, I poderá apontar para objetos a mais de 50km de b. Já os elementos de C' e C'' são representações internas de objetos em CIDADES, armazenadas em memória principal. Assim, o passo P_{121} corresponde à fase de filtragem P_{122} e P_{123}, à fase de leitura e refinamento.

O processamento da subconsulta P_{13} é agora simples, bastando extrair de cada elemento de C' os valores dos atributos de interesse e eliminando aqueles que não possuem mais de 50 mil habitantes, criando assim o conjunto C, que é a resposta da consulta Q. Os elementos de C serão pares de valores, correspondendo ao nome e à população de uma cidade.

O processamento de P_{121}, P_{122}, P_{123} e P_3 poderá ser concatenado em *pipeline*, ou seja, cada elemento identificado em P_{121} poderá ser imediatamente processado por P_{122} e filtrado por P_{123}, eventualmente gerando um par em C, ao ser tratado em P_{13}. Isto evita a criação explícita dos resultados intermediários I, C' e C'', levando a uma razoável economia no processamento global da consulta.

O segundo plano, P_2, inverte a ordem de execução das restrições de Q:

P_{21}. Determine a posição b de Belo Horizonte;

P_{22}. Determine o conjunto C' de todas as cidades com mais de 50 mil habitantes;

P_{23}. Determine o subconjunto C de C' de cidades a 50km de b.

Os elementos de C' serão triplas em memória principal indicando o nome, população e localização de objetos em CIDADES com mais de 50 mil habitantes. O processamento da subconsulta convencional P_{22} será particularmente eficiente se partirmos do pressuposto que o banco de dados inclui um índice convencional sobre CIDADES, que chamaremos de ID-POP, invisível aos usuários, que permita identificar todos os objetos de CIDADES cuja população esteja em uma faixa de valores. Tal índice será provavelmente preciso, no sentido de retornar exatamente o subconjunto de cidades cuja população está na faixa de valores dada. Porém, note que o conjunto C' corresponderá a todas as cidades brasileiras com mais de 50 mil habitantes. Neste caso, a execução de P_{22} dá-se da seguinte forma:

P_{221}. Através de ID-POP, determine que objetos em CIDADES possuem população com mais de 50 mil habitantes, criando o conjunto J;

P_{222}. Leia da memória secundária para a memória principal a representação interna dos objetos indicados por J, selecionando seu nome, população e localização e criando o conjunto C'.

O processamento da subconsulta P_{23} é então simplificado, bastando selecionar do conjunto C' (suposto já em memória principal) aqueles elementos que estão de fato a menos de 50km de b, criando o conjunto C. Novamente, o processamento de P_{221}, P_{222} e P_{23} poderá ser sincronizado, evitando a criação de J e C'.

O terceiro plano, P_3, constrói dois conjuntos independentemente para, em seguida, computar a sua interseção:

P_{31}. Determine a posição b de Belo Horizonte;

P_{32}. Determine o conjunto C' de todas as cidades com mais de 50 mil habitantes;

P_{33}. Determine o conjunto C'' de cidades a 50km de b;

P_{34}. Determine a interseção C de C' e C''.

A execução detalhada deste plano é uma combinação da discutida para os dois planos anteriores.

Até este ponto, descrevemos apenas o detalhamento de três planos plausíveis para execução da consulta. A estimativa do custo de cada plano depende essencialmente de estimar o número de objetos identificados pelo índice ID-CIDADES e pelo índice ID-POP e está fora do escopo deste texto. Por exemplo, o plano P_1 será mais vantajoso do que P_2 se ID-CIDADES for razoavelmente eficiente, ou seja, não retornar muitas cidades que não estão a menos de 50km de Belo Horizonte, e se o número de cidades brasileiras com mais de 50 mil habitantes for muito grande. Ou seja, se a construção da resposta intermediária registrada no conjunto I for mais barata do que a registrada no conjunto J.

7.3.2 Exemplo de processamento de junção espacial

Suponha agora que o banco de dados geográfico contém também um conjunto de geo-objetos, AEROPORTOS, tal que, para cada aeroporto

brasileiro, há um objeto em AEROPORTOS cujos atributos convencionais correspondem aos dados convencionais do aeroporto e cuja localização é dada novamente pela latitude e longitude do aeroporto. Suponha ainda que o banco inclua um índice espacial sobre AEROPORTOS, chamado ID-AEROPORTOS, que permita identificar todos os aeroportos cuja localização esteja a uma certa distância de um dado ponto.

Considere a seguinte consulta "Selecione todas as cidades que são sede de municípios e que são atendidas por aeroportos pavimentados". Esta consulta exemplifica uma junção espacial. Para torná-la mais precisa, suponha que uma cidade é atendida por um aeroporto se a cidade dista menos de 50km do aeroporto. A consulta reformulada seria então:

Q. *Selecione o nome de cada cidade e o nome de cada aeroporto tais que a cidade é sede de município, o aeroporto é pavimentado e a cidade dista menos de 50km do aeroporto.*

Novamente há vários planos de execução possíveis para esta consulta, sendo os principais discutidos a seguir. Todos os planos, sob certo aspecto, reduzem o processamento da junção espacial ao processamento de várias seleções por região.

Considere o plano P_1 inicialmente:

P_{11}. Determine o conjunto R contendo o nome n e localização l de cada aeroporto pavimentado;

P_{12}. Inicialize o conjunto C como vazio;

P_{13}. Para cada elemento $(n,l) \in R$, faça:

  P_{131}. Determine o conjunto dos nomes c de cidades que são sede de município e que estão a menos de 50km de l, acrescentando o par (c,n) a C.

O processamento de P_1 começa com a execução da subconsulta convencional P_{11}, que retorna o nome e a localização de todos os aeroportos pavimentados. Para cada um destes, executa-se uma seleção por região para determinar as sedes dos municípios que estão a menos de 50km do aeroporto. O processamento destas seleções segue então de acordo com uma das estratégias discutidas no exemplo anterior.

O segundo plano, P_2, é simétrico a este, recuperando primeiro o conjunto de todas as cidades que são sede de município para depois determinar quais aeroportos pavimentados estão a menos de 50km de cada cidade. Como supusemos que há um índice espacial sobre o conjunto de aeroportos, este segundo plano compete com o primeiro. A determinação de qual dos dois planos é mais vantajoso depende de qual conjunto for menos numeroso: os aeroportos pavimentados ou as cidades que são sede de município.

O terceiro plano, P_3, primeiro constrói de forma independente dois conjuntos para, em seguida, computar a sua interseção:

P_{31}. Determine o conjunto R contendo o nome n e localização l de cada aeroporto pavimentado;

P_{32}. Determine o conjunto S contendo o nome c e localização s de cada sede de município;

P_{33}. Determine o conjunto C tal que $(c,l) \in C$ se e somente se $(n,l) \in R$ e $(c,s) \in S$ e a distância de l a s é menor do que 50km.

O passo P_{33} esconde uma iteração sobre os elementos de R e sobre os elementos de S. Em um caso, por exemplo, a cada elemento de R, todos os elementos de S seriam visitados para determinar quais satisfazem a condição imposta. Logo, o plano P_3 nada mais é do que uma reformulação do plano P_1.

É possível, no entanto, implementar outras formas de computar o passo P_{33} que tornam o plano P_3 diferente dos demais. Discutiremos a seguir, brevemente, como aplicar *o método de varredura do espaço* para computar este passo. Intuitivamente, este método consiste em varrer o espaço em uma dada direção processando todos os objetos encontrados. No caso abaixo, usaremos um meridiano, varrendo o plano em ordem crescente de valor de longitude:

P_{331}. Ordene inicialmente os elementos de R e S em valor crescente da longitude das suas localizações;

P_{332}. Para cada elemento de menor longitude ainda não processado, faça:

P_{3321}. Suponha que este elemento seja um par $(n,l) \in R$ (se for um par $(c,s) \in S$, o processamento é inteiramente semelhante);

P_{3322}. Determine todos os pares (c,s)∈S tais que a distância de l a s é menor do que 50km;

P_{3323}. Considere o par (n,l) e todos os pares (c,s) encontrados como processados.

Novamente a escolha entre os planos depende de uma estimativa do custo de cada um, discussão que está fora do escopo deste texto.

7.3.3 Exemplo de processamento de superposição de geo-campos

Considere agora que o banco de dados geográfico contém dois geo-campos temáticos, SOLOS e VEGETAÇÃO, indicando os tipos de solo e de vegetação do Estado de São Paulo. Suponha que os geo-campos possuam representações por subdivisão planar. Suponha ainda, por simplicidade, que as duas representações utilizem a mesma escala e projeção de tal forma que seja possível operar sobre elas sem conversão.

Considere agora a consulta Q, formulada como:

Q. *Crie um geo-campo temático com o remanescente de mata atlântica em latossolo roxo na região B*

onde B é uma *janela*, ou seja, um retângulo definido por um par (ie,sd), no mesmo sistema de coordenadas cartográficas das representações, indicando o canto inferior esquerdo e superior direito do retângulo. Esta consulta difere das anteriores por criar um campo temático a partir de dois outros armazenados no banco de dados, tendo uma restrição dada por um retângulo.

Discutiremos a execução de apenas um plano, P_1, para esta consulta:

P_{11}. Utilizando SOLOS, crie o campo temático SOLOS-B indicando os tipos de solo em B;

P_{12}. Utilizando VEGETAÇÃO, crie o campo temático VEGETAÇÃO-B indicando os tipos de vegetação em B;

P_{13}. Crie o campo temático VEGET-REM, resposta da consulta, aplicando uma variante da operação de superposição de campos a SOLOS-B e VEGETAÇÃO-B;

Ao contrário dos exemplos anteriores em que a otimização do plano dependia da existência de índices espaciais, o processamento será facilitado aqui se supusermos a existência de estruturas de

armazenamento eficientes para as representações que facilitem recuperar as parcelas das representações que cobrem o retângulo B. Suponha então que as representações envolvidas sejam armazenadas em estruturas que permitam trazer para memória principal apenas as páginas que contêm os dados de vegetação ou solos na região B ou, pelo menos, em uma região B' que cubra B e seja significativamente menor do que a região abrangida pelas representações.

Em uma implementação mais simples, o subsistema de armazenamento recuperará as páginas apropriadas, filtrando-as se necessário, e materializará as representações vetoriais dos campos SOLOS-B e VEGETAÇÃO-B. Sobre estes campos intermediários, o processador de consultas aplicará então a operação de superposição de representações apropriada para criar o campo temático VEGET-REM, resposta da consulta.

Já em uma implementação mais sofisticada, o subsistema de armazenamento tratará os campos SOLOS-B e VEGETAÇÃO-B como *visões* não materializadas dos campos originais. Mais explicitamente, quando o processador de consultas executar a operação de superposição de representações, o subsistema de armazenamento recuperará, para memória principal, as páginas contendo os dados de vegetação ou solos à medida que se tornem necessárias para a operação, sendo de fato criada apenas a representação do campo temático VEGET-REM.

7.4 Computação de operações básicas

Conforme discutido brevemente na Seção 7.2, dada uma consulta Q, o processador de consultas é responsável por preparar um plano de execução para Q, consistindo de operações de mais baixo nível, implementadas pelo subsistema de armazenamento.

Esta seção aborda então implementações alternativas para duas das principais operações que um subsistema de armazenamento deve oferecer, que correspondem diretamente à seleção e à junção espaciais. As implementações dependem da utilização ou não de índices espaciais e, conseqüentemente, exibem desempenho bastante diferenciado.

7.4.1 Computação de seleções espaciais

Uma *seleção espacial* é uma expressão da forma σ[B(x)], onde B(x), chamado de *predicado de seleção*, é uma expressão booleana com apenas uma variável livre, x, tal que B(x) envolve operações espaciais. Dado um conjunto S de geo-objetos, σ[B(x)](S)=S' se e somente se S' é o conjunto de todos os objetos s∈S tais que B(s) é verdadeira.

Um exemplo de seleção espacial é a expressão σ[`distancia(x,p)<100`], onde x é a variável livre e p é uma constante do tipo ponto.

Seja S um conjunto de geo-objetos no que se segue. Usaremos r.e.m. para abreviar o termo "retângulo envolvente mínimo".

A estratégia mais simples para computar σ[B(x)](S), denominada *seleção por pesquisa exaustiva*, é apresentada em pseudo-código na Figura 7.2. Consiste em trazer a representação geométrica de cada objeto s∈S para memória principal e testar se B(s) é verdadeira. Possui custo proporcional à cardinalidade de S, se os objetos de S não estiverem agrupados em memória secundária, ou proporcional ao número de páginas ocupadas por S, em caso contrário. Esta estratégia é adotada quando não há índices para S, ou quando a cardinalidade de S é suficientemente pequena para não justificar estratégias mais sofisticadas.

```
SELEÇÃO_POR_PESQUISA_EXAUSTIVA(S,B;R)
S  - conjunto de geo-objetos
B  - expressão booleana envolvendo comparações
     espaciais com apenas uma variável livre x
R  - subconjunto de S dos objetos satisfazendo B
Inicio
  R = ∅
  Para cada s em S faça
     Inicio Recupere s para memória principal
            Se s satisfaz B, acrescente s a R
     Fim
Fim
```

Figura 7.2 – Seleção por pesquisa exaustiva.

Conforme discutido no Capítulo 6, quando a cardinalidade esperada de S é suficientemente grande e a freqüência de acesso a S é alta, costuma-se definir um índice para acelerar o acesso aos objetos de S.

A estratégia para computar σ[B(x)](S), denominada *seleção por índice,* é apresentada em pseudo-código na Figura 7.3 e possui duas etapas. A *etapa de filtragem* utiliza um índice para selecionar parte dos objetos de S, criando um conjunto P de apontadores para objetos em S. A *etapa de refinamento* recupera para memória principal a representação geométrica de cada objeto s apontado por P, acrescentando s à resposta, se B(s) for verdadeira.

```
   SELEÇÃO_POR_INDICE(S,B,I,C;R)
S  -  conjunto de geo-objetos
B  -  predicado de seleção
I  -  índice sobre S
C  -  expressão booleana envolvendo comparações
      espaciais com apenas uma variável livre
      que pode ser resolvida por I
R  - conjunto dos objetos de S que satisfazem B
Inicio
   R, P = ∅
   FILTRO(I,C;P)              % Etapa 1: Filtragem
   Para cada p em P faça      % Etapa 2: Refinamento
   Inicio
      Recupere o objeto s de S apontado por p
      Se s satisfizer B, acrescente s a R
   Fim
Fim
FILTRO(I,C;P)
Inicio
   P = ∅
   Acrescente a P todos os apontadores para
      objetos de S que satisfazem C, usando I

Fim
```

Figura 7.3 – Seleção por índice.

Normalmente, o índice não resolve B diretamente, mas uma outra expressão booleana C, também com uma variável livre, tal que:

(1) para todo geo-objeto s, se C(s) = falso então B(s) = falso

Assim, se um geo-objeto s não é selecionado pelo índice (ou seja, C(s) é falso), então s não pertence à resposta da seleção (ou seja, B(s) também é falso). Porém, o índice pode selecionar um objeto s que não pertence à resposta (ou seja, pode existir um geo-objeto s tal que $C(s) \wedge \neg B(s)$).

O resto desta seção refina a função auxiliar FILTRO para árvores-R.

Dados dois inteiros $m>1$ e $n>1$, recorde que uma árvore-R de ordem (m,n) para S é uma árvore de busca balanceada tal que (veja exemplo na Figura 7.4):

- todas as folhas estão no mesmo nível;
- cada folha contém entre $\lceil n/2 \rceil$ e n entradas;
- cada nó interior contém entre $\lceil m/2 \rceil$ e m filhos, exceto a raiz, que contém entre 2 e m filhos;
- para cada objeto $s \in S$, existe uma (única) entrada em uma (única) folha M, consistindo de um ponteiro para s em conjunto com o r.e.m. de s; dizemos que s é *apontado* por M;
- cada nó interior N contém uma entrada para cada filho M de N, consistindo de um ponteiro para M e o r.e.m. de M, ou seja, o r.e.m. de todos os retângulos armazenados em M.

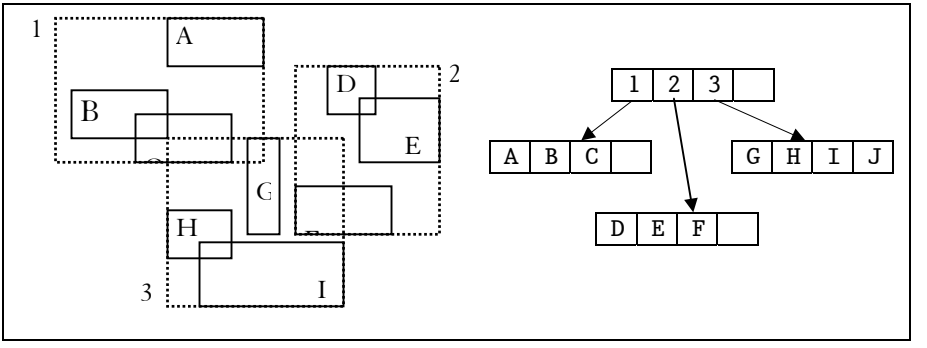


Figura 7.4 – Exemplo de uma árvore-R.

A computação de $\sigma[B(x)](S)$ é guiada pelos retângulos armazenados na árvore-R, dependendo evidentemente da expressão booleana B.

Considere, por exemplo, a computação da seleção σ[`contem(x,p)`](S), onde p é uma constante do tipo ponto.

Voltando à Figura 7.4, para determinar quais objetos s∈S contém p, proceda da seguinte forma. Observe inicialmente que os retângulos 1 e 3, armazenados na raiz, contêm p; estes retângulos correspondem ao primeiro e ao segundo filho da raiz. Em seguida, repita o processo descendo por estes nós. Assim, descendo pelo primeiro filho da raiz, observe que o retângulo C contém p e acrescente o ponteiro a ele associado a uma lista temporária L. Descendo pelo terceiro filho da raiz, observe que nenhum retângulo aí armazenado contém p. Como último passo, recupere para memória principal o objeto s apontado pelo (único) ponteiro em L e teste se s de fato contém p; se contiver, σ[`contem(x,p)`](S) ={s}; caso contrário, σ[`contem(x,p)`](S)=∅.

De forma geral, o uso de árvores-R como índices coloca um desafio adicional pois as folhas da árvore armazenam retângulos aproximando a geometria dos objetos, e não a geometria em si. Além disto, cada nó interior N contém uma entrada para cada filho M de N, consistindo de um ponteiro para M e o r.e.m. dos retângulos armazenados em M.

Portanto, para que árvores-R possam ser utilizadas na etapa de filtragem, devemos definir um segunda expressão booleana C', que chamaremos de *companheira* de C, tal que:

(2) para todo geo-objeto s, para todo retângulo r tal que r é igual ou contém o r.e.m. de s, se C'(r) = falso então C(s) = falso

Dizemos que uma expressão booleana C(x), com uma variável livre x varrendo geo-objetos, *pode ser filtrada por árvores-R* se somente se é possível definir uma expressão booleana companheira para C.

Note que C' não é a expressão booleana C da Figura 7.3. Porém, de (1) e (2) podemos deduzir diretamente que

(3) para todo geo-objeto s, para todo retângulo r tal que r é igual ou contém o r.e.m. de s, se C'(r) = falso então B(s) = falso

Ou seja, apesar de trabalhar com retângulos, através da noção de expressão companheira, as arvores-R podem ser utilizadas como filtros.

Freqüentemente, C' é a própria expressão C. Por exemplo, se tivermos C(x)=`contem`(x,k), onde x é uma variável varrendo geo-objetos e k uma constante espacial, então C(r)=`contem`(r,k), para todo retângulo que contém ou é igual ao r.e.m. de x.

Porém, isto nem sempre é verdade. Tome, por exemplo, C(x)=`toca`(x,k), onde x é uma variável varrendo geo-objetos e k uma constante espacial. Neste caso, C(x) não implica em C(r), para todo retângulo que contém ou é igual ao r.e.m. de x. Por exemplo, teríamos que definir C' como:

C'(r) = `superpõe`(r,k) $\vee$ `toca`(r,k)

Em outras situações, não é realmente possível definir C' de tal forma a tornar árvores-R úteis para filtrar C(x). Por exemplo, tomemos C(x) = `disjunto`(x,k), onde x é uma variável varrendo geo-objetos e k uma constante espacial. Para este operador, teríamos que definir C' como:

C'(r) = `disjunto`(r,k) $\vee$ `superpõe`(r,k)

o que seria inútil como filtro.

De posse destas noções, o refinamento da função auxiliar FILTRO para árvores-R é imediato e mostrado na Figura 7.5. Note que esta função continua a receber a expressão booleana C sobre objetos de S, por compatibilidade com FILTRO, e internamente determina (por um processo não especificado) a expressão C' companheira de C, que supõe-se existir.

O pseudo-código apresentado consiste de um caminhamento em amplitude na árvore, com o auxílio da variável Q, eliminando os nós cujos r.e.m. não satisfazem C', e retornando os apontadores para objetos, contidos nas folhas, cujos r.e.m.'s satisfazem C'. Portanto, consistentemente com a estratégia de seleção por índice, esta função não acessa a geometria dos objetos.

Por fim, simplificadamente, o custo de uma pesquisa por índice, utilizando árvores-R, é proporcional ao número de nós da árvore-R lidos, somado ao número de objetos apontados pela árvore-R cujo retângulo envolvente satisfaz C'. Uma análise detalhada pode ser encontrada em (Aboulnaga e Naughton, 2000; Kriegel et al., 1990; Ciferri et al., 1995).

```
FILTRO_ARVORE_R(A,C;P)

A  -  apontador não nulo para raiz de uma árvore-R
      sobre um  conjunto de geo-objetos S.
      Cada nó N é uma estrutura da forma:
      lt(N)   –  lista de entradas;
      rem(N)  –  r.e.m. das entradas em lt(N)
      Cada entrada E é uma estrutura da forma:
      Nó interior:
      pt(E)   –  apontador para a raíz de uma
                 sub-árvore de A
      rem(E)  –  retângulo envolvente mínimo da
                 sub-árvore apontada por E
      Folha:
      pt(E)   –  apontador para um objeto em S
      rem(E)  –  retângulo envolvente mínimo
                 do objeto apontado por E
C  -  expressão booleana envolvendo comparações
      espaciais com apenas uma variável livre
      que pode ser filtrada por A
P  -  conjunto de apontadores para objetos de S

Inicio
   Determine a expressão C' companheira de C
   P = ∅
   Q = {A}
   Enquanto Q ≠ ∅ faça:
   Inicio
      Selecione q em Q, removendo-o de Q
      Se q for uma folha,
         Para cada entrada E em ls(q),
            Se C'(rem(E)), acrescente pt(E) a P
      Senão
         Para cada entrada E em ls(q),
            Se C'(rem(E)), acrescente pt(E) a Q
   Fim
Fim
```

Figura 7.5 – Filtro por árvores-R.

7.4.2 Computação de junções espaciais

Uma *junção espacial* é uma expressão da forma $\chi[J(x,y)]$, onde J(x,y), chamado de *predicado de junção*, é uma expressão booleana envolvendo apenas comparações espaciais com duas variáveis livres x e y. Dados dois conjuntos S e T de geo-objetos, $\chi[J(x,y)](S,T)=R$ se e somente se R é o conjunto de todos os pares de objetos $(s,t)\in S\times T$ tais que J(s,t) é verdadeira.

Um exemplo de junção espacial é a expressão `χ[distancia(x,y)<100]`, onde x e y são variáveis. O predicado de junção define o conjunto de pares de geo-objetos, x e y, que estão a menos de 100 e a mais de 50 (metros) um do outro.

Computar $\chi[J(x,y)](S,T)$ é mais complexo do que computar junções não espaciais pois não há uma ordem total entre geo-objetos que preserve proximidade espacial. Ou seja, não há uma função que mapeie qualquer par (S,T) de conjuntos de geo-objetos em uma seqüência Q de geo-objetos tal que todo objeto em $S\cup T$ ocorre uma única vez em Q e, para todo $(s,t)\in S\times T$, se s e t estão espacialmente próximos, então s e t estão próximos em Q. Esta característica dos geo-objetos afeta as estratégias tradicionais para computar junção, tornando-as inaplicáveis ou menos eficientes.

Esta seção discute algumas estratégias para computar junção espacial, seguindo basicamente (Brinkhoff et al., 1993) (Brinkhoff et al., 1994) (Gunther et al., 1993) (Patel e DeWitt, 1996).

Considere inicialmente a estratégia de *junção por pesquisa exaustiva* para computar $\chi[J(x,y)](S,T)$, apresentada na Figura 7.6. Esta estratégia consiste em duas iterações aninhadas varrendo todos os pares de objetos (s,t) em $S\times T$, testando J(s,t) para cada um deles. Ou seja, consiste de uma pesquisa exaustiva sobre $S\times T$, o espaço de busca de $\chi[J(x,y)](S,T)$.

Esta estratégia levanta dois problemas básicos relativos à computação de uma junção espacial. Primeiro, a transferência dos geo-objetos da memória secundária para memória principal pode ser uma operação de alto custo, quando as geometrias dos objetos são extensas. Esta transferência pode, de fato, forçar a liberação de geo-objetos de T para permitir a reutilização de espaço na área de trabalho em memória

principal. Desta forma, o mesmo objeto em T poderá ser transferido várias vezes para memória principal. No pior caso, cada objeto em T será lido n vezes, onde n é a cardinalidade de S. Assim, o próprio aninhamento das iterações torna-se ainda mais ineficiente.

Segundo, o teste do predicado de junção pode também ser uma operação de alto custo, principalmente se a geometria dos objetos consistir de um grande número de pontos. O problema é agravado pelo fato de que este teste será realizado para todos os pares $(s,t) \in S \times T$.

```
JUNÇÃO_POR_PESQUISA_EXAUSTIVA(S,T,J;R)

S,T  - conjuntos de objetos espaciais
J    - expressão booleana envolvendo comparações
       espaciais com duas variáveis livres
R    - conjunto de pares de objetos em S×T que
       satisfazem J

Inicio
  R = ∅
  Para cada s em S faça
  Inicio
     Recupere   s de memória secundária
                para memória principal
     Para cada t em T faça
     Inicio
        Se t não está em memória principal,
          recupere t  de memória secundária
                      para memória principal
        Se J(s,t),
          acrescente (s,t) a R
     Fim
   Fim
Fim
```

Figura 7.6 – Junção por pesquisa exaustiva.

Uma segunda estratégia para computar $\chi[J(x,y)](S,T)$, chamada de *junção por particionamento e intercalação* (Patel e DeWitt, 1996), opera em duas etapas, filtragem e refinamento, sem recorrer a índices. De forma semelhante à seleção espacial, assumiremos que a etapa de

filtragem aplica-se a um predicado K(x,y) tal que, para todo x, y, se K(x,y) é verdadeiro então J(x,y) também é verdadeiro. Além disto, o predicado K deve ser tal que:

(4) para todo par (s_1,s_2) de geo-objetos, com r.e.m.'s r_1 e r_2 respectivamente, se r_1 e r_2 não se superpõem então $K(s_1,s_2)$ = falso

O passo inicial da etapa de filtragem cria e armazena em memória secundária dois conjuntos temporários, S^0 e T^0, da seguinte forma. Cada geo-objeto $s \in S$ é lido para memória principal, gerando um objeto $s^0 \in S^0$ da forma $s^0=(p,r)$, onde p é o identificador de s em memória secundária, denotado $id(s^0)$, e r é o r.e.m. de s, denotado $rem(s^0)$. O mesmo processo é aplicado a T, gerando T^0.

O próximo passo desta etapa determina todos os pares de objetos $(s^0,t^0) \in S^0 \times T^0$ tais que $rem(s^0)$ e $rem(t^0)$ se superpõem. Para cada par (s^0,t^0) que passar no teste, o par de identificadores $(id(s^0),id(t^0))$ é acrescentado à resposta desta fase.

Se os conjuntos temporários S^0 e T^0 puderem ser simultaneamente trazidos para memória principal, uma técnica de varredura do plano (Preparata e Shamos, 1988) poderá ser adotada para determinar quais retângulos se superpõem. Por exemplo, um conjunto de retângulos pode ser varrido em ordem crescente da coordenada do canto inferior esquerdo, como ilustra a Figura 7.7. Um algoritmo, baseado em uma técnica de varredura, para determinar quais pares de retângulos $(r,s) \in A_1 \times A_2$ se superpõem é apresentado na Figura 7.8.

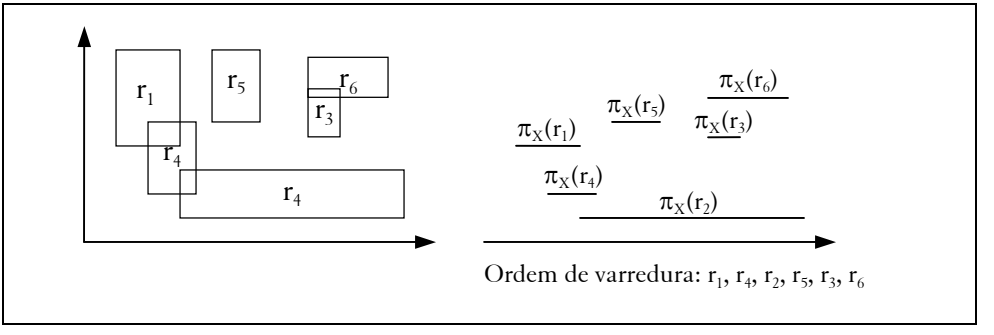


Figura 7.7 – Exemplo de varredura do plano para retângulos.

```
VARREDURA_DO_PLANO(A1,A2;B)
A1,A2 -  conjuntos de retângulos com lados paralelos
         aos eixos x e y;
         xe(r) e xd(r) denotam as coord. x
         dos vértices à esquerda e à direita de r
B     -  (r,s)∈A1×A2 tal que r e s se sobrepõem
Inicio
   R = ∅
   C = A1 ∪ A2
   Ordene C crescentemente por xe(r)
   Enquanto C ≠ ∅ faça
   Inicio
      Seja r∈C com menor valor de xe(r)
      Remova r de C e suponha que r∈Ai
      % A(i+1) : soma módulo 2
      Para cada s∈A(i+1) até que xe(s)>xd(r) faça
      % xe(r)≤xe(s)≤xd(r): r e s se superpõem no eixo x
        Se r e s também se superpõem no eixo y,
            acrescente (r,s) a B
   Fim
Fim
```

Figura 7.8 – Técnica de varredura para determinar superposição de retângulos.

Se os conjuntos não puderem ser trazidos para memória principal, então cada conjunto deve ser particionado em p conjuntos não necessariamente disjuntos $S^1,...,S^p$ e $T^1,...,T^p$, respectivamente. Estes novos conjuntos são tais que S^k e T^k, para $k=1,...,p$, podem ser simultaneamente trazidos para memória e, para todo $(s^0,t^0)\in S^0\times T^0$ tais que $rem(s^0)$ e $rem(t^0)$ se superpõem, se $s^0\in S^k$ então $t^0\in T^k$.

A criação destas partições segue a seguinte tática (Patel e DeWitt, 1996):

(a) Ao criar S^0 e T^0, determine R, o r.e.m. de todos os retângulos em S^0 e T^0.

(b) Considerando o tamanho da área de trabalho disponível em memória principal e o tamanho de cada elemento em S^0 e T^0,

determine o número p de partes em que R deve ser dividido, criando retângulos $R^1,...,R^p$ (ver Figura 7.9).

(c) Os conjuntos $S^1,...,S^p$ e $T^1,...,T^p$ são criados da seguinte forma:

para cada $k \in [1,p]$, para cada $s^0 \in S^0$,
$s^0 \in S^k$ se e somente se $rem(s^0)$ e R^k se superpõem

para cada $k \in [1,p]$, para cada $t^0 \in T^0$,
$t^0 \in T^k$ se e somente se $rem(t^0)$ e R^k se superpõem

O número p de partes em que R deve ser dividido pode ser estimado da seguinte forma.

Sejam $|S^0|$ e $|T^0|$ as cardinalidades de S^0 e T^0, respectivamente. Seja M o espaço disponível em memória principal e N o tamanho de cada objeto em S^0 e T^0. Como é necessário manter S^k e T^k simultaneamente em memória principal, temos que:

$$p = \lceil (|S^0| + |T^0|) * N/M \rceil$$

Após o particionamento, para cada $k \in [1,p]$, os conjuntos S^k e T^k são suficientemente pequenos para serem simultaneamente trazidos para memória principal.

Novamente a técnica de varredura do plano pode ser adotada para determinar todos os pares de objetos $(s^k,t^k) \in S^k \times T^k$ tais que $rem(s^k)$ e $rem(t^k)$ se superpõem. Para cada par (s^k,t^k) que passar no teste, $(id(s^k),id(t^k))$ é acrescentado à resposta desta fase. Isto conclui a etapa de filtragem. O resultado é um conjunto F de pares de identificadores de objetos em $S^0 \times T^0$.

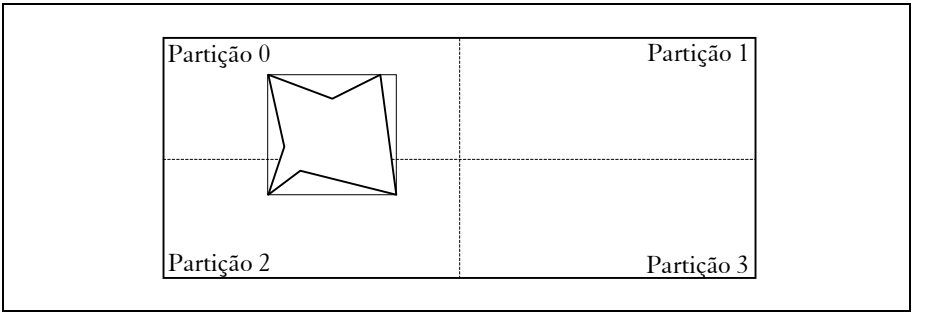


Figura 7.9 – Divisão dos conjuntos originais contendo os r.e.m.'s.

A etapa de refinamento procede da seguinte forma. Para evitar acesso aleatório aos objetos de S e T, os pares no resultado F da etapa de filtragem são ordenados tendo como chave a primeira coordenada seguida da segunda coordenada. Duplicatas são eliminadas neste processo, gerando o conjunto G. Em seguida, para cada par $(d,e) \in G$, o objeto $s \in S$ tal que $d = id(s)$ é trazido para a área de trabalho, se já lá não estiver. Para cada par $(f,g) \in G$ tal que $f = d$, o objeto $t \in T$ tal que $g = id(t)$ também é trazido para a área de trabalho. Se $J(s,t)$ for satisfeito, então o par (s,t) é finalmente adicionado à resposta de $\chi[J(x,y)](S,T)$.

Como na seleção espacial, uma outra forma de reduzir o custo de computar uma junção espacial $\chi[J(x,y)](S,T)$ consiste em utilizar índices.

Uma estratégia genérica, denominada *junção por índice*, também possui duas etapas, exceto que a estratégia utiliza um índice para a etapa de filtragem. A Figura 7.10 apresenta a estratégia em pseudo-código e a Figura 7.11 ilustra a estratégia esquematicamente.

De acordo com (Brinkhoff et al., 1994), a Etapa 2 é a de maior custo, tanto em termos de acesso a memória secundária para recuperar a geometria exata dos objetos, quando em termos de tempo de processamento para computar o predicado de junção.

Note que a estratégia, como apresentada na Figura 7.10, é apenas um *framework*, que deve ser refinado com implementações específicas em cada etapa. Para a etapa de filtragem, por exemplo, (Brinkhoff et al., 1993; Brinkhoff et al., 1994) usam árvores-R* e (Patel e DeWitt, 1996; Lo e Ravishankar, 1996) propõem uma estrutura baseada em *hash*. Já para a etapa de refinamento, (Brinkhoff et al., 1994) adota um algoritmo de varredura do plano, conforme anteriormente apresentado.

O resto desta seção discute o refinamento da etapa de filtragem para o caso em que os índices são árvores-R. (Brinkhoff et al., 1993; Brinkhoff et al., 1994; Gunther et al., 1993).

Seja $K(x,y)$ o predicado a ser filtrado pelas árvores-R. Novamente, para que árvores-R possam ser utilizadas, devemos definir uma segunda expressão booleana K', que chamaremos de *companheira* de K, tal que:

(5) para todo par (s_1,s_2) de geo-objetos, para todo par (r_1,r_2) de retângulos tais que r_i é igual ou contém o r.e.m. de s_i, para i=1,2, se $K'(r_1,r_2)$ = falso então $K(s_1,s_2)$ = falso

O refinamento proposto, apresentado como *filtro de junção por árvore-R* na Figura 7.12, consiste de uma dupla pesquisa por amplitude nas duas árvore-R. A variável Q contém apenas os pares de retângulos (r_1,r_2) tais que $K'(r_1,r_2)$=verdadeiro. Pela propriedade acima de K' e pela construção das árvores-R, para todo par $(s,t)\in S\times T$, se K(s,t)=verdadeiro, então $(pt(s),pt(t))\in P$, onde P é a resposta devolvida pelo filtro (mas o converso é falso, ou seja, poderá existir $(s,t)\in S\times T$ tal que K(s,t)=falso e $(pt(s),pt(t))\in P$ pelo próprio fato das árvores-R trabalharem com aproximações, ou seja, não serem filtros exatos.

```
JUNÇÃO_POR_INDICES(S,T,J,U,V,K;R)

S,T  - conjuntos de objetos espaciais
J    - expressão booleana envolvendo comparações
       espaciais com duas variáveis livres
U,V  - índices sobre S e T, respectivamente
K    - expressão booleana envolvendo comparações
       espaciais com duas variáveis livres
       a ser filtrada por U e V
R    - conjunto de pares de objetos em S×T
       que satisfazem J
Inicio
   R = ∅
   % Etapa 1: Filtragem por índice
   - Use os índices espaciais U e V sobre S e T
     para computar o conjunto R1 de pares de retângulos, u e
     v, tais que K(u,v)
   % Etapa 2: Refinamento
   - Para cada entrada (u,v) em R2, acesse as geometrias do
     par de objetos (s,t) correspondente a (u,v)
   - se J(s,t), acrescente (s,t) à resposta R
Fim
```

Figura 7.10 – Junção por índices.

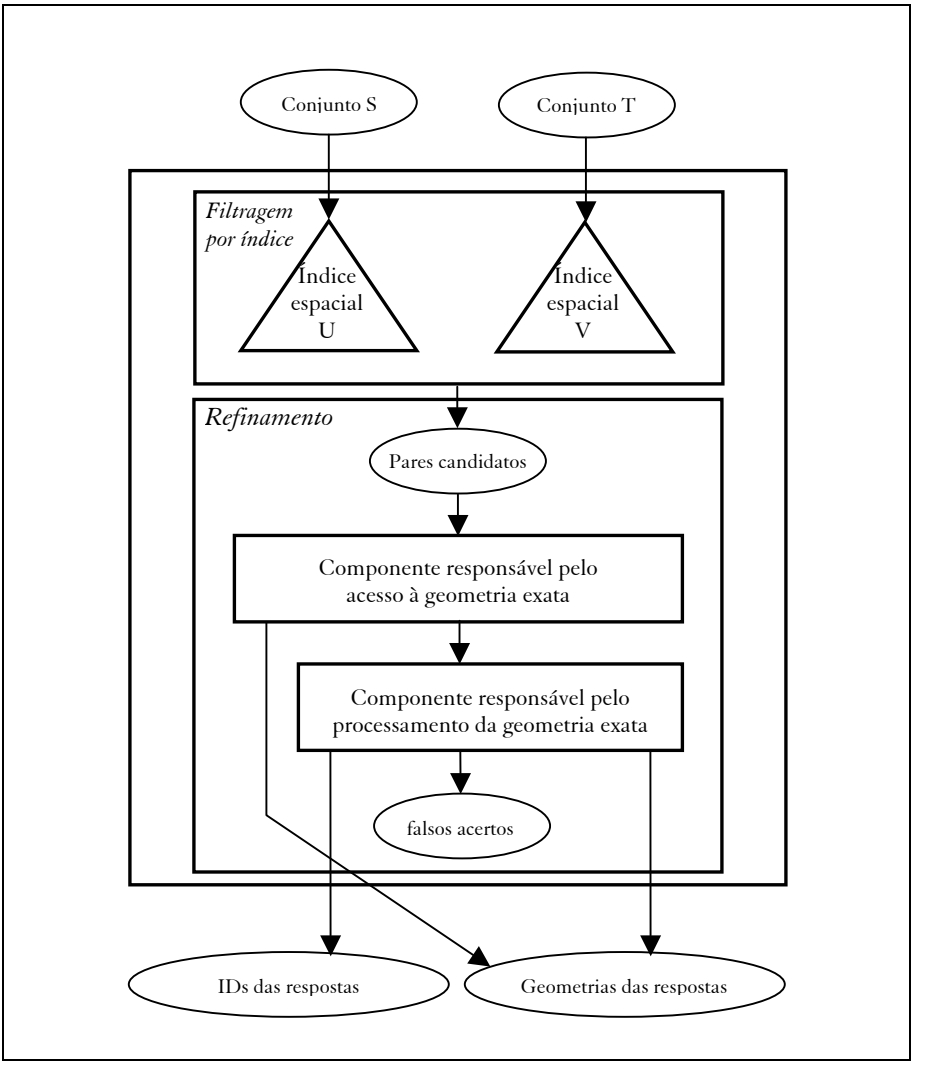


Figura 7.11 – Esquema da junção por índices. Fonte: adaptado de Kriegel et al., (1993).

```
FILTRO_DE_JUNCAO_POR_ARVORE_R(A,B,K;P)

A,B- apontadores para as raízes de duas árvores-R
     sobre conjuntos de geo-objetos S e T
     (com a estrutura indicada na Figura 7.5).
     Supõe-se que A e B não sejam nulos.
K  - expressão booleana envolvendo comparações
     espaciais com duas variáveis livres
     que pode ser filtrada por árvores-R
P  - conjunto de pares de apontadores para
     objetos de S e T

Inicio
   Determine a expressão K' companheira de K
   P = ∅
   Q = {(A,B)}
   Enquanto Q ≠ ∅ faça:
   Inicio
      Selecione (p,q) em Q, removendo-o de Q
      Se p e q forem folhas,
         Para cada entrada E em ls(p),
            Para cada entrada F em ls(q),
               Se K'(rem(E),rem(F)),
                  acrescente (pt(E),pt(F)) a P
      Se p for folha e q for nó interior,
         Para cada entrada F em ls(q),
            Se K'(rem(p),rem(F)),
               acrescente (p,pt(F)) a Q
      Se q for folha e p for nó interior,
         Para cada entrada E em ls(p),
            Se K'(rem(E),rem(q)),
               acrescente (pt(E),q) a Q
   Fim
Fim
```

Figura 7.12 – Filtro de junção por árvore-R.

7.4.3 Computação de superposições de mapas temáticos

Um polígono é *simples* se e somente se não há um par de arestas não-consecutivas compartilhando um ponto. Um *polígono simples com furos* é um polígono simples ou um par (p,F) onde p é um polígono simples e F é um conjunto não vazio de polígonos simples com furos tais que, para todo $q \in F$, q está contido no interior de p e, para todo q, $q' \in F$, q e q' são disjuntos.

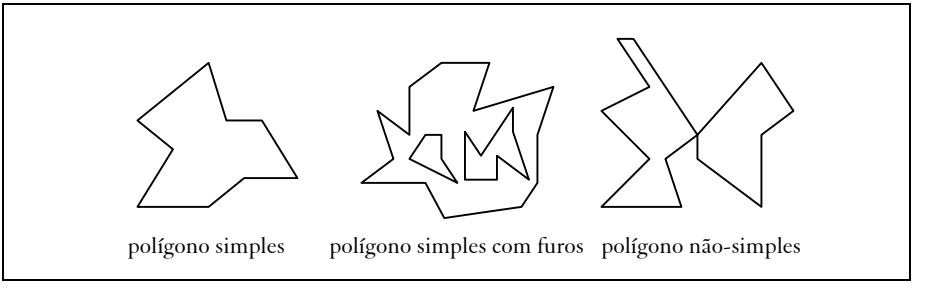


Figura 7.13 – Exemplos de polígonos (Kriegel et al., 1992).

Seja $\wp$ o conjunto dos polígonos simples com furos. Um *conjunto de temas* é qualquer conjunto finito não vazio. Um *mapa temático vetorial* sobre um conjunto de temas T é uma função parcial $M: \wp \rightarrow T$ tal que, para todo $p,q \in dom(M)$, p e q são disjuntos. Denotaremos por $\Im[T]$ o conjunto dos mapas temáticos sobre um conjunto de temas T.

Esquemas de armazenamento, em memória secundária, para mapas temáticos vetoriais são variantes dos esquemas de armazenamento de polígonos onde cada polígono possui um atributo indicando o seu tema. Desta forma, índices para conjuntos de polígonos podem ser utilizados diretamente para indexar os domínios dos mapas temáticos.

Sejam T, U e V conjuntos de temas e $f: T \times U \rightarrow V$ uma função total. A *superposição de mapas temáticos induzida por* f é a função $\theta[f]: \Im[T] \times \Im[U] \rightarrow \Im[V]$ definida como $\theta[f](M,N) = O$ se e somente se

- dom(O) é o conjunto dos polígonos com furos obtidos pela interseção dois-a-dois de um polígono em dom(M) com um polígono em dom(N);
- $O(p) = f(q,r)$ se e somente se p faz parte da interseção de q e r

Há três questões que devem ser levadas em consideração na computação da superposição $\theta[f](M,N)=O$ de dois mapas temáticos vetoriais: (1) M e N são normalmente grandes e, portanto, devem ser trazidos progressivamente para memória principal; (2) além da geometria, é necessário armazenar e recuperar o tema de cada polígono nos domínios de M e N; (3) a etapa de refinamento deve ser seguida por um pós-processamento para computar os polígonos no domínio do mapa resultante O e o valor associado a cada um destes polígonos.

Apesar destas questões, estratégias para computar $\theta[f](M,N)$ podem ser obtidas modificando-se as estratégias para junção espacial apresentadas anteriormente, considerando-se como predicado de junção a superposição de polígonos, e acrescentando-se uma etapa de pós-processamento como indicado acima. De fato, variantes da estratégia para computar $\theta[f](M,N)$ denominada *particionamento por varredura do plano* (Kriegel et al., 1992) podem ser derivadas da junção por particionamento e intercalação e da junção por índice.

7.5 Gerência da área de trabalho

7.5.1 Principais questões na gerência da área de trabalho

A definição de uma estratégia para gerência da área de trabalho em memória principal suscita algumas questões importantes, discutidas nesta seção, para que efetivamente contribua para um melhor desempenho do processador de consultas.

No que segue, usaremos o termo objeto para designar indistintamente, páginas físicas, objetos lógicos ou conjuntos destes.

Em primeiro lugar, há a questão do tipo de objeto – físico ou lógico - que será mantido na área de trabalho. Normalmente, os sistema de gerência de banco de dados objeto-relacionais trabalham com páginas físicas na área de trabalho, enquanto que os sistemas orientados-a-objeto estritos podem trabalhar com páginas físicas, objetos lógicos ou conjuntos de objetos. Uma estratégia de gerência da área de trabalho é chamada de *semântica* quando trabalha com objetos lógicos ou conjuntos de objetos e utiliza características destes objetos ou conjuntos para implementar seus algoritmos.

Associada a esta primeira questão, está a decisão de que nível de granularidade a estratégia de gerência adota. A estratégia pode controlar páginas físicas ou objetos lógicos individualmente, ou trabalhar com *segmentos físicos*, compostos por páginas físicas e *segmentos semânticos*, compostos por objetos lógicos.

A terceira questão é a política de substituição dos objetos mantidos na área de trabalho. A maioria dos sistemas usa uma variante da estratégia usualmente chamada *LRU – Least Recentently Used,* que propõe substituir o objeto que está há mais tempo na área de trabalho. Uma variante desta estratégia será discutida na seção seguinte.

Além destas, há a questão de criar políticas de antecipação que tragam objetos para a área de trabalho antes de serem solicitados por uma consulta. Naturalmente, uma boa política de antecipação pode acelerar o processamento de consultas, mas depende de um bom entrosamento com o processador de consultas.

7.5.2 Área de trabalho utilizando segmentos semânticos

Esta seção discute as características básicas de estratégias para gerência da área de trabalho que a organizem em segmentos lógicos, seguindo (Ren et al., 2003).

Ao processar uma nova consulta Q, o relacionamento entre o seu resultado e os segmentos lógicos na área de trabalho recai nos 4 casos mostrados na Figura 7.14. No caso 4, o resultado de Q está inteiramente contido na área de trabalho e nenhum novo objeto precisa ser trazido de memória secundária. Porém, informação adicional mantida junto aos segmentos lógicos poderá ser atualizada para refletir o processamento de Q. Em todos os outros casos, o resultado de Q deverá ser, em parte ou na sua totalidade, adicionado à área de trabalho e, novamente, informação adicional será gerada ou atualizada.

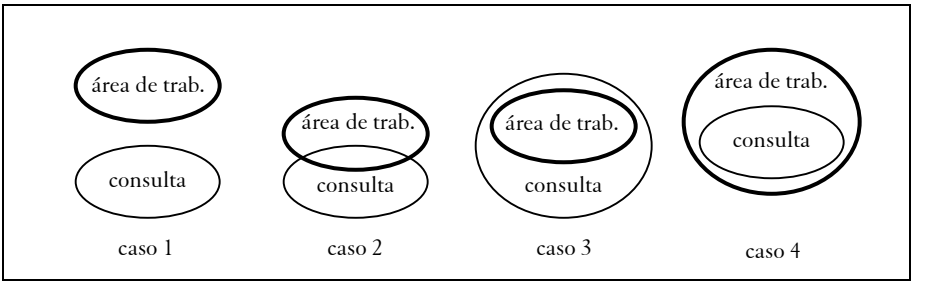


Figura 7.14 – Relacionamentos entre o resultado de uma nova consulta e a área de trabalho (Ren et al., 2003).

Mais precisamente, suponha que o conteúdo da área de trabalho possa ser representado pelo predicado C e seja R o resultado de uma consulta Q, com predicado de consulta Q_P.

A *política de inclusão* determina como tratar R em presença de C:

1. $(\neg Q_P \Rightarrow C)$: o resultado de Q deve ser *inteiramente incluído* na área de trabalho (Caso 1 da Figura 7.14).
2. $(Q_P \Rightarrow C)$: o resultado de Q deverá ser *ignorado* pois está inteiramente contido na área de trabalho (Caso 4 da Figura 7.14).
3. Nenhum dos casos acima: o resultado de Q deverá ser *parcialmente incluído* na área de trabalho (Casos 2 e 3 da Figura 7.14).

Usualmente, as implementações da política de inclusão testam o resultado de uma consulta contra cada segmento semântico em separado e tentam chegar a uma conclusão.

Suponha que R, o resultado da consulta, se superpõe a um segmento semântico S.

A *política de colapsamento* determina como tratar este tipo de superposição, escolhendo entre 3 opções:

1. *Colapsar totalmente* R e S, criando um novo segmento semântico $S_1 = R \cup S$.
2. *Colapsar parcialmente* R e S, criando dois novos segmentos semânticos, $S_1 = R$ e $S_2 = S - R$.
3. *Não colapsar* R e S, criando três novos segmentos semânticos, $S_1 = R - S$, $S_2 = S - R$ e $S_3 = S \cap R$.

Colapsar totalmente R e S reduz o número de segmentos lógicos na área de trabalho, o que simplifica a sua gerência e reduz o tempo de processamento do critério de inclusão. Por outro lado, esta política tende a criar segmentos lógicos muito grandes, embora só uma parte dos objetos no segmento seja efetivamente usada. Não colapsar R e S produz o efeito inverso pois tende a gerar segmentos lógicos com uma granularidade muito fina. Colapsamento parcial é normalmente a melhor escolha pois abre a possibilidade de balancear o tamanho dos segmento lógicos.

A *política de substituição* determina qual segmento semântico deve ser removido da área de trabalho quando o resultado de uma nova consulta não puder ser acomodado.

Assumindo a política de colapsamento parcial, a política de substituição tradicional, LRU, de remover o segmento lógico que está há mais tempo na área de trabalho não é adequada pois os objetos em um segmento resultam das respostas de várias consultas e, portanto, possuem diferentes tempos de residência na área de trabalho. Neste caso, a política LRU pode ser substituída por uma versão mais sofisticada, chamada de *LRU dinâmica,* ou D-LRU (Ren et al., 2003). Brevemente, seja R novamente o resultado de uma consulta e S um segmento semântico que se sobrepõe a R. Suponha que S seja decomposto em $S_1 = R - S$, $S_2 = S - R$ e $S_3 = S \cap R$. A política D-LRU mantém a data de S para o segmento S_2 e cria S_1 e S_3 com a data de execução de Q. Se a área de trabalho não pode acomodar o resultado R da consulta, os segmentos mais antigos são descartados, como na política LRU tradicional, até que haja espaço livre suficiente.

Há estratégias mais sofisticadas que utilizam uma noção de distância semântica para selecionar o objeto a substituir, como a apresentada em (Dar et al., 1996).

7.5.3 Exemplo de estratégia de gerência da área de trabalho

Esta seção exemplifica o uso dos conceitos apresentados na seção anterior para construir uma estratégia de gerência da área de trabalho que melhore o desempenho de aplicações de exploração visual dos dados,

típicas de sistemas de informação geográfico. Esta seção segue basicamente (Doshi et al., 2003).

Suponha que, na aplicação de exploração visual dos dados, o usuário interaja diretamente com uma *janela de visualização* através das operações usuais de *aproximação, afastamento, deslocamento* e *enquadramento* (*zoom-in, zoom-out, panning, fitting*). Estas operações geram então consultas aos dados.

A área de trabalho armazena segmentos semânticos, definidos por conjuntos de geo-objetos. Cada segmento semântico está associado ao r.e.m. dos objetos nele contidos.

O resto desta seção discute, em detalhe, políticas de antecipação de acesso aos geo-objetos. Em uma aplicação de exploração visual dos dados, uma política de antecipação é especialmente interessante por duas razões: (1) o usuário passa considerável tempo analisando os dados visualizados, deixando o processador e o disco ociosos; logo, é possível utilizar o processador e o disco para pré-carregar dados sem afetar a exploração dos dados; (2) dadas as características das operações, é possível prever qual será o próximo movimento do usuário.

Em geral, uma política de antecipação deve trabalhar gradativamente, à medida que descobre novas características do comportamento do usuário ou dos dados acessados. No início do processo de exploração visual, pouca ou nenhuma informação está disponível. Com o tempo, mais características do comportamento do usuário ou dos dados acessados tornam-se evidentes, melhorando a qualidade dos dados trazidos antecipadamente para a área de trabalho.

Mais especificamente, para aplicações de exploração visual dos dados, assumimos que a política de antecipação deve detectar se o usuário: (1) tende a manter a direção da operação de deslocamento (da janela de visualização) ou alterá-la; (2) se os dados sob análise visual possuem regiões de interesse às quais o usuário tende a voltar. Baseadas nestas suposições, podemos definir as seguintes políticas de antecipação (Doshi et al., 2003): P1 – aleatória; P2 – direção de deslocamento; P3 – foco do usuário.

A política de antecipação aleatória, ilustrada na Figura 7.15(a), escolhe aleatoriamente a direção do deslocamento da janela de visualização, dando pesos iguais às quatro possíveis direções. Esta política naturalmente se aplica quando não é possível detectar características úteis no comportamento do usuário ou do acesso aos dados.

A política de antecipação baseada na direção de deslocamento, ilustrada na Figura 7.15(b), é semelhante à anterior, exceto que dá maior peso a uma direção, se o usuário moveu a janela de visualização duas vezes seguidas naquela direção.

Já a política de antecipação baseada no foco do usuário é mais sofisticada e utiliza indicações sobre o próximo deslocamento e sobre regiões de interesse do usuário. Uma região é considerada como uma *região de interesse* em uma sessão de trabalho quando o usuário a visita mais do que k vezes, onde k é um parâmetro da política. Uma região de interesse é descrita por uma data e um retângulo. Enquanto nenhuma região de interesse estiver próxima da janela de visualização, esta política trabalha de forma semelhante à política baseada na direção de deslocamento. Quando alguma região de interesse torna-se próxima à janela de visualização, esta política antecipa a carga dos objetos na direção da região de interesse, e não na direção dos últimos deslocamentos. Esta política reflete então a intuição de que o usuário provavelmente governa a visualização em direção à região de interesse e lá permanecerá por algum tempo.

Por fim, a política de substituição combina a estratégia de D_LRU com a política de antecipação. Assim, os segmentos semânticos mais antigos e fora da direção indicada pela política de antecipação devem ser descartados primeiro, até que seja liberada memória suficiente para acomodar os novos a serem trazidos para memória principal.

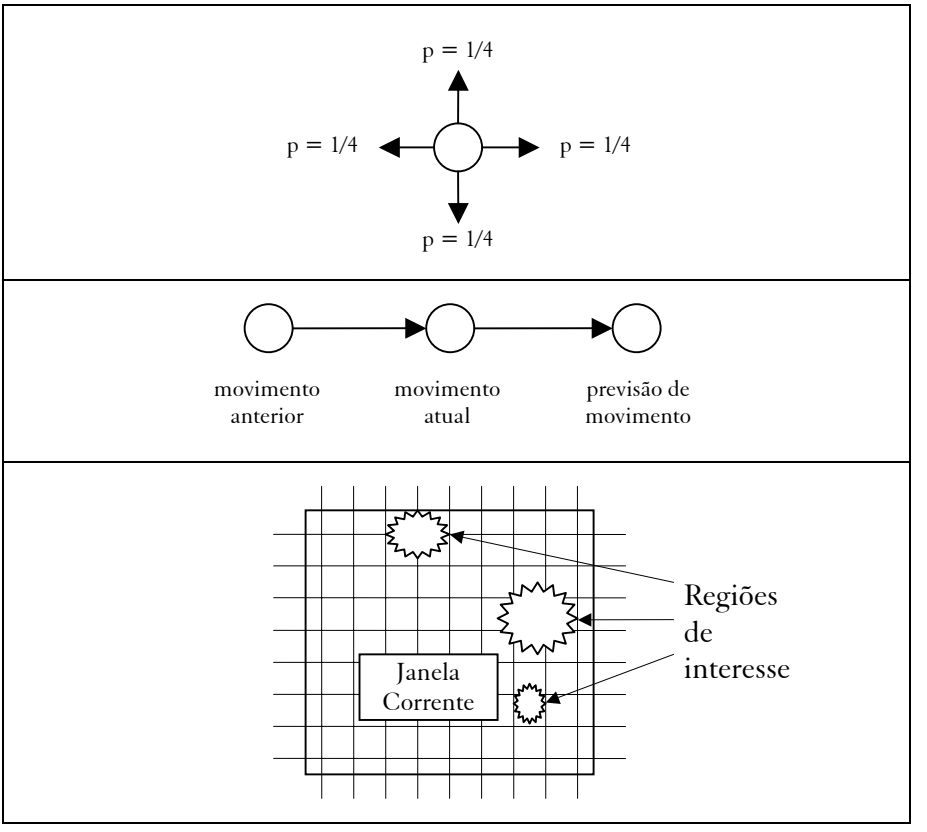


Figura 7.15 – Relacionamentos entre o resultado de uma nova consulta e a área de trabalho (Ren et al., 2003).

7.6 Gerência de transações

7.6.1 Transações em bancos de dados geográficos

Em aplicações convencionais de banco de dados, um usuário modifica os dados através de uma seqüência de operações elementares que devem ser executadas como um todo. As operações são tipicamente curtas e envolvem um volume pequeno de dados. Por exemplo, uma transferência de fundos é implementada através de duas atualizações, em contas bancárias distintas, que devem ser ambas executadas ou ambas canceladas, para evitar erros.

Esta situação configura uma *transação,* ou seja, uma seqüência de operações que o sistema de gerência de banco de dados deve processar até o fim, ou garantir que o banco de dados não reflita nenhuma delas, desfazendo aquelas que tenham sido total ou parcialmente executadas. Além disto, o sistema deve processar as operações sem interferência de outras transações e garantir que, se as operações começam em um estado consistente do banco de dados, elas terminam em um estado também consistente. Mecanismos para garantir estas propriedades foram amplamente investigados no contexto de aplicações convencionais.

Os usuários de bancos de dados geográficos também executam transações desta natureza, principalmente ao modificar os atributos convencionais de objetos. Porém, eles podem apresentar um padrão de comportamento muito diferente, mais próximo do encontrado em ambientes de desenvolvimento de software, projeto de circuitos VLSI e criação de documentos hipermídia, entre outros. Nestes ambientes, uma equipe de usuários trabalha cooperativamente para produzir uma nova configuração dos objetos. Cada usuário cria uma parte de um todo em várias etapas, possivelmente trabalhando sobre uma situação já existente, que não pode ser congelada. Além disto, a equipe pode criar configurações alternativas dos objetos, ou seja, modificações que não necessariamente serão efetivadas.

O conjunto de modificações criado pela equipe de usuários configura uma *transação longa e aninhada*. Neste contexto, o termo transação é freqüentemente utilizado como sinônimo de *sessão de trabalho*. A

transação é longa pois normalmente o trabalho da equipe se desenvolve por muito tempo, possivelmente semanas. Ela é cooperativa porque envolve modificações criadas por usuários distintos, de forma coordenada. Finalmente, a transação pode conter coleções completas de modificações organizadas em *subtransações aninhadas*. Informalmente, uma transação T contém uma subtransação aninhada T' quando um parte das operações de T é agrupada e tratada como uma transação T' de tal forma que as operações em T' possam ser desfeitas ou refeitas pelo sistema como um todo, ou tornadas condicionalmente visíveis a outros usuários.

Por exemplo, considere um banco de dados geográfico contendo dados sobre topografia, vegetação, áreas de preservação ambiental e rede viária de uma região. Suponha que uma equipe de dois engenheiros esteja desenvolvendo o traçado de um novo oleoduto, cada um trabalhando em trechos diferentes e contíguos do oleoduto. Este projeto envolve então trabalho cooperativo, de longa duração, sobre os objetos no banco, configurado como uma única transação longa, contendo duas subtransações aninhadas, uma para cada membro da equipe. Este exemplo será elaborado de forma um pouco diferente na Seção 7.6.3.

7.6.2 Mecanismos para implementação de transações

Esta seção discute alguns mecanismos genéricos propostos para implementação de transações com as características descritas na seção anterior, incluindo tratamento de versões. A descrição segue (Dias et al., 1995) (Soares et al., 1993). Esta seção é opcional, podendo o leitor passar diretamente para a Seção 7.6.3.

Paralelamente aos mecanismos sofisticados para manipulação de dados, descritos abaixo, o usuário poderá ainda manipular diretamente tais dados, para cobrir casos simples como, por exemplo, atualização do valor de um atributo convencional.

Em geral, um ambiente para trabalho cooperativo deve permitir tanto compartilhamento de conjuntos de dados entre usuários quanto a definição de conjuntos privativos a um grupo de usuários. Para tanto, supomos que os dados estão organizados em uma hierarquia de bancos de dados e que cada dado só pertence a um destes bancos. Se B' é filho de

B na hierarquia, dizemos que B' é um banco *subordinado* a B. (Note que a hierarquia possui um número arbitrário de níveis). Cada banco está associado a um grupo de usuários com permissão para realizar um certo conjunto de operações sobre os dados no banco ou em bancos subordinados a ele. Esta hierarquia de bancos reflete o fato de transações conterem outras transações aninhadas. Assim, um grupo de usuários poderá criar um banco de dados para conter os dados com que irá trabalhar e, recursivamente, criar outros bancos subordinados a este, associando-os a transações aninhadas conduzidas por subgrupos de usuários.

Ortogonalmente a estes conceitos, consideramos que um dado pode estar em um de três estados: *pronto*, *trabalho* ou *obsoleto* (respectivamente, *committed*, *uncommitted* ou *obsolete*). Um dado d pode ainda estar associado a um conjunto de outros dados, tratados como suas *versões*, e organizados sob forma de uma árvore, onde d é a raiz. Esta árvore é mantida de tal forma que, se uma versão está pronta, todos os seus ancestrais na árvore também estão prontos; e, se uma versão é obsoleta, todos os seus descendentes também são obsoletos. Os dados são manipulados através do elenco de operações descritas a seguir.

A operação *update* permite atualizar um dado em trabalho, sem alterar o seu estado.

A operação *commit* transforma em pronto um dado em trabalho.

A operação *delete* remove um dado d de um banco B. Se d for um dado em trabalho, ele é efetivamente destruído; se d for um dado pronto, ele é tornado obsoleto. O usuário também pode remover um banco B, provocando a remoção de todos os dados e bancos subordinados a B, recursivamente. O sistema se encarregará de manter dados obsoletos apenas enquanto eles forem referenciados por outros dados.

Há três operações para criação de dados: *create* cria um dado d' completamente novo em um banco B'; *create-version* cria uma nova versão d' de um dado d pronto ou em trabalho no mesmo banco de dados B' que contém d; *check-out* cria um novo dado d' em B' como uma versão de um dado pronto d existente em B, onde B' é um banco subordinado a B. O usuário também poderá utilizar esta operação para criar em bloco versões de um conjunto de dados. Note que o usuário não pode mover d

de B para B', mas apenas usar a operação *check-out* para criar uma versão d' de d em B'. Em todos estes três casos, d' é considerado um dado em trabalho após a sua criação.

A operação *check-in* permite mover um dado pronto d' de um banco B' para o banco B a que B' se subordina. O usuário também poderá utilizar esta operação para mover em bloco um conjunto de dados de um banco para outro. Há duas variantes desta operação. A primeira delas, *check-in de adição*, de fato move d' de B' para B, exceto se d' for uma versão de um dado d existente em B e d' não tiver sido modificado, caso em que d' é descartado, evitando assim a criação de uma réplica de d em B. A segunda, *check-in por substituição*, move d' de B' para B, se d' for um novo dado, ou move d' de B' para B e remove o dado d existente em B (através da operação *delete*, se d' for uma versão de d e d' tiver sido modificado). Esta última operação corresponde portanto à tradicional operação de atualização.

Estas operações mantêm as árvores de versões consistentes através de ações colaterais. Assim, a operação *commit* aplicada a um dado se propaga recursivamente para todos os seus ancestrais na árvore, transformando-os também de dados em trabalho para prontos. A operação *delete* aplicada a um dado se propaga recursivamente para todos os seus descendentes na árvore, destruindo-os ou tornando-os obsoletos. Este efeito pode ser provocado também por uma operação de *check-in por substituição* ao chamar implicitamente uma operação *delete*.

Note que as árvores de versões cruzam a hierarquia de bancos de dados. Portanto, quando um usuário executa uma operação de *commit* sobre um dado em um banco B, ele poderá afetar um dado que pertence ao banco a que B se subordina, e assim recursivamente. Assim, o usuário deverá possuir permissão para executar *commit* também sobre dados nestes outros bancos. Porém, a operação *delete* não causa problemas. Por definição, se o usuário possui permissão para aplicar a operação de *delete* sobre dados de B, ele também possui permissão para aplicar *delete* a dados em bancos subordinados a B, recursivamente. A mesma observação se aplica à operação de *check-in por substituição*. Assim, como efeito colateral de uma operação sobre um dado d, um usuário poderá interferir com bancos de dados de outros usuários que contêm versões de d.

Para minimizar este problema, podemos lançar mão de mecanismos de controle de acesso, permitindo o bloqueio de dados em três modos: *compartilhado*, *exclusivo* e *versão-exclusivo*. Quando um usuário (ou grupo de usuários) bloqueia um dado em modo compartilhado, ele indica uma potencial intenção de substituí-lo por outra versão; em modo exclusivo, ele proíbe outros usuários de substituí-lo por outra versão; em modo versão-exclusivo, ele proíbe outros usuários de criar versões do dado.

Este recurso se integra às operações descritas anteriormente. Em uma operação de *check-out* que cria um novo dado d' em B' como uma versão de um dado d em B; d é bloqueado em modo compartilhado para o usuário que executa a operação. Ao executar um *check-in por substituição* sobre d', se d' tiver sido modificado, o bloqueio de d é escalonado para o modo exclusivo e os outros usuários que possuíam bloqueios compartilhados sobre d têm seus bloqueios descartados. Isto os impede de retornar versões de d, forçando-os a refazer o *check-out*, agora sobre d' (dado que substituiu d).

Esta discussão é simplificada, pois não explora em detalhe as referências cruzadas entre dados, quer seja porque um dado é uma componente de outro, quer seja porque há um relacionamento explícito entre ambos. Também não considera a criação de versões ou o bloqueio de partes de um conjunto de dados. Esta extensão é particularmente importante pois geo-objetos ou geo-campos freqüentemente cobrem áreas geográficas maiores do que a região de interesse do usuário. A seção seguinte apresenta um exemplo de modificação de um banco de dados geográfico que utiliza extensões das operações acima descritas cobrindo este ponto.

7.6.3 Exemplo de transação sobre banco de dados geográfico

Considere um banco de dados geográfico B contendo, para o Estado do Rio de Janeiro:

- um geo-campo T capturando a sua topografia;
- uma coleção de geo-objetos chamada de ÁREAS-DE-PRODUÇÃO, cujos elementos descrevem as áreas de proteção, descritas por uma representação por subdivisão planar P; e

- duas coleções de geo-objetos, REDE-VIÁRIA e OLEODUTOS, com a descrição das estradas e oleodutos, cujas localizações encontram-se em uma representação complexa V.

Suponha agora que uma equipe tenha sido designada para elaborar uma proposta para um oleoduto do terminal da Petrobrás no Município de Angra dos Reis para o Porto de Sepetiba, sem cruzar áreas de proteção ambiental e aproveitando ao máximo os cortes já feitos para as estradas. O trabalho da equipe configura uma transação longa e cooperativa, construída interativamente com descrito a seguir.

Inicialmente a equipe define uma janela R cobrindo toda a região de trabalho. Em seguida, define o banco de dados privado B' e cria, através da operação *check-out*, versões T_R e V_R dos objetos T e V apenas para a região R e uma versão OL de OLEODUTOS. Suponha que a equipe tenha permissão apenas para consultar as coleções ÁREAS DE PROTEÇÃO e REDE-VIÁRIA e a representação P.

Em seguida, a equipe define o oleoduto, inserindo um novo geo-objeto O em OL, e alterando V_R para incluir a representação de O.

A equipe também modifica o geo-campo T_R para indicar os novos cortes necessários à passagem do oleoduto (mas não modifica P, por restrições de projeto).

Ao término, a equipe introduz o resultado do projeto no banco original B através da operação de *check-in por substituição*, estendida para tratar de versões que se referem apenas a uma parte da área geográfica coberta pelos objetos.

Em paralelo, uma segunda equipe poderá estar trabalhando no traçado de uma nova estrada no Município de Angra dos Reis, com restrições semelhantes às anteriores: a estrada não poderá cruzar áreas de proteção ambiental e deverá aproveitar ao máximo os cortes já feitos para oleodutos.

Novamente, a equipe define uma janela S cobrindo sua região de trabalho e define o banco de dados privado E, criando, através da operação *check-out,* versões T_S e V_S dos objetos T e V apenas para a região S e uma versão RV de REDE-VIÁRIA.

De forma semelhante, esta equipe define a estrada, inserindo um novo geo-objeto E na coleção RV, alterando V_S para incluir a representação de E, e modificando o geo-campo T_S para indicar novos cortes necessários à passagem da nova estrada.

Ao término, a equipe tentará introduzir o resultado deste segundo projeto no banco original B, através da operação de *check-in por substituição*.

Duas situações poderão ocorrer. Primeiro, se as regiões R e S forem disjuntas, então o trabalho de uma equipe não afeta diretamente o da outra e a alteração do banco B ocorrerá normalmente. Porém, a implementação da operação de *check-in por substituição* deverá ser suficientemente sofisticada para alterar V e T apenas para a região S de tal forma que não destrua as modificações efetuadas nestes campos pela equipe que criou o oleoduto.

Segundo, se as regiões R e S não forem disjuntas, o trabalho de uma equipe potencialmente afeta o da outra. À guisa de ilustração, façamos uma análise mais detalhada desta situação. A primeira equipe altera V acrescentando a representação de O e a segunda acrescentando a representação de E. Portanto, se o sistema versionar as componentes de V, e não V como um todo, será simples implementar a operação de *check-in por substituição* de tal forma que o trabalho de uma equipe não interfira com o da outra do ponto de vista de V: basta reconhecer que ambas as equipes simplesmente criaram novas componentes de V. Argumento semelhante se aplica a T, exceto que, neste caso, como T é um geo-campo, será necessário analisar a representação utilizada para T. Por exemplo, se a representação for por isolinhas, será necessário versionar as isolinhas em si; assim uma equipe não interferirá com a outra se alterarem isolinhas distintas.

Naturalmente, toda esta discussão sobre a operação de *check-in por substituição* é afetada pelos mecanismos de controle de acesso, que também deverão ser revistos para acomodar os refinamentos baseados no uso de regiões ou de componentes.

7.7 Leituras suplementares

Resumos dos principais aspectos de implementação de sistema de gerência de bancos de dados geográficos podem ser encontrados em referências gerais (Guting, 1994) (Medeiros e Pires, 1994) (Shekhar et al., 1999) (Rigaux et al, 2001) (Shekhar, 2002). Descrições de extensões dos principais sistema de gerência de bancos de dados para tratar dados geográficos encontram-se em (Ravada e Sharma, 1999) (David, 2001).

Políticas específicas para otimização de consultas espaciais podem ser encontradas em referências seminais sobre o assunto (Aref e Samet, 1991) (Kriegel et al., 1993) (Brinkhoff et al., 1993) (Brinkhoff et al., 1994). Em particular, a idéia básica de processar uma consulta espacial em vários passos, através de aproximações da geometria dos objetos, foi introduzida em (Kriegel et al., 1993). Propostas de *benchmarks* para consultas espaciais podem ser encontrados em (Stonebraker et al. 1993) (Cifferi, 1995). Uma biblioteca de algoritmos para processamento de consultas espaciais é apresentada em (Bercken et al. 2001). A questão do processamento de consultas para objetos representados em alta resolução é discutida em (Kriegel et al., 2003).

Há bem poucas referências sobre gerência de transações para o caso específico de sistemas de gerência de bancos de dados geográficos. Porém, o padrão de transações para estas aplicações é semelhante às chamadas *transações longas e aninhadas* (Lynch, 1983) (Weikum, 1991), típicas de ambientes de desenvolvimento de software, ambientes de projeto de VLSI e hipertexto, entre outros. Um modelo semântico, genérico para transações, que se aplica também a bancos de dados geográficos, foi proposto em (Brayner et al., 1999). Um modelo de transações recente e específico para bancos de dados geográficos encontra-se (Kadri-Dahmani e Osmani, 2003).

A discussão sobre processamento de consultas espaciais apresentada neste capítulo aborda apenas aplicações convencionais envolvendo dados geográficos. Há, no entanto, uma vasta gama de aplicações de outra natureza, que necessitam revisar a semântica das consultas e, conseqüentemente, as técnicas de processamento de consultas.

Podemos apontar aplicações que necessitam: tratar objetos espaço-temporais (Kim et al., 2002); trabalhar com dados geográficos

representando redes, onde a noção de espaço Euclidiano não é adequada (Papadias et al., 2003); considerar consultas dependentes de localização em ambientes para computação móvel (Ayse et al., 2001) (Zhang et al., 2003) (Manical et al., 2004); e processar objetos móveis (Porkaew et al., 2001) (Chon et al., 2002).

Referências

ABOULNAGA, A.; NAUGHTON, J. F. Accurate estimation of the cost of spatial selections. In: 16th International Conference on Data Engineering. San Diego, CA, USA, 2000. p. 123-134.

AREF, W.; SAMET, H. Optimization for Spatial Query Processing. In: 17th International Conference on Very Large Data Bases. Morgan Kaufmann Publishers Inc., Barcelona, Spain, 1991. p. 81-90.

AYSE, Y.; DUNHAM, H. M.; KUMAR, V. M. Location dependent query processing. In: 2nd ACM international workshop on Data engineering for wireless and mobile access. Santa Barbara, CA, USA, 2001. p. 47-53.

BERCKEN, J.; BLOHSFELD, B.; DITTRICH, J.-P.; KRAMER, J.; SCHAFER, T.; SCHNEIDER, M.; SEEGER, B. XXL - A Library Approach to Supporting Efficient Implementations of Advanced Database Queries. In: 27th International Conference on Very Large Data Bases. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001. p. 39-48.

BRAYNER, A.; HARDER, T.; RITTER, N. Semantic Serializability: A Correctness Criterion for Processing Transactions. **Data & Knowledge Engineering**, v. 31, n.1, p. 1-24, 1999.

BRINKHOFF, T.; HORN, H.; KRIEGEL, H.-P.; SCHNEIDER, H. A Storage and Access Architecture for Efficient Query Processing in Spatial Database Systems. In: Third International Symposium on Advances in Spatial Databases. **Lecture Notes In Computer Science.** Springer-Verlag London, UK, 1993a. p. 357-376.

BRINKHOFF, T.; KRIEGEL, H.-P.; SCHNEIDER, H.; SEEGER, B. GENESYS: A System for Efficient Spatial Query Processing. In: 1994 ACM SIGMOD International Conference on Management of Data. Minneapolis, MI, USA, 1994a. p. 519.

BRINKHOFF, T.; KRIEGEL, H.-P.; SCHNEIDER, R.; SEEGER, B. Multi-step Processing of Spatial Joins. In: 1994 ACM SIGMOD International Conference on Management of Data. Minneapolis, USA, 1994b. p. 197-208.

BRINKHOFF, T.; KRIEGEL, H.-P.; SEEGER, B. Efficient Processing of Spatial Joins. In: 1993 ACM SIGMOD International Conference on Management of Data. Washington, DC, USA, 1993b. p. 237-246.

CHON, D. H.; AGRAWAL, D.; EL ABBADI, A. Query Processing for Moving Objects with Space-Time Grid Storage Model. In: Third International Conference on Mobile Computing. 2002. p. 121.

CIFFERI, R. Um Benchmark Voltado a Análise de Desempenho de Sistemas de Informações Geográficas. Campinas, SP, Brasil: UNICAMP, 1995. Departamento de Ciência da Computação, 1995.

DAR, S.; FRANKLIN, M. J.; JONSSON, B. T.; SRIVATAVA, D.; TAN, M. Semantic Data Caching and Replacement. In: 22th International Conference on Very Large Data Bases. Morgan Kaufmann Publishers Inc., 1996. p. 330-341.

DAVID, W. A. DB2 Spatial Extender - Spatial data within the RDBMS. In: 27th International Conference on Very Large Data Bases. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001. p. 687-690.

DIAS, E.; GRANADO, S.; MAGALHÃES, G. Uso de Versões na Garantia de Consistência em Ambientes Mistos de Projetos e Operação. In: Simpósio Brasileiro de Banco de Dados. 1995. p. 321-334.

DOSHI, P. R.; RUNDENSTEINER, E. A.; WARD, M. O. Prefetching for Visual Data Exploration. In: Eighth International Conference on Database Systems for Advanced Applications. 2003. p. 195.

GUNTHER, O.; BECKER, L.; HINRICHS, K.; FINKE, U. Efficient computation of spatial joins. In: 9th International Conference on Data Engineering. Vienna, Austria, 1993. p. 50-59.

GUTING, R. An Introduction to Spatial Database Systems. **VLDB Journal**, v. 3, n.4, p. 357-399, 1994.

KADRI-DAHMANI, H.; OSMANI, A. Updating Data in GIS: How to Maintain Database Consistency? In: Enterprise Information Systems IV. Kluwer Academic Publishers, Ciudad Real, Spain, 2003. p. 163-169.

KIM, H. D.; RYU, H. K.; PARK, H. C. Design and implementation of spatiotemporal database query processing system. **Journal of Systems and Software**, v. 60, n.1, p. 37-49, 2002.

KRIEGEL, H.-P.; BRINKHOFF, T.; SCHNEIDER, R. An Efficient Map Overlay Algorithm Based on Spatial Access Methods and Computational Geometry. In: **Geographic Database Management Systems.** Springer-Verlag, Capri, 1992. p. 194-211.

KRIEGEL, H.-P.; BRINKHOFF, T.; SCHNEIDER, R. Efficient Spatial Query Processing in Geographic Database Systems. **Data Engineering Bulletin**, v. 16, n.3, p. 10-15, 1993.

KRIEGEL, H.-P.; PFEIFLE, M.; POTKE, M.; SEIDL, T. Spatial Query Processing for High Resolutions. In: 8th International Conference on

Database Systems for Advanced Applications (DASFAA'03). Kyoto, Japan, 2003. p. 17.

KRIEGEL, H.-P.; SCHIWIETZ, M.; SCHNEIDER, R.; SEEGER, B. Performance comparison of point and spatial access methods. In: First Symposium on Design and Implementation of Large Spatial Databases.Lecture Notes in Computer Science. Springer-Verlag, Santa Barbara, CA, USA, 1990. p. 89-114.

LO, M.-L.; RAVISHANKAR, C. V. Spatial hash-joins. In: 1996 ACM SIGMOD International Conference on Management of Data. Montreal, Quebec, Canada, 1996. p. 247-258.

LYNCH, N. A. A New Correctness Criterion for Database Concurrency Control. **ACM Transactions on Database Systems**, v. 8, n.4, p. 484-502, 1983.

MANICAL, H.; CAMARGO, S. M.; CIFERRI, A. D. C.; CIFERRI, R. R. Processamento de Consultas Espaciais Baseado em Cache Semântico Dependente de Localização. In: VI Simpósio Brasileiro de Geoinformática – GeoInfo 2004. Campos de Jordão, SP, 2004. p.

MEDEIROS, C. B.; PIRES, F. Databases for GIS. **ACM SIGMOD Record**, v. 23, n.1, p. 107-115, 1994.

PAPADIAS, D.; ZHANG, J.; MAMOULIS, N.; Y., T. Query Processing in Spatial Network Databases. In: 29th Very Large Data Base Conference. Berlin, Germany, 2003. p.

PATEL, J. M.; DEWITT, D. J. Partition based spatial-merge join. In: 1996 ACM SIGMOD International Conference on Management of Data. Montreal, Quebec, Canada, 1996. p. 259-270.

PORKAEW, K.; LAZARIDIS, I.; MEHROTRA, S. Querying Mobile Objects in Spatio-Temporal Databases. In: 7th International Symposium on Advances in Spatial and Temporal Databases.Lecture Notes In Computer Science. Springer-Verlag, Redondo Beach, CA, USA, 2001. p. 59-78.

PREPARATA, F. P.; SHAMOS, M. I., eds., 1985, **Computational Geometry: An Introduction**: New York, NY, Springer-Verlag.

RAVADA, S.; SHARMA, J. Oracle8i Spatial: Experiences with Extensible Databases. In: 6th International Symposium on Advances in Spatial Databases. Springer-Verlag, London, UK, 1999. p. 355-359.

REN, Q.; DUNHAM, M. H.; KUMAR, V. Semantic Caching and Query Processing. **IEEE Transactions on Knowledge and Data Engineering**, v. 15, n.1, p. 192-210, 2003.

RIGAUX, P.; SCHOLL, M.; VOISARD, A. **Spatial Databases with Application to GIS**. San Francisco: Morgan Kaufmann Publishers Inc, 2001.

SHEKHAR, S.; CHAWLA, S.; RAVADA, S.; FETTERER, A.; LIU, X.; LIU, C. T. Spatial Databases: Accomplishments and Research Needs. **IEEE Transactions on Knowledge and Data Engineering**, v. 11, n.1, p. 45-55, 1999.

SOARES, L.; CASANOVA, M.; RODRIGUES, N. Um Modelo Conceitual Hipermídia com Nós de Composição e Controle de Versões. In: VII Simpósio Brasileiro de Engenharia de Software. 1993. p. 365-381.

STONEBRAKER, M.; FREW, J.; GARDELS, K.; MEREDITH, J. The Sequoia 2000 Benchmark. In: 1993 ACM SIGMOD International Conference on Management of Data. Washington, D.C., USA, 1993. p. 2-11.

WEIKUM, G. Principles and Realization Strategies of Multilevel Transaction Management. **ACM Transactions on Database Systems**, v. 16, n.1, p. 132-180, 1991.

ZHANG, J.; ZHU, M.; PAPADIAS, D.; TAO, Y.; LEE, D. L. Location-based spatial queries. In: 2003 ACM SIGMOD International Conference on Management of Data. San Diego, CA, USA, 2003. p. 443-454.

8 SGBD com extensões espaciais

Gilberto Ribeiro de Queiroz
Karine Reis Ferreira

8.1 Introdução

Este capítulo apresenta duas extensões espaciais de SGBD convencionais, uma da comunidade de software aberto (PostGIS) e outra comercial (Oracle Spatial).

O PostGIS (Ramsey, 2002) estende o PostgreSQL, seguindo as especificações da SFSSQL. O PostgreSQL é um sistema de gerência de banco de dados objeto-relacional, gratuito e de código fonte aberto (Stonebraker et al, 1990). Foi desenvolvido a partir do projeto Postgres, iniciado em 1986, na Universidade da Califórnia em Berkeley.

O Spatial (Ravada e Sharma, 1999) (Murray, 2003) estende o modelo objeto-relacional do sistema de gerência de banco de dados Oracle. Esta extensão baseia-se nas especificações do OpenGIS.

8.2 Cenários

Esta seção apresenta alguns cenários que serão empregados ao longo do capítulo para ilustrar os principais recursos das extensões espaciais consideradas. Os exemplos que iremos apresentar, representam as informações de distritos, bairros e rede de drenagem da cidade de São Paulo; os municípios que formam a grande São Paulo (São Caetano do Sul, Suzano, Guarulhos entre outras). A Figura 8.1 mostra a parte geométrica dos distritos de São Paulo. Para cada distrito associaremos um polígono e os atributos código do distrito (COD), sigla de abreviação (SIGLA) e o nome do distrito (DENOMINACAO). Aos pontos que representam bairros da cidade de São Paulo (Figura 8.2) iremos associar

os atributos nome do bairro (BAIRRO) e o nome do distrito (DISTR). Às linhas da Figura 8.3 (mapa de drenagem) associaremos um único atributo, a classe de drenagem (CLASSE). E, aos polígonos dos municípios da grande São Paulo, os atributos código do município (CODMUNIC), nome do município (NOMEMUNICP) e população (POPULACAO).

Figura 8.1 – Mapa de Distritos da Cidade de São Paulo.

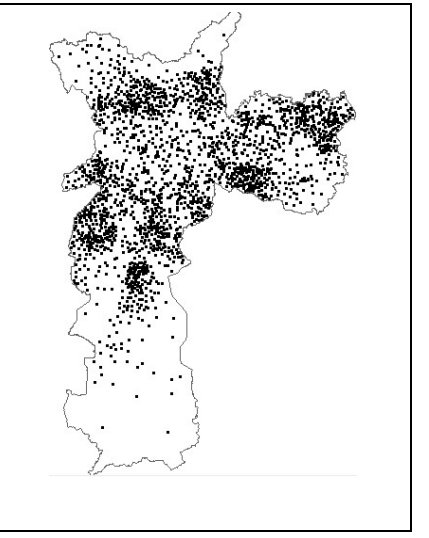


Figura 8.2 – Mapa de Bairros da Cidade de São Paulo.

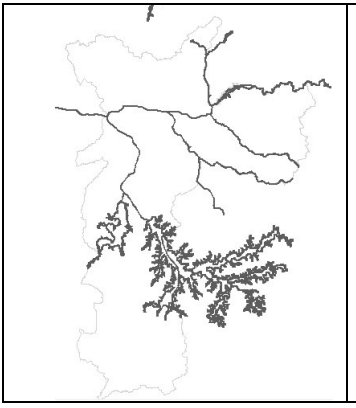


Figura 8.3 – Mapa de drenagem.

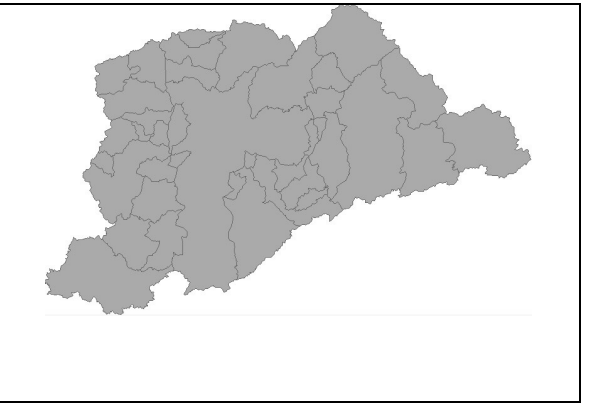


Figura 8.4 – Mapa dos Municípios da Grande São Paulo.

A partir dessas informações, iremos construir alguns cenários envolvendo consultas espaciais que serão utilizados durante a explicação dos recursos de cada uma das extensões.

Cenário 1: "Recuperar o nome de todos os municípios da grande São Paulo que são vizinhos ao município de São Paulo". A Figura 8.5 ilustra esse cenário, com o município de São Paulo destacado em preto e os municípios vizinhos destacados em cinza claro

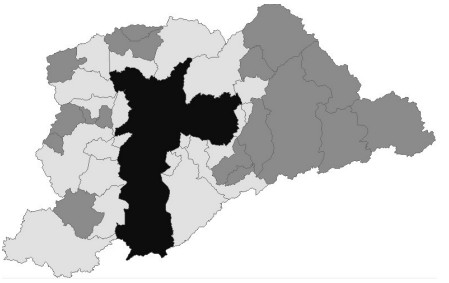


Figura 8.5 – Municípios vizinhos à cidade de São Paulo.

Cenário 2: "Recuperar o nome de todos os municípios da grande São Paulo que são vizinhos ao distrito Anhanguera da cidade de São Paulo". A Figura 8.6 ilustra este cenário, com o distrito Anhanguera de São Paulo destacado em preto e os municípios vizinhos a este distrito, destacados em cinza claro.

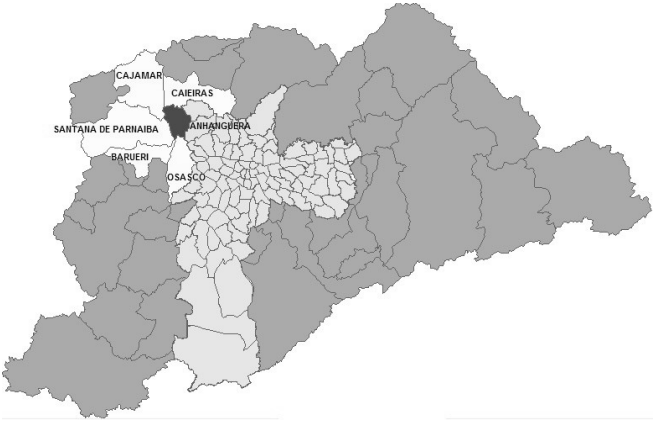


Figura 8.6 – Municípios vizinhos ao distrito Anhanguera.

Cenário 3: "Recuperar o número de bairros contidos no distrito Grajaú". A Figura 8.7 ilustra o cenário, com o distrito Grajaú destacado em cinza escuro e os municípios contidos neste distrito de cinza claro.

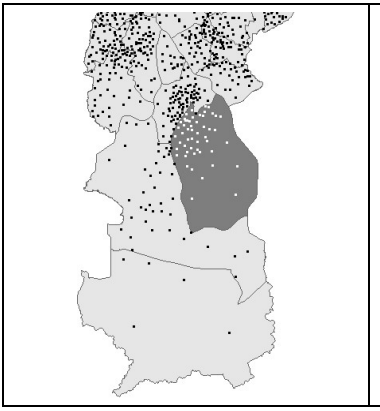


Figura 8.7 – Bairros contidos no distrito de Grajaú.

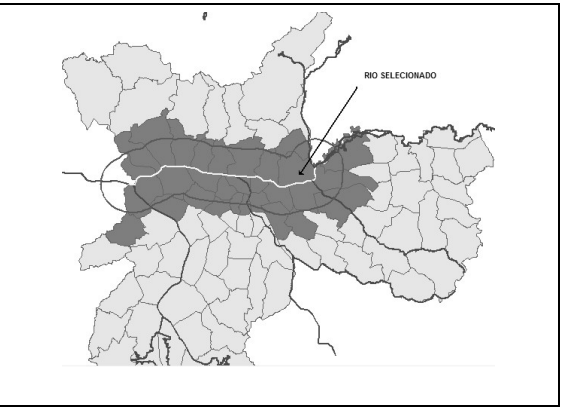


Figura 8.8 – Buffer de 3Km ao redor de um rio e distritos nesta região de influência.

Cenário 4: "Recuperar todos os distritos que estão num raio de 3km de um determinado rio". A Figura 8.8 ilustra o resultado desta consulta, com o rio selecionado destacado em branco e os distritos em cinza escuro contidos no raio de 3km do rio. A linha em cinza escuro ao redor do rio selecionado representa um buffer gerado com o qual os distritos foram testados (relacionamento intercepta).

Cenário 5: "Recuperar todos os bairros que estejam a menos de 3km do bairro Boacava".

8.3 PostGIS para PostgreSQL

8.3.1 Características principais do PostgreSQL

O PostgreSQL é um sistema gerenciador de banco de dados objeto-relacional, gratuito e de código fonte aberto, desenvolvido a partir do projeto Postgres, iniciado em 1986, na Universidade da Califórnia em Berkeley, sob a liderança do professor Michael Stonebraker. Em 1995, quando o suporte a SQL foi incorporado, o código fonte foi disponibilizado na Web (http://www.postgresql.org). Desde então, um grupo de desenvolvedores vem mantendo e aperfeiçoando o código fonte sob o nome de PostgreSQL.

Em sua versão de distribuição oficial, ele apresenta tipos de dados geométricos (Figura 8.9), operadores espaciais simples e indexação espacial através de uma R-Tree nativa ou através de R-Tree implementada no topo do mecanismo de indexação GiST (Hellerstein et al, 1995).

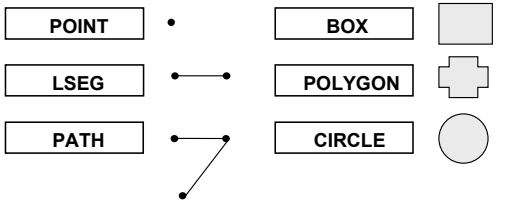


Figura 8.9 – Tipos Geométricos do PostgreSQL.

A implementação da R-Tree nativa possui uma severa limitação em seu uso, uma coluna do tipo polígono não pode exceder 8Kbytes. Na prática, é muito comum um SIG manipular dados representados, por exemplo, por polígonos maiores do 8Kbytes cada, o que torna o uso desse índice inviável. Uma alternativa é o uso da segunda R-Tree.

O método de indexação chamado GiST foi introduzido por Hellerstein et al (1995) e implementado no PostgreSQL. Atualmente, ele é mantido por Signaev e Bartunov (http://www.sai.msu.su/~megera/postgres/gist) e não possui restrições de tamanho do dado a ser indexado. GiST é uma abreviação de *Generalized Search Trees*, consistindo em um índice (árvore balanceada) que pode fornecer tanto as funcionalidades de uma B-Tree quanto de uma R-Tree e suas variantes. A principal vantagem desse índice é a possibilidade de definição do seu "comportamento".

Quanto aos poucos operadores espaciais existentes, estes realizam a computação apenas sobre o retângulo envolvente das geometrias e não diretamente nelas. Já os tipos de dados simples, como polígonos, não permitem a representação de buracos, não existindo também geometrias que permitam representar objetos mais complexos como os formados por conjuntos de polígonos.

Portanto, a integração de um SIG através desses tipos geométricos requer muito esforço – implementação de novas operações espaciais como união, interseção, testes topológicos sobre a geometria exata, definição de um modelo de suporte a polígonos com buracos, entre outras.

Mas, como dito anteriormente, um dos pontos fortes desse SGBD é seu potencial de extensibilidade, o que possibilitou o desenvolvimento de uma extensão geográfica mais completa, chamada PostGIS. Antes de entrar em maiores detalhes dessa extensão, apresentaremos um pouco do mecanismo de extensibilidade para que o leitor possa entender melhor o núcleo da extensão PostGIS.

8.3.2 Extensibilidade do PostgreSQL

O mecanismo de extensibilidade do PostgreSQL permite incorporar capacidades adicionais ao sistema de forma a torná-lo mais flexível para o gerenciamento de dados para cada classe de aplicação. No caso dos SIG, isso significa a possibilidade do desenvolvimento de uma extensão geográfica capaz de armazenar, recuperar e analisar dados espaciais. A seguir serão apresentados alguns passos envolvidos na criação de uma extensão do PostgreSQL. Os exemplos foram adaptados do tipo de dados POLYGON existente na distribuição oficial.

8.3.2.1 Tipos de dados definidos pelo usuário

A extensão de tipos de dados no PostgreSQL pode ser feita em linguagem C (Kernighan e Ritchie, 1990) através da definição de uma estrutura de dados, que é responsável pela representação do tipo em memória. O trecho de código abaixo ilustra a definição de um tipo chamado `TeLinearRing`, que representa uma linha fechada composta por um conjunto de coordenadas (do tipo `TeCoord2D`). Os tipos definidos pelo usuário podem ser de tamanho fixo ou variável, como no caso do exemplo abaixo (variável):

```
typedef struct
{ int32 size;             //struct length
  int32 ncoords_;         //number of coords
  TeCoord2D coords_[1];   //variable length array
} TeLinearRing;
```

Além da estrutura de dados, é necessário definir outras duas rotinas[1] que fazem a conversão do tipo de acordo com a sua representação em memória para ASCII e vice-versa. Estas rotinas são conhecidas como funções de entrada e saída, e elas associam uma representação textual externa para o dado. Essas funções são utilizadas internamente para permitir que o tipo seja usado diretamente em comandos SQL. Por exemplo, para o tipo TeLinearRing, poderíamos optar pelo seguinte formato de representação: *[(X1, Y1), (X2, Y2), ..., (Xn, Yn)]*. O trecho de código abaixo ilustra a implementação de uma rotina de entrada para o tipo `TeLinearRing`[2]:

```
00   PG_FUNCTION_INFO_V1(TeLinearRing_in);
01   Datum TeLinearRing_in(PG_FUNCTION_ARGS)
02   { char* str = PG_GETARG_CSTRING(0);
03     TeLinearRing* ring; int ncoords;
04     int size;        char* s;
05
06     if((ncoords = number_of_coords(str, ')')) <= 0)
07       elog(ERROR, "Bad TeLinearRing external
08                    representation '%s'", str);
09     size = offsetof(TeLinearRing, coords_[0]) +
10            sizeof(ring->coords_[0]) * ncoords;
11     ring = (TeLinearRing*)palloc(size);
12     MemSet((char *) ring, 0, size);
13     ring->size = size;  ring->ncoords_ = ncoords;
14     if((!decode(ncoords, str, &s,
15                 &(ring->coords_[0]), '[', ']')) ||
16         (*s != '\0'))
17         elog(ERROR, "Bad TeLinearRing external
18               representation '%s'", str);
19     if(!is_TeLinearRing(ring))
20     { pfree(ring);
21       ring = NULL;
22       elog(ERROR, "In a TeLinearRing the first point
23            must be the same as the last point '%s'",
24            str);
25     }
26     PG_RETURN_TeLine2D_P(ring);
27   }
```

[1] A partir da versão 7.4, pode-se definir das funções de I/O no formato binário.
[2] As funções `number_of_coords` e `decode` serão omitidas por questões de espaço.

Os passos envolvidos na definição desta rotina são:

- Na linha 02, podemos observar que a rotina recebe um polígono (`TeLinearRing`) na forma textual (`str`).
- Na linha 06, a função `number_of_coords` determina quantas coordenadas formam a fronteira do polígono.
- Na linha 11, é alocada memória suficiente para um polígono com o número de coordenadas determinado na linha 06.
- Finalmente, na linha 14, a *string* é decodificada e as coordenadas são armazenadas no vetor de coordenadas da fronteira do polígono.

A função de saída é mais simples, recebe o dado na representação interna, devendo convertê-lo para a representação externa. O trecho de código abaixo ilustra a função de saída para o tipo `TeLinearRing`:

```
PG_FUNCTION_INFO_V1(TeLinearRing_out);
Datum TeLinearRing_out(PG_FUNCTION_ARGS)
{TeLinearRing *ring = (TeLinearRing*)
               PG_GETARG_TeLinearRing_P(0);
 char* str;
 str = palloc(ring->ncoords_ * (P_MAXLEN + 3) + 2);

 if(!encode(ring->ncoords_, ring->coords_, str, '[',
            ']', "Unable to format TeLinearRing"))
   elog(ERROR, "Unable to format TeLinearRing");
 PG_RETURN_CSTRING(str); }
```

Depois de criado o tipo, uma biblioteca compartilhada deve ser gerada para ser integrada dinamicamente ao servidor de banco de dados. É necessário registrar o tipo através do comando SQL: `CREATE TYPE`, informando as rotinas de entrada e saída:

```
CREATE FUNCTION telinearring_in(opaque)
    RETURNS telinearring
    AS '/opt/tepgdatatypes.so'
    LANGUAGE 'c';

CREATE FUNCTION telinearring_out(opaque)
    RETURNS opaque
    AS '/opt/tepgdatatypes.so'
    LANGUAGE 'c';

CREATE TYPE telinearring
```

```
( alignment = double,
  internallength = VARIABLE,
  input   = telinearring_in,
  output  = telinearring_out,
  storage = main );
```

Agora podemos utilizar o tipo `TeLinearRing` dentro de comandos SQL, como:

- Criar uma tabela com uma coluna do tipo TeLinearRing:
- `CREATE TABLE tab_poligonos_simples (id INTEGER, geom TeLinearRing);`
- Inserir um quadrado (4 x 4):
- `INSERT INTO tab_poligonos_simples VALUES(1, '[(0, 0), (4, 0), (4, 4), (0, 4), (0, 0)]');`
- Recuperar todos os polígonos:

```
SELECT * FROM tab_poligonos_simples;
Resultado:
id | geom
-----------------------------------------
1  | '[(0, 0), (4, 0), (4, 4), (0, 4), (0, 0)]'
```

8.3.2.2 Funções e operadores definidos pelo usuário

Além da flexibilidade do sistema de tipos, o PostgreSQL permite a criação de métodos que operem sobre as instâncias dos tipos definidos. Essas funções podem ser integradas ao servidor informando o nome da função, a lista de argumentos e o tipo de retorno. Essas informações são registradas no catálogo do sistema.

O trecho de código abaixo ilustra a definição de uma função para calcular a área de um polígono simples representado por um `TeLinearRing`.

```
PG_FUNCTION_INFO_V1(TeLinearRingArea);
Datum TeLinearRingArea(PG_FUNCTION_ARGS)
{TeLinearRing *r = (TeLinearRing *)
                   PG_DETOAST_DATUM(PG_GETARG_DATUM(0));
 double area = 0.0;   int npoints = 0;
 int    loop;      npoints = r->ncoords_;

 for(loop = 0; loop < (npoints - 1); ++loop)
```

```
{  area += ((r->coords_[loop].x_ *
             r->coords_[loop + 1].y_) -
            (r->coords_[loop + 1].x_ *
             r->coords_[loop].y_)); }
area *= 0.5;
PG_RETURN_FLOAT8(area); }
```

Assim como os tipos, as funções também devem ser colocadas em uma biblioteca compartilhada para poderem ser integradas ao servidor de banco de dados. É necessário registrar a função através do comando SQL: `CREATE FUNCTION`, como mostrado abaixo:

```
CREATE FUNCTION area(telinearring)
RETURNS float4
AS '/opt/tepgfunctions.so', 'telinearringarea'
LANGUAGE 'c' WITH (isstrict);
```

Outra funcionalidade importante oferecida pelo PostgreSQL é que os nomes das funções podem ser sobrecarregadas desde que os parâmetros sejam de tipos diferentes. Depois de criada uma função, é possível definir um operador para ela através do comando SQL: `CREATE OPERATOR`. A importância da definição do operador está ligada ao otimizador de consultas que pode usar as informações contidas na definição do operador para decidir a melhor estratégia para a consulta (ou simplificação desta). Outra funcionalidade oferecida pelo PostgreSQL é a definição de funções agregadas, que podem ser construídas de forma análoga às funções sobre tipos, porém são registradas com o comando SQL: `CREATE AGGREGATE`.

8.3.2.3 Extensão do mecanismo de indexação

O PostgreSQL possui quatro mecanismos de indexação (B-Tree, R-Tree, GiST e HASH). Esses índices podem ser usados para os tipos de dados e funções definidos pelo usuário, bastando para isso registrar no catálogo do sistema as informações das operações necessárias a cada índice. Por exemplo, para um tipo que deseje ser indexado pela B-Tree, é necessário indicar ao sistema as operações de comparação "<", "<=", "=", ">=" e ">" para que o índice saiba como realizar buscas.

O exemplo abaixo ilustra a implementação do "operador <" para o tipo `TeCoord2D` (coordenada com os campos x_ e y_), sendo que os demais operadores diferem apenas à forma da chamada:

```
static int
tecoord2d_cmp_internal(TeCoord2D* a, TeCoord2D* b)
{  if(a->x_ < b->x_)
     return -1;
   if(a->x_ > b->x_)
     return 1;
   if(a->y_ < b->y_)
     return -1;
   if(a->y_ > b->y_)
     return 1;
   return 0;
}

PG_FUNCTION_INFO_V1(tecoord2d_lt);
Datum
tecoord2d_lt(PG_FUNCTION_ARGS)
{   TeCoord2D *a = (TeCoord2D*) PG_GETARG_POINTER(0);
    TeCoord2D *b = (TeCoord2D*) PG_GETARG_POINTER(1);
    PG_RETURN_BOOL(tecoord2d_cmp_internal(a, b) < 0); }

Datum
tecoord2d_cmp(PG_FUNCTION_ARGS)
{ TeCoord2D    *a = (TeCoord2D*) PG_GETARG_POINTER(0);
  TeCoord2D    *b = (TeCoord2D*) PG_GETARG_POINTER(1);
  PG_RETURN_INT32(tecoord2d_cmp_internal(a, b)); }
```

Agora, podemos registrar a função e definir o operador através da seguinte seqüência de comandos:

```
CREATE FUNCTION tecoord2d_lt(TeCoord2D, TeCoord2D) RETURNS
bool
    AS 'filename', 'tecoord2d_lt'
    LANGUAGE C IMMUTABLE STRICT;

CREATE FUNCTION tecoord2d_cmp(TeCoord2D, TeCoord2D)
    RETURNS integer
    AS 'filename', 'tecoord2d_cmp'
    LANGUAGE C IMMUTABLE STRICT;
```

```
CREATE OPERATOR < (
   leftarg = TeCoord2D, rightarg = TeCoord2D,
   procedure = tecoord2d_lt,
   commutator = > , negator = >= ,
   restrict = scalarltsel, join = scalarltjoinsel
);
```

E, por último, criamos a classe de operação requerida pelo índice *B-tree:*

```
CREATE OPERATOR CLASS tecoord2d_ops
    DEFAULT FOR TYPE TeCoord2D USING btree AS
        OPERATOR   1       < ,
        OPERATOR   2       <= ,
        OPERATOR   3       = ,
        OPERATOR   4       >= ,
        OPERATOR   5       > ,
        FUNCTION 1 tecoord2d_cmp(TeCoord2D, TeCoord2D);
```

8.3.3 A extensão espacial PostGIS do PostgreSQL

A comunidade de software livre vêm desenvolvendo a extensão espacial PostGIS, construída sobre o PostgreSQL. Atualmente, a empresa Refractions Research Inc (http://postgis.refractions.net) mantém a equipe de desenvolvimento dessa extensão, que segue as especificações da SFSSQL.

8.3.3.1 Tipos de dados espaciais

A Figura 8.10 ilustra os tipos espaciais suportados pelo PostGIS e embutidos na SQL do PostgreSQL.

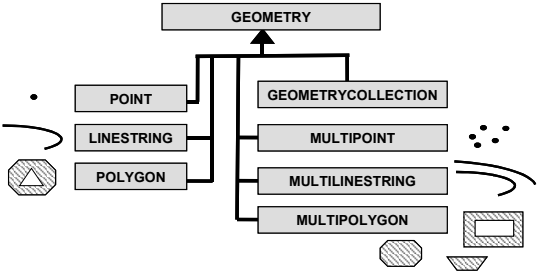


Figura 8.10 - Tipos de dados espaciais do PostGIS.

Esses tipos possuem a seguinte representação textual:

- `Point: (0 0 0)`
- `LineString: (0 0, 1 1, 2 2)`
- `Polygon: ((0 0 0, 4 0 0, 4 4 0, 0 4 0, 0 0 0), ( 1 0 0, ...), ...)`
- `MultiPoint: (0 0 0, 4 4 0)`
- `MultiLineString: ((0 0 0, 1 1 0, 2 2 0), (4 4 0, 5 5 0, 6 6 0))`
- `MultiPolygon: (((0 0 0, 4 0 0, 4 4 0, 0 4 0, 0 0 0), (...), ...), ...)`
- `GeometryCollection: (POINT(2 2 0), LINESTRING((4 4 0, 9 9 0))`

Como exemplo, mostraremos os comandos em SQL para gerar tabelas para armazenar os atributos e as geometrias[3]:

- Dos distritos de São Paulo, mostrados na Figura 8.1.

```
CREATE TABLE distritossp
( cod       SERIAL,
  sigla           VARCHAR(10),
  denominacao     VARCHAR(50),
  PRIMARY KEY  (cod)
);
SELECT AddGeometryColumn('terralibdb', 'distritossp',
'spatial_data', -1, 'POLYGON', 2);
```

- Dos bairros de São Paulo, mostrados na Figura 8.2.

```
CREATE TABLE bairrossp
( cod       SERIAL,
  bairro        VARCHAR(40),
  distr     VARCHAR(40),
  PRIMARY KEY      (cod)
);
SELECT AddGeometryColumn('terralibdb',
   'bairrossp', 'spatial_data',  -1,     'POINT', 2);
```

- Do mapa de drenagem, mostrado na Figura 8.3:

[3] Aqui consideramos a existência de um banco de dados chamado terralibdb

```
CREATE TABLE drenagemsp
( cod         SERIAL,
  classe         VARCHAR(255) NULL,
  PRIMARY KEY    (cod)
);
SELECT AddGeometryColumn('terralibdb', 'drenagemsp',
'spatial_data', -1, 'LINESTRING', 2);
```

- Do mapa de municípios da grande São Paulo (Figura 8.4)

```
CREATE TABLE grande_sp
( cod           SERIAL,
  nomemunicp    VARCHAR(255) NULL,
  populacao     INTEGER,
  PRIMARY KEY  (cod)
);
SELECT   AddGeometryColumn('terralibdb',   'grande_sp',
'spatial_data', -1, 'POLYGON', 2);
```

Observe que a criação de uma tabela com tipo espacial é construída em duas etapas. Na primeira, definimos os atributos básicos (alfanuméricos) e na segunda, usamos a função `AddGeometryColumn` para adicionar a coluna com o tipo espacial. Essa função implementada no PostGIS e especificada no OpenGIS, realiza todo o trabalho de preenchimento da tabela de metadado "`geometry_columns`". Os parâmetros dessa função são:

- nome do banco de dados;
- nome da tabela que irá conter a coluna espacial;
- nome da coluna espacial;
- sistema de coordenadas em que se encontram as geometrias da tabela;
- tipo da coluna espacial, que serve para criar uma restrição que verifica o tipo do objeto sendo inserido na tabela;
- dimensão em que se encontram as coordenadas dos dados.

As tabelas de metadado do PostGIS seguem as especificações da SFSSQL e são representadas pelas seguintes tabelas:

Tabela 8.1 – Tabela de metadado do sistema de coordenadas

spatial_ref_sys

Attribute	Type	Modifier
srid	INTEGER	PK
auth_name	VARCHAR(256)	
auth_srid	INTEGER	
srtext	VARCHAR(2048)	
proj4text	VARCHAR(2048)	

Tabela 8.2 – Tabela de metadado das tabelas com colunas espaciais

geometry_columns		
Attribute	Type	**Modifier**
f_table_catalog	VARCHAR(256)	PK
f_table_schema	VARCHAR(256)	PK
f_table_name	VARCHAR(256)	PK
f_geometry_column	VARCHAR(256)	PK
coord_dimension	INTEGER	
srid	INTEGER	FK
type	VARCHAR(30)	

Após criar as tabelas de dados, podemos inserir as informações usando o comando SQL INSERT. Para isso, podemos usar a representação textual das geometrias em conjunto com a função `GeometryFromText` que recebe a representação textual e mais o sistema de coordenadas em que se encontra a geometria:

```
INSERT INTO bairrossp (bairro, spatial_data)
VALUES('JARDIM DOS EUCALIPTOS',
GeometryFromText('POINT(321588.628426 7351166.969244)',
-1));
INSERT INTO drenagemsp (classe, spatial_data)VALUES('RIOS',
GeometryFromText('LINESTRING(344467.895137 7401824.476217,
```

```
344481.584686 7401824.518728, 344492.194756 7401825.716359,
…)', -1));

INSERT INTO distritossp (denominacao, sigla, spatial_data)
VALUES('MARSILAC', 'MAR',
GeometryFromText('POLYGON((335589.530575 7356020.721956,
335773.784959 7355873.470174, …))', -1));
```

Podemos também recuperar as informações em cada tabela. Por exemplo, o comando abaixo seleciona a linha do bairro "Vila Mariana":

```
SELECT bairro, AsText(spatial_data) geom FROM bairrossp WHERE
bairro = 'VILA MARIANA';

Resultado:
    bairro    |    geom
--------------+-----------------------------------
 VILA MARIANA | POINT(334667.138663 7388890.076491)
(1 row)       |
```

Aqui, empregamos a função `AsText` para obter a representação textual, pois a partir das versões mais recentes o PostGIS utiliza o formato binário do OpenGIS (WKB) como o padrão nas consultas.

8.3.3.2 Indexação espacial

As colunas com tipos espaciais podem ser indexadas através de uma R-Tree construída no topo do GiST. A sintaxe básica para criação de um índice é a seguinte:

```
CREATE INDEX sp_idx_name ON nome_tabela
  USING GIST (coluna_geometrica GIST_GEOMETRY_OPS);
```

Para as tabelas do nosso exemplo, poderíamos construir os seguintes índices espaciais:

```
CREATE INDEX sp_idx_bairros ON bairrossp USING GIST
(SPATIAL_DATA GIST_GEOMETRY_OPS)

CREATE INDEX sp_idx_bairros ON distritossp USING GIST
(SPATIAL_DATA GIST_GEOMETRY_OPS)

CREATE INDEX sp_idx_bairros ON drenagemsp USING GIST
(SPATIAL_DATA GIST_GEOMETRY_OPS)
```

```
CREATE INDEX sp_idx_bairros ON grande_sp USING GIST
(SPATIAL_DATA GIST_GEOMETRY_OPS)
```

Os índices espaciais são usados em consultas que envolvam predicados espaciais, como no caso de consultas por janela, onde um retângulo envolvente é informado e as geometrias que interagem com ele devem ser recuperadas rapidamente.

O operador **&&** pode ser usado para explorar o índice espacial. Por exemplo, para consultarmos os municípios da grande São Paulo que interagem com o retângulo envolvente de coordenadas: 438164.882699, 7435582.150681 e 275421.967006, 7337341.000355, podemos construir a seguinte consulta:

```
SELECT * FROM grande_sp
 WHERE 'BOX3D(438164.882699 7435582.150681,
             275421.967006 7337341.000355)'::box3d
        && spatial_data);
```

Com o uso do operador **&&**, apenas alguns registros precisarão ser pesquisados para responder à pergunta acima.

8.3.3.3 Consultas espaciais

Outro grande destaque desta extensão é o grande número de operadores espaciais disponíveis, entre alguns deles podemos citar:

- Operadores topológicos conforme a Matriz de 9-Interseções DE:

  `equals(geometry, geometry)`

  `disjoint(geometry, geometry)`

  `intersects(geometry, geometry)`

  `touches(geometry, geometry)`

  `crosses(geometry, geometry)`

  `within(geometry, geometry)`

  `overlaps(geometry, geometry)`

  `contains(geometry, geometry)`

  `relate(geometry, geometry)`: retorna a matriz de intersecção.

- Operador de construção de mapas de distância:

  `buffer(geometry, double, [integer])`

- Operador para construção do Fecho Convexo:

```
convexhull(geometry)
```

- Operadores de conjunto:

```
intersection(geometry, geometry)
geomUnion(geometry, geometry)
symdifference(geometry, geometry)
difference(geometry, geometry)
```

- Operadores Métricos:

```
distance(geometry,geometry)
area(geometry)
```

- Centróide de geometrias:

```
Centroid(geometry)
```

- Validação (verifica se a geometria possui auto-interseções):

```
isSimple(geometry)
```

O suporte aos operadores espaciais é fornecido através da integração do PostGIS com a biblioteca GEOS (Geometry Engine Open Source) (Refractions, 2003). Essa biblioteca é uma tradução da API Java JTS (Java Topology Suite) (Vivid Solutions, 2003) para a linguagem C++. A JTS é uma implementação de operadores espaciais que seguem as especificações da SFSSQL. Para exemplificar o uso desses operadores e funções, mostraremos os comandos em SQL para executar as consultas dos cenários de 1 a 5, apresentados no início desse capítulo.

Cenário 1 – Usando o operador *touches*, uma possível consulta seria:

```
SELECT d2.nomemunicp
FROM grande_sp d1, grande_sp d2
WHERE touches(d1.spatial_data, d2.spatial_data)
  AND (d2.nomemunicp <> 'SAO PAULO')
  AND (d1.nomemunicp = 'SAO PAULO');
```

```
Resultado:
```

nomemunicp	nomemunicp	nomemunicp
COTIA	JUQUITIBA	CAIEIRAS
ITAPECERICA DA SERRA	ITAQUAQUECETUBA	SAO BERNARDO DO CAMPO
EMBU-GUACU	EMBU	DIADEMA
SANTANA DE PARNAIBA	TABOAO DA SERRA	SAO CAETANO DO SUL
SANTO ANDRE	BARUERI	MAUA
GUARULHOS	CAJAMAR	FERRAZ DE VASCONCELOS
MAIRIPORA	OSASCO	POA

```
(21 rows)
```

Na consulta acima, o operador *touches* retorna verdadeiro caso as geometrias de *d2* toquem na geometria de São Paulo. Esse é um exemplo de junção espacial entre duas relações (no nosso caso a mesma relação foi empregada duas vezes). Todas as geometrias da relação *d1*, com exceção da geometria São Paulo, foram avaliadas no teste topológico *touches*, pois o índice espacial não foi empregado. Em tabelas com grandes números de objetos, é importante a utilização desse índice. Ele pode ser explorado empregando-se o operador && em conjunto com os predicados da consulta anterior. Nossa consulta pode ser reescrita como:

```
SELECT d2.nomemunicp
FROM distritossp d1, distritossp d2
WHERE touches(d1.spatial_data, d2.spatial_data)
  AND (d2.nomemunicp <> 'SAO PAULO')
  AND (d1.spatial_data && d2.spatial_data)
  AND (d1.nomemunicp = 'SAO PAULO');
```

Cenário 2 – Novamente iremos empregar o operador *touches*:

```
SELECT grande_sp.nomemunicp
  FROM distritossp, grande_sp
 WHERE touches(distritossp.spatial_data,
               grande_sp.spatial_data)
   AND (distritossp.spatial_data &&
        grande_sp.spatial_data)
   AND (distritossp.denominacao = 'ANHANGUERA');
```

```
Resultado:
```

nomemunicp	nomemunicp
BARUERI	OSASCO
SANTANA DE PARNAIBA	CAIEIRAS
CAJAMAR	

```
(5 rows)
```

Cenário 3 – Para este cenário, podemos utilizar o operador *contains*:

```
SELECT COUNT(*)
  FROM bairrossp pt, distritossp pol
 WHERE contains(pol.spatial_data, pt.spatial_data)
   AND (pol.spatial_data && pt.spatial_data)
   AND pol.denominacao = 'GRAJAU';

Resultado:
count
-------
52
(1 row)
```

Cenário 4 – Aqui empregaremos os operadores *buffer* e *intersects*:

```
SELECT grande_sp.nomemunicp
  FROM grande_sp, drenagemsp
 WHERE intersects(buffer(drenagemsp.spatial_data, 3000),
                         grande_sp.spatial_data)
   AND drenagemsp.cod = 59;
Resultado:
```

Denominação	denominacao	denominacao
AGUA RASA	JAGUARA	SANTANA
ALTO DE PINHEIROS	JAGUARE	SAO DOMINGOS
BARRA FUNDA	LAPA	SE
BELEM	LIMAO	TATUAPE
BOM RETIRO	MOOCA	VILA FORMOSA
BRAS	PARI	VILA GUILHERME
CANGAIBA	PENHA	VILA LEOPOLDINA
CARRAO	PERDIZES	VILA MARIA
CASA VERDE	PIRITUBA	VILA MATILDE
CONSOLACAO	REPUBLICA	VILA MEDEIROS
FREGUESIA DO O	RIO PEQUENO	
JACANA	SANTA CECILIA	

```
(34 rows)
```

Cenário 5 – A resposta a essa pergunta poderia ser traduzida, de início, na seguinte consulta:

```
SELECT b1.bairro
  FROM bairrossp b1, bairrossp b2
 WHERE (distance(b1.spatial_data, b2.spatial_data)
                 < 3000)
   AND b1.bairro <> 'BOACAVA'
   AND b2.bairro = 'BOACAVA' order by b1.bairro;
```

Resultado:

Bairro	bairro	bairro
ALTO DA LAPA	PINHEIROS	VILA INDIANA
ALTO DE PINHEIROS	SICILIANO	VILA IPOJUCA
BELA ALIANCA	SUMAREZINHO	VILA LEOPOLDINA
BUTANTA	VILA ANGLO BRASILEIRA	VILA MADALENA
JAGUARE	VILA HAMBURGUESA	VILA RIBEIRO DE BARROS
LAPA	VILA IDA	VILA ROMANA

(18 rows)

No entanto, podemos reescrever essa mesma consulta de forma mais eficiente, utilizando o índice espacial. A estratégia básica é montar um retângulo de 6Km por 6Km centrado no bairro BOACAVA, de forma que somente seja necessário calcular a distância para os pontos que estejam dentro do retângulo. Reescrevendo a consulta temos:

```
SELECT b1.nome, astext(b1.spatial_data)
  FROM bairros b1, bairros b2
 WHERE (distance(b1.spatial_data, b2.spatial_data)
                 < 3000)
   AND (expand(b2.spatial_data, 3000) &&
        b1.spatial_data)
   AND b1.nome <> 'BOACAVA'
   AND b2.nome = 'BOACAVA' ORDER BY b1.nome;
```

8.4 Oracle Spatial

Oracle Spatial (Murray, 2003) é uma extensão espacial desenvolvida sobre o modelo objeto-relacional do SGDB Oracle. Este modelo permite definir novos tipos de dados através da linguagem de definição de dados SQL DDL, e implementar operações sobre esses novos tipos, através da linguagem PL/SQL (Urman, 2002), uma extensão da SQL (Lassen et al, 1998). Esta extensão é baseada nas especificações do OpenGIS e contém um conjunto de funcionalidades e procedimentos que permitem armazenar, acessar, modificar e consultar dados espaciais de representação vetorial.

O Oracle Spatial é formado pelos seguintes componentes:

- Um modelo próprio de dados chamado MDSYS que define a forma de armazenamento, a sintaxe e semântica dos tipos espaciais suportados.
- Mecanismo de indexação espacial.

- Um conjunto de operadores e funções para representar consultas, junção espacial e outras operações de análise espacial.
- Aplicativos administrativos.

8.4.1 Tipos de dados espaciais

O modelo de dados do Spatial consiste em uma estrutura hierárquica de elementos, geometrias e planos de informação (*layers*). Cada plano é formado por um conjunto de geometrias, que por sua vez são formadas por um conjunto de elementos.

Cada elemento é associado a um tipo espacial primitivo, como ponto, linha ou polígono (com ou sem ilhas), os quais são mostrados na Figura 8.11.

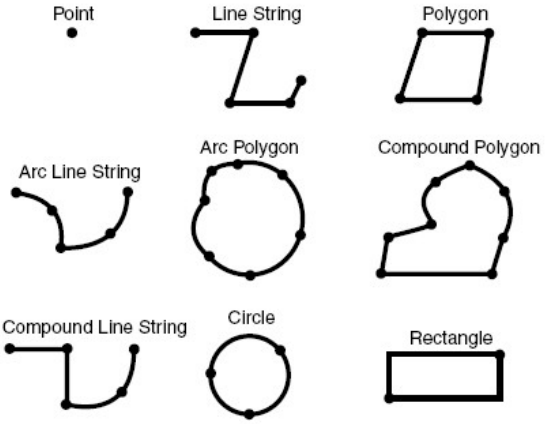


Figura 8.11 – Tipos espaciais primitivos do Oracle Spatial

Os tipos espaciais bidimensionais são compostos por pontos formados por duas ordenadas X e Y, freqüentemente correspondentes à longitude e latitude. A extensão também suporta o armazenamento e indexação de tipos tridimensionais e tetradimensionais, mas as funções e operadores só funcionam para os tipos bidimensionais.

Uma geometria pode ser formada por um único elemento, ou por um conjunto homogêneo (multipontos, multilinhas ou multipolígonos) ou heterogêneo (coleção) de elementos. Um plano de informação é formado

por uma coleção de geometrias que possuem um mesmo conjunto de atributos.

Baseado no modelo objeto-relacional, o Spatial define um tipo de objeto, para representar dados espaciais, chamado `SDO_GEOMETRY`, como mostrado a seguir.

```
CREATE TYPE sdo_geometry AS OBJECT (
SDO_GTYPE       NUMBER,
SDO_SRID    NUMBER,
SDO_POINT       SDO_POINT_TYPE,
SDO_ELEM_INFO   SDO_ELEM_INFO_ARRAY,
SDO_ORDINATES   SDO_ORDINATE_ARRAY);
```

Este objeto contém a geometria em si, suas coordenadas, e informações sobre seu tipo e projeção. Em uma tabela espacial, os atributos alfanuméricos da geometria são definidos como colunas de tipos básicos (`VARCHAR2`, NUMBER, DATE, dentre outros), e a geometria como uma coluna do tipo SDO_GEOMETRY. Em uma tabela espacial, cada instância do dado espacial é armazenada em uma linha, e o conjunto de todas as instâncias dessa tabela forma um plano de informação.

O objeto SDO_GEOMETRY é composto pelos seguintes atributos:

- SDO_GTYPE: formado por quatro números, onde os dois primeiros indicam a dimensão da geometria e os outros dois o seu tipo. Os tipos podem ser: 00 (não conhecido), 01 (ponto), 02 (linha ou curva), 03 (polígono), 04 (coleção), 05 (multipontos), 06 (multilinhas) e 07 (multipolígonos);
- SDO_SRID: utilizado para identificar o sistema de coordenadas, ou sistema de referência espacial, associado à geometria;
- SDO_POINT: é definido utilizando um objeto do tipo SDO_POINT_TYPE, que contém os atributos X, Y e Z para representar as coordenadas de um ponto. Somente é preenchido se a geometria for do tipo ponto, ou seja, se os dois últimos números do SDO_GTYPE forem iguais a "01";
- SDO_ELEM_INFO: é um vetor de tamanho variável que armazena as características dos elementos que compõem a geometria. As coordenadas de cada elemento são armazenadas em um vetor

variável chamado SDO_ORDINATES e são interpretadas através de três números armazenados no SDO_ELEM_INFO:

- SDO_STARTING_OFFSET: indica qual a posição da primeira coordenada do elemento no SDO_ORDINATES;
- SDO_ETYPE: indica o tipo do elemento;
- SDO_INTERPRETATION: indica como o elemento deve ser interpretado juntamente com o SDO_ETYPE.

- `SDO_ORDINATES`: é um vetor de tamanho variável que armazena os valores das coordenadas da geometria.

Como exemplo, mostraremos os comandos em SQL para gerar tabelas para armazenar os atributos e as geometrias:

- Dos distritos de São Paulo, mostrados na Figura 8.1

```
CREATE TABLE  DistritosSP (
cod       NUMBER(32) NOT NULL ,
sigla         VARCHAR2(20),
denominacao     VARCHAR2(200),
spatial_data    MDSYS.SDO_GEOMETRY,
PRIMARY KEY (cod))
```

- Dos bairros de São Paulo, mostrados na Figura 8.2.

```
CREATE TABLE BairrosSP (
geom_id      NUMBER(32) NOT NULL,
bairro       VARCHAR2(200),
distr        VARCHAR2(200),
spatial_data    MDSYS.SDO_GEOMETRY,
PRIMARY KEY (geom_id))
```

- Do mapa de drenagem, mostrado na Figura 8.3

```
CREATE TABLE DrenagemSP (
geom_id      NUMBER(32) NOT NULL,
classe       VARCHAR2(100)  NULL,
spatial_data    MDSYS.SDO_GEOMETRY,
PRIMARY KEY (geom_id))
```

As tabelas criadas anteriormente contêm colunas de tipos básicos como NUMBER e VARCHAR2 para armazenar atributos, e uma coluna do tipo SDO_GEOMETRY para armazenar geometrias. Note que não é preciso especificar, na criação, o tipo de geometria que será armazenado.

Mostramos a seguir alguns exemplos de comandos SQL para inserir dados nessas tabelas criadas:

```
INSERT INTO DistritosSP (cod, sigla, denominacao,
spatial_data)  VALUES (1, 'VMR', 'VILA MARIA'
MDSYS.SDO_GEOMETRY(2003, NULL, NULL,
MDSYS.SDO_ELEM_INFO_ARRAY( 1, 1003, 1 ),
MDSYS.SDO_ORDINATE_ARRAY(6,10, 10,1, 14,10, 10,14, 6,10)))

INSERT INTO DrenagemSP ( geom_id, classe, spatial_data)
VALUES (1, 'RIO',
MDSYS.SDO_GEOMETRY(2002, NULL, NULL,
MDSYS.SDO_ELEM_INFO_ARRAY( 1, 2, 1 ),
MDSYS.SDO_ORDINATE_ARRAY(10,10, 10,14, 6,10, 14,10)))

INSERT INTO BairrosSP ( geom_id, bairro, distr, spatial_data)
VALUES ( 1, 'JARDIM SHANGRILA', 'GRAJAU'
MDSYS.SDO_GEOMETRY(2001, NULL,
MDSYS.SDO_POINT_TYPE(32.628, 736.944, NULL ), NULL, NULL))
```

Observe que para incluir uma geometria através de um comando SQL é preciso montar um objeto `SDO_GEOMETRY`. Através do atributo `SDO_GTYPE`, podemos identificar qual o tipo da geometria contida no `SDO_GEOMETRY`. A geometria inserida na tabela DistritosSP é um polígono (`SDO_GTYPE=2003`), na DrenagemSP é uma linha (`SDO_GTYPE=2002`) e na BairrosSP é um ponto (`SDO_GTYPE=2001`), todos bidimensionais. Além disso, todas as geometrias são formadas por um único elemento cujo tipo pode ser identificado através dos atributos `SDO_ETYPE` e `SDO_INTERPRETATION`.

A Tabela 8.3 apresenta os tipos de elementos possíveis. Note que a primeira coordenada de um polígono tem que ser igual à última, e seus anéis externos (`SDO_ETYPE=1003`) devem estar no sentido anti-horário, e os internos (`SDO_ETYPE=2003`), no sentido horário. Nesse exemplo não estamos usando o sistema de coordenadas fornecido pela extensão (`SDO_SRID=NULL`).

Através de um comando SQL simples, é possível recuperar o objeto `SDO_GEOMETRY`. A seguir mostraremos uma consulta e o resultado retornado pelo Spatial:

```
SELECT spatial_data FROM DistritosSP
WHERE denominacao = 'PARELHEIROS'
```

```
Resultado:
SPATIAL_DATA
----------------------------------------------------
SDO_GEOMETRY(2003, NULL, NULL,
SDO_ELEM_INFO_ARRAY(1, 1003, 1),
SDO_ORDINATE_ARRAY(370139,955, 7408390,5, 369854,277,
7407971,06, 370014,832, 7408419,18, 370139,955, 7408390,5,
369854,277, 7407971,06, 370014,832, 7408419,18, 370139,955,
7408390,5))
```

Tabela 8.3 – Tipos de elementos espaciais

SDO_ ETYPE	SDO_ INTERPRETATION	*Descrição do tipo*
0	Nenhum valor	Tipo não suportado
1	1	Ponto
1	n>1	Conjunto de n pontos
2	1	Linha formada por vértices conectados por segmentos retos
2	2	Linha formada por vértices conectados por arcos circulares
1003 ou 2003	1	Polígono simples composto por vértices conectados por segmentos retos
1003 ou 2003	2	Polígono simples composto por vértices conectados por arcos circulares
1003 ou 2003	3	Retângulo otimizado composto por dois pontos
1003 ou 2003	4	Círculo formado por três pontos da circunferência
4	n>1	Linha composta por alguns vértices conectados por segmentos de reta e outros por arcos
1005 ou 2005	n>1	Polígono composto por alguns vértices conectados por segmentos de reta e outros por arcos

O modelo MDSYS apresenta dois conjuntos de tabelas de metadados que são utilizadas por funcionalidades internas da extensão, como por exemplo, nas consultas espaciais:

- Tabelas de metadados sobre geometrias armazenadas, chamadas `USER_SDO_GEOM_METADATA` e `ALL_SDO_GEOM_METADATA`.
- Tabelas de metadados sobre indexação espacial, chamadas `USER_SDO_INDEX_METADATA` e `ALL_SDO_INDEX_INFO`.

As tabelas de metadados sobre geometrias armazenam, para cada tabela espacial: o seu nome (`TABLE_NAME`); o nome da coluna de tipo geométrico (`COLUMN_NAME`); todas as dimensões das geometrias, cada uma com um mínimo retângulo envolvente e uma tolerância (`DIMINFO`); e o sistema de coordenadas (`SRID`).

Sua estrutura é mostrada a seguir:

```
TABLE_NAME   VARCHAR2(32),
COLUMN_NAME     VARCHAR2(32),
DIMINFO      SDO_DIM_ARRAY,
SRID         NUMBER
```

Após criar e inserir os dados em uma tabela espacial, o usuário deve registrar seu metadado. A seguir, mostraremos um exemplo de um comando em SQL para inserir o metadado referente à tabela espacial DistritosSP criada anteriormente.

```
INSERT INTO USER_SDO_GEOM_METADATA
VALUES ( 'DistritosSP' ,'spatial_data' ,
MDSYS.SDO_DIM_ARRAY(
    MDSYS.SDO_DIM_ELEMENT('X',275.9670,429.567,0.0005),
    MDSYS.SDO_DIM_ELEMENT('Y',833.0355,582.15,0.0005)),
    NULL)
```

Note que as geometrias armazenadas na tabela DistritosSP possuem duas dimensões X e Y, portanto, existem duas entradas no vetor `SDO_DIM_ARRAY`, uma para cada dimensão contendo seu mínimo retângulo envolvente e sua tolerância.

8.4.2 Indexação espacial

Indexação espacial, como qualquer outro tipo de indexação, fornece mecanismos para limitar o conjunto de busca, aumentando assim a performance das consultas e da recuperação dos dados espaciais. O Spatial fornece dois tipos de indexação espacial, R-tree e Quadtree,

podendo ser utilizados simultaneamente. Porém, o Oracle recomenda fortemente o uso de R-tree ao invés de Quadtree, por causa do seu desempenho. Mais informações sobre tipos de indexação espacial podem ser encontrados no capítulo 6.

O usuário pode criar uma R-tree utilizando os parâmetros default do `MDSYS` ou especificando cada parâmetro como, por exemplo, o tamanho da memória utilizada e o número de dimensões a serem indexadas. A sintaxe de um comando em SQL pra gerar uma R-tree com parâmetros default do `MDSYS` é:

```
CREATE INDEX index_name ON table_name (spatial_column_name)
INDEXTYPE IS MDSYS.SPATIAL_INDEX;
```

Como exemplo, mostraremos os comandos em SQL usados para gerar os índices espaciais das tabelas criadas anteriormente:

```
CREATE INDEX DistritosSP_IDX ON
DistritosSP(SPATIAL_DATA) INDEXTYPE IS
MDSYS.SPATIAL_INDEX

CREATE INDEX DrenagemSP_IDX ON
DrenagemSP (SPATIAL_DATA)
INDEXTYPE IS MDSYS.SPATIAL_INDEX

   CREATE INDEX BairrosSP_IDX ON BairrosSP(SPATIAL_DATA)
INDEXTYPE IS MDSYS.SPATIAL_INDEX
```

Ao inserir, remover ou modificar geometrias de uma tabela, a performance do índice espacial gerado inicialmente pode ser degradada. Para isso, a extensão fornece um conjunto de funções para avaliar a performance dos índices, como a função `SDO_TUNE.QUALITY_DEGRADATION`, e para reconstruí-lo, usando o comando `ALTER INDEX REBUILD`.

Após a criação de índices espaciais, a extensão atualiza, automaticamente, as tabelas de metadados sobre indexação citadas na seção 8.2.1.1. Essas tabelas são mantidas pela extensão e não devem ser alteradas pelos usuários.

8.4.3 Consultas espaciais

O Oracle Spatial utiliza um modelo de consulta baseado em duas etapas, chamadas de primeiro e segundo filtro, como mostrado na Figura 8.12. O primeiro filtro considera as aproximações das geometrias, pelo critério do

mínimo retângulo envolvente (MBR), para reduzir a complexidade computacional. Este filtro é de baixo custo computacional e seleciona um subconjunto menor de geometrias candidatas, que será passado para o segundo filtro. O segundo filtro trabalha com as geometrias exatas, por isso é computacionalmente mais caro e só é aplicado ao subconjunto resultante do primeiro filtro. Retorna o resultado exato da consulta.

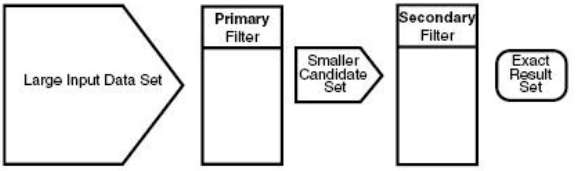


Figura 8.12 – Modelo de Consulta.

Para executar consultas e operações espaciais, o Oracle Spatial fornece um conjunto de operadores e funções que são utilizados juntamente com a linguagem SQL.

Os operadores, alguns mostrados na Tabela 8.4, são usados na cláusula WHERE e utilizam indexação espacial. Portanto, só podem ser executados sobre colunas espaciais já indexadas. As funções, algumas mostradas na Tabela 8.5, são definidas como subprogramas em PL/SQL, e utilizadas na cláusula WHERE ou em subconsultas, podendo ser executadas sobre colunas espaciais não indexadas. Devido ao fato dos operadores sempre explorarem a indexação, é recomendável usá-los, em lugar de funções, quando possível.

Tabela 8.4 – Principais operadores espaciais

Operador	*Descrição*
`SDO_FILTER`	Implementa o primeiro filtro do modelo de consulta, ou seja, verifica se os mínimos retângulos envolventes das geometrias têm alguma interação entre si.
`SDO_RELATE`	Avalia se as geometrias possuem uma determinada

	relação topológica.
SDO_WITHIN_ DISTANCE	Verifica se duas geometrias estão dentro de uma determinada distância.
SDO_NN	Identifica os *n* vizinhos mais próximos de uma geometria

A sintaxe de cada operador é mostrada a seguir:

```
SDO_FILTER   (geometry1 SDO_GEOMETRY,
              geometry2 SDO_GEOMETRY)

SDO_RELATE (geometry1 SDO_GEOMETRY,
              geometry2 SDO_GEOMETRY,
       param VARCHAR2)

SDO_WITHIN_DISTANCE (geometry1  SDO_GEOMETRY,
                           aGeom   SDO_GEOMETRY,
                           params  VARCHAR2);

SDO_NN (geometry1   SDO_GEOMETRY,
          aGeom    SDO_GEOMETRY, param  VARCHAR2,
       [, number NUMBER]);
```

Os operadores SDO_FILTER e SDO_RELATE recebem os parâmetros em comum:

- geometry1: uma coluna do tipo SDO_GEOMETRY de uma tabela que deve estar indexada.
- geometry2: um objeto do tipo SDO_GEOMETRY. Esse objeto pode ser uma instância em memória ou estar armazenado em alguma tabela espacial, que não precisa ser indexada.

Além dos parâmetros citados anteriormente, o SDO_RELATE recebe ainda o tipo da relação topológica a ser computada (param), que pode ser EQUAL, DISJOINT, TOUCH, INSIDE, COVERS, COVERREDBY, OVERLAPBDYINTERSECT, ON, CONTAINS, OVERLAPBDYDISJOINT e ANYINTERACT. Este operador é baseado no Modelo de 9-Interseções mostrado no capitulo 2.

Os operadores SDO_WITHIN_DISTANCE e SDO_NN recebem os parâmetros em comum:

- `geometry1`: uma coluna do tipo `SDO_GEOMETRY` de uma tabela que deve estar indexada.
- `aGeom`: uma instância do tipo `SDO_GEOMETRY`.

Além dos parâmetros em comum, os operadores `SDO_WITHIN_DISTANCE` e `SDO_NN` recebem outros, como a distância a ser considerada e o número de vizinhos que devem ser retornados. A extensão ainda fornece alguns operadores que não serão abordados neste capitulo, como por exemplo, `SDO_JOIN`, `SDO_TOUCH`, `SDO_INSIDE`, `SDO_ON`, dentre outros.

As funções fornecidas pelo Spatial podem ser agrupadas em:

- Relação (verdadeiro/falso) entre duas geometrias: `RELATE` e `WITHIN_DISTANCE`.
- Validação: `VALIDATE_GEOMETRY_WITH_CONTEXT`, `VALIDATE_LAYER_WITH_CONTEXT`.
- Operações sobre uma geometria: `SDO_ARC_DENSIFY`, `SDO_AREA`, `SDO_BUFFER`, `SDO_CENTROID`, `SDO_CONVEXHULL`, `SDO_LENGTH`, `SDO_MAX_MBR_ORDINATE`, `SDO_MIN_MBR_ORDINATE`, `SDO_MBR`, `SDO_POINTONSURFACE`.
- Operações sobre duas geometrias: `SDO_DISTANCE`, `SDO_DIFFERENCE`, `SDO_INTERSECTION`, `SDO_UNION`, `SDO_XOR`.

Tabela 8.5 – Algumas funções espaciais

Função	*Descrição*
`SDO_BUFFER`	Gera uma nova geometria ao redor ou dentro de uma outra, considerando uma distância passada como parâmetro.
`SDO_AREA`	Calculam, respectivamente, a área e o perímetro ou comprimento de uma geometria.
`SDO_ LENGTH`	
`SDO_DISTANCE`	Calcula a distância entre duas geometrias.

SDO_INTERSECTION SDO_UNION SDO_DIFFERENCE	Geram uma nova geometria resultante da interseção, união e diferença, respectivamente, entre outras duas.

Para exemplificar o uso desses operadores e funções, mostraremos os comandos em SQL para executar as consultas dos cenários de 1 a 5, apresentados no início desse capítulo.

Cenário 1 – Essa consulta é realizada sobre as geometrias da tabela espacial MunicipiosSP. Para respondê-la, podemos usar tanto o operador SDO_RELATE quanto o operador SDO_TOUCH na cláusula WHERE da SQL:

```
SELECT t1.nomemunicp
FROM  MunicipiosSP t1, MunicipiosSP t2
WHERE SDO_RELATE   (t1.spatial_data,
t2.spatial_data, 'mask=TOUCH') = 'TRUE'
AND   t2.nomemunicp = 'SAO PAULO'

SELECT t1.nomemunicp
FROM MunicipiosSP t1, MunicipiosSP t2
WHERE
SDO_TOUCH (t1.spatial_data, t2.spatial_data) =
'TRUE' AND   t2.nomemunicp = 'SAO PAULO'
```

Resultado:

nomemunicp	nomemunicp	nomemunicp
COTIA	JUQUITIBA	CAIEIRAS
ITAPECERICA DA SERRA	ITAQUAQUECETUBA	SAO BERNARDO DO CAMPO
EMBU-GUACU	EMBU	DIADEMA
SANTANA DE PARNAIBA	TABOAO DA SERRA	SAO CAETANO DO SUL
SANTO ANDRE	BARUERI	MAUA
GUARULHOS	CAJAMAR	FERRAZ DE VASCONCELOS
MAIRIPORA	OSASCO	POA

(21 rows)

Cenário 2 – Essa consulta é realizada sobre as geometrias de duas tabelas espaciais distintas, MunicipiosSP e DistritosSP. Podemos usar tanto o operador SDO_RELATE quanto o operador SDO_TOUCH na cláusula WHERE:

```
SELECT t1.nomemunicp
FROM  MunicipiosSP t1, DistritosSP t2
WHERE SDO_RELATE (t1.spatial_data, t2.spatial_data,
    'mask= TOUCH') = 'TRUE'
AND   t2.denominacao = 'ANHANGUERA'
```

```
SELECT t1.nomemunicp
FROM  MunicipiosSP t1, DistritosSP t2
WHERE SDO_TOUCH (t1.spatial_data, t2.spatial_data) = 'TRUE'
AND   t2.denominacao = 'ANHANGUERA'
```

Resultado:

nomemunicp	nomemunicp
BARUERI	OSASCO
SANTANA DE PARNAIBA	CAIEIRAS
CAJAMAR	

(5 rows)

Cenário 3 – Essa consulta utiliza a função de agregação COUNT juntamente com o operador SDO_RELATE ou SDO_INSIDE.

```
SELECT COUNT(*)
FROM BairrosSP t1, DistritosSP t2
WHERE SDO_RELATE (t1.spatial_data, t2.spatial_data,
'mask=INSIDE') = 'TRUE'
AND   t2.denominacao = 'GRAJAU'

SELECT COUNT(*)
FROM BairrosSP t1, DistritosSP t2
WHERE  SDO_INSIDE (t1.spatial_data, t2.spatial_data) =
'TRUE'  AND   t2.denominacao = 'GRAJAU'

Resultado:
count
52
(1 row)
```

Cenário 4 – Para realizar essa consulta utilizamos dois operadores espaciais, SDO_BUFFER e SDO_RELATE. Para gerar o buffer, o operador usará a tolerância definida na tabela de metadados. Observe que o operador SDO_RELATE recebe como parâmetro a combinação de três relações topológicas.

```
SELECT t1.denominacao
FROM  DistritosSP t1, DreanagemSP t2,
   user_sdo_geom_metadata m
WHERE SDO_RELATE (t1.spatial_data,
   SDO_GEOM.SDO_BUFFER(t2.spatial_data, m.diminfo, 3000),
       'mask=INSIDE+TOUCH+ OVERLAPBDYINTERSECT')
   = 'TRUE'
   ANDm.table_name = ' DreanagemSP '
```

```
   ANDm.column_name = 'spatial_data'
   ANDt2.geom_id = 55
```

Resultado:

denominacao	denominacao	denominacao
AGUA RASA	JAGUARA	SANTANA
ALTO DE PINHEIROS	JAGUARE	SAO DOMINGOS
BARRA FUNDA	LAPA	SE
BELEM	LIMAO	TATUAPE
BOM RETIRO	MOOCA	VILA FORMOSA
BRAS	PARI	VILA GUILHERME
CANGAIBA	PENHA	VILA LEOPOLDINA
CARRAO	PERDIZES	VILA MARIA
CASA VERDE	PIRITUBA	VILA MATILDE
CONSOLACAO	REPUBLICA	VILA MEDEIROS
FREGUESIA DO O	RIO PEQUENO	
JACANA	SANTA CECILIA	

(34 rows)

Cenário 5 – Esta consulta utiliza a função SDO_DISTANCE na cláusula WHERE.

```
SELECT t1.BAIRRO
FROM BairrosSP t1, BairrosSP t2
WHERE SDO_GEOM.SDO_DISTANCE (t1.spatial_data,
t2.spatial_data, 0.00005) < 3000
ANDt2.bairro = 'BOACAVA'
```

Resultado:

bairro	bairro	bairro
ALTO DA LAPA	PINHEIROS	VILA INDIANA
ALTO DE PINHEIROS	SICILIANO	VILA IPOJUCA
BELA ALIANCA	SUMAREZINHO	VILA LEOPOLDINA
BUTANTA	VILA ANGLO BRASILEIRA	VILA MADALENA
JAGUARE	VILA HAMBURGUESA	VILA RIBEIRO DE BARROS
LAPA	VILA IDA	VILA ROMANA

(18 rows)

8.5 Leituras suplementares

Além das extensões apresentadas neste capítulo, ainda temos o IBM DB2 Spatial Extender (IBM, 2002), o Informix Spatial e Geodetic Datablade (IBM, 2003) e a extensão do MySQL (Yarger et al, 1999).

A extensão espacial deste último SGBD encontra-se em construção (MySQL AB, 2003). O seu projeto segue as especificações da SFSSQL, e os únicos recursos disponibilizados até o momento são os tipos de dados espaciais semelhantes aos fornecidos pelo PostGIS (Point, LineString, Polygon, MultiLineString, MultiPolygon, MultiPoint e GeometryCollection) e o mecanismo de indexação através de R-Tree.

O quadro abaixo apresenta um resumo comparativo entre os SGBDs com suporte espacial:

Tabela 8.6 – Quadro Comparativo entre os SGBDs com Suporte Espacial.

Recurso	Oracle Spatial	PostgreSQL comTipos Geométricos	PostgreSQL comPostGIS	MySQL
Tipos espaciais	SFSSQL	Tipos simples	SFSSQL	SFSSQL
Indexação espacial	R-Tree e QuadTree	R-Tree nativa ou R-Tree sobre GiST	R-Tree sobre GiST	R-Tree
Operadores topológicos	Matriz 9-Interseções	Não	Matriz 9-Interseções DE	Em desenv.
Operadores de conjuntos	Sim	Não	Sim	Em desenv.
Operador de *buffer region*	Sim	Não	Sim	Em desenv.
Transformação entre sistemas de coordenadas	Sim	Não	Sim	Não
Tabelas de metadados das colunas geométricas	Sim (diferente do OGIS)	Não	Sim (conforme OGIS)	Não

Referências

HELLERSTEIN, J. M.; NAUGHTON, J. F.; PFEFFER, A. Generalized search trees for databases systems. In: international Conference in VLDB, 21., set. 1995, Zurich, Switzerland. **Proceeding.** San Francisco: Morgan Kaufman, 1995, 562-573.

IBM. **IBM DB2 Spatial Extender:** user's guide and reference. Disponível em: <http://www.ibm.com.br>. Acesso em: jun. 2002.

IBM. **Working with the geodetic and spatial datablade modules**. Disponível em: <http://www.ibm.com>. Acesso em: out. 2003.

KERNIGHAN, B. W.; RITCHIE, D. M. **C: a linguagem de programação -** padrão ANSI. Brasil: Campus, 1990.

LASSEN, A. R. O., J.; OSTERBYE, K., 1998, Object Relational Modeling, Centre for Object Technology (COT).

MURRAY, C., 2003, Oracle® Spatial User's Guide and Reference 10g Release 1 (10.1), Redwood City, Oracle Corporation, p. 602.

MySQL AB. **MySQL reference manual**. Disponível em: <http://www.mysql.com>. Acesso em: nov. 2003.

RAMSEY, P. **PostGIS manual**. Disponível em: <http://postgis.refractions.net/documentation>. Acesso em: jan. 2002.

RAVADA, S.; SHARMA, J. Oracle8i Spatial: experiences with extensible databases. In: International Symposium on Spatial Databases, 6., jul. 1999, Hong Kong, China. **Proceedings.** Berlin: Springer-Verlag, 1999, p. 355-359.

REFRACTIONS Inc. **GEOS API documentation**. Disponível em: <http://geos.refrcations.net>. Acesso em: nov. 2003.

STONEBRAKER, M.; ROWE, L. A.; HIROHAMA, M. The implementation of Postgres. **IEEE Transactions on Knowledge and Data Engineering**, v. 2, n. 1, p. 125-142, mar. 1990.

URMAN, S. **Oracle 9i Programacao PL/SqL.** Rio de Janeiro: Editora Campos, 2002. 552 p.

VIVID SOLUTIONS. **JTS technical specifications**. Disponível em: <http://www.vividsolutions.com/jts/jtshome.htm>. Acesso em: nov. 2003.

YARGER, R. J.; REESE, G.; KING, T. **MySQL & mSQL**. EUA: O'Reilly, 1999.

9 *Integração e interoperabilidade entre fontes de dados geográficos*

Marco Antonio Casanova
Daniela Francisco Brauner
Gilberto Câmara
Paulo de Oliveira Lima Júnior

9.1 Introdução

Este capítulo aborda inicialmente o problema básico de intercâmbio de dados geográficos. Em seguida, apresenta um breve resumo da arquitetura e das tecnologias relacionadas a integração e interoperabilidade no contexto de dados geográficos, ilustrado com exemplos. Focaliza então a definição de mapeamentos entre fontes de dados. Um exemplo simples ilustra as dificuldades encontradas para criar mapeamentos entre fontes que apresentem heterogeneidade nos dados, tanto em formato e estrutura, quanto em interpretação ou significado. Por fim, o capítulo trata da construção de mediadores capazes de distribuir consultas entre diversas fontes e combinar os resultados devolvidos em uma única resposta para o usuário.

O Capítulo 10 tratará em detalhe as questões de interoperabilidade especificas para o contexto da Internet, enquanto que o Capítulo 11 resumirá as propostas do *Open Geospatial Consortium* (OGC) endereçando interoperabilidade.

A motivação para este e os próximos dois capítulos nasce do crescente volume de fontes independentes de dados geográficos, interligadas entre si, fato que abre uma oportunidade única para intercâmbio de dados em tempo hábil, reduzindo custos e agilizando processos de decisão. No entanto, para tal, as aplicações devem ser capazes de interpretar e processar dados oriundos de diversas fontes, bem como localizar e acessar as próprias fontes.

Referências para problemas específicos sobre integração e interoperabilidade entre fontes de dados geográficos encontram-se ao longo do texto e sugestões para leitura adicional, ao final do capítulo.

9.2 Intercâmbio de dados geográficos

9.2.1 Problemas inerentes ao intercâmbio de dados geográficos

Entendemos por intercâmbio de dados a capacidade de compartilhar e trocar informações e processos entre diferentes usuários de informação. O grande desafio do intercâmbio de dados é enfrentar a diversidade de modelos conceituais[1] dos SIGs disponíveis no mercado. Esta diversidade faz com que muitas organizações produtoras de informação georreferenciada sigam regras conceituais vinculadas ao sistema por elas utilizado. O resultado é um ambiente heterogêneo, onde cada organização tem sua maneira de organizar a informação espacial.

A falta de modelos conceituais comuns acarreta problemas na troca de dados entre organizações utilizando SIGs distintos. Estes problemas incluem distorção de dados, perdas de qualidade da informação e de definições de atributos e informação sobre georreferenciamento. A tarefa de compartilhamento de dados geográficos deve envolver processos para garantir que a informação não seja perdida ou corrompida na transferência, e ferramentas para prevenir inconsistências resultantes de conjuntos de dados redundantes. Dada a variedade de usuários e diversidade do uso do dado espacial e sistemas de computação, é conveniente dispor de mecanismos de intercâmbio para compartilhar dados entre diferentes sistemas de computação.

Em SIGs, realizar intercâmbio de dados não é uma tarefa simples, devido a complexidade da informação geográfica envolvida, ocorrendo incompatibilidades em vários níveis. O problema vem sendo estudado em diferentes níveis, como a conversão entre formatos de dados próprios de cada SIG, a conversão entre semânticas de bancos de dados distintos e o

[1] Segundo Thomé (1998) entende-se por modelos conceituais a semântica do funcionamento de cada SIG, e a maneira como os dados devem estar organizados.

desenvolvimento ou uso de modelos gerais de dados geográficos propostos por diferentes organizações.

9.2.2 Conversão sintática de dados geográficos

O armazenamento dos dados geográficos em um SIG é organizado em estruturas próprias que descrevem características dos dados, por exemplo, coordenadas dos pontos que formam um polígono representando geometricamente uma dada entidade geográfica. As entidades geográficas possuem uma representação geométrica ou geometria e atributos associados. A geometria vetorial é baseada nas primitivas: ponto, linha e polígonos, que podem ser derivadas para formar estruturas mais complexas. A Figura 9.1 mostra exemplos de geometrias vetoriais.

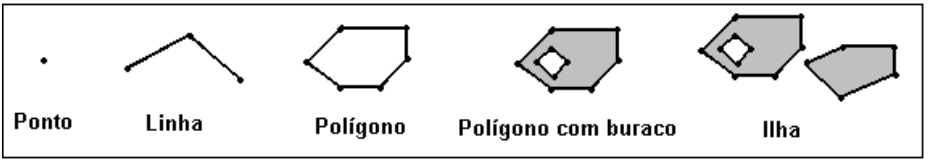
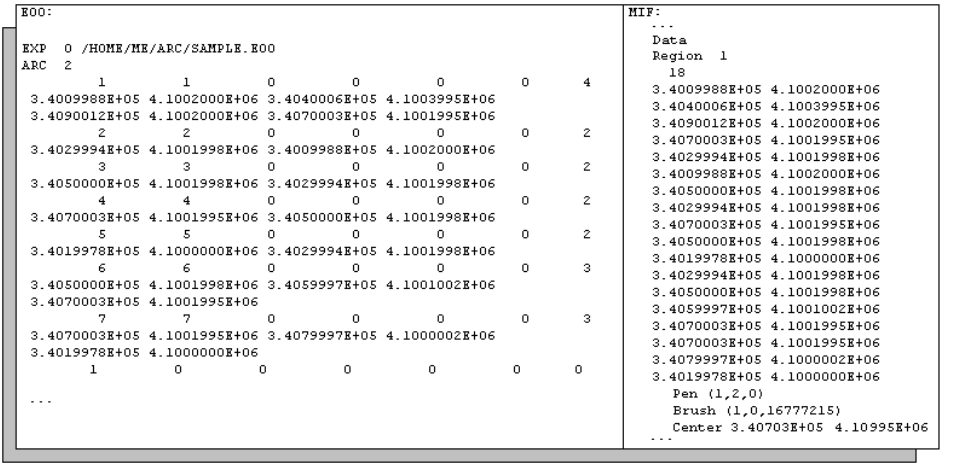


Figura 9.1 – Geometrias vetoriais usadas em SIGs.

Normalmente a organização dos dados nos SIGs é distribuída em "camadas" ("*layers*" ou "planos de informação"), onde cada camada contém uma variável geográfica distinta, por exemplo uma imagem de satélite de uma região, os municípios desta região, a sua geomorfologia, ou hidrologia. Cada camada é representada internamente em estruturas lógicas próprias de cada SIG e armazenada em arquivos distintos de acordo com o formato próprio do sistema utilizado. Este esquema de codificação e os arquivos de sistema ou de exportação possuem uma sintaxe própria para descrever as entidades (geometria e atributos), ou seja, a forma de escrever o dado. Consideramos este esquema próprio de cada sistema para armazenar e documentar seus dados, o nível sintático.

Tradicionalmente, muito do trabalho realizado em intercâmbio de dados geográficos trata de transformações estruturais (Gardels, 1996). A abordagem mais básica é a conversão sintática direta de formatos, que procura realizar a interpretação e tradução dos arquivos de informação geográfica em diferentes formatos, permitindo que um sistema

compreenda os dados provenientes de outros sistemas. Isto é eficiente desde que o desenvolvedor da conversão tenha conhecimento dos formatos envolvidos para não comprometer a qualidade dos dados no processo de conversão. Cada sistema tem sua própria definição e nomenclatura para as diferentes formas de geometria. A Figura 9.2 mostra dois trechos de arquivos em diferentes formatos de exportação. O lado esquerdo é um fragmento de um arquivo com extensão .E00 proveniente do software Arc/Info o da direita é um fragmento de um arquivo de extensão .MIF proveniente do software MapInfo. Ambos descrevem uma mesma entidade, com a mesma geometria e as mesmas coordenadas, mas com uma sintaxe própria.



```
E00:

EXP  0 /HOME/ME/ARC/SAMPLE.E00
ARC  2
         1          1          0          0          0          0       4
 3.4009988E+05 4.1002000E+06 3.4040006E+05 4.1003995E+06
 3.4090012E+05 4.1002000E+06 3.4070003E+05 4.1001995E+06
         2          2          0          0          0          0       2
 3.4029994E+05 4.1001998E+06 3.4009988E+05 4.1002000E+06
         3          3          0          0          0          0       2
 3.4050000E+05 4.1001998E+06 3.4029994E+05 4.1001998E+06
         4          4          0          0          0          0       2
 3.4070003E+05 4.1001995E+06 3.4050000E+05 4.1001998E+06
         5          5          0          0          0          0       2
 3.4019978E+05 4.1000000E+06 3.4029994E+05 4.1001998E+06
         6          6          0          0          0          0       3
 3.4050000E+05 4.1001998E+06 3.4059997E+05 4.1001002E+06
 3.4070003E+05 4.1001995E+06
         7          7          0          0          0          0       3
 3.4070003E+05 4.1001995E+06 3.4079997E+05 4.1000002E+06
 3.4019978E+05 4.1000000E+06
        1          0          0          0          0          0       0
...
```

```
MIF:
  ...
  Data
  Region  1
    18
  3.4009988E+05 4.1002000E+06
  3.4040006E+05 4.1003995E+06
  3.4090012E+05 4.1002000E+06
  3.4070003E+05 4.1001995E+06
  3.4029994E+05 4.1001998E+06
  3.4009988E+05 4.1002000E+06
  3.4050000E+05 4.1001998E+06
  3.4029994E+05 4.1001998E+06
  3.4070003E+05 4.1001995E+06
  3.4050000E+05 4.1001998E+06
  3.4019978E+05 4.1000000E+06
  3.4029994E+05 4.1001998E+06
  3.4050000E+05 4.1001998E+06
  3.4059997E+05 4.1001002E+06
  3.4070003E+05 4.1001995E+06
  3.4070003E+05 4.1001995E+06
  3.4079997E+05 4.1000002E+06
  3.4019978E+05 4.1000000E+06
    Pen (1,2,0)
    Brush (1,0,16777215)
    Center 3.40703E+05  4.10995E+06
  ...
```

Figura 9.2 – Arquivos diferentes - E00 x MIF

Entendendo a estrutura específica de um formato, é possível escrever um código que trata as características de cada sistema envolvido, viabilizando a conversão ou importação direta, atingindo desta forma um grau de interoperabilidade no nível sintático.

Por fim, cabe salientar que, apesar dos avanços no uso de gerenciadores de dados geográficos, a primeira geração de SIGs possui suporte limitado a banco de dados e utiliza arquivos para armazenamento e exportação de dados espaciais, o que torna possível encontrar um acervo relevante de dados espaciais em arquivos de

diferentes formatos. Assim, a abordagem mais básica para intercâmbio de dados geográficos é a conversão sintática direta, que procura realizar a tradução dos arquivos de informação geográfica entre diferentes formatos.

Para permitir este tipo de conversão, os SIGs trabalham com duas alternativas:

- Oferecer um formato de exportação ASCII de fácil legibilidade, como DXF (Autocad), MID/MIF (MapInfo), E00 (Arc/Info) e SPR (Spring);
- Documentar as estruturas de dados internas, como é o caso do SHP (ArcView).

9.2.3 Uso de metadados

Metadados são "dados sobre os dados", descrevem o conteúdo, condição, histórico, localização e outras características do dado, (FGDC, 2001). O objetivo do seu uso é ter um mecanismo para identificar qual dado existe, a sua qualidade, como acessá-lo e usá-lo. Assim, os metadados tratam a interoperabilidade em nível de gerenciamento da informação, facilitando a recuperação de uma informação contida em um banco de dados.

Por exemplo, uma base de dados contendo mapas com a informação sobre aptidão climática ao plantio de várias culturas pode incluir, em seus metadados, uma descrição referente ao tipo de cultura contido nos mapas, por exemplo: Aptidão climática ao plantio de abacaxi ou Aptidão climática ao plantio de algodão, o que identifica o tipo de cultura a que o dado se refere, facilitando a consulta ao banco.

Há propostas de padrões nos Estados Unidos e no Canadá (FGDC, 2001), com o objetivo de fornecer terminologia e definições comuns para conceitos relacionados aos metadados geográficos. A seguir descrevemos a proposta do FDGC, como exemplo de esquema de metadados.

O FGDC (*Federal Geographic Data Committee*) é um comitê entre agências para promover a coordenação do uso, troca e disseminação de dados espaciais nos EUA. O FGDC (2001) propõe um padrão que especifica os elementos necessários para suportar os principais usos de metadados: ajudar a manter um investimento interno em dado espacial, pelas organizações; prover informação sobre o domínio de dados de uma

organização; prover informação para processar e interpretar dados transferidos de uma fonte externa.

O padrão estabelece um conjunto comum de terminologia e definições para a documentação do dado geográfico, incluindo elementos para os seguintes tópicos: identificação, qualidade do dado, organização espacial do dado, referência espacial, informação sobre entidade e atributo, distribuição e referência do metadado (NSDI, 1997).

O padrão permite ao usuário saber: qual dado está disponível, se o dado atende suas necessidades específicas, onde achar o dado e como acessar o dado. Muitos dados estão disponíveis com este formato de metadados nos EUA onde, estados, governos locais ou do setor privado são incentivados a adotar o padrão para documentar seus dados.

O FGDC também patrocina a criação de uma *Clearinghouse* (*National Geospatial Data Clearinghouse*) um *Website* que guia usuários ao melhor dado espacial para seus projetos por meio de pesquisa a metadados. A intenção não é centralizar todos os dados geográficos em um local, mas prover *links* na Internet para distribuir *Websites* onde os dados são produzidos e mantidos. Gerenciadores documentam e disponibilizam seus dados, de acordo com o padrão, para a *Clearinghouse*, assim usuários podem achar facilmente uma informação, o que promove interoperabilidade entre organizações.

Como sua ênfase é na disponibilidade da informação, o padrão FGDC não especifica a maneira pela qual a informação está organizada nem o processo de transferência. Com exceção da parte de entidades e atributos, que pode revelar parte do significado do dado, as demais partes não descrevem a semântica da informação espacial.

O grande problema da proposta do FGDC, e do uso de metadados em geral, é a excessiva ênfase em informações que descrevem o processo de produção dos dados. Com relação à sintaxe, o padrão limita-se a indicar qual o formato em que os dados estão disponíveis. No aspecto semântico, suas informações são muito limitadas, pois o FGDC não adota o "modelo padrão" de geoinformação (campos e objetos). Adicionalmente, o padrão do FGDC reflete os compromissos inevitáveis do "projeto de comitê", pois requer uma excessiva quantidade de informações (de aplicação questionável), com dezenas de formulários.

Em resumo, a substancial burocracia envolvida em adotar o padrão FGDC não se traduz em benefícios proporcionais. Estes fatos talvez expliquem porque sua adoção ainda está limitada e porque o consórcio OpenGIS propõe seu próprio formato para metadados.

9.2.4 Uso de ontologias

O grande fator limitante de conversão de dados são as diferenças de entendimento entre comunidades de usuários distintas. Diferentes visões da realidade geográfica sempre existirão por pessoas com culturas diferentes, pois a própria natureza é complexa e leva a percepções distintas. Neste caso seria interessante conviver com estas diferentes formas de conhecimento sobre a realidade e tentar criar mecanismos para implementar e combinar diferentes visões, ou seja, representar o conhecimento geográfico no computador buscando interoperabilidade pela equivalência semântica dos conceitos entre sistemas distintos. Neste sentido, são propostos trabalhos relacionados a Ontologias e seu uso para interoperabilidade e concepção de SIGs baseados em Ontologias (Fonseca et al., 2000).

A Ontologia é uma disciplina filosófica que vem desde o estudo feito por Aristóteles sobre as categorias e a metafísica, e pode ser definida como o estudo do Ser e de suas propriedades. Para a comunidade de Inteligência Artificial, ontologias são teorias que especificam um vocabulário relativo a um certo domínio (Fonseca et al., 2000) e descrevem uma dada realidade usando o conjunto de premissas de acordo com o sentido intencional das palavras deste vocabulário.

O uso de ontologias no desenvolvimento e uso de sistemas de informação leva ao que chamamos de Sistemas de Informação baseados em ontologias (Guarino, 1998). Fonseca *et al.*,(2000) propõe um SIG baseado em ontologias, composto por um editor de ontologias, por um servidor de ontologias, por ontologias especificadas formalmente e por classes derivadas de ontologias. A Figura 9.3 mostra o esquema de um SIG baseado em ontologias.

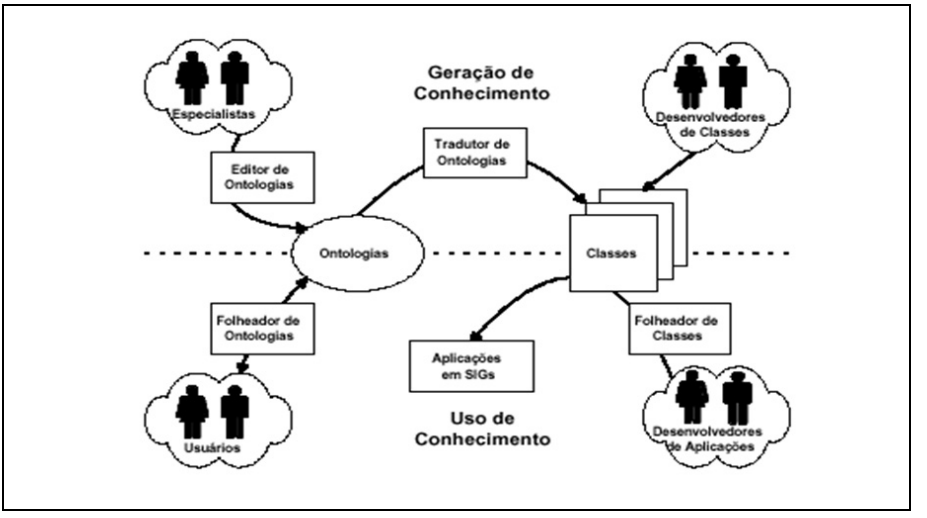


Figura 9.3 – Esquema de SIG baseado em ontologias (Fonseca et al., 2000).

Na visão apresentada acima, os especialistas especificam formalmente seu conhecimento em ontologias, que são administradas por um servidor de ontologias. As ontologias são traduzidas para componentes de software por técnicas de orientação a objetos, constituindo um conjunto de classes, gerenciadas por desenvolvedores de classes, que formam uma estrutura hierárquica representando o mundo geográfico (Fonseca et al., 2000). Desenvolvedores de aplicações utilizam o conhecimento traduzido em componentes de software (classes) para criar SIGs que também podem ser usados para troca de informações. O servidor permite o folheamento de ontologias e comunica-se com SIGs por meio de mediadores responsáveis por extrair informações destes e criar instâncias das classes que vão conter tal informação e o conhecimento extraído das ontologias.

Segundo Fonseca et al. (2000) a interoperabilidade semântica poderia ser resolvida através do uso de classes derivadas de ontologias, onde toda a manipulação de informações seria feita baseada nas definições das entidades geográficas presentes nas ontologias.

A interoperabilidade plena requer não só uma equivalência sintática entre as entidades representadas pelos sistemas, mas inclui também a equivalência de conceitos e significados destas entidades. Por exemplo, duas comunidades de informação podem utilizar nomes diferentes para o mesmo conceito (como no caso de "rio" e "curso de água"). Ou ainda, um único conceito para uma comunidade (i.e., "rio") pode ser expresso com níveis maiores de detalhe por outra (i.e., "rios perenes", "rios temporários", "riachos"). Neste sentido, é necessário que os formatos de intercâmbio de dados disponham de um mecanismo que suporte o conceito de Ontologias e comunidades de informação geográfica. Com isto, interpretações diferentes de uma mesma realidade geográfica possam ser identificadas e facilmente trocadas.

9.3 Estratégias para integração e interoperabilidade e exemplos

9.3.1 Estratégias para integração e interoperabilidade

As estratégias para tratar dos problemas de integração e interoperabilidade entre sistemas de informação geográfica podem ser classificadas em quatro categorias principais (Gupta et al., 1999).

A abordagem mais simples consiste em definir catálogos de metadados e dicionários geográficos. Um *catálogo de metadados* armazena descrições de coleções de dados armazenadas em diversas fontes (Nebert, 2002), oferecendo serviços de localização, consulta e gerência de metadados, assim como serviços de solicitação de dados, que são repassados às fontes. Um catálogo, no entanto, tipicamente não armazena os dados em si.

Um *dicionário geográfico (gazetteer)* (Atkinson, 2002) define um vocabulário consistindo do identificador, localização e parte dos atributos dos geo-objetos de interesse. Um dicionário geográfico tipicamente cobre uma região bem definida, como um país, por exemplo.

A *estratégia de federação* (Sheth e Larson, 1990) assume que as fontes de dados mantêm autonomia, mas cooperaram para oferecer suporte a operações globais, que acessam dados em mais de uma fonte. Uma federação é *fracamente acoplada* se a responsabilidade de criar e manter a federação é dos usuários, não existindo controle centralizado. Já uma

federação é *fortemente acoplada* quando existe controle centralizado para acesso às fontes de dados.

A *estratégia de armazém de dados (data warehouse)* (Miller e Han, 2001) consiste em: (1) extrair os dados das diversas fontes; (2) transformar os dados extraídos para um modelo comum; (3) armazenar os dados transformados em um único repositório. Este enfoque é viável quando o número de fontes é pequeno e relativamente estável, não necessitando constantes atualizações nos dados extraídos. O processo de transformação pode se tornar particularmente difícil face à heterogeneidade dos dados, conforme já discutido na Seção 9.2.

A *estratégia de mediação* (Gupta et al., 1999) baseia-se em uma arquitetura de 3 níveis: camada de adaptação, com as fontes de dados acessadas através de adaptadores; camada de aplicação, com as aplicações que desejam acessar as fontes; camada de mediação, com um ou mais mediadores, que registra as fontes de dados conhecidas, e processa as consultas produzidas pelas aplicações. A estratégia de mediação será discutida em detalhe na Seção 9.5.

Por fim, a *estratégia híbrida* combina a estratégia de armazém de dados com mediação. Por exemplo, a arquitetura descrita em (Voisard e Schweppe, 1998) considera 4 camadas: a camada de aplicação, que recebe requisições das aplicações; a camada de serviços virtuais, que oferece um visão uniforme do sistema (um banco de dados virtual); a camada de serviços concretos, que gerencia as tarefas dos vários componentes; e a camada de serviços de sistema, que trata de serviços internos do sistema.

9.3.2 Exemplo de catálogo de metadados e dicionário geográfico

O projeto da *Alexandria Digital Library (ADL)* (Smith e Frew, 1995; Smith, 1996; Frew et al., 2000) exemplifica a construção combinada de um catálogo de metadados e de um dicionário geográfico, disponível na Web.

Os metadados para informação geo-referenciada combinam elementos dos padrões de metadados USMARC e FGDC (USMARC, 1976; FGDC, 2001), já discutido na Seção 9.2.3. O primeiro é o padrão de metadados adotado para bibliotecas convencionais do governo dos

EUA, enquanto que o segundo define metadados primariamente para catalogar objetos geográficos em formato digital, e é adotado pelas agências do governo dos EUA. A especificação completa dos metadados mantidos pela ADL contém cerca de 350 campos.

O dicionário geográfico da ADL contém nomes e características de acidentes geográficos oriundos do *US Geological Survey's Geographic Names Information System* e do *US Board of Geographic Names*. O primeiro lista o nome de cerca de 1,8 milhões de acidentes geográficos, organizados hierarquicamente em 15 categorias. Já o segundo contém os nomes de cerca de 4,5 milhões de acidentes geográficos, inclusive acidentes submarinos.

A arquitetura da ADL (ver Figura 9.4) segue um modelo de 3 níveis: *servidores ADL* gerenciam as coleções de dados; *mediadores ADL* implementam serviços sobre as coleções de dados; *clientes ADL* oferecem os serviços aos usuários finais.

Mais detalhadamente, um servidor ADL é responsável por manter uma coleção de metadados descrevendo os objetos catalogados e por implementar os mecanismos de consulta aos metadados, de acordo com os serviços definidos pela ADL. Uma entrada no catálogo pode incluir referências a representações digitais do objeto, disponíveis *online* ou *offline,* ou a representações não-digitais. Os servidores são autônomos, desde que ofereçam os serviços definidos pela ADL. Assim, uma instituição pode implementar um servidor ADL e publicar os seus dados através de um mediador ADL.

Um cliente ADL é responsável por oferecer os serviços ADL aos usuários finais, quer sejam usuários humanos ou agentes de software. Um cliente ADL deve manter o estado das sessões e suportar interações complexas com os usuários finais.

A peça central da arquitetura é o mediador ADL, que esconde a heterogeneidade dos servidores ADL através de uma coleção de serviços padronizados, oferecidos aos clientes ADL. Estes serviços são o cerne do projeto, pois expõem a funcionalidade pretendida para o catálogo e o dicionário geográfico da ADL.

Os serviços do mediador ADL não mantém estados de sessões. Ou seja, cada chamada a um serviço do mediador é tratada como uma

transação por si só, e qualquer relacionamento entre duas chamadas deve ser codificado nos seus parâmetros. Esta decisão de projeto simplifica a implementação do mediador, mas torna mais difícil implementar consultas que são refinadas em sucessivas interações com o usuário.

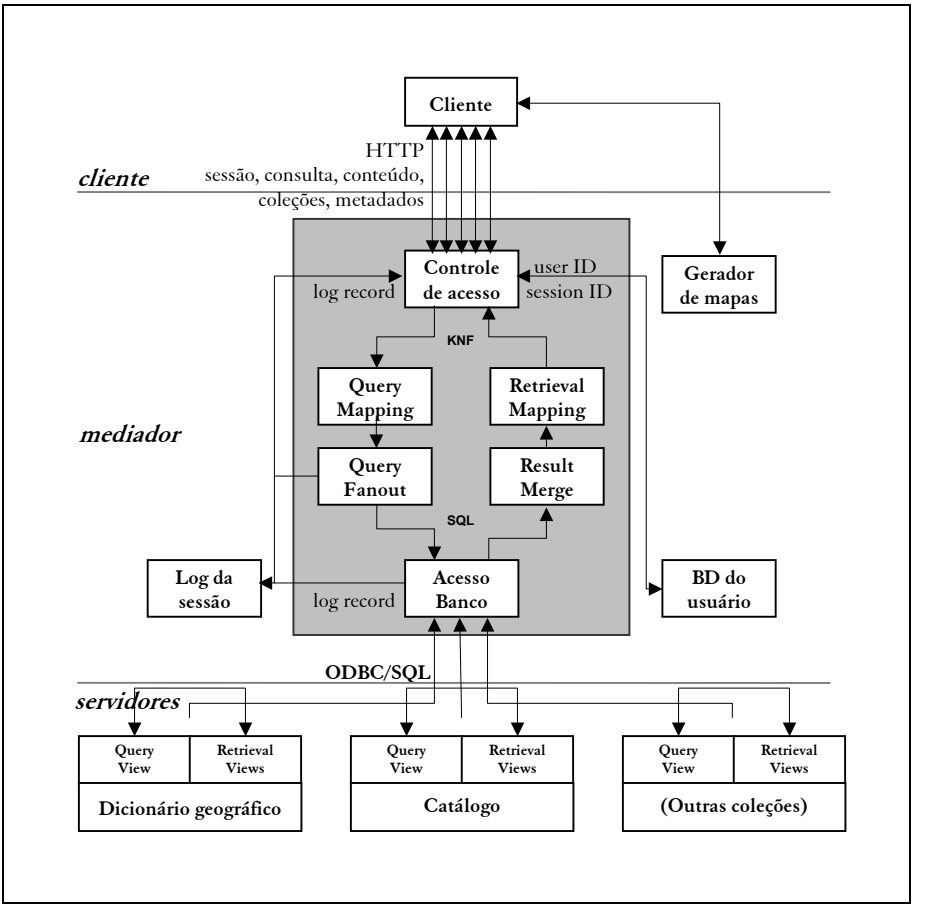


Figura 9.4 – Arquitetura da ADL (Frew et al., 2000).

9.3.3 Exemplo de armazém de dados geográficos

O *Projeto TerraServer* (Barclay, 2000) exemplifica a construção de um armazém de dados geográficos, combinado com um dicionário geográfico. O TerraServer representa o maior repositório público de

imagens de sensoriamento remoto e mapas topográficos disponível na Web.

O TerraServer foi projetado para atender simultaneamente a milhares de acessos através da Web. Um usuário pode pesquisar o repositório de quatro formas diferentes:

1. Clicando em um mapa de baixa resolução da Terra, que indica onde há dados armazenados no repositório.
2. Indicando o nome de um local.
3. Indicando as coordenadas de interesse.
4. Selecionando o nome de um local de uma lista de locais bem conhecidos.

O repositório do TerraServer foi criado a partir de quatro fontes:

USGS Digital Ortho-Quadrangles (DOQ): imagens aéreas em cinza ou infravervelho na resolução de 1m. As imagens foram orto-retificadas de tal forma que 1 pixel corresponde a $1m^2$. Este acervo cobre aproximadamente 40% do território dos EUA, correspondendo a 3 milhões de quilômetros quadrados.

USGS Digital Raster Graphics (DRG): versões digitalizadas dos mapas topográficos do USGS. As imagens foram re-amostradas de tal forma que 1 pixel corresponda à potência de 2 mais próxima. Este acervo cobre todo o território dos EUA, correspondendo a 10 milhões de quilômetros quadrados.

Imagens do SPIN-2™: imagens em cinza na resolução de 1,56m, originárias de satélites militares russos. As imagens foram rotacionadas, com o norte para cima, mas não estão orto-retificadas; foram re-amostradas de tal forma que 1 pixel corresponde a $2m^2$. Este acervo cobre parte da Europa Ocidental, EUA e o Oriente, correspondendo a 1 milhão de quilômetros quadrados.

Encarta Shaded Relief: mapa em cores naturais do globo terrestre, indicando o relevo, obtido do CD-ROM da Enciclopédia Encarta. A imagem cobre continuamente o globo entre +80º e -80º de latitude, em uma resolução de aproximadamente $1km^2$ por pixel.

A arquitetura do TerraServer segue as três camadas usuais:

Cliente: um navegador normal que suporte HTTP 1.1 e HTML 3.2.

Aplicação: processa as requisições HTTP , repassando-as ao servidor de banco de dados.

Servidor de banco de dados: armazena os dados, servindo as requisições da aplicação.

A aplicação gera páginas dinamicamente, em HTML 3.2, e as envia ao navegador. Um usuário pode realizar operações de aproximação, afastamento e deslocamento nas imagens recebidas, sem necessidade de recursos especiais.

O banco de dados armazena mapas e imagens recortados em unidades menores, chamadas de *blocos (tiles)* nesta seção. Os blocos são agrupados logicamente em coleções contíguas, chamadas de *cenas*. Os blocos são indexados por tema, resolução, cena e localização.

O banco de dados mantém uma tabela para cada tema e resolução. Cada linha de cada uma destas tabelas contém os metadados de um bloco e um campo do tipo BLOB *(binary long object)*, armazenando o próprio bloco, no formato JPEG ou GIF.

Cada bloco é armazenado no banco de dados, redundantemente, em resoluções mais baixas, formando uma pirâmide de 7 níveis (ver Capítulo 13 para detalhes de armazenamento piramidal). O acervo do USGS/DOQ é compatível com resoluções de 1 a 64m; o acervo do USGS/DRG, de 2 a 128m; e o acervo do SPIN, de 1 a 64m.

O banco de dados também armazena um dicionário geográfico, permitindo ao usuário localizar cenas por nome de locais. O dicionário contém cerca de 1,5 milhões de nomes, incluindo sinônimos, e relaciona cada nome com o sistema de coordenadas utilizado pelo TerraServer. O dicionário contém ao todo 4 milhões de linhas e ocupa 3.3 GB.

O TerraServer entrou em operação em junho de 1998 e atualmente está entre os 1000 Web sites mais visitados. A média diária situa-se em: perto de 40 mil visitas; 3,6 milhões de acessos a blocos; 69GB transferidos.

9.3.4 Exemplo de mediador

A *Missão ao Planeta Terra (Mission to Planet Earth - MTPE)* é um programa da NASA para estudar processos ligados a mudanças

climáticas, a partir de dados gerados por inúmeros satélites orbitando o planeta Terra. O *EOS Data and Information System (EOSDIS)* (Kobler, 1995) é o componente responsável por prover acesso fácil e rápido aos dados gerados no contexto do MTPE.

O EOSDIS é um sistema distribuído, organizado em três segmentos principais: o *Flight Operations Segment (FOS)* gerencia e controla os satélites e instrumentos do EOS; o *Communications and System Management Segment (CSMS)* fornece a infraestrutura de comunicação e gerência do sistema; e o *Science Data Processing Segment (SDPS)* trata do armazenamento, processamento e distribuição dos dados.

Em mais detalhe, o SDPS é o componente do EOS responsável por: aquisição de dados brutos; geração de dados derivados a partir dos dados brutos; arquivamento e distribuição dos dados derivados aos usuários. O SDPS está organizado em sete subsistemas principais:

Aquisição: recebe dados brutos e dispara processos para arquivá-los e processá-los.

Servidor de dados: fornece serviços de arquivamento físico e distribuição de dados, via FTP ou mídia removível.

Planejamento. fornece serviços para planejamento das atividades.

Processamento: executa os processos para geração de dados derivados.

Cliente: fornece uma interface para pesquisar e recuperar dados.

Interoperabilidade: fornece serviços para pesquisar e localizar serviços de dados.

Gerência de dados: fornece serviços para localizar e acessar dados.

O projeto Missão ao Planeta Terra gera vários Terabytes de dados diariamente e acumula um volume total de dados que chega a vários Petabytes, em formatos próprios, coletivamente chamados HDF-EOS. Portanto, o armazenamento e disseminação dos dados gerados representa um substancial desafio.

O projeto *NASA HDF-EOS Web GIS Software Suite (NWGISS)* (Di et al., 2001) visa exatamente tornar os dados gerados pelo EOSDIS disponíveis a outras aplicações que sigam as especificações do OGC (ver Capítulo 11 para um resumo dos padrões definidos pelo OGC e mencionados nesta seção). A arquitetura do NWGISS (ver Figura 9.5)

consiste de três componentes principais: um servidor de mapas; um servidor de geo-campos *(coverage server)*; e um servidor de catálogo. O NWGISS inclui ainda um cliente OGC WCS.

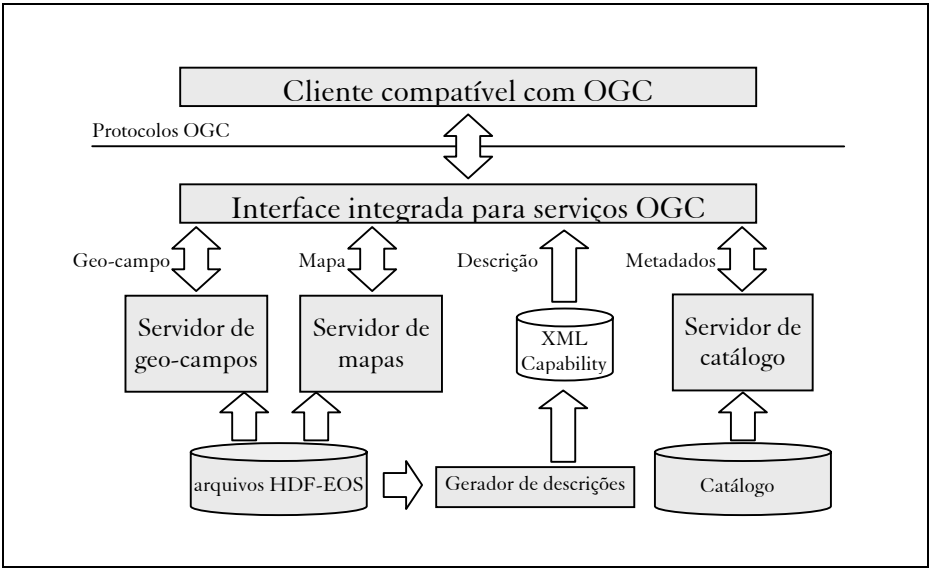


Figura 9.5 – Arquitetura do NWGISS.

O servidor de mapas do NWGISS implementa os serviços definidos no OGC WMS para todos os tipos de dados do formato HDF-EOS. Em particular, gera o geo-referenciamento do mapa em tempo de execução, se necessário. Da mesma forma, o servidor de geo-campos do NWGISS implementa os serviços definidos no OGC WCS para três formatos de dados, HDF-EOS, NITF e GeoTIFF. O servidor re-amostra, recorta e remonta geo-campos em tempo real, bem como transforma geo-campos de formato.

O cliente de geo-campos do NWGISS implementa a especificação do cliente OGC WCS. O cliente atua como um mediador para acessar geo-campos, nos formatos HDF-EOS, NITF e GeoTIFF, armazenados em servidores OGC WCS, não se limitando ao servidor OGC WCS implementado pelo NWGISS. O cliente permite acessar, visualizar, geo-retificar, re-projetar e reformatar geo-campos.

Como mediador, o cliente NWGISS é bastante limitado pois permite apenas selecionar geo-campos de mais de uma fonte, utilizando seus descritores, e visualizá-los em conjunto, entre outras operações.

9.4 Mapeamentos entre fontes de dados

9.4.1 Estratégias para definição de mapeamentos

Para interpretar e processar dados obtidos de diversas fontes, as aplicações devem ser capazes de tratar dados heterogêneos, tanto em formato e estrutura, quanto em interpretação ou significado. De fato, a heterogeneidade estrutural e semântica são problemas que bancos de dados distribuídos vem enfrentando desde longa data (Özsu e Valduriez, 1999).

Uma primeira estratégia para resolver o problema de tratar dados heterogêneos consiste em gerar mapeamentos entre pares de fontes de dados. Esta estratégia torna-se impraticável quando o número de fontes aumenta, ou quando não se conhece à priori as fontes disponíveis.

Uma segunda estratégia propõe uma descrição comunitária dos dados, chamada *esquema global ou federado*, *esquema mediado* ou *esquema de referência*, dependendo do enfoque adotado para atingir interoperabilidade (ver Seção 9.3.1). A esta descrição comunitária são então mapeadas as descrições das fontes de dados, chamadas de *esquemas locais* ou *esquemas exportados*, novamente dependendo do enfoque adotado. Esta estratégia simplifica o problema pois evita criar mapeamentos dois-a-dois entre os esquemas locais, mas ainda requer que as fontes sejam conhecidas à priori.

Uma terceira estratégia, proposta mais recentemente, adota ontologias para formalizar o esquema de referência e os esquemas locais. Utiliza ainda técnicas de alinhamento entre ontologias para simplificar a geração dos mapeamentos (Wache et al., 2001) (Uschold e Grüninger, 2001) (Mena et al. 2000).

Por fim, a definição dos mapeamentos entre as descrições locais e a descrição comunitária pode se beneficiar de uma análise dos metadados das fontes. Porém, segundo Haas & Carey (2003), a inexistência de metadados é um dos motivos para o fracasso de tentativas para atingir

interoperabilidade entre fontes de dados. Além disso, metadados não necessariamente garantem a não ambigüidade dos termos, pois um mesmo termo pode ser usado em diferentes contextos, em diferentes línguas ou até mesmo de forma errônea. Uma possível solução seria enfatizar o uso, uniforme e rigoroso, de metadados para completar a descrição dos conceitos pertinentes às fontes de dados. A própria descrição comunitária pode atuar como um vocabulário compartilhado, servindo de referencial à priori para a definição dos esquemas locais (Brauner, 2005).

9.4.2 Exemplo de definição de mapeamentos

Esta seção exemplifica questões compartilhadas pelas estratégias de federação, armazém de dados e mediação no que diz respeito à definição de mapeamentos.

Usaremos os termos *esquema de referência* para designar a descrição comunitária das fontes de dados e *esquema local* para descrever os dados visíveis em cada fonte de dados. Assumiremos que as descrições conterão definições de classes de objetos, denotando conjuntos, e de propriedades dos objetos, denotando funções mapeando objetos em objetos ou objetos em valores de dados.

Adotaremos a notação de teoria de conjuntos quando necessário. Desta forma, evitaremos escolher um particular modelo de dados para descrever os esquemas locais e o esquema de referência.

Um esquema local descreve as classes e propriedades dos dados que uma fonte deseja compartilhar, e o esquema de referência descreve as classes e propriedades dos dados oferecidos aos usuários como uma visão unificada das fontes. O esquema de referência contém ainda mapeamentos entre as classes e propriedades do esquema de referência e as classes e propriedades de cada fonte.

Há dois problemas que devem ser tratados na definição dos mapeamentos:

identificação de objetos: como definir uma forma universal de identificar os objetos nas várias fontes e, em particular, como identificar a ocorrência do mesmo objeto em fontes diferentes.

transformação das propriedades: como re-mapear os valores das propriedades oriundos de uma ou mais fontes para o esquema de referência, ou entre si.

Como exemplo, considere duas fontes de dados, F_A e F_B, descrevendo aeroportos. Suponha que o esquema local E_A da fonte F_A contenha (onde C é o conjunto de caracteres adotado e $\Re$ denota o conjunto dos números reais):

(1) classes: Aeroporto
propriedades: Código: Aeroporto $\rightarrow C^3$
Cidade: Aeroporto $\rightarrow C^{50}$
Loc: Aeroporto $\rightarrow \Re^2$

e que o esquema local E_B da fonte F_B contenha:

(2) classes: Airport
propriedades: Code: Airport $\rightarrow C^3$
Name: Airport $\rightarrow C^{20}$
Coord: Airport $\rightarrow \Re^2$

Considere que o esquema de referência E_R contenha:

(3) classes: R-Aeroporto
propriedades: R-Código: R-Aeroporto $\rightarrow C^3$
R-Loc: R-Aeroporto $\rightarrow \Re^2$
R-Nome: R-Aeroporto $\rightarrow C^{20}$

Suponha que as propriedades Código, de E_A, e Code, de E_B, de fato armazenem os códigos universais dos aeroportos, com três caracteres. O mapeamento entre E_R, E_A e E_B pode então ser definido, por exemplo, como:

(4) R-Aeroporto =
$\{x \in \text{Aeroporto} / \exists y(y \in \text{Airport} \wedge \text{Código}(x) = \text{Code}(y))\}$

(5) $(\forall x \in \text{Aeroporto})(\text{R-Código}(x) = \text{Código}(x))$

(6) $(\forall x \in \text{Aeroporto})(\text{R- Loc}(x) = \text{Loc}(x))$

(7) $(\forall x \in \text{Aeroporto})(\text{R-Nome}(x) = n \Leftrightarrow$
$(\exists y \in \text{Airport})(\text{R-Cod}(x) = \text{Code}(y) \wedge \text{Name}(y) = n))$

Note que a sentença (4) indica que a classe R-Aeroporto de E_R é definida com base na classe Aeroporto de E_A. Assim, um aeroporto só

estará disponível através do esquema de referência se e somente se for um aeroporto cadastrado na fonte F_A, por força da forma como o mapeamento foi definido (ver sentença (4)). Portanto, um aeroporto cadastrado na fonte F_B que não existir na fonte F_A não será visível através de E_R.

Assim, as sentenças (5) e (6) podem transferir diretamente as propriedades de aeroportos definidas em E_A para o esquema de referência E_R. Note que, arbitrariamente, E_R não contém a cidade do aeroporto. De fato, em geral, o esquema de referência não é a união dos esquemas locais.

A sentença (7) é mais complexa pois a fonte F_B pode conter aeroportos não cadastrados na fonte F_A. Portanto, é necessário verificar se o aeroporto existe na fonte F_A ao transferir a propriedade Name de E_B para o esquema de referência E_R.

Assuma agora que as propriedades Código e Code não representem os códigos universais dos aeroportos. Isto impediria identificar diretamente, via o código, quando dois aeroportos nas fontes F_A e F_B são o mesmo aeroporto. No entanto, se assumirmos que as propriedades Loc e Coord contém as coordenadas dos aeroportos no mesmo sistema de geo-referenciamento, então podemos substituir a sentença (7) por:

(8) $(\forall x \in \text{Aeroporto})(\text{R-Nome}(x) = n \Leftrightarrow$
$(\exists y \in \text{Airport})(\textit{Prox}(\text{Loc}(x),\text{Coord}(y)) \wedge \text{Name}(y) = n))$

onde *Prox* é uma relação de proximidade espacial que é verdadeira se e somente se dois pontos no espaço tiverem as mesmas coordenadas dentro de uma certa tolerância.

De fato, a adoção de um sistema universal de geo-referenciamento oferece também um sistema universal de identificação de objetos geográficos, ou pelo menos permite definir relacionamentos universais entre objetos geográficos.

Se assumirmos agora que as propriedades Loc e Coord contém as coordenadas dos aeroportos em sistemas de geo-referenciamento distintos, mas há suficiente informação nos metadados de Loc e Coord sobre os sistemas adotados, então devemos substituir a sentença (7) por:

(9) $(\forall x \in \text{Aeroporto})(\text{R-Nome}(x) = n \Leftrightarrow$
$(\exists y \in \text{Airport})(\textit{Prox}(\textit{Remap}(\text{Loc}(x)), \text{Coord}(y)) \wedge \text{Name}(y) = n))$

onde *Remap* é uma função que re-mapeia as coordenadas dadas por Loc para o sistema de geo-referenciamento adotado para Coord.

Este é um exemplo simples do problema de transformação das propriedades, no contexto de dados geográficos. A Seção 9.3.3 apresenta outros exemplos, quando discute a fase de transformação dos dados.

9.5 Construção de mediadores

9.5.1 Arquitetura de mediadores

Conforme adiantado na Seção 9.3.1, uma particular estratégia para integrar um conjunto de fontes de dados heterogêneas baseia-se na construção de um *sistema de mediação* com uma arquitetura em 3 camadas (ver Figura 9.6):

Camada de Aplicação: compreende as aplicações que desejam acessar as fontes.

Camada de Mediação: contém um ou mais mediadores fornecendo serviço de mediação para fontes de dados. Um mediador centraliza informações fornecidas por adaptadores, criando um *esquema mediado* das fontes de dados. O mediador também decompõe as consultas submetidas pelas aplicações em consultas a serem executadas pelos adaptadores, e reúne os resultados parciais para formar a resposta à consulta original.

Camada de Adaptação: contém os adaptadores responsáveis pelo acesso às fontes de dados. Cada adaptador esconde a heterogeneidade da fonte de dados, tornando o acesso à fonte transparente para o mediador. Para cada fonte de dados existe um adaptador que exporta algumas informações sobre a fonte, tais como: seu esquema, informações sobre seus dados e sobre seus recursos para processamento das consultas.

Ao participar de um sistema de mediação, uma fonte de dados, através do seu adaptador, deve ser capaz de expor um *esquema exportado*, descrevendo os dados que deseja tornar visíveis. Deve também oferecer serviços que permitam: (1) processar consultas sobre o esquema exportado; (2) transformar os resultados locais para os padrões definidos

para intercâmbio de dados no sistema de mediação; e (3) aceitar temporariamente dados externos, convertendo-os para o formato local.

Por sua vez, o sistema de mediação deve ser capaz de expor um *esquema mediado* com a descrição comunitária dos dados, sobre o qual as aplicações definirão consultas ao sistema. O sistema deve ser capaz de executar consultas definidas sobre o esquema mediado, aplicando as transformações necessárias sobre os dados geográficos, além dos serviços usuais para definição e controle da execução de sub-consultas às fontes de dados, combinação dos resultados e reestruturação dos dados convencionais. Esta seção aborda estes pontos em separado, seguindo (Gupta et al., 2000).

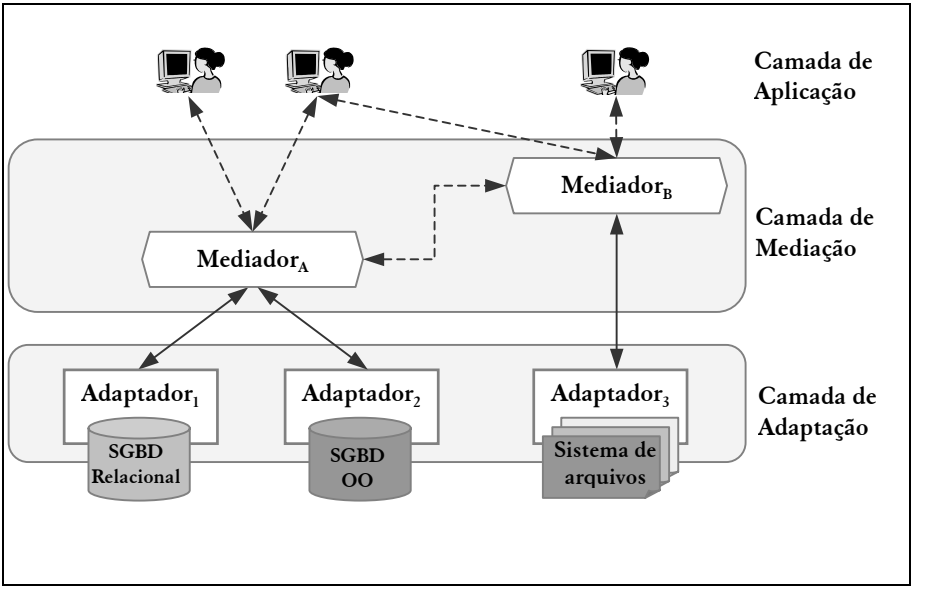


Figura 9.6 – Arquitetura mediador-adaptador (Gupta et al., 2000).

9.5.2 Serviços de adaptação

Acesso ao esquema exportado

Uma fonte de dados, através do seu adaptador, deve ser capaz de expor um esquema exportado, descrevendo os dados que deseja tornar visíveis,

e o mapeamento entre este esquema e o esquema mediado. No caso de dados geográficos, o esquema exportado deve conter substancialmente mais informação para os atributos geográficos, conforme discutido a seguir.

Seja Q uma consulta sobre o esquema mediado. O mediador utiliza os esquemas exportados e os mapeamentos existentes no esquema mediado para selecionar fontes de dados e decompor Q em sub-consultas a serem enviadas às fontes selecionadas.

O processo de escolha das fontes deverá levar em consideração o custo de converter os dados geográficos armazenados em cada fonte para o formato adequado para processar a consulta. Diferentemente de dados convencionais, este custo poderá ser bastante alto e variar substancialmente de fonte para fonte, inclusive quanto à precisão do processo de conversão. Portanto, cada esquema exportado e o esquema mediado devem conter informação adicional sobre os atributos geográficos para subsidiar a decisão da escolha da fonte e da transformação necessária.

Em geral, cada atributo geográfico G deve estar associado a um *esquema de representação*, semelhante à noção de *measurement framework* definida em (Chrisman, 1997). O esquema de representação de G pode associar dois tipos de informação a G: (1) metadados, de forma semelhante aos esquemas de metadados discutidos na Seção 9.2.3 em outro contexto; (2) outros atributos cujos valores são necessários para interpretar o valor de G. O esquema de representação pode estar diretamente associado a G, ou ser herdado do objeto O ao qual G se aplica, ou da coleção de objetos a que pertence O. A Seção 9.5.4 apresenta exemplos do processamento de consultas em um mediador que utiliza esquemas de representação.

Processamento de consultas sobre o esquema exportado

Uma fonte de dados, através do seu adaptador, deve ser capaz de processar consultas sobre o esquema exportado, devolvendo os resultados no formato definido pelo mediador.

O formato utilizado pelo mediador para passar uma consulta para o adaptador deve ser especificado à priori, quando o sistema de mediação é

criado. Por exemplo, o sistema de mediação pode adotar o padrão WFS, definido pela OGC (ver Capítulo 11), que especifica como as consultas devem ser passadas para as fontes de dados.

Transformação de dados

Uma fonte de dados, através do seu adaptador, deve ser capaz de aplicar transformações aos dados antes de enviá-los ao mediador.

A transformação mais básica consiste em converter os dados locais para o formato de intercâmbio de dados adotado pelo sistema de mediação. Por exemplo, o sistema de mediação pode adotar o padrão GML, definido pela OGC (ver Capítulo 11), que especifica um formato de transporte para dados geográficos.

De forma geral, uma *transformação de dados* é qualquer função $f: D^n \rightarrow R$ em que os argumentos de f representam tanto dados armazenados na fonte, quanto parâmetros governando a transformação a ser aplicada aos dados.

Junto com a interface da transformação, o adaptador deve expor um modelo de custo, a ser utilizado pelo mediador ao montar o plano de processamento de uma consulta. Em geral, o modelo de custo captura a complexidade computacional da transformação, em função dos parâmetros de entrada e de uma estimativa do volume de dados recebidos como entrada e do volume de dados produzidos pela transformação.

9.5.3 Serviços de mediação

Acesso ao esquema mediado

O mediador deve ser capaz de expor o esquema mediado às aplicações, e fornecer meios para mantê-lo. Em particular, deve permitir o cadastramento de uma nova fonte de dados, com seu esquema exportado e mapeamento para o esquema mediado.

As questões levantadas pela exposição do esquema mediado às aplicações são semelhantes àquelas relativas à exposição do esquema exportado. Em particular, o esquema mediado deve associar cada atributo geográfico G a um esquema de representação.

Processamento de consultas sobre o esquema mediado

Um mediador deve ser capaz de processar consultas sobre o esquema mediado, devolvendo os resultados nos formatos solicitados pelas aplicações.

O mediador executa os seguintes passos ao processar uma consulta Q definida sobre o esquema mediado:

Seleção de fontes:

- baseando-se nos mapeamentos armazenados no esquema mediado e nos atributos utilizados na consulta, o mediador seleciona conjuntos de fontes capazes de produzir o resultado da consulta.

Otimização:

- para cada conjunto de fontes, o mediador utiliza os mapeamentos do esquema mediado para criar um plano, contendo sub-consultas de tal forma que cada uma possa ser inteiramente processada por uma fonte no conjunto, e que juntas produzam o resultado da consulta original.
- o mediador estima o custo de processamento de cada plano e escolhe o de menor custo estimado.

Execução:

- o mediador controla a execução do plano escolhido no passo de otimização.

Outras estratégias mais sofisticadas envolvendo transferências temporárias de dados de uma fonte para outra, por exemplo, podem ser definidas de forma similar às estratégias de otimização de consultas utilizadas em um banco de dados distribuído (Özsu e Valduriez, 1999).

Serviços adicionais

Um mediador pode utilizar as informações sobre as fontes de dados que armazena para oferecer serviços adicionais às aplicações.

Um primeiro exemplo de serviço adicional seria prover uma interface abrangente para que os usuários ou aplicações possam explorar o esquema mediado, incluindo os metadados mantidos sobre as fontes de

dados. Desta forma, o mediador atuaria de forma semelhante a um catálogo de metadados, conforme ilustrado na Seção 9.3.2 e discutido em detalhe (Brauner, 2005).

Um exemplo deste tipo de serviço adicional, designado por *processamento cooperativo de consultas*, consistiria em aplicar transformações nas consultas submetidas para: corrigir as consultas; resolver ambigüidades; gerar consultas alternativas, quando a consulta submetida falha; fornecer explicações sobre as respostas; complementar as consultas fornecendo informação adicional à explicitamente solicitada.

9.5.4 Exemplos de processamento de consultas sobre o esquema mediado

Considere duas fontes de dados, F_A e F_B, descrevendo aeroportos. Suponha que os esquemas sejam definidos da seguinte forma:

(10) Esquema exportado por F_A:

classes:	Aeroporto		
propriedades:	Código:	Aeroporto →	C^3
	Cidade:	Aeroporto →	C^{50}
	Loc:	Aeroporto →	$\Re^2$
	Ruído:	Aeroporto →	$2^{\Re_2}$

onde

- Código é o código universal de 3 letras identificando os aeroportos nos diversos países;
- Ruído(x) associa um geo-campo vetorial a cada aeroporto x representando o nível de ruído no entorno do aeroporto, e armazenado como uma grade regular de amostras pontuais (indicado na função apenas como um conjunto de pontos no $\Re^2$, assumindo que o espaçamento da grade é o mesmo para todos os aeroportos);

(11) Esquema exportado por F_B:

classes:	Airport		
propriedades:	Code:	Airport →	C^3
	Name:	Airport →	C^{20}
	Coord:	Airport →	$\Re^2$
	Noise:	Airport →	$\wp(\Re^2)$

onde

- Code é um código próprio da fonte FB (ou seja, não é o código universal);
- Noise(x) associa um geo-campo vetorial a cada aeroporto x representando o nível de ruído no entorno do aeroporto, e armazenado como curvas de nível (representadas como elementos do conjunto $\wp(\Re^2)$);

(12) Esquema mediado E_M :

classes: R-Aeroporto

propriedades: R-Código: R-Aeroporto $\rightarrow C^3$
R-Loc: R-Aeroporto $\rightarrow \Re^2$
R-Nome: R-Aeroporto $\rightarrow C^{20}$
R-Ruído: R-Aeroporto $\rightarrow 2^{\Re 2} \times \wp(\Re)$

onde

- R-Ruído(x) = (g,c) se e somente se g for a grade regular com amostras de ruído do aeroporto x, dada pela fonte F_A, e c for a curva de nível do ruído do aeroporto x, se este for cadastrado na fonte F_B, ou for o conjunto vazio, em caso contrário:

Considere ainda os seguintes mapeamentos, idênticos aos da Seção 9.4.2:

(13) R-Aeroporto =
$\{x \in \text{Aeroporto} / \exists y(y \in \text{Airport} \wedge \text{Código}(x) = \text{Code}(y))\}$

(14) $(\forall x \in \text{Aeroporto})(\text{R-Código}(x) = \text{Código}(x))$

(15) $(\forall x \in \text{Aeroporto})(\text{R- Loc}(x) = \text{Loc}(x))$

(16) $(\forall x \in \text{Aeroporto})(\text{R-Nome}(x) = n \Leftrightarrow$
$(\exists y \in \text{Airport})(\textit{Prox}(\textit{Remap}(\text{Loc}(x)), \text{Coord}(y)) \wedge \text{Name}(y) = n))$

e um último mapeamento capturando o significado pretendido para R-Ruído:

(17) $(\forall x \in \text{Aeroporto})(\text{R-Ruído}(x) = (\text{Ruído}(x),c) \Leftrightarrow$
$((\exists y \in \text{Airport})(\textit{Prox}(\textit{Remap}(\text{Loc}(x)), \text{Coord}(y))) \wedge c = \text{Noise}(y)))\vee$
$(\neg\exists y \in \text{Airport})(c = \varnothing)))$

Considere a seguinte consulta:

Q_1 : *Qual o nome e a localização do aeroporto com código GIG?*

O mediador procederá então da seguinte forma:

Seleção de fontes:

- de (14), (15) e (16), o mediador necessita acessar as duas fontes.

Otimização:

- novamente de (14), (15) e (16), o mediador decompõe Q_1 em duas sub-consultas, Q_{1A} e Q_{1B}, a serem enviadas a F_A e F_B, respectivamente:

Q_{1A} : Selecione a localização p=Loc(x) do aeroporto x
tal que Código(x)=GIG

Q_{1B} : Selecione o nome n=Name(y) do aeroporto y
tal que *Prox*(p', (Coord(y)))

- considere um único plano:

P_1 : Envie Q_{1A} à fonte F_A;
Espere o resultado contendo a localização p do aeroporto;
Re-mapeie a localização p para o sistema adotado na fonte FB, gerando uma nova representação p' da localização do aeroporto.
Envie Q_{1B} à fonte F_B;
Espere o resultado contendo o nome n do aeroporto;
Devolva a resposta (n,p);

Execução:

- o mediador executa o plano P_1.

Esta breve descrição deixa em aberto um ponto importante. É necessário que o mediador tenha acesso ao esquema de representação de Loc e Coord para que possa re-mapear p do sistema de geo-referenciamento de F_A para o sistema de F_B. Da mesma forma, o adaptador da fonte F_B, ou a própria fonte F_B, deve ter acesso ao esquema de representação de Coord para poder computar apropriadamente o relacionamento *Prox*.

Caso este segundo problema não possa ser resolvido por F_B ou seu adaptador, o mediador deve gerar um plano alternativo, substituindo a qualificação "*Prox*(p', (Coord(y)))" por "*dist*(p', (Coord(y))<k", onde o

mediador decide o valor de k a partir dos esquemas de geo-referenciamento de Loc e Coord.

Considere agora a seguinte consulta:

Q_2 : *Obtenha o nível de ruído, em curvas de nível, do aeroporto na localização p.*

O mediador procederá então da seguinte forma:

Seleção de fontes:

- de (17), o mediador pode acessar qualquer uma das duas fontes.

Otimização:

- novamente de (17), há pelo menos 4 planos possíveis.
- O primeiro plano acessa a fonte F_A através da consulta:

Q_{2A1} : Selecione o nível de ruído C=Ruído(x) do aeroporto x tal que Loc(x)=p

O plano consiste de:

P_{2A1} : Envie Q_{2A1} à fonte F_A;
Espere o resultado contendo o nível de ruído C do aeroporto;
Transforme (no mediador) C para curvas de nível C';
Devolva a resposta C';

- O segundo plano acessa a fonte F_A através da consulta, que já inclui a transformação:

Q_{2A2} : Selecione o nível de ruído C=Ruído(x) do aeroporto x tal que Loc(x)=p, re-mapeando-o para curvas de nível

O plano consiste de:

P_{2A2} : Envie Q_{2A2} à fonte F_A;
Espere o resultado contendo o nível de ruído C do aeroporto; Devolva a resposta C;

- O terceiro plano acessa a fonte F_B através da consulta:

Q_{2A1} : Selecione o nível de ruído D=Noise(x) do aeroporto x tal que Coord(x)=p'

O plano consiste de:

P_{2B1} : Re-mapeie p para p' no sistema adotado em F_B;
Envie Q_{2B1} à fonte F_B;

Espere o resultado com o nível de ruído C' do aeroporto;
Se existir, devolva C' como resposta;
Caso contrário, execute o plano P_{2A};

- O quarto plano acessa a fonte F_B através da consulta: Q_{2A2} : Selecione o nível de ruído D=Noise(x) do aeroporto x tal que Coord(x)=*Remap*(p)

  O plano consiste de:

  P_{2B2} : Envie Q_{2B2} à fonte F_B;
  Espere o resultado com o nível de ruído C' do aeroporto;
  Se existir, devolva C' como resposta;
  Caso contrário, execute o plano P_{2A1} (ou o plano P_{2A2});
- o mediador deve estimar o custo de processar os planos e escolher um deles.

Execução:

- execute o plano escolhido no passo de otimização.

Novamente, a descrição deixa vários pontos em aberto. Por exemplo, os planos P_{2A1} e P_{2A2} diferem na capacidade do mediador e do adaptador de F_A transformarem o geo-campo armazenado em F_A de uma grade regular para curvas de nível. Dependendo da capacidade de cada um destes componentes, um dos dois planos prevalecerá.

Além disto, esta transformação pode ser muito mais cara do que tentar acessar diretamente a curva de nível armazenada em F_B, mesmo correndo o risco de não encontrar o aeroporto em F_B e ter que recorrer a F_A de qualquer maneira. Portanto, os planos P_{2B1} e P_{2B2} podem ser muito mais eficientes do que os primeiros dois planos.

Da mesma forma, os planos P_{2B1} e P_{2B2} diferem em qual componente realizará a conversão de p para o sistema de geo-referenciamento adotado por F_B.

Em qualquer um dos casos, novamente é necessário que o mediador tenha acesso aos esquemas de representação de Ruído e Noise para decidir qual dos planos é viável, ou de menor custo.

9.6 Leituras Suplementares

Recomenda-se inicialmente um estudo da série de padrões relativos a dados geográficos publicados pela *International Standards Organization* (ISO):

- ISO 19115 Geographic Information – Metadata standard
- ISO 19107:2003 Geographic Information – Spatial Schema
- ISO 19109 Geographic Information – Rules for Application Schema
- ISO 19110 Geographic Information – Methodology for Feature Cataloguing
- ISO 19111:2003Geographic Information – Spatial Referencing by Coordinates
- ISO 19112:2003 Geographic Information – Spatial Referencing by Geographic Identifier

Em particular, o ISO 19115 define um esquema de metadados para descrever informação geográfica e serviços. O modelo em UML deste padrão está disponível como o ISO TC211 Harmonized Model. Estes padrões podem ser encontrados sob forma de ontologias em Islam et. al. (2004).

Há várias ontologias bastante interessantes definindo vocabulários geográficos. Duas delas merecem destaque:

- *CYC Geographic Vocabulary*, disponível em: http://www.cyc.com/cyc-2-1/vocab/geography-vocab.html
- *Alexandria Digital Library Feature Type Thesaurus*, disponível em: http://www.alexandria.ucsb.edu/gazetteer/FeatureTypes/ver070302/index.htm)

Descrições de várias arquiteturas e tecnologias para facilitar a integração e interoperabilidade entre fontes de dados geográficos podem ser encontradas na literatura.

Abordagens gerais para o problema de integração e interoperabilidade cobrem desde catálogos de dados geográficos a mediadores capazes de processar consultas a fontes distintas (Abel at al., 1998) (DeVogele et al., 1998) (Bishr, 1998) (Laurini, 1998) (Jacobsen e Voisard, 1998)

(Bergmann et al., 2000) (Tanin et al., 2002). Enfoques baseados em XML ou em *Web services* podem ser encontrados em (Gupta et al., 1999) (Gupta et al., 2000) (Alameh, 2003) (Manpuria et al., 2003). Há inúmeras iniciativas baseadas nos padrões do OGC, como (Boucelma et al., 2002) (Essid et al., 2004), para citar algumas referências. Mais recentemente, várias abordagens para construção de mediadores baseadas em ontologias foram propostas na tentativa de explorar o maior poder de expressão das linguagens para descrever ontologias e as tecnologias para alinhamento de ontologias (Mena et al., 2000; Wiegand e Zhou, 2001).

As áreas de armazém de dados e mineração de dados espaciais merecem atenção especial pelo potencial das aplicações. Um ponto de partida é o livro texto de Miller e Han (2001).

Referências

ABEL, D. J.; OOI, B. C.; TAN, K.-L.; TAN, S. H. Towards integrated geographical information processing. **International Journal of Geographical Information Science**, v. 12, n.4, p. 353-371, 1998.

ALAMEH, N. Chaining Geographic Information Web Services. **IEEE Internet Computing archive**, v. 7, n.5, p. 22-29, 2003.

ATKINSON, R. F., J., 2002, **Gazetteer Service Profile of the Web Feature Service Implementation Specification**, Open Geoscience Consortium.

BARCLAY, T.; GRAY, J.; SLUTZ, D. Microsoft TerraServer: A Spatial Data Warehouse. **ACM SIGMOD Record**, v. 29, n.2, p. 307-318, 2000.

BERGMANN, A.; BREUNIG, M.; CREMERS, A. B.; SHUMILOV, S. Towards an Interoperable Open GIS. In: International Workshop on Emerging Technologies for Geo-Based Applications. Ascona, Switzerland, 2000. p. 283-296.

BISHR, Y. Overcoming the semantic and other barriers to GIS interoperability. **International Journal of Geographical Information Science**, v. 12, n.4, p. 299-314, 1998.

BOUCELMA, O.; LACROIX, Z. E. M. A WFS-Based Mediation System for GIS Interoperability. In: 10th ACM international Symposium on Advances in Geographic Information Systems. McLean, VA, USA, 2002. p. 23-28.

BRAUNER, D. F. **Uma Arquitetura para Catálogos de Objetos baseados em Ontologias**. Rio de Janeiro: Pontifícia Universidade Católica do Rio de Janeiro, 2005.Departamento de Informática., 2005.

CHRISMAN, N. **Exploring Geographic Information Systems.** New York: John Wiley & Sons, 1997.

DEVOGELE, T. P., C.; SPACCAPIETRA, S. On spatial database integration. **International Journal Geographical Information Science**, v. 12, n.4, p. 335-352, 1998.

DI, L.; YANG, W.; DENG, M.; DENG, M.; MCDONALD, K. The prototype NASA HDF-EOS Web GIS Software Suite (NWGISS). In: NASA Earth Science Technologies Conference. Greenbelt, Maryland, USA, 2001. p.

ESSID, M.; BOUCELMA, O.; COLONNA, F. M.; LASSOUED, Y. Query Processing in a Geographic Mediation System. In: 12th Annual ACM International Workshop on Geographic Information Systems. ACM Press New York, Washington DC, USA, 2004. p. 101-108.

FEDERAL GEOGRAPHIC DATA COMMITTEE (FGDC). **Content Standard for Digital Geospatial Metadata WorkBook**. Reston, VA. Disponível em: <http://www.fgdc.gov/metadata/constan.html>.

FONSECA, F.; EGENHOFER, M.; BORGES, K. Ontologias e Interoperabilidade Semântica entre SIGs. In: Workshop Brasileiro em Geoinformática (GeoInfo2000), 2., São Paulo, 2000. **Anais**. São José dos Campos: INPE, 2000. p. 45 - 52.

FREW, J.; FREESTON, M.; FREITAS, N.; HILL, L.; JANÉE, G.; LOVETTE, K.; NIDEFFER, R.; SMITH, T.; ZHENG, Q. The Alexandria Digital Library architecture. **International Journal on Digital Libraries**, v. 2, n.4, p. 259-268, 2000.

GARDELS, K. The Open GIS Approach to Distributed Geodata and Geoprocessing. In: Third International Conference/Workshop on Integrating GIS and Environmental Modeling. Santa Fe, NM, USA, 1996. p. 21-25.

GUARINO, N. **Formal ontology and information systems**. Amsterdam, Netherlands: IOS Press, 1998, p.3-15.

GUPTA, A.; MARCIANO, R.; ZASLAVSKY, I.; BARU, C. Integrating GIS and Imagery through XML-Based Information Mediation. In: International Workshop on Integrated Spatial Databases. Portland, ME, USA, 1999. p. 211.

GUPTA, A.; ZASLAVSKY, I.; MARCIANO, R. Generating Query Evaluation Plans within a Spatial Mediation Framework. In: 9th International Symposium on Spatial Data Handling. Beijing, China, 2000. p. 8a18-8a31.

HAAS, L.; CAREY, M. Will federated databases ever go anywhere? In: Lowell Database Research Self-Assessment Meeting. Lowell, Massachusetts, USA, 2003.

ISLAM, A.S.; BERMUDEZ, L.; BERAN, B; FELLAH, S.; PIASECKI, M, **Ontologies for ISO Geographic Information Standards**. http://loki.cae.drexel.edu/~wbs/ontology/2004/09/bug/iso-19115/index.htm

JACOBSEN, A. H.; VOISARD, A. CORBA – Based Open Geographic Information Systems. In: European Conference on Parallel and Distributed Systems. 1998. p.

KOBLER, B.; BERBERT, J.; CAULK, P.; HARIHARAN, P. C. Architecture and design of storage and data management for the NASA Earth observing system Data and Information System (EOSDIS). In: 14th IEEE Symposium on Mass Storage Systems. Monterey, California, USA, 1995. p. 65.

LAURINI, R. Spatial multi-database topological continuity and indexing: a step towards seamless GIS data interoperability. **International Journal of Geographical Information Science**, v. 12, n.4, p. 373-402, 1998.

MANPURIA, V.; ZASLAVSKY, I.; BARU, C. Web Services for Accuracy-Based Spatial Query Rewriting in a Wrapper-Mediator System. In: Fourth International Conference on Web Information Systems Engineering Workshops (WISEW'03). Rome, Italy, 2003. p. 63-71.

MENA, E.; ILLARRAMENDI, A.; KASHYAP, V.; SHETH, A. OBSERVER: An Approach for Query Processing in Global Information Systems based on Interoperation across Pre-existing Ontologies. **International Journal on Distributed And Parallel Databases (DAPD)**, v. 8, n.2, p. 223-272, 2000.

MILLER, J. H.; HAN, J. **Geographic Data Mining and Knowledge Discovery**. Bristol, PA, USA: Taylor & Francis, Inc., 2001.

NATIONAL SPATIAL DATA INFRASTRUCTURE (NSDI). **Geospatial Metadata.** Disponível em: <http://www.fgdc.gov/publications/documents/metadata/metafact.pdf>.

NEBERT, D., 2002, Catalog Services Specification, Version 1.1.1, OpenGIS© Implementation Specification, Open Geoscience Consortium.

ÖZSU, M. T.; VALDURIEZ, P. **Principles of distributed database systems**. Prentice-Hall, Inc., 1999.

SHETH, A.; LARSON, J. Federated database systems for managing distributed, heterogeneous and autonomous databases. **ACM Computing Surveys**, v. 22, n.3, p. 183-236, 1990.

SMITH, T. R. A Digital Library for Geographically Referenced Materials. **IEEE Computer**, v. 29, n.5, p. 54-60, 1996.

SMITH, T. R.; FREW, J. Alexandria Digital Library. **Communications of the ACM**, v. 38, n.4, p. 61-62, 1995.

TANIN, E.; BRABEC, F.; SAMET, H. Remote Access to Large Spatial Databases. In: 10th ACM International Symposium on Advances in Geographic Information Systems. McLean, Virginia, USA, 2002. p. 5-10.

THOMÉ, R. **Interoperabilidade em geoprocessamento: Conversão entre modelos conceituais de sistemas de informação geográfica e comparação com o padrão OpenGIS. 1998**. 196 f. (INPE-7266-TDI/708). Dissertação (Mestrado em Computação Aplicada) – Instituto Nacional de Pesquisas Espaciais, 1998.

USCHOLD, M.; GRÜNINGER, M. **Ontologies: Principles, methods and applications**. Knowledge Engineering Review, v. 11, n.2, p. 93 -155, 2001.

USMARC, 1976, A MARC Format, in OFFICE, L. O. C. M. D., ed., Washington, D.C., Library of Congress Information Systems Office.

VOISARD, A.; SCHWEPPE, H. Abstraction and Decomposition in Interoperable GIS. **International Journal of Geographic Information Science**, v. 12, n.4, p. 315 - 333, 1998.

WACHE, H.; VÖGELE, T.; VISSER, U.; STUCKENSCHMIDT, H.; SCHUSTER, G.; NEUMANN, H.; HÜBNER, S. Ontology-Based Integration of Information – A Survey of Existing Approaches. In: IJCAI-01 Workshop: Ontologies and Information Sharing. Seattle, WA, USA, 2001. p. 108 -117.

WIEGAND, N.; ZHOU, N. Ontology-Based Geospatial XML Query System. In: 4th AGILE Conference on Geografical Information Science. 2001.

10 *Disseminação de dados geográficos na Internet*

Clodoveu A. Davis Jr.
Ligiane Alves de Souza
Karla A. V. Borges

10.1 Introdução

Este capítulo explora diversas possibilidades de disseminação de dados geográficos através da Internet. Apresenta inicialmente a arquitetura de três camadas, típica de sistemas de informação baseados na Web. Em seguida, discute as três principais formas de disseminação de dados geográficos: (1) a disseminação direta, usando características gráficas típicas dos *browsers*, complementadas ou não por ferramentas e recursos adicionais; (2) as bibliotecas digitais de informações geográficas; e (3) uma visão da emergente área de infra-estruturas de dados espaciais. Por fim, tece algumas considerações finais, destacando principalmente o outro lado da disseminação de dados geográficos na Internet: os mecanismos de busca voltados para localização e dados geográficos.

A Internet rapidamente se tornou o meio preferencial para disseminação de dados. Sua (quase) universalidade, associada a custos de acesso cada vez mais baixos, motivou o desenvolvimento de toda uma nova classe de sistemas de informação, com uma arquitetura diferenciada em relação a seus predecessores.

Esse movimento se estende aos dados geográficos: atualmente, todos os principais fornecedores de software para SIG dispõem de alternativas para acesso a dados geográficos através da Web. Apesar dessa forte tendência, constata-se uma grande variedade de estilos de implementação, recursos tecnológicos e arquiteturas internas das soluções, cada qual refletindo um conjunto de preocupações e voltada para um nicho de aplicação mais ou menos específico.

De forma coerente com a multiplicidade cultural da Internet, todas as alternativas encontram-se em uso, cada qual em seu nicho; não se pode apontar uma única alternativa tecnológica "vencedora", ou preponderante sobre todas as demais. Existe ainda muito espaço para a expansão dessa área de conhecimento e do mercado dela decorrente.

10.2 Arquitetura de sistemas de informação baseados na Web

A arquitetura dos SGBD acompanhou, de modo geral, as tendências de descentralização e modularização observadas nas arquiteturas de *hardware* (Elmasri e Navathe, 2004).

Na era dos mainframes, os SGBD eram igualmente monolíticos, incluindo todas as funções de acesso aos dados no mesmo ambiente em que eram processadas as aplicações e produzidas as telas de interface com o usuário.

Com a queda dos preços dos equipamentos e ascensão dos computadores pessoais, passamos a ter a possibilidade de transferir parte do processamento para um equipamento que era, anteriormente, um mero terminal "burro". Isso levou ao desenvolvimento da *arquitetura cliente/servidor*, inicialmente concebida não apenas para o gerenciamento de dados, mas para que se pudesse dispor de máquinas capazes de prestar qualquer tipo de serviço especializado em uma rede de computadores, tais como gerenciamento de impressão, de arquivos, de backup, entre outros (Figura 10.1).

Nessa arquitetura, os processos relacionados com o acesso do usuário aos serviços são chamados processos *cliente*, enquanto os processos que lidam com os recursos especializados são denominados processos *servidor*. Observe que essa denominação é frequentemente confundida com a designação de computadores de maior ou menor porte – o que às vezes se justifica pois, na maioria dos casos, equipamentos encarregados de executar processos servidores para uma grande quantidade de processos cliente precisariam, em princípio, ser mais robustos e poderosos.

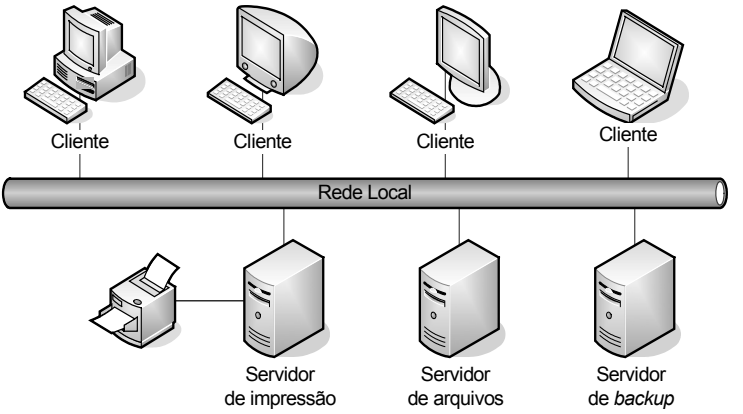


Figura 10.1 – Arquitetura cliente/servidor (2 camadas). Fonte: adaptado de (Elmasri e Navathe, 2004).

A arquitetura cliente-servidor adapta-se bem às necessidades de SGBD, e está hoje totalmente incorporada aos produtos comerciais. Nesse caso, o servidor de banco de dados provê, como serviço, respostas a consultas enviadas por processos cliente. Por esse motivo, em SGBD relacionais, o servidor de banco de dados é também denominado *servidor de consultas*, ou *servidor SQL* (Elmasri e Navathe, 2004).

Assim, de acordo com a arquitetura cliente-servidor aplicada aos SGBD, a camada do cliente fica com as funções de gerenciamento da interface com o usuário, de dicionário de dados, de interfaceamento com linguagens de programação, entre outras, em um nível mais alto.

A camada do servidor gerencia as funções de armazenamento em disco, controle de concorrência, backup e recuperação de dados, e outras funções de mais baixo nível (Figura 10.2).

Entre essas duas camadas, a comunicação ocorre por meio do estabelecimento de uma conexão através da rede. Uma das alternativas para essa conexão é o uso de um padrão de conectividade, denominado *Open Database Connectivity* (ODBC), que neutraliza, por meio de uma interface comum de programação (ou API, *Application Programming Interface*), as diferenças entre os diversos programas envolvidos, desta forma evitando a incompatibilidade. Um padrão de conectividade

semelhante, denominado JDBC, foi criado para que clientes escritos na linguagem Java pudessem também acessar os SGBD por meio de uma interface padronizada.

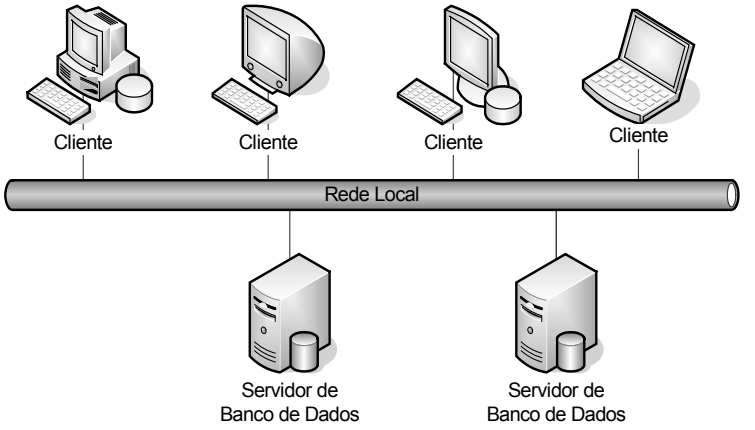


Figura 10.2 – Arquitetura cliente/servidor aplicada a SGBD. Fonte: adaptado de (Elmasri e Navathe, 2004).

A arquitetura cliente-servidor original tem sido denominada *arquitetura de duas camadas*, já que seus componentes estão logicamente distribuídos entre dois níveis, o do cliente e o do servidor. A simplicidade e a facilidade de adaptação de arquiteturas anteriores para o funcionamento em duas camadas justificam sua grande popularidade. Por outro lado, sua escalabilidade pode deixar a desejar, caso exista a necessidade de se ter um número muito grande de clientes para um servidor. Isso levou à proposição e à implementação da *arquitetura de três camadas*, tipicamente empregada em sistemas baseados na Web.

A arquitetura de três camadas acrescenta, entre a camada do cliente e a camada do servidor, uma camada intermediária, que recebe o nome de *middleware*, ou *servidor de aplicações* (*application server*) (Figura 10.3). Em bancos de dados, essa camada é responsável por manter regras de negócio, restrições e outros elementos necessários para a aplicação. Desta forma, a as três camadas são compostas, basicamente, de interface com o usuário

(cliente), regras de negócio (aplicação) e acesso a dados (servidor). Como um servidor de aplicação pode também controlar o acesso de usuários aos dados, e rotear as requisições para algum servidor selecionado por ele sem a interferência do usuário, a arquitetura de três camadas possibilita melhorar sensivelmente a escalabilidade dos sistemas de informação apoiados em um SGBD.

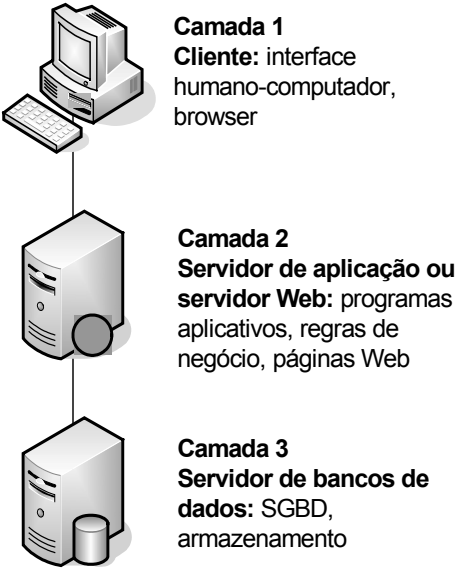


Figura 10.3 – Arquitetura de três camadas. Fonte: adaptado de (Elmasri e Navathe, 2004).

10.3 Disseminação direta

A *World-Wide Web* (Web) foi originalmente concebida, de forma bastante despretensiosa para os padrões atuais, para trabalhar com "documentos contendo texto, e eventualmente imagens" (Berners-Lee et al., 1994). De fato, além da possibilidade de navegação usando hipertexto, os primeiros clientes de servidores Web – os hoje onipresentes *browsers* – permitiam apenas a apresentação de imagens simples, em formato GIF ou JPEG. O protocolo *HyperText Transfer Protocol* (HTTP) e a linguagem de

marcação *HyperText Markup Language* (HTML), base do funcionamento da Web, permitem a elaboração, o preenchimento online e o envio do conteúdo de formulários de cliente para servidor, provendo assim alguma interatividade, embora bem mais limitada que os recursos usualmente encontrados em aplicações gráficas convencionais.

Ao longo da evolução da Web, algumas alternativas para acesso a dados geográficos foram sendo implementadas dentro dessas limitações. A evolução dos padrões ligados à Internet, capitaneados pelo *World-Wide Web Consortium* (W3C)[1], acompanhados pela evolução tecnológica na área de sistemas distribuídos, possibilitou uma enorme ampliação dessas alternativas, em particular após o surgimento da linguagem Java (Sun Microsystems, 1994). Apresentamos, nas subseções a seguir, as principais alternativas utilizadas histórica e atualmente para disseminação de dados geográficos através da Web.

10.3.1 Mapas estáticos em formato de imagem

A forma mais básica de disseminação de dados geográficos na Web é, naturalmente, a publicação de mapas estáticos em formato de imagem, embutidas em páginas Web. Apesar de a interatividade ser nula, é possível produzir e manter disponíveis, para consulta e referência histórica, grandes conjuntos de mapas temáticos. A existência de documentos eletrônicos contendo mapas estáticos possibilita o desenvolvimento de esforços de preservação (Thomaz, 2004) independentes dos bancos de dados de onde (espera-se) eles tenham se originado. A Figura 10.4 apresenta um exemplo de uso dessa alternativa.

[1] http://www.w3c.org

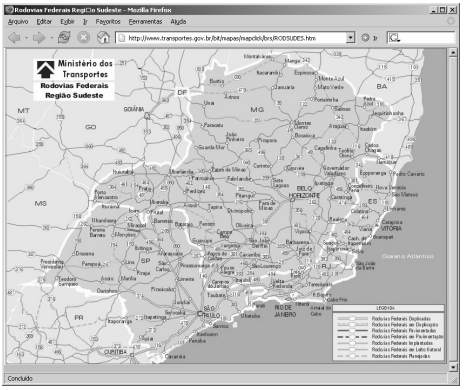


Figura 10.4 – Mapa estático, publicado em formato JPEG. Fonte: Web site do Ministério dos Transportes (www.transportes.gov.br)

10.3.2 Mapas gerados a partir de formulários

Muito adotada na segunda metade da década de 1990, esta alternativa consiste em oferecer ao usuário um formulário para preenchimento. Neste formulário são solicitadas informações quanto à região geográfica de interesse (muitas vezes solicitando uma referência explícita a um número de mapa em uma coleção ou articulação), à composição do mapa (camadas que deveriam aparecer) e mesmo alguns elementos de composição visual (cores, espessura de linhas, cores ou hachuras de preenchimento). Quando o usuário termina o preenchimento do formulário, as informações são transmitidas a um servidor, que recupera os dados necessários e converte o mapa final para um formato de imagem, como GIF ou JPEG. Esta imagem é então inserida numa página Web criada dinamicamente, e transmitida para o usuário.

Considerando um nível mínimo de interatividade, esta alternativa é talvez a mais natural do ponto de vista dos *browsers*, uma vez que requer apenas a apresentação de imagens. No entanto, é uma alternativa limitada, por diversos motivos. Em primeiro lugar, porque não permite que o usuário navegue pelo mapa, interagindo diretamente com os objetos apresentados, sem que haja a necessidade de re-geração da imagem. Além disso, a transmissão de imagens apresenta um compromisso entre

tamanho, qualidade e resolução, por um lado, e a velocidade de transmissão, pelo outro. Por fim, existe o problema de sobrecarga no servidor, que precisa construir o mapa em formato matricial, muitas vezes a partir de dados vetoriais, e transmiti-lo para o cliente. Este último fator claramente reduz a escalabilidade dessa solução. Observe que qualquer operação simples, como *zoom* ou *pan*, exige a formação de um novo mapa-imagem (sempre com processamento no servidor) e nova transmissão.

Um exemplo desse tipo de recurso está apresentado parcialmente na Figura 10.5, referente a um gerador de mapas do *United States Geological Survey* (USGS). Além de prover as coordenadas de um retângulo envolvente para a área que deseja visualizar, o usuário deve decidir sobre (1) acrescentar ou não uma borda ao redor da área escolhida, (2) a maneira de lançar pontos sobre o mapa, (3) a inclusão ou não de dados topográficos, (4) o lançamento de texto sobre o mapa, indicando as coordenadas onde fazê-lo, (5) a inclusão ou não de limites políticos, e (6) a inclusão ou não de rios. Apenas então o usuário pode clicar um botão e disparar o processo de montagem da imagem.

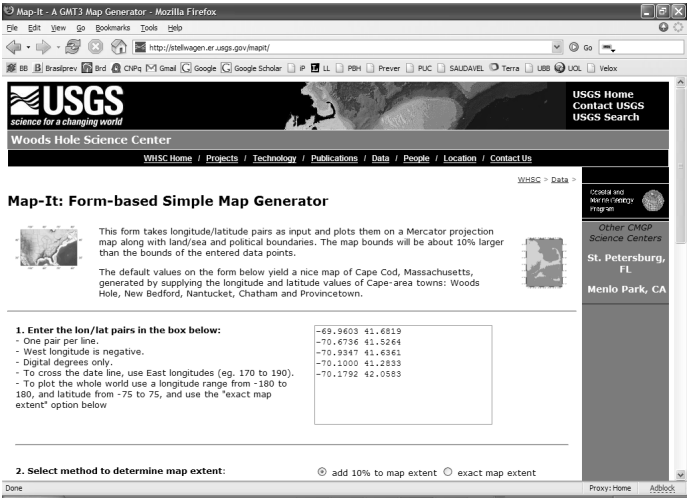


Figura 10.5 – Mapa baseado em formulário. Fonte: http://stellwagen.er.usgs.gov/mapit/. O gerador de mapas é baseado na biblioteca de código aberto GMT (http://gmt.soest.hawaii.edu/).

10.3.3 Navegação baseada em mapas-chave

Outra alternativa para acesso a dados geográficos na Web é a que apresenta para o usuário um mapa chave, em formato de imagem. O usuário deve indicar com o *mouse* uma região de seu interesse, gerando uma navegação para outro mapa ou imagem mais detalhado, ou clicar em ícones periféricos à imagem para navegar para regiões adjacentes, mantendo a escala de visualização. Eventualmente, podem existir ícones que ativam funções mais sofisticadas, como medição na tela, identificação de elementos ou ativação/desativação de camadas.

Esta abordagem permite um grau um pouco maior de flexibilidade, mas não resolve os problemas principais da alternativa anterior, ou seja, custos de processamento e transmissão, além de não resolver completamente o problema de navegação. O grau de interatividade com o usuário é baixo, já que, nesse modelo, não há interação direta entre o usuário e um banco de dados propriamente dito, apenas com o que seria a materialização, em formato de imagem, de um instantâneo (*snapshot*) de parte do conteúdo do banco. No exemplo apresentado na Figura 10.6, a barra de atividade que aparece no centro da tela indica que o servidor está compilando o mapa em formato de imagem para envio ao cliente, na próxima etapa da interação.

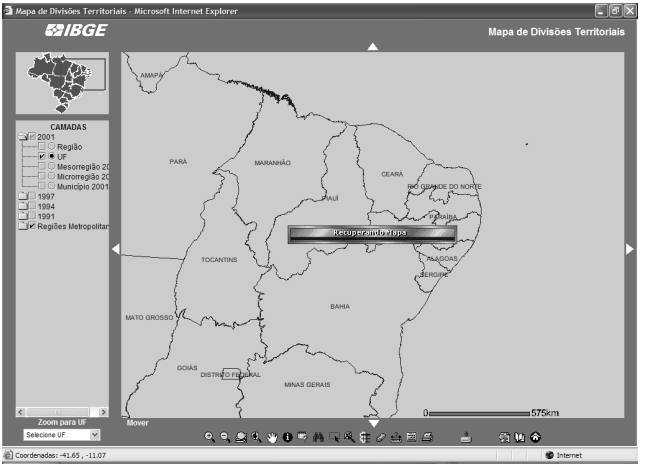
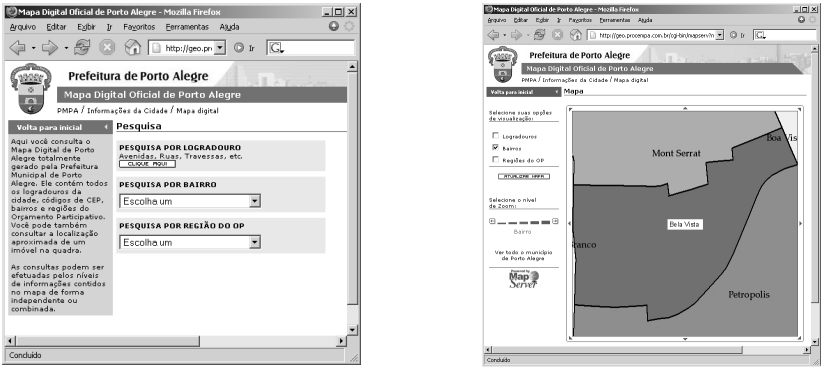


Figura 10.6 – Navegação baseada em mapas-chave. Fonte: Web site do IBGE, servidor de mapas (www.ibge.gov.br).

Outro exemplo é apresentado na Figura 10.7, este o mapa digital do município de Porto Alegre, desenvolvido usando o software gratuito e de fonte aberto MapServer (mapserver.gis.umn.edu). A Figura 10.7a apresenta uma tela inicial, em que o usuário informa parâmetros para uma consulta. O resultado aparece na Figura 10.7b.



(a) (b)

Figura 10.7 – Mapa digital oficial de Porto Alegre, desenvolvido com MapServer.

10.3.4 Transmissão de dados vetoriais

Uma alternativa mais interessante do que a transmissão de imagens é a transmissão de objetos geográficos com representação vetorial. Desta maneira, o usuário fica livre para decidir a região de interesse, bem como para ativar ou desativar as camadas que deseja.

Os objetos vetoriais transmitidos são mantidos na memória da máquina cliente, para que possam ser reaproveitados no caso de operações de *zoom* ou *pan*, ganhando tempo para aumentar a interatividade. O usuário pode também interagir diretamente com os objetos do mapa, consultando atributos e acessando funções de atualização de dados.

Outra possibilidade interessante é a aplicação ao mapa vetorial do conceito de *hipermapa*, simulando nos símbolos e objetos vetoriais disponíveis a operação dos *links* de hipertexto comuns nas páginas da Web. Por exemplo, bastaria clicar sobre o símbolo de um hospital num

mapa urbano para consultar seus atributos associados, ou para navegar diretamente para a página do hospital na Internet.

A transmissão de dados geográficos em formato vetorial pela Internet tem um obstáculo: nenhum dos *browsers* atuais está preparado (nativamente) para receber e apresentar informações neste formato. Para que isso seja possível, existem duas alternativas. A primeira, adotada por alguns desenvolvedores de SIG, consiste em criar um *plug-in*, ou seja, um programa que funciona no computador do usuário, conectado ao *browser*.

Este *plug-in* reconhece os dados vetoriais à medida em que chegam, geralmente agrupados em um arquivo com formato padronizado, ou usando um formato específico de codificação de objetos, e os exibe na tela. Esta alternativa tem a desvantagem de exigir que o usuário faça o *download* dos *plug-ins* a partir do *site* do desenvolvedor e os instale. Como os *plug-ins* são específicos para os principais *browsers* do mercado, que estão em constante evolução, é preciso atualizá-los periodicamente. Em uma variação mais drástica desta alternativa, existem *browsers* completos, desenvolvidos pelos fabricantes de software para SIG, construídos em torno da transmissão de dados vetoriais. A Figura 10.8 apresenta um exemplo.

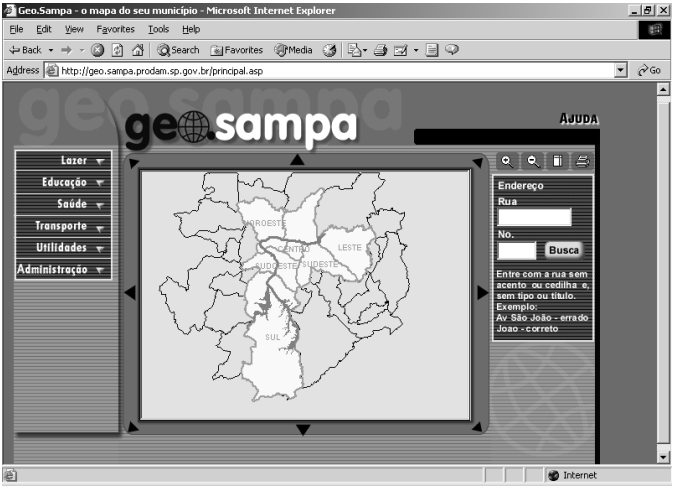


Figura 10.8 – Visualização composta utilizando plug-in ActiveX de um software comercial. Fonte: http://geo.sampa.prodam.com.br.

Outra alternativa consiste em criar uma aplicação (*applet*) na linguagem Java (Fonseca e Davis, 1999), que é transmitida no momento do acesso e executada na máquina do usuário, dispensando procedimentos complicados de instalação ou mesmo a ocupação de área em disco. A aplicação desaparece da máquina do usuário no momento em que é desativada. Assim, novas versões não precisam ser distribuídas, pois estarão disponíveis instantaneamente a partir do momento de sua instalação no servidor. Os dados são recebidos e tratados objeto por objeto, facilitando a implementação de caches locais. Cada objeto precisa ser transmitido uma única vez, sendo que operações posteriores de *zoom* ou *pan* podem apenas utilizar os dados já presentes na cache.

Este tipo de arquitetura já foi inclusive proposta como modelo de interoperabilidade (Fonseca e Davis, 1997) (Fonseca e Davis, 1999). Atualmente, é implementada por diversos produtos, dentre os quais o ALOVmap, um publicador de mapas gratuito que pode funcionar tanto isolado em um computador, quanto através de um servidor Web (Figura 10.9).

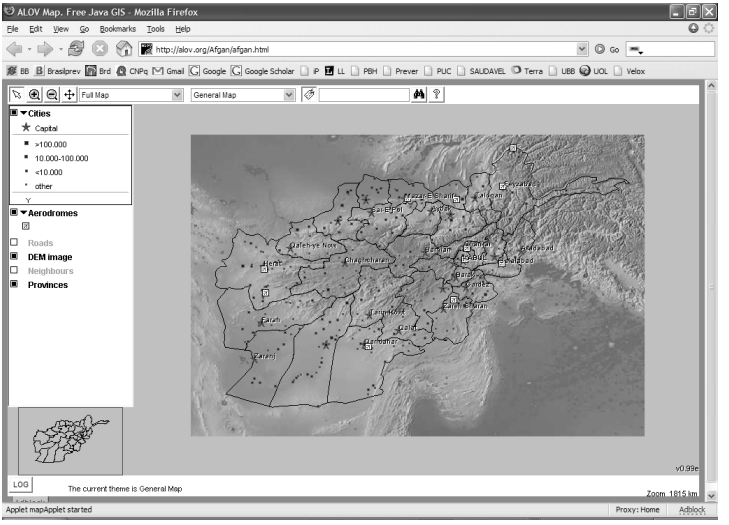


Figura 10.9 – Mapa do Afeganistão apresentado pelo software ALOVmap.
Fonte: http://alov.org/Afgan/afgan.html.

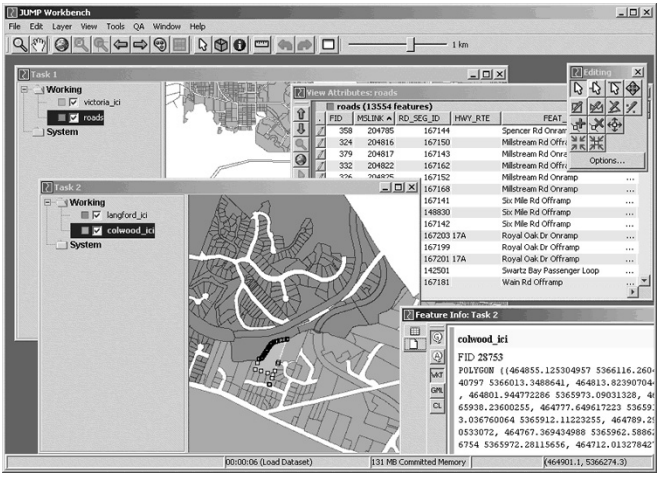


Figura 10.10 – JUMP: *Java Unified Mapping Platform*, um aplicativo para visualização e manipulação de dados geográficos na Internet. Fonte: www.jump-project.org.

Outro exemplo é o projeto Java Unified Mapping Platform (JUMP), um produto de fonte aberto para visualização e manipulação de dados geográficos através da Internet, e que conta com suporte a alguns dos principais padrões do OGC (ver Capítulo 11), como a linguagem GML e o modelo de objetos (Figura 10.10).

Em 2004, o W3C (organismo de padronização da WWW) aprovou a especificação final da linguagem *Simple Vector Graphics* (SVG) (World-Wide Web Consortium (W3C, 2000), através da qual é possível apresentar gráficos vetoriais usando um *browser* comum. Atualmente, esse tipo de recurso exige a instalação de um *plug-in*, pois o padrão é muito recente e ainda não foi incorporado às versões atualmente em uso dos *browsers*.

A linguagem SVG é baseada em XML, assim como a linguagem *Geography Markup Language* (GML), esta padronizada pelo OGC (OpenGIS Consortium, 2004), conforme discutido no Capítulo 11. A combinação entre GML e SVG cria ótimas oportunidades para disseminação de dados geográficos na Internet. É possível, por exemplo, transformar um conjunto de dados GML em um arquivo SVG para

visualização na Web, determinando os parâmetros dessa transformação em um arquivo XSLT (e*Xtensible Stylesheet Language Transformations*)(W3C, 1999).

Um exemplo desse tipo de aplicação foi apresentado em (Mathiak, Kupfer et al. 2004). É possível codificar apresentações bastante complexas usando SVG, contando inclusive com recursos de animação, som e um refinado acabamento gráfico, o que permite antever uma gama de tipos diferentes de apresentação para dados geográficos, com recursos que irão muito além dos tradicionais mapas estáticos da cartografia.

A Figura 10.11 apresenta um exemplo de uso de SVG para apresentação de dados.

Por fim, a Figura 10.12 sumariza as alternativas de acesso a bancos de dados geográficos através da Internet.

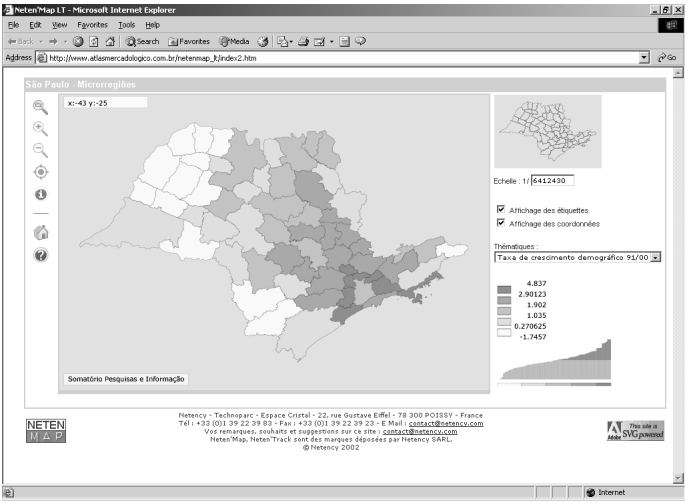


Figura 10.11 – Visualização usando SVG. Fonte: www.atlasmercadologico.com.br.

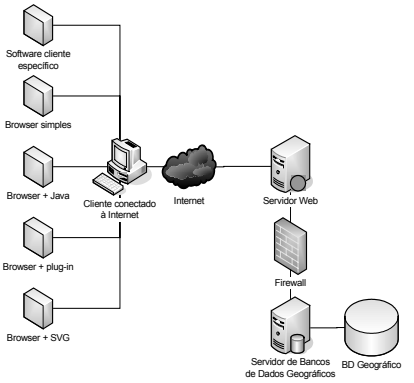


Figura 10.12 – Disseminação de dados geográficos através da Internet – resumo das alternativas.

10.4 Biblioteca digital de informações geográficas

Os SIG estão evoluindo para além de sua comunidade de usuários tradicionais e se tornando parte integrante da infra-estrutura de sistemas de informação de muitas organizações. Uma conseqüência positiva desse fato é o aumento significativo no número e no volume das fontes de dados espaciais disponíveis para acesso através de redes de computadores.

Essa evolução representa um novo paradigma na forma de utilização da informação geográfica, baseado no conceito de *biblioteca digital de informações geográficas* (BDG), ou *centros de dados geográficos*, que são bibliotecas digitais especializadas em dados geo-referenciados, fornecendo uma infra-estrutura para a criação, estruturação, armazenamento, organização, processamento, recuperação e distribuição de dados geo-referenciados (Câmara et al., 1996) (Oliveira et al., 1999).

10.4.1 Alexandria Digital Library

O projeto da *Alexandria Digital Library* (ADL) (UCSB, 2005) já foi discutido no Capítulo 9 como um exemplo de um catálogo de metadados combinado com um dicionário geográfico.

A ADL é, principalmente, um dos mais importantes projetos de criação de uma biblioteca digital de informações geográficas. A ADL fornece acesso público a um acervo de mais de 15.000 itens digitais e não digitais

(mapas, imagens e dados espaciais) do *Map and Imagery Laboratory* (MIL) da Universidade da Califórnia e também a outros acervos.

Recordando a discussão do Capítulo 9, sua arquitetura segue o modelo de três camadas: *servidores* gerenciam as coleções de dados espaciais, um *middleware* implementa os serviços de acesso às coleções via protocolo HTTP e *clientes* são utilizados pelos usuários da biblioteca para a consulta e navegação pelas coleções (Frew, et al. 2000).

Na interface de consulta Web da ADL (Figura 10.13), o usuário pode utilizar um mapa para selecionar a área de interesse da sua pesquisa e também especificar, para a busca ao catálogo, um conjunto de palavras-chave, o período, o tipo do objeto (se mapa, dado, imagem, etc.), seu formato, a fonte original, e o método de *ranking* utilizado na resposta. Os dados sobre os itens localizados são então exibidos, incluindo um *link* para o dado espacial, se esse estiver disponível.

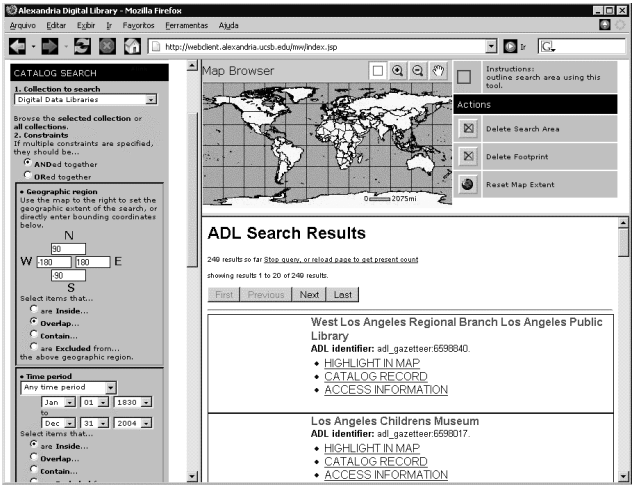


Figura 10.13 – Biblioteca Geográfica Digital Alexandria.

10.4.2 Maine Library of Geographic Information

Nesta BGD, o estado norte-americano do Maine torna disponíveis, para acesso público, dados geográficos em formato digital sobre seu território (Maine Digital Geographic Library, 2005). Destaca-se a qualidade dos metadados da biblioteca, de acordo com uma divisão em sete itens: identificação, qualidade dos dados, organização dos dados, sistema de referência utilizado, informações dos atributos dos dados, informações de distribuição e referência dos metadados.

A busca (Figura 10.14) pode ser feita com uso de palavras-chave ou pela seleção de um dos cinco grupos temáticos disponíveis: cidades, condados, quadrículas, unidades hídricas ou todo o estado. Cada um desses grupos possui uma relação de dados digitais disponíveis. A seleção de um grupo mostra a lista dos dados acompanhada de seus respectivos metadados e endereços para *download* dos dados, quase sempre em formato *shape*.

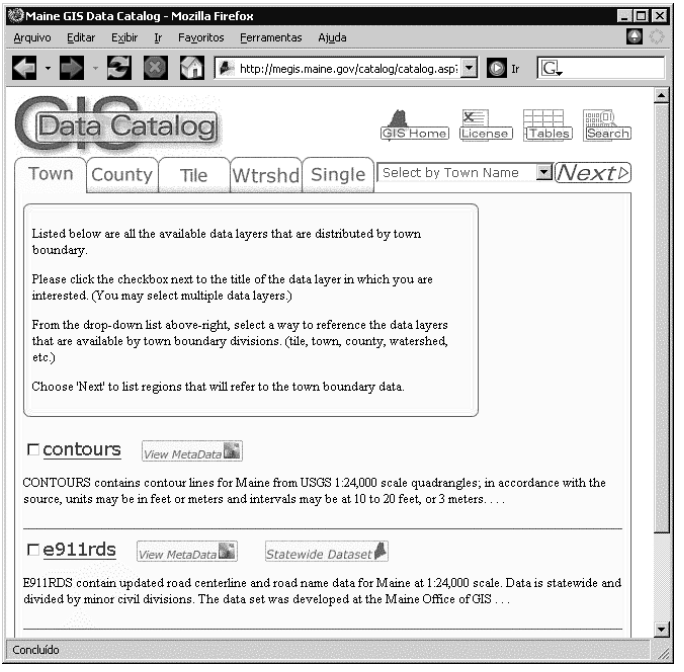


Figura 10.14 – Biblioteca Geográfica Digital do Maine.

10.4.3 GeoConnections Discovery Portal

Mantida por um entidade canadense, formada por membros do governo e da iniciativa privada, esta BDG tem por objetivo de longo prazo o desenvolvimento de uma infra-estrutura para a divulgação de dados espaciais, especialmente os que se referem ao território do Canadá (Natural Resources Canada, 2005). Estão disponíveis tanto dados gratuitos quanto com valor associado.

O sistema de busca é capaz de localizar dados pesquisando por área, assunto, palavra-chave, período de tempo ou por qualquer combinação desses itens. O resultado de uma consulta pode ser observado na Figura 10.15. Os metadados apresentam um bom nível de detalhamento, e trazem informações sobre a identificação dos dados, sua distribuição e as referências dos metadados.

Figura 10.15 – Geoconnections Discovery Portal.

10.5 Infra-estruturas de dados espaciais

O acesso à informação tem sido uma das grandes dificuldades de todo pesquisador, usuário, desenvolvedor ou especialista envolvido em temas ligados à representação computacional do espaço. Inicialmente, nos primórdios do geoprocessamento, o problema se concentrava na construção dos bancos de dados geográficos: fontes de dados, processos de conversão, técnicas e tecnologias para levantamento de dados em campo (Montgomery,e Schuch, 1993).

Com o aumento gradual do volume de informação digital disponível em meio digital, encontrar padrões para facilitar o intercâmbio de dados passou a ser o problema (Lima, 2002) (Lima et al., 2002) (United States Geological Survey, 2005) (Davis Jr., 2002), conforme discutido no Capítulo 9.

Rapidamente, no entanto, percebeu-se que o intercâmbio puro e simples dos dados não seria suficiente, se o receptor não tivesse como avaliar se os dados atendem ou não às suas necessidades. Isso levou ao estabelecimento de padrões e ao início de esforços de desenvolvimento de metadados espaciais (Soares, 1999). Mas, mesmo com os metadados, permaneciam ainda eventuais dúvidas sobre aspectos semânticos ligados aos dados, problemas esses resultantes de visões variadas do mundo por pessoas de diferentes origens e formações. Esse problema, juntamente com o anterior, motivou o desenvolvimento de estudos de interoperabilidade semântica (Fonseca et al. 2000) (Vckovski, 1998), já aludido no Capítulo 9. Atualmente, mesmo sem haver conclusões definitivas sobre os problemas de semântica, existe um movimento para criar *infra-estruturas de dados espaciais* (IDE) (Bernard, 2005) (Smits, 2002).

Diversas iniciativas têm buscado compreender melhor os processos segundo os quais elementos da infra-estrutura espacial são desenvolvidos e utilizados. Uma dessas iniciativas é o projeto INSPIRE (*Infrastructure for Spatial Information in Europe*) (Smits, 2002), que pretende tornar a informação geográfica digital mais acessível e interoperável para uma ampla gama de aplicações, usando uma arquitetura baseada em serviços. Outra delas é a *Global Spatial Data Infrastructure Association* (GSDI), uma organização que pretende promover o desenvolvimento de uma infra-estrutura de dados espaciais em escala planetária, como suporte à

condução de ações de desenvolvimento sustentável e impacto social, econômico e ambiental (GSDI Association, 2005).

A idéia principal das infra-estruturas de dados espaciais é oferecer serviços de acesso à informação geográfica, com base em grandes catálogos de acervos de informação, tornando indiferentes, aos olhos do cliente da IDE, o local, meio e estrutura física de armazenamento. Como o acesso aos dados é realizado apenas através de serviços, é possível *encapsular* a estrutura física dos dados, seguindo um dos preceitos da orientação por objetos. O usuário também não precisaria conhecer o local onde os dados estão armazenados, pois cada provedor de dados se encarrega de registrar, junto a um serviço de catalogação, que dados possui, onde estão, como estão organizados, e onde estão os metadados. Basta, então, que o usuário consulte um serviço para determinar se os dados que procura estão disponíveis, consulte outro para avaliar detalhes sobre a fonte e o processo de captura usando outro serviço e, caso esteja satisfeito com as características dos dados, acione um terceiro serviço para recuperá-los.

O foco no conceito de IDE é uma conseqüência natural da evolução da Web e de sua arquitetura, diante das diretrizes que vêm sendo traçadas por seu órgão normatizador, o W3C.

Em sua concepção inicial, o objetivo da Web era fornecer conteúdo exclusivo para uso e interpretação humanos. Isso significa que, em sua origem, o foco de desenvolvimento da Web restringiu-se a questões relacionadas à apresentação. O crescimento explosivo da Web, entretanto, fez surgir uma enorme demanda por aplicações que fossem capazes de operar e interagir nesse ambiente. A apresentação deixou de ser a preocupação central para dividir seu lugar com o conteúdo.

As primeiras abordagens para a interoperação entre aplicações na Web, porém, possuíam um caráter *ad hoc* e baseavam-se na exploração da própria infra-estrutura básica da Internet. Muitas vezes, essa interoperação era realizada através de APIs que incorporavam módulos encarregados de extrair o conteúdo de páginas Web (McIlraith et al., 2001).

Essa mudança de foco da apresentação para o conteúdo foi cristalizada no conceito da *Web Semântica*, "uma extensão da Web na qual a informação possui um significado bem definido, possibilitando que

computadores e pessoas trabalhem em cooperação" (Berners-Lee, 2001). Alguns passos importantes em direção à Web Semântica já foram dados, como o uso crescente do padrão XML para o intercâmbio de dados e a especificação de uma grande quantidade de serviços Web. Percebe-se, assim, que a área de geoinformação também tem se movimentado em direção à Web Semântica, tanto por meio dos padrões da OGC, quanto por iniciativa da comunidade de pesquisa em geoinformação (Egenhofer, 2002).

O conceito de serviço Web surgiu para prover uma arquitetura sistemática e mais ampla para a interação entre aplicações, fundamentada sobre os protocolos Web já existentes e o padrão XML (Curbera et al., 2002).

O funcionamento proposto para as IDE aproxima-se bastante da arquitetura de serviços do OpenGIS Consortium, abordados no Capítulo 11. No modelo OGC (Figura 10.16), órgãos públicos, empresas e instituições geradoras de informação espacial proveriam acesso aos seus dados através de serviços Web de diversas naturezas, conforme o tipo de dado e as peculiaridades de seu uso. Cada serviço é registrado em um servidor central, através do qual os usuários poderão descobrir sobre a existência ou não de determinado dado ou serviço, e obter o caminho de acesso a servidores de metadados, através dos quais poderão verificar se a qualidade e demais características do dado atendem ou não às suas necessidades. Poderão também, eventualmente, escolher entre diversos provedores do mesmo tipo de serviço.

Observe que os serviços podem ser ou não baseados em dados: serviços para transformação de coordenadas, por exemplo, podem prover apenas o processamento de dados do usuário. Outros serviços são totalmente baseados em dados, porém não os fornecem diretamente para o usuário, possibilitando assim que o administrador do serviço tenha a liberdade de estruturá-lo internamente da maneira que achar melhor.

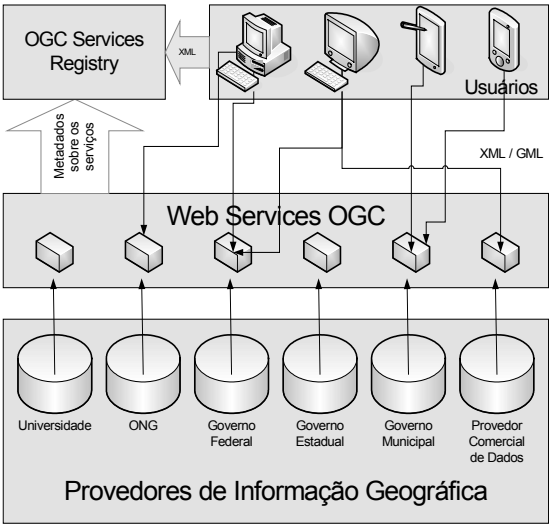


Figura 10.16 – Arquitetura de serviços OGC.

10.6 Leituras Suplementares

A grande variedade de alternativas para disseminação de dados geográficos pela Internet não deixa dúvidas quanto à enorme demanda que existe por informação espacial simples de acessar, de obter e usar. A atual disposição de países desenvolvidos em promover a criação de infra-estruturas de dados espaciais reflete essa demanda: trata-se do reconhecimento de que a informação é um bem da sociedade que, estando disponível no tempo certo, com qualidade adequada, de maneira livre e a baixo custo, pode fomentar uma ampla gama de iniciativas, públicas, privadas, individuais ou do terceiro setor.

Sugerimos aos leitores um aprofundamento sobre infra-estruturas de dados espaciais, seguindo projetos como o INSPIRE e outros, além de um maior aprofundamento nas propostas do OGC para arquiteturas de sistemas geográficos apoiadas em serviços Web (OpenGIS Consortium, 2003).

Resta ainda comentar sobre o outro lado da disseminação cada vez mais ampla de dados na Web: a necessidade de *localizar* esses dados. Existem

hoje muitas iniciativas no sentido de ampliar o escopo das busca tradicionais, tais como Google, Yahoo ou Altavista. Os dois primeiros têm em operação serviços de busca apoiados em nomes de locais (local.google.com e local.yahoo.com), que combinam buscas por palavra-chave com um sistema de páginas amarelas.

Algumas iniciativas de extrair o contexto geográfico presente no conteúdo de uma página Web estão sendo conduzidas (Laender et al., 2005), visando, entre outras coisas, produzir "índices espaciais" para a localização de páginas Web (Jones et al., 2002) e constituindo uma área denominada *recuperação da informação espacial* (*spatial information retrieval*). Por enquanto, sabe-se principalmente que o interesse em se contar com uma "Web local" é muito grande, e que isso só será viabilizado quando conseguirmos entender melhor a maneira segundo a qual as pessoas relacionam-se com a geografia em seu dia-a-dia (Hiramatsu e Ishida, 2001).

Referências

BERNARD, L.; CRAGLIA, M. SDI: From Spatial Data Infrastructure to Service Driven Infrastructure. In: Research Workshop on Cross-Learning between Spatial Data Infrastructures (SDI) and Information Infrastructures (II),2005,Enschede, Holanda.

BERNERS-LEE, T.; CAILLIAU, R.; LUOTONEN, A.; NIELSEN, H. F.; SECRET, A. The World-Wide Web. Communications of the ACM, v. 37, n.8, p. 76-82, 1994.

BERNERS-LEE, T.; HENDLER, J.; LASSILA, O. The Semantic Web. **Scientific American**, v. 184, n.5, p. 34-43, 2001.

CÂMARA, G.; CASANOVA, M.; HEMERLY, A.; MAGALHÃES, G.; BAUZER-MEDEIROS, C., 1996, Anatomia de Sistemas de Informação Geográfica, Curitiba, Sagres.

CURBERA, F.; DUFTLER, M.; KHALAF, R.; NAGY, W.; MUKHI, N.; WEERAWARANA, S. Unraveling the Web Services Web: an introduction to SOAP, WSDL and UDDI. **IEEE Internet Computing**, v. 6, n.2, p. 86-93, 2002.

DAVIS JR., C. A., 2002, Intercâmbio de Informação Geográfica: a experiência de padronização e cooperação em Minas Gerais. In: PEREIRA, G. C., ROCHA, M. C. F., ed., **Dados Geográficos: aspectos e perspectivas**: Salvador (BA), Rede Baiana de Tecnologias da Informação Espacial (REBATE), p. 43-54.

EGENHOFER, M. Toward the semantic geospatial web. In: 10th ACM international symposium on Advances in geographic information systems table of contents, 2002, McLean, Virginia, USA.

ELSMARI, R.; NAVATHE, S. **Fundamentals of Database Systems.** Pearson Education, 2004.

FONSECA, F. T.; DAVIS JR., C. A. Using the Internet to access geographic information: an OpenGIS interface prototype. In: International Conference on Interoperating Geographic Information Systems (Interop 97), 1997, Santa Barbara (CA). NCGIA and OpenGIS Consortium (OGC), p. 283-293.

FONSECA, F.; DAVIS, C., 1999, Using the Internet to Access Geographic Information: An OpenGis Prototype. In: GOODCHILD, M.; EGENHOFER, M.; FEGEAS, R.; KOTTMAN, C., eds., **Interoperating**

Geographic Information Systems: Norwell, MA, Kluwer Academic Publishers, p. 313-324.

FONSECA, F.; EGENHOFER, M.; BORGES, K. Ontologias e Interoperabilidade Semântica entre SIGs. In: Workshop Brasileiro em Geoinformática (GeoInfo2000), 2., São Paulo, 2000. **Anais**. São José dos Campos: INPE, 2000. p. 45 - 52.

FREW, J.; FREESTON, M.; FREITAS, N.; HILL, L.; JANÉE, G.; LOVETTE, K.; NIDEFFER, R.; SMITH, T.; ZHENG, Q. The Alexandria Digital Library Architecture. **International Journal on Digital Libraries**, v. 2, n.4, p. 259-268, 2000.

GSDI ASSOCIATION, 2005, Global Spatial Data Infrastructure Web Site.

HIRAMATSU, K.; ISHIDA, T. An Augmented Web Space for Digital Cities. In: IEEE Symposium on Applications and the Internet (SAINT 2001),2001,San Diego (CA). p. 105-112.

JONES, C. B.; PURVES, R.; RUAS, A.; SANDERSON, M.; SESTER, M.; VAN KREVELD, M.; WEIBEL, R. Spatial information retrieval and geographic ontologies: an overview of the SPIRIT project. In: 25th Annual ACM SIGIR Conference on Research and Development on Information Retrieval,2002, Tampere, Finland.

LAENDER, A. H. F.; BORGES, K. A. V.; CARVALHO, J. C. P.; MEDEIROS, C. M. B.; SILVA, A. S.; DAVIS JR., C. A., 2005, Integrating Web Data and Geographic Knowledge into Spatial Databases. In: MANOLOPOULOS, Y.; PAPADOPOULOS, A.; VASSILAKOPOULOS, M., eds., **Spatial Databases: technologies, techniques and trends**: Hershey (PA), Idea Group Publishing.

LIMA, P. GeoBR: Intercâmbio Sintático e Semântico de Dados Espaciais.São José dos Campos: INPE, 2002.Dissertação de Mestrado, 2002.

LIMA, P.; CÂMARA, G.; QUEIROZ, G. R. GeoBR: Intercâmbio Sintático e Semântico de Dados Espaciais. In: **IV Simpósio Brasileiro de GeoInformática (GeoInfo 2002)**,2002,Caxambu (MG). p. 139-146.

MAINE DIGITAL GEOGRAPHIC LIBRARY, 2005.

MATHIAK, B.; KUPFER, A.; NEUMANN, K. Using XML languages for modeling and Web-visualization of geographical legacy data. In: GeoInfo 2004,Campos do Jordão (SP). Sociedade Brasileira de Computação (SBC).

MCILRAITH, S. A.; SON, T. C.; ZENG, H. Semantic Web Services. **IEEE Intelligent Systems**, v. 16, n.2, p. 46-53, 2001.

MONTGOMERY, G. E.; SCHUCH, H. C. GIS Data Conversion Handbook. John Wiley & Sons,1993.

NATURAL RESOURCES CANADA, 2005, GeoConnections Discovery Portal.

OLIVEIRA, J. L.; GONÇALVES, M. A.; MEDEIROS, C. M. B. A framework for designing and implementing the user interface of a geographic digital library. **International Journal on Digital Libraries**, p. 190-206, 1999.

OPENGIS CONSORTIUM, 2003, OpenGIS Reference Model.

OPENGIS CONSORTIUM, 2004, Geography Markup Language (GML) 3.1.0.

SMITS, P., 2002, INSPIRE Architecture and Standards Position Paper, Architecture and Standards Working Group.

SOARES, V. G.; SALGADO, A. C. Consultas visuais em sistemas de informações geográficas baseadas em padrões de metadados espaciais. In: I GeoInfo 1999,Campinas (SP).

SUN MICROSYSTEMS, 1994, The Java language: a white paper.

THOMAZ, K. P. **A preservação de documentos eletrônicos de caráter arquivístico: novos desafios, velhos problemas.**Belo Horizonte: UFMG, 2004.Tese de Doutorado, Escola de Ciência da Informação, 2004.

UCSB, 2005, The Alexandria Digital Library.

UNITED STATES GEOLOGICAL SURVEY, 2005, USGS Spatial Data Transfer Standard Information Site.

VCKOVSKI, A. **Interoperable and Distributed Processing in GIS.** CRC Press,1998.

WORLD-WIDE WEB CONSORTIUM (W3C), 1999, XSL Transformations (XSLT) Version 1.0.

WORLD-WIDE WEB CONSORTIUM (W3C), 2000, Scalable Vector Graphics (SVG) 1.0 Specification.

11 O Open Geospatial Consortium

Clodoveu A. Davis Jr.
Karla A. V. Borges
Ligiane Alves de Souza
Marco Antonio Casanova
Paulo de Oliveira Lima Júnior

11.1 Introdução

Este capítulo resume o modelo conceitual, o formato de intercâmbio de dados e os serviços propostos pelo Open Geospatial Consortium (OGC).

O Open Geospatial Consortium (OGC, 2005a) é um consórcio com mais de 250 companhias, agências governamentais e universidades, criado para promover o desenvolvimento de tecnologias que facilitem a interoperabilidade entre sistemas envolvendo informação espacial e localização (Gardels, 1996) (Percivall, 2003). Os produtos do trabalho do OGC são apresentados sob forma de especificações de interfaces e padrões de intercâmbio de dados.

11.2 Modelo conceitual

O modelo conceitual do OGC é baseado em uma classe abstrata denominada *feature*. Uma *feature* é considerada pelo OpenGIS uma abstração de um fenômeno do mundo real; é uma feição geográfica se é associada com uma localidade relativa à Terra.

A classe abstrata FEATURE tem duas especializações FEATURE WITH GEOMETRY, que almeja capturar o conceito de geo-objetos, e COVERAGES, que pretende capturar o conceito de geo-campos.

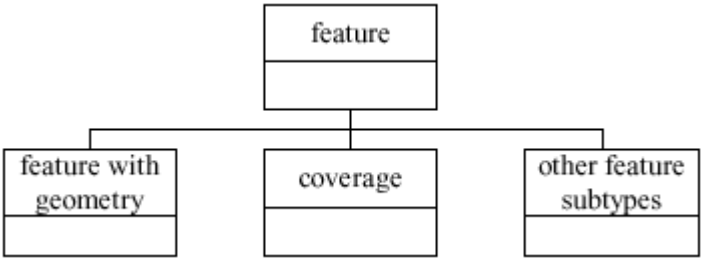


Figura 11.1 – Subtipos de feature, adaptado de (OGC, 1999).

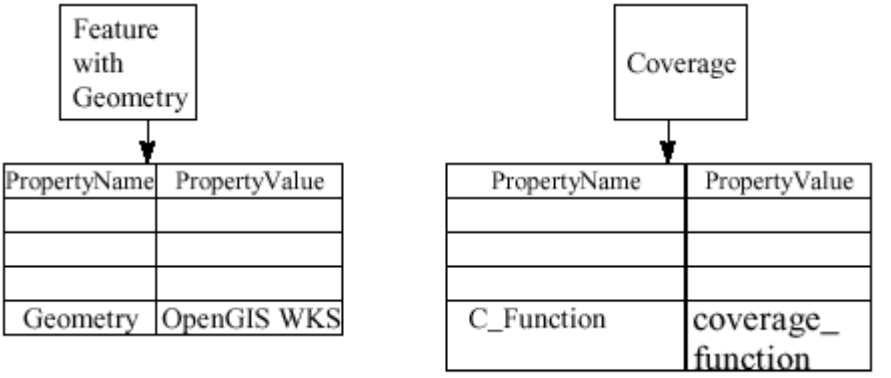


Figura 11.2 – Relação entre Feature e Coverage, adaptado de (OGC, 1999).

O padrão OpenGIS relaciona diretamente cada tipo de *coverage* com uma geometria particular, num relacionamento de especialização. Isto faz com que o conteúdo semântico de uma *coverage*, ou seja, os tipos de dados geográficos representados, sejam confundidos com seu conteúdo sintático, a estrutura de dados utilizada. Esta relação é incoerente com a definição proposta pelo padrão OpenGIS para *coverage*, que fala em função, domínio e intervalo.

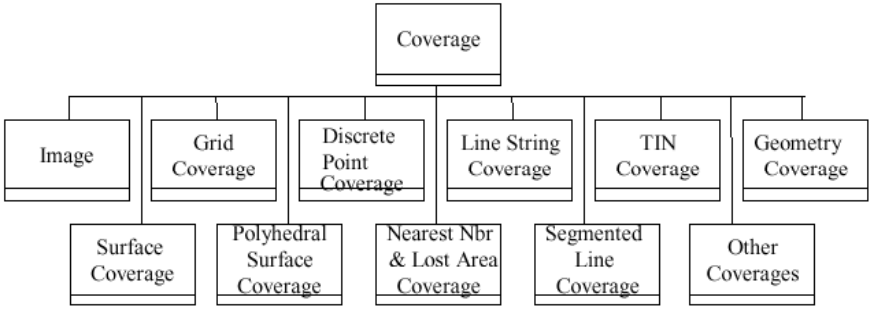


Figura 11.3 – Subtipos de Coverage, adaptado de (OGC, 1999).

11.3 Geography Markup Language

Seguindo a tendência do uso de padrões para intercâmbio de dados, o OpenGIS usa o padrão XML (eXtensible Markup Language) para definir uma forma de codificar dados geográficos. Para isso especificou a linguagem GML (Geography Markup Language) (Cox, 2003). A GML (Geography Markup Language) foi especificada para o transporte e armazenamento de informação geográfica, incluindo propriedades espaciais e não espaciais das feições geográficas (OGC, 1999).

O objetivo da GML é oferecer um conjunto de regras com as quais um usuário pode definir sua própria linguagem para descrever seus dados. Para tanto, a GML é baseada em esquemas XML (XML Schema). O esquema XML define os elementos (*tags*) usados em um documento que descreve os dados. Sua versão 3.0 inclui esquemas que contêm os modelos de geometria, feições (features) e superfícies. Os esquemas estão publicados nas especificações do OGC (Cox, 2003) os principais são os seguintes:

- **BasicTypes**: que engloba uma série de componentes simples e genéricos para representação arbitraria de atributos, nulos ou não.
- **Topology**: o qual especifica as definições do esquema geométrico dos dados, bem como sua descrição.

- **CoordinateReference Systems**: para sistemas de referência de coordenadas.
- **Temporal Information and Dynamic Feature**: Este esquema estende aos elementos características temporais dos dados geográficos e suas funções dinamicamente definidas.
- **Definitions and Dictionaries**: definições das condições de uso dentro de documentos com certas propriedades ou informações de referentes à propriedade padrão.
- **Metadata**: Este esquema é utilizado para definir as propriedades dos pacotes de dados que podem ser utilizados através de outros dados já existentes.

De posse destes esquemas, um usuário pode definir o seu próprio esquema para sua aplicação. Mas há algumas exigências a seguir para obter conformidade:

- Desenvolvedores de esquemas de aplicação devem assegurar que seus tipos são subtipos dos correspondentes tipos da GML: gml:AbstractFeatureType ou gml:AbstractFeatureCollectionType para feições e gml:AbstractGeometryType ou gml:GeometryCollectionType para a geometria.
- Um esquema de aplicação não pode mudar o nome, definição ou tipo de dado dos elementos obrigatórios da GML;
- Definições de tipos abstratos podem ser livremente estendidas ou restritas;
- Esquema de aplicação deve estar disponível a qualquer um que receba o dado estruturado por aquele esquema;
- Os esquemas relevantes devem especificar um "*namespace*" que não deve ser http://www.opengis.net/gml.

Desta forma um desenvolvedor de esquemas pode criar seus próprios tipos e *tags* e uma aplicação GML poderá fazer uso dos dados. Por exemplo, consideramos dois arquivos, um para o esquema (exemplo.xsd) e outro com os dados (exemplo.xml):

```
...
1.<complexType name="hidrografia">
2.     <complexContent>
3.          <extension base="gml:AbstractFeatureType">
4.                  <sequence>
5.                          <element ref="gml:centerLineOf"/>
6.                  </sequence>
7.          </extension>
8.     </complexContent>
9.</complexType>
...
```

Figura 11.4 – Trecho de um esquema de aplicação.

A Figura 11.4 é um fragmento de um arquivo exemplo.xsd que define um esquema de aplicação, mostrando a criação de um tipo, no caso hidrografia. Seguindo as regras, a linha 3 faz com que hidrografia seja subtipo de `gml:AbstractFeatureType`. Este tipo pode ser usado na criação de uma *tag*, por exemplo:

```
...
<element name="rio" type="ex:hidrografia" substitutionGroup="gml:_Feature"/>
...
```

Figura 11.5 – Criação de uma *tag*.

O fragmento mostrado na Figura 11.5 é parte do mesmo esquema e define a criação de uma *tag* `<rio>` do tipo hidrografia que pode ser usada para descrever um determinado rio no arquivo exemplo.xml. Também é definido que o elemento tem o atributo `substitutionGroup` igual a `gml:_Feature`, o que garante a uma aplicação a leitura dos dados. O "ex" em `ex:hidrografia` indica que o nome hidrografia é do domínio `ex`, declarado no início do arquivo:

```
1. <?xml version="1.0" encoding="UTF-8"?>
2. <!-- File: city.xsd -->
3. <schema targetNamespace="http://www.opengis.net/examples"
4.      xmlns:ex="http://www.opengis.net/examples"
5.      xmlns:xlink="http://www.w3.org/1999/xlink"
6.      xmlns:gml="http://www.opengis.net/gml"
7.      xmlns="http://www.w3.org/2000/10/XMLSchema"
8.      elementFormDefault="qualified" version="2.03">
...
```

Figura 11.6 – Exemplo de *namespace*.

No início do arquivo foi definido o "*namespace*" `ex`, como mostra a linha 4 da Figura 11.6. Um fragmento do arquivo contendo os dados é ilustrado pela Figura 11.7:

```
...
1. <rio>
2.    <gml:description>O rio principal</gml:description>
3.    <gml:name>Rio Grande</gml:name>
4.    <gml:centerLineOf>
5.       <gml:LineString  srsName="http://www.opengis.net/gml/srs/epsg.xml#432>
6.          <gml:coord><gml:X>0</gml:X><gml:Y>50</gml:Y></gml:coord>
7.          <gml:coord><gml:X>7</gml:X><gml:Y>60</gml:Y></gml:coord>
8.          <gml:coord><gml:X>1</gml:X><gml:Y>50</gml:Y></gml:coord>
9.       </gml:LineString>
10.   </gml:centerLineOf>
11. </rio>
...
```

Figura 11.7 – Fragmento de um arquivo de dados em XML.

As *tags* **`<gml:description>`** (linha 2) e `<gml:name>` (linha 3) da Figura 11.7 não foram criadas no esquema de aplicação, mas sim herdadas do tipo AbstractFeatureType, já que hidrografia é deste tipo e `<rio>` foi definido como hidrografia. Para a transferência de dados é necessária a transferência do arquivo com o esquema também, só assim uma aplicação que procura por `<_Feature>` poderá saber que `<rio>` é `<_Feature>`.

Os esquemas da GML sozinhos não são adequados para criar uma instância de documento. Estes devem ser estendidos pela criação de esquemas de aplicação para domínios específicos, seguindo as regras

descritas na especificação. Isto exige um investimento na elaboração de esquemas.

A GML possui pontos, linhas, polígonos e coleções geométricas (MultiPoint, MultiPolygon) definidos por coordenadas cartesianas uni, bi ou tridimensionais associados a eventuais sistemas de referência espacial. Mas as localizações espaciais são definidas apenas por coordenadas cartesianas, coordenadas projetivas não estão previstas.

Uma vantagem no uso de XML é a flexibilidade oferecida para criar "tags" que expressam o significado do dado descrito, obtendo-se um documento rico semanticamente. Mas, considere a seguinte situação: dois usuários de domínios diferentes representam uma determinada entidade, pela GML, como <rio> e <curso_de_agua>. Em uma troca de dados entre os usuários, os esquemas também devem ser compartilhados, pois só assim uma aplicação poderá saber que <rio> ou <curso_de_agua> são da classe <_Feature> definida pelo esquema Feature.xsd da GML, e então processá-los adequadamente. Desta forma o problema de acesso aos dados é resolvido. Mas não há como saber que <rio> é <curso_de_agua> e vice-versa. O aspecto semântico não é considerado de forma efetiva a promover a interoperabilidade. Para amenizar este problema, pode-se acrescentar *tags* que descrevem as entidades e suas relações, ou que identifiquem sinônimos.

11.4 Descrição dos serviços

11.4.1 *Framework* arquitetural dos serviços

As especificações do OGC baseiam-se em um *framework* arquitetural, chamado de *OpenGIS Services Framework* (Percivall, 2003), que especifica o escopo, objetivos e comportamento de uma série de componentes. Neste sentido, o *framework* representa uma arquitetura de referência para desenvolvimento de aplicações geográficas, no espírito da ISO 19119.

A definição dos componentes segue o paradigma de *Web services* e, portanto, está sujeita a restrições conceituais e de implementação. As restrições conceituais endereçam funcionalidade e incluem orientação a serviços, auto-descrição dos serviços e operação sem estados persistentes *(stateless operation)*. As restrições de implementação endereçam questões

relativas a interoperabilidade, incluindo a adoção de formatos de intercâmbio em XML, utilização de protocolos comuns à Internet.

Porém, convém salientar que os serviços originalmente especificados pelo OGC não seguem as recomendações do W3C para definição de serviços Web, como SOAP para intercâmbio de dados, WSDL para descrição dos serviços e UDDI para registro dos serviços. Apenas, mais recentemente, a série de propostas de especificação conhecidas coletivamente como *OpenGIS Web Service 2 initiative* (Sonnet, 2004) definiram interfaces que utilizam os padrões do W3C. Porém, tais especificações ainda são tratadas como propostas de mudança. Ainda, recentemente, a OGC iniciou um experimento em larga escala para testar o conceito de Web semântica geospacial (OGC, 2005b).

Brevemente, o *OpenGIS Services Framework* compreende (ver Figura 11.8):

Padrões de Codificação: especificações de formatos de intercâmbio e armazenamento de dados geográficos, incluindo descrições de sistemas de geo-referenciamento, geometria, topologia, e outras características. O *Geography Markup Language (GML)* é um formato de documentos XML para intercâmbio de dados geográficos.

Serviços do cliente: componentes que, do lado do cliente, interagem com os usuários e que, do lado do servidor, interagem com os servidores de aplicação e de dados.

Serviços de registro: componentes que oferecem mecanismos para classificar, registrar, descrever, pesquisar, manter e acessar informação sobre os recursos na rede. Incluem o *Web Registry Service (WRS)*.

Serviços de processamento de workflow: componentes que oferecem mecanismos para composição de serviços que operam sobre dados e metadados geográficos. Incluem o *Sensor Planning Service (SPS)* e o *Web Notification Service (WNS)*.

Serviços de visualização: componentes que oferecem suporte específico para visualização de dados geográficos, resultando em produtos como mapas, visões em perspectiva do terreno, imagens anotadas, visões dinâmicas de dados espaço-temporais. Incluem o *Web Map Service*

(WMS), o *Coverage Portrayal Service (CPS)* e o *Style Management Service (SMS)*.

Serviços de dados: componentes que oferecem os serviços básicos de acesso aos dados geográficos. Incluem o *Web Object Service (WOS),* o *Web Feature Service (WFS),* o *Sensor Collection Service (SCS),* o *Image Archive Service (IAS)* e o *Web Coverage Service (WCS)*.

O resto desta seção resume os serviços definidos pelo OGC. Para maiores detalhes, recomenda-se uma visita ao Website do OGC (OGC, 2005a).

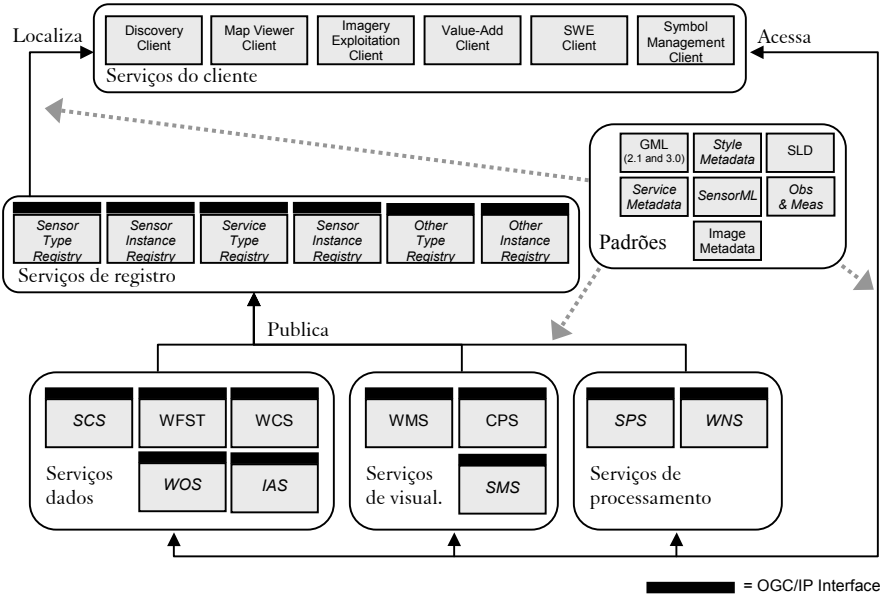


Figura 11.8 – Componentes do OGC Web Service Framework.

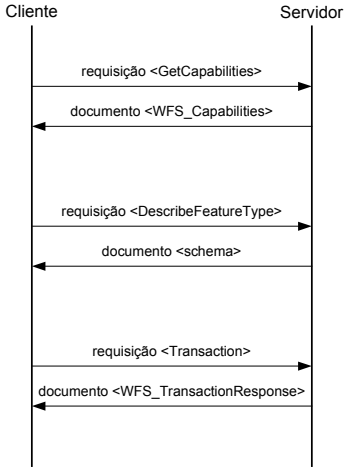


Figura 11.9 – Exemplo de execução do serviço WFS.

11.4.2 Web Feature Service

A especificação OpenGIS *Web Feature Service* (WFS) define um serviço para que clientes possam recuperar objetos (*features*) espaciais em formato GML de servidores WFS. O serviço pode ser implementado pelo servidor em duas versões: básica, onde apenas operações de consulta ficam disponíveis, ou transacional, que implementa o serviço completo, que inclui operações de inserção, exclusão, atualização, consulta de objetos (*features*) geográficos.

As seguintes operações são definidas para o serviço:

- *getCapabilities*: descreve as características do servidor.
- *describeFeatureType*: descreve a estrutura dos tipos de objeto que podem ser servidos.
- *getFeature*: retorna instâncias dos objetos disponíveis na base de dados. O cliente pode selecionar quais objetos deseja por critérios espaciais ou não.
- *transaction*: utilizado para a execução de operações de modificação dos objetos (inserção, exclusão e atualização).
- *lockFeature*: bloqueia uma ou mais instâncias durante uma transação.

A Figura 11.9 ilustra uma seqüência de execução típica do serviço WFS. Como em todos os demais serviços, tanto as requisições quanto as respostas são documentos XML.

11.4.3 Web Coverage Service

Para o acesso a dados que representam fenômenos com variação contínua no espaço, o consórcio OpenGIS criou a especificação OpenGIS *Web Coverage Service* (WCS). O serviço é específico para o tratamento de dados modelados como geo-campos, em complementação ao serviço WFS, que trata de dados modelados como geo-objetos, isto é, que representam entidades espaciais discretas e bem definidas.

É importante mencionar que o serviço não retorna imagens das *coverages* como resposta das requisições, mas sim dados sobre a semântica dos fenômenos representados.

Três operações são implementadas no serviço:

- *getCapabilities*: fornece uma descrição do servidor e informações básicas acerca das coleções de dados disponíveis.
- *describeCoverage*: recupera uma descrição completa das coverages.
- *getCoverage*: recupera uma *coverage* (valores e propriedades de um conjunto de localizações geográficas) no servidor.

11.4.4 Web Map Service

A especificação OpenGIS *Web Map Service* (WMS) define um serviço para a produção de mapas na Internet. Neste sentido, o mapa é uma representação visual dos dados geográficos e não os dados de fato. Os mapas produzidos são representações geradas em formatos de imagem, como PNG, GIF e JPEG, ou em formatos vetoriais, como o SVG.

Quando o cliente requisita um mapa utilizando o serviço, um conjunto de parâmetros deve ser informado ao servidor: as camadas desejadas, os estilos que devem ser aplicados sobre as camadas, a área de cobertura do mapa, a projeção ou sistema de coordenadas geográficas, o formato da imagem gerada e também o seu tamanho.

O serviço possui as seguintes operações:

- getCapabilities: obtém os metadados do servidor, que descrevem o conteúdo e os parâmetros aceitos.

- *getMap*: obtém a imagem do mapa que corresponde aos parâmetros informados.
- *getFeatureInfo*: recupera informações sobre um elemento (feature) particular de um mapa.

11.4.5 Catalog Service

A Internet possui uma vasta quantidade de informação geoespacial distribuída em vários formatos e mantida por inúmeras instituições. A organização dessa informação com base na construção de catálogos é uma forma de facilitar sua distribuição, localização e acesso.

A especificação OpenGIS *Catalog Services* (OCS) introduz um serviço para a publicação e busca em coleções de informações descritivas (metadados) de dados espaciais e objetos relacionados. Os metadados de um catálogo representam as características dos recursos que podem ser pesquisados e apresentados para a avaliação e processamento tanto de homens quanto de máquinas.

Na especificação, é definida uma linguagem de consulta comum a todas as interfaces do serviço, a *Common Catalog Query Language*, que possui uma sintaxe semelhante a uma cláusula Where do SQL. Um esquema de metadados básico (*core metadata schema*), baseado na norma ISO19115 - *Geographic Information Metadata*, é proposto para facilitar a interoperabilidade dos catálogos. Assim, uma mesma consulta pode ser executada em diferentes catálogos. Um conjunto de atributos mínimo é definido para as consultas e também para as respostas das consultas.

As operações disponíveis no serviço são as seguintes:

- *getCapabilities*: permite que um cliente recupere metadados descrevendo as características do servidor.
- *query*: operação que executa uma consulta no catálogo e retorna um conjunto de zero ou mais referências que satisfazem à consulta.
- *present*: recupera os metadados de uma ou mais referências.
- *describeRecordType*: retorna a definição do tipo de uma ou mais referências.

- *getDomain*: retorna a domínio (tipos de valores possíveis) de um atributo.
- *initialize*: utilizado para iniciar uma sessão interativa, para a qual um identificador único é gerado.
- *close*: encerra uma sessão interativa.
- *status*: recupera a situação de uma operação iniciada anteriormente, ainda em execucação ou já encerrada.
- *cancel*: permite que um cliente cancele uma operação.
- *transaction*: utilizada para que um cliente solicite um conjunto de ações de inserção, remoção ou alteração de itens do catálogo.
- *harvestResource*: operação que tenta recuperar recursos de uma localização específica e que pode ser reprocessada de tempos em tempos.
- *order*: permite que um cliente execute uma operação de compra de um recurso, negociando preço e outros fatores.

11.4.6 Web Terrain Service

A especificação OpenGIS *Web Terrain Service* (WTS) é, na verdade, uma especialização do WMS que incorpora modelos de elevação de terreno, com perspectiva e renderização tridimensional de mapas. O resultado produzido, assim como no WMS, é uma representação pictórica dos dados geográficos.

As operações *getCapabilities* e *getMap* seguem a definição do WMS. A diferença fica por conta da inclusão da operação *getView*:

- *getView*: obtém uma cena 3D, que é uma visão de um lugar a partir do ponto de vista de um observador. A operação exige o fornecimento de alguns parâmetros para a composição da cena: o ponto de interesse, a distância e o ângulo entre o ponto de interesse e o observador da cena, o ângulo representando o campo de visão e o azimute.

11.4.7 Web Coordinate Transformation Service

Este serviço especifica uma interface para a conversão de dados de um sistema de coordenadas espaciais (CRS - *coordinate reference system*) para outro. O serviço recebe como entrada objetos geográficos digitais, que

podem ser objetos vetoriais (*features*) ou matriciais (*coverages*), que estão georreferenciados em um CRS e retorna os mesmos objetos em outro CRS especificado.

O OpenGIS *Web Coordinate Transformation Service* (WCTS) define sete operações que podem ser requisitadas pelos clientes.

- *getCapabilities*: como em todos os outros serviços Web do OpenGis, esta operação retorna as propriedades do servidor.
- *transform*: requisição para a transformação de coordenadas de um conjunto de objetos. O CRS dos objetos deve ser informado, assim como o novo CRS desejado.
- *isTransformable*: o retorno dessa requisição indica se o servidor WCTS consegue processar a transformação entre dois CRS especificados e também se podem ser processados tanto *features* quanto *coverages*.
- *getTransformation*: utilizada para que um cliente consulte as definições das transformações que o servidor pode processar de um CRS para outro.
- *describeTransformation*: esta requisição recupera a definição de uma ou mais transformações pelo seu identificador.
- *describeCRS*: um cliente pode recuperar a definição de um ou mais CRS com essa requisição.
- *describeMethod*: recupera uma ou mais definições de métodos das operações.

11.4.8 Geolinking Service

Um *geolink* ocorre quando a geometria de um objeto espacial não é armazenada juntamente com seus dados alfanuméricos, mas apenas um identificador geográfico para a geometria (por exemplo, o nome de uma cidade). O identificador geográfico se refere, portanto, a uma geometria armazenada em outro banco de dados. O serviço de Geolinking tem por objetivo executar um *join* entre os atributos alfanuméricos e as geometrias que compartilham uma chave em comum (o identificador geográfico).

Os identificadores geográficos podem incluir nomes de lugares, códigos postais, códigos de área telefônicos e outros. Muitos bancos de

dados corporativos possuem dados dessa natureza, mas não utilizam seu potencial geográfico. O OpenGIS *Geolinking Service* é uma alternativa para o georreferenciamento dessas bases de dados. A especificação ainda está em fase de discussão.

As operações do serviço são:

- *getCapabilities*: recupera informações gerais sobre o servidor.
- *geoLink*: usado para instruir o servidor a acessar um conjunto de dados específico para o geolink e processar o *join* solicitado.

11.4.9 Web Gazetteer Service

Esta especificação adiciona ao protocolo WFS alguns recursos específicos para a implementação de interfaces para consulta, inserção e atualização de objetos armazenados em *gazetteers digitais* (Souza et al., 2004).

Os recursos explorados que vão além do WFS são: o acesso aos relacionamentos hierárquicos entre os termos do *gazetteer*, baseado nos conceitos de termo mais geral (BT – *broader term*), termo mais específico (NT – *narrower term*) e termo relacionado (RT – *related term*); e a recuperação de propriedades específicas de gazetteers, tais como o tipo dos lugares. Trata-se de outro serviço ainda em fase de discussão. Os operações são as mesmas do WFS, com pequenas modificações.

11.4.10 Web Registry Service

Um problema derivado da aceitação e implementação pela comunidade dos serviços Web propostos pelo OpenGIS passa a ser a localização dos servidores espalhados pela rede. A solução encontrada para o problema foi a especificação de mais um tipo de serviço Web.

O objetivo da especificação OpenGIS *Web Registry Service* (WRS) é propor um serviço capaz de fornecer uma estrutura para a localização dos servidores na Web. Os administradores dos servidores os registrariam em um ou mais servidores WRS para que esses pudessem ser encontrados. O catálogo do WRS fornece a localização e as características dos servidores OpenGIS nele registrados. Não seria necessário sequer executar a operação *getCapabilities* em cada servidor, já que o WRS informa aos clientes as características de cada servidor cadastrado. Essa especificação ainda se encontra em fase de discussão.

Eis as operações do serviço:

- *getCapabilities*: retorna as características do servidor.
- *getDescriptor*: retorna os servidores registrados que atende à consulta.
- *registerService*: registra um servidor OpenGIS no servidor WRS.

11.5 Leituras complementares

O documento *OpenGIS® Reference Model* (Percivall, 2003) oferece um bom resumo da proposta de trabalho do OGC. Para detalhes sobre os serviços, recomenda-se uma visita ao website do OGC (OGC, 2005a).

Referências

COX, S.; DAISEY, P.; LAKE, R.; PORTELE, C.; WHITESIDE, A. (ed), 2003. **OpenGIS® Geography Markup Language (GML) Implementation Specification**. Open Geospatial Consortium, Inc.

GARDELS, K., 1996. The Open GIS Approach to Distributed Geodata and Geoprocessing. In: Third International Conference/Workshop on Integrating GIS and Environmental Modeling. Santa Fe, NM, USA, 1996. p. 21-25.

OGC, 1999. **The OpenGIS Abstract Specification – Topic 6: The Coverage Type and its Subtypes.** Open Geospatial Consortium, Inc.

OGC, 2005a, OpenGIS Consortium Inc.

OGC, 2005b, OGC to Begin Geospatial Semantic Web Interoperability Experiment. Press Release, 12/04/2005. http://www.opengeospatial.org/press/?page=pressrelease&year=0&prid=222

PERCIVALL, G. (ed), 2003. **OpenGIS® Reference Model**. Document number OGC 03-040 Version: 0.1.3. Open Geospatial Consortium, Inc.

SONNET, J. (ed), 2004**. OWS 2 Common Architecture: WSDL SOAP UDDI**. Discussion Paper OGC 04-060r1, Version: 1.0.0. Open Geospatial Consortium, Inc.

SOUZA, L. A.; DELBONI, T.; BORGES, K. A. V.; DAVIS JR., C. A.; LAENDER, A. H. F. Locus: Um Localizador Espacial Urbano. In: VI Simpósio Brasileiro de GeoInformática (GeoInfo 2004),2004,Campos do Jordão (SP). Sociedade Brasileira de Computação (SBC), p. 467-478.

12 *Descrição da TerraLib*

Lúbia Vinhas
Karine Reis Ferreira

12.1 Introdução

Esse capítulo descreve a biblioteca TerraLib em seus aspectos mais relevantes em termos de bancos de dados geográficos, incluindo o modelo conceitual do banco de dados, o modelo de armazenamento de geometrias e dados descritivos, e os mecanismos de manipulação do banco em diferentes níveis de abstração.

A TerraLib é um projeto de software livre que permite o trabalho colaborativo entre a comunidade de desenvolvimento de aplicações geográficas, servindo desde à prototipação rápida de novas técnicas até o desenvolvimento de aplicações colaborativas. Sua distribuição é feita através da Web no site www.terralib.org.

TerraLib é uma biblioteca de classes escritas em C++ para a construção de aplicativos geográficos, com código fonte aberto e distribuída como um software livre. Destina-se a servir como base para o desenvolvimento cooperativo na comunidade de usuários ou desenvolvedores de SIG's – Sistemas de Informação Geográfica.

TerraLib fornece funções para a decodificação de dados geográficos, estruturas de dados espaço-temporais, algoritmos de análise espacial além de propor um modelo para um banco de dados geográficos (Câmara et al. 2002). A arquitetura da biblioteca é mostrada na Figura 12.1. Existe um módulo central, chamado *kernel*, composto de estruturas de dados espaço-temporais, suporte a projeções cartográficas, operadores espaciais e uma interface para o armazenamento e recuperação de dados espaço-temporais em bancos de dados objeto-relacionais, além de

mecanismos de controle de vizualização. Em um módulo composto de *drivers* a interface de recuperação e armazenamento é implementada. Esse módulo também contém rotinas de decodificação de dados geográficos em formatos abertos e proprietários. Funções de análise espacial são implementadas utilizando as estruturas do *kernel*. Finalmente, sobre esses módulos podem ser construídas diferentes interfaces aos componentes da TerraLib em diferentes ambientes de programação (Java, COM, C++) inclusive para a implementação de serviços OpenGIS como o WMS – Web Map Server (OGIS, 2005).

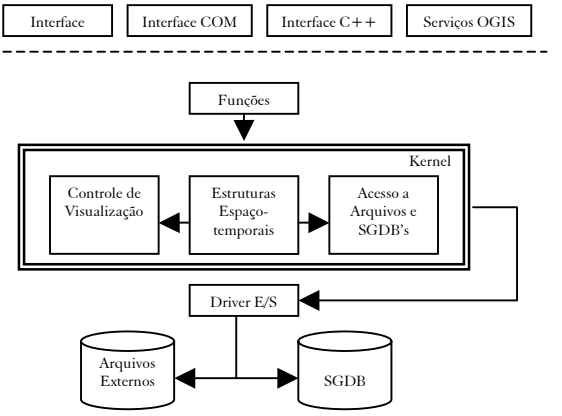


Figura 12.1 – Arquitetura da TerraLib.

Uma das características mais importantes da TerraLib é a sua capacidade de integração a sistemas de gerenciamento de bancos de dados objeto-relacionais (SGBD-OR) para armazenar os dados geográficos, tanto sua componente descritiva quanto sua componente espacial. Essa integração é o que permite o compartilhamento de grandes bases de dados, em ambientes coorporativos, por aplicações customizadas para diferentes tipos de usuários. A TerraLib trabalha em um modelo de arquitetura em camadas (Davis e Câmara, 2001), funcionando como a camada de acesso entre o banco e a aplicação final.

Como exemplo de um aplicativos geográfico construído sobre a TerraLib, podemos citar o TerraView (INPE/DPI, 2005). Na Figura 12.2 ilustramos como o TerraView utiliza a TerraLib como camada de acesso

a um banco de dados sob o controle de um SGDB-OR como o MySQL (MYSQL, 2005), por exemplo.

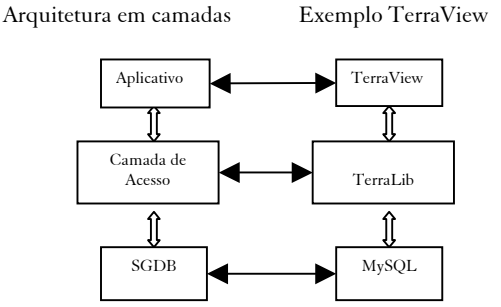


Figura 12.2 – Exemplo de uso da TerraLib como camada de acesso ao banco de dados.

Enquanto biblioteca de software, a TerraLib é compilada em um ambiente multiplataforma, Windows e Linux, e em diferentes compiladores C++. A TerraLib usa extensivamente os mecanismos mais atuais da linguagem C++, como a STL – Standard Template Libray, classes parametrizadas e programação multi-paradigma (Stroustroup, 1997).

Esse capítulo é ilustrado com trechos de código C++ que utilizam a TerraLib. Vale lembrar, que esses exemplos, por questão de clareza, contêm apenas os trechos mais relevantes para ilustrar a informação sendo destacada. Maiores detalhes sobre as classes, estruturas de dados e funções da biblioteca, usadas nos exemplos, podem ser obtidos na documentação de código da TerraLib disponível em www.terralib.org.

12.2 Modelo conceitual

A TerraLib propõe não somente um modelo de armazenamento de dados geográficos em um SGBD-OR, mas também um modelo conceitual de banco de dados geográfico, sobre o qual são escritos seus algoritmos de processamento. As entidades que formam o modelo conceitual são:

Banco de Dados – representa um repositório de informações contendo tanto os dados geográficos quanto o seu modelo de organização. Um banco de dados pode ser materializado em diferentes SGDB's – Sistemas Gerenciadores de Bancos de Dados, comerciais ou de domínio público. O único requisito da TerraLib é que o SGDB possua a capacidade de armazenar campos binários longos, ou uma extensão própria capaz de criar tipos abstratos espaciais, e que possa ser acessado por alguma camada de *software*.

Layer – um *layer* representa uma estrutura de agregação de um conjunto de informações espaciais que são localizadas sobre uma região geográfica e compartilham um conjunto de atributos, ou seja, um *layer* agrega coisas semelhantes. Como exemplos de *layers* podem ser citados os mapas temáticos (mapa de solos, mapa de vegetação), os mapas cadastrais de objetos geográficos (mapa de municípios do Distrito Federal) ou ainda dados matriciais como cenas de imagens de satélites. Independentemente da representação computacional adotada para tratar o dado geográfico, matricial ou vetorial, um *layer* conhece qual a projeção cartográfica da sua componente espacial.

Layers são inseridos no banco de dados através da importação de arquivos de dados geográficos em formatos de intercâmbio como shapefiles, ASCII-SPRING, MID/MIF, GeoTiff, JPEG ou dbf. A biblioteca fornece as rotinas de importação desses arquivos. *Layers* também podem ser gerados a partir de processamentos executados sobre outros *layers* já armazenados no banco.

Representação – trata do modelo de representação da componente espacial dos dados de um *layer* e pode ser do tipo vetorial ou matricial. Na representação vetorial, a TerraLib distingue entre representações formadas por pontos, linhas ou áreas e também outras representações mais complexas formadas a partir dessas como células e redes.

Para representações matriciais, a TerraLib suporta a representação de grades regulares multi-dimensionais.

A TerraLib permite que um mesmo geo-objeto de um *layer* possua diferentes representações vetoriais (ex. um município pode ser representado pelo polígono que define os seus limites, bem como pelo ponto onde está localizado em sua sede). A entidade representação da

TerraLib guarda informações como o seu menor retângulo envolvente ou a resolução horizontal e vertical de uma representação matricial.

O termo representação espacial, no contexto da TerraLib, é muitas vezes usado de maneira análoga ao termo geometria.

Projeção Cartográfica – serve para representar a referência geográfica da componente espacial dos dados geográficos. As projeções cartográficas permitem projetar a superfície terrestre em uma superfície plana. Diferentes projeções são usadas para minimizar as diferentes deformações inerentes ao processo de projeção de um elipsóide em um plano. Cada projeção é definida a partir de certo número de parâmetros como o Datum planimétrico de referência, paralelos padrão e deslocamentos (Snyder, 1987).

Tema – serve principalmente para definir uma seleção sobre os dados de um *layer*. Essa seleção pode ser baseada em critérios a serem atendidos pelos atributos descritivos do dado e/ou sobre a sua componente espacial.

Um tema também define o visual, ou a forma de apresentação gráfica da componente espacial dos objetos do tema. Para o caso de dados com uma representação vetorial a componente espacial é composta de elementos geométricos como pontos, linhas ou polígonos. Para os dados com uma representação matricial, sua componente espacial está implícita na estrutura de grade que a define, regular e com um espaçamento nas direções X e Y do plano cartesiano.

Os temas podem definir também formas de agrupamento dos dados de um *layer*, gerando grupos, os quais possuem legendas que os caracterizam.

Vista – serve para definir uma visão particular de um usuário sobre o banco de dados. Uma vista define quais temas serão processados ou visualizados conjuntamente. Além disso, como cada tema é construído sobre um *layer* com sua própria projeção geográfica, a vista define qual será a projeção comum para visualizar ou processar os temas que agrega.

Visual – um visual representa um conjunto de características de apresentação de primitivas geométricas. Por exemplo, cores de preenchimento e contorno de polígonos, espessuras de contornos e

linhas, cores de pontos, símbolos de pontos, tipos e transparência de preenchimento de polígonos, estilos de linhas, estilos de pontos, etc.

Legenda – uma legenda caracteriza um grupo de dados, dentro de um tema, apresentados com o mesmo visual, quando os dados do tema são agrupados de alguma forma.

As entidades que formam o modelo conceitual estão representadas tanto nas classes que compõe a biblioteca quanto em um conjunto de tabelas no banco de dados. A seção seguinte mostra como as entidades do modelo conceitual são representadas nas classes da TerraLib.

12.3 Classes do modelo conceitual

O conceito de um banco de dados TerraLib é independente do SGDB onde será fisicamente armazenado e é implementado em uma classe abstrata chamada `TeDatabase`. Essa classe abstrata contém os métodos necessários para criar, popular e consultar um banco de dados. A classe `TeDatabase` é derivada em classes concretas, chamadas *drivers*, que resolvem para diferentes SGDB's comerciais e de domínio público, as particularidades de cada um de forma que as aplicações possam criar bancos de dados em diferentes gerenciadores. A TerraLib fornece alguns *drivers* em sua distribuição padrão, como mostrado na Figura 12.3. *Drivers* para outros gerenciadores podem ser criados através da implementação dos métodos virtuais definidos na classe.

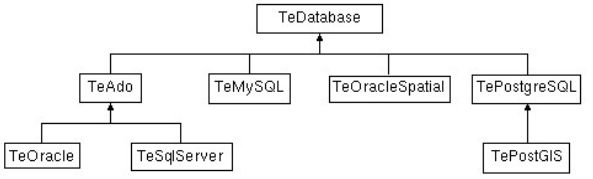


Figura 12.3 – *Drivers* para bancos de dados fornecidos pela TerraLib.

Tipicamente, as aplicações que usam a TerraLib processam ponteiros para a classe `TeDatabase`, inicializados com instancias concretas de algum *driver* como mostrado no Exemplo 12.1.

```
TeDatabase* myDb = 0;
if (op == “ado”)
    myDb = new TeAdo(); // Usa ACCESS através da biblioteca
ADO
else
    myDb = new TeMySQL();   // Usa MySQL
```

Exemplo 12.1 – A classe `TeDatabase`.

Um ***Layer*** existe em memória como uma instância da classe `TeLayer`. Um exemplo de criação de um *layer* através da importação de um arquivo de dados em formato MID/MIF é mostrado no Exemplo 12.2. O arquivo contém os dados de um cadastro de municípios de um estado e a rotina de importação vai criar um *layer* no banco de dados com as geometrias e atributos dos municípios descritos no arquivo.

```
TeLayer*                    lmun                    =
TeImportMIF("../data/Municipios.mid",myDb,"Municipios");
```

Exemplo 12.2 – Importação de um arquivo de dados.

Uma **Representação** existe em memória como uma instância da estrutura `TeRepresentation`. Cada *layer* possui a informação de quais as representações geométricas (ou geometrias) possui. Assim, após a execução do código acima podemos recuperar algumas informações sobre a representação geométrica dos municípios como o seu menor retângulo envolvente (ver Exemplo 12.3).

```
TeRepresentation*          munPol          =          lmun-
>getRepresentation(TePOLYGONS);
// recupera o menor retângulo envolvente das geometrias do
tipo
// polígono do layer de municípios
TeBox = mumPol->box_;
```

Exemplo 12.3 – A classe `TeRepresentation`.

Uma **Projeção** existe em memória como uma instância da classe `TeProjection`. A TerraLib também fornece uma classe para descrever

um datum planimétrico chamada de `TeDatum`. A `TeProjection` é uma classe abstrata que define métodos para converter coordenadas de uma projeção para coordenadas geográficas (latitude e longitude) e vice-versa.

A TerraLib fornece em sua distribuição um conjunto de especializações da classe `TeProjection` representando as projeções mais comuns como UTM, Mercator ou Policônica. O Exemplo 12.4 mostra como converter coordenadas entre uma projeção UTM, Datum SAD69 e uma projeção Policônica, Datum WGS84.

```
TeDatum dSAD69 = TeDatumFactory::make("SAD69");
TeDatum dWGS84 = TeDatumFactory::make("WGS84");
TeUtm* pUTM = new TeUtm(dSAD69,-45.0*TeCDR);
TePolyconic* pPolyconic = new TePolyconic(dWGS84,-45.0*TeCDR);

TeCoord2D pt1(340033.47, 7391306.21);   // em projeção UTM
// Conversão de UTM para policônica
pUTM->setDestinationProjection(pPolyconic);
// Converte uma coordenada de UTM para coordenada geográfica
TeCoord2D ll = pUTM->PC2LL(pt1);
// Converte de geográfica para policônica
TeCoord2D pt2 = pPolyconic->LL2PC(ll);
// Converte uma coordenada de policônica para coordenada UTM
pPolyconic->setDestinationProjection(pUTM);
ll = pPolyconic->PC2LL(pt2);
pt1 = pUTM->LL2PC(ll);
```

Exemplo 12.4 – Conversão de coordenadas para diferentes projeções.

A classe `TeLayer` possui a indicação de qual é a sua projeção cartográfica como mostrado no Exemplo 12.5.

```
TeProjection* munProj = lmun->projection();
```

Exemplo 12.5 – Recuperação da projeção de um *layer*.

Temas existem em memória como instâncias da classe `TeTheme` e **Vistas** como instâncias da classe `TeView`. O Exemplo 12.6 mostra como criar uma nova vista e um novo tema a partir do *layer* criado no Exemplo 12.2.

```
// Cria uma vista com a mesma projeção do layer
TeView* view = new TeView("Municipios ", user);
view->projection(munProj);
// salva a vista no banco de dados
myDb->insertView(view))
// cria um tema com todos os objetos do layer
TeTheme* theme = new TeTheme("t_municipios ", lmun);
view->add(theme);
```

Exemplo 12.6 – Criação de um Tema.

Uma ilustração que resume o relacionamento entre as principais classes que formam o modelo conceitual da TerraLib é mostrado na Figura 12.4. Um banco TerraLib é formado por um conjunto de *layers*, cada *layer* possui uma projeção cartográfica. Em um banco podem ser criadas *n* vistas e cada vista possui sua própria projeção cartográfica. Cada vista pode conter *n* temas sendo que cada tema é criado a partir de um *layer*.

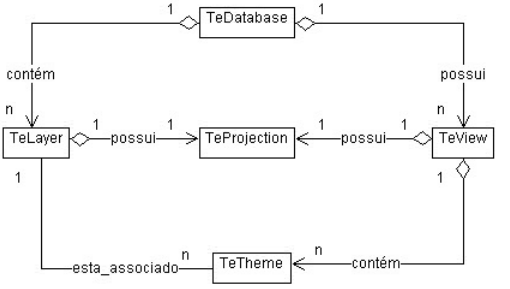


Figura 12.4 – Relacionamento entre as classes do modelo conceitual de TerraLib.

12.4 Modelo do banco de dados

Fisicamente, um banco de dados TerraLib é formado por um conjunto de tabelas em um SGBD-OR, onde são armazenados tanto os dados geográficos (suas geometrias e seus atributos) quanto um conjunto de

informações sobre a organização desses dados no banco, ou seja, o modelo conceitual da biblioteca. Essas tabelas podem ser divididas em dois tipos:

1. *Tabelas de Metadados*: possuem nome e formato pré-definido e são usadas para guardar o modelo conceitual da TerraLib;
2. *Tabelas de Dados*: são usadas para guardar os dados em si, tanto em sua componente espacial quanto descritiva.

As tabelas de metadados são automaticamente criadas quando se cria um novo banco TerraLib. Para acessar bancos de dados já existentes, as aplicações abrem conexões a eles, uma aplicação pode manter conexões a mais que um banco de dados ao mesmo tempo. Ao criar ou abrir uma conexão a um banco de dados as aplicações devem informar os parâmetros exigidos pelo SGDBOR onde está armazenado o banco, como mostra o Exemplo 12.7. Ao final da execução da aplicação todas as conexões devem ser fechadas.

```
// Criando um novo banco TerraLib
TeDatabase* db = new TeMySQL();
db->newDatabase(dbname, user, password, host);
// Acessando um banco de dados já criado
db->connect(host,user,pass,dbname,0);
//... processamento

// Fecha a conexão
db->close();
```

Exemplo 12.7 – Criação e conexão a um banco TerraLib.

As tabelas de metadados servem para armazenar os conceitos descritos na Seção 12.2. Cada *layer* criado no banco gera um registro na tabela chamada `TE_LAYER`, o campo `layer_id` contém a identificação única de cada *layer* no banco de dados. Os outros campos dessa tabela armazenam o nome e o mínimo retângulo envolvente do *layer*, ou seja, o mínimo retângulo envolvente de todas as geometrias associadas ao *layer*.

Cada representação geométrica associada a um *layer* gera um registro na tabela `TE_REPRESENTATION`. Cada tabela de atributos associada a um *layer* gera um registro na tabela `TE_LAYER_TABLE`.

Cada vista criada no banco de dados gera um registro em uma tabela chamada de `TE_VIEW`. Cada instância de projeção cartográfica é armazenada no banco na tabela `TE_PROJECTION`. *Layers* e vistas possuem referência a um registro da tabela de projeções. Cada tema criado gera um registro na tabela `TE_THEME`. Cada tema possui uma referência para vista no qual está definido.

Para compreender melhor as tabelas do modelo de metadados e os relacionamentos entre elas é interessante observar o conteúdo dessas tabelas após a execução de uma seqüência típica de operações:

- Criar um banco de dados;
- Importar um dado geográfico de um arquivo para um *layer* do banco de dados;
- Criar uma *vista*;
- Criar um *tema* usando o *layer* criado e inseri-lo na *vista*.

A Figura 12.5 mostra as tabelas do modelo conceitual e seus relacionamentos após a execução da seqüência de operações. Por questões de simplicidade apenas alguns campos das tabelas são mostrados.As tabelas de dados serão descritas mais adiante após falarmos sobre o modelo de geometrias e de atributos de TerraLib.

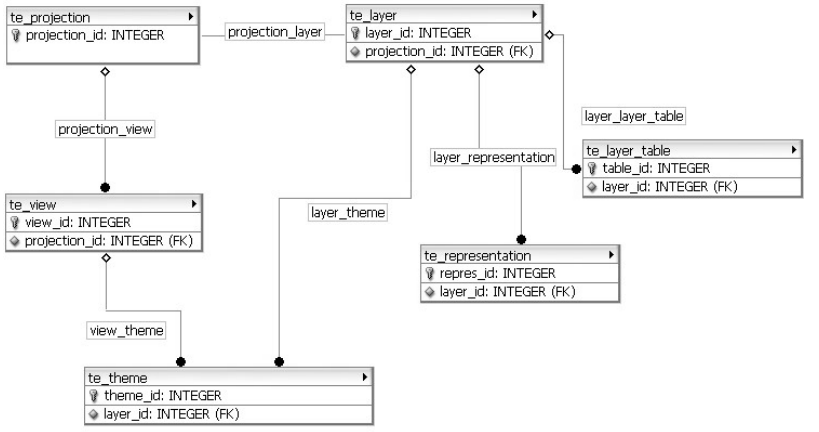


Figura 12.5 – Principais tabelas do modelo conceitual da TerraLib.

12.5 Modelo de geometrias

Conforme descrito na Seção 12.2, na TerraLib os dados geográficos são agregados em *layers*. *Layers* são formados por conjuntos de objetos, onde cada objeto possui uma identificação única, um conjunto de atributos descritivos e uma representação geométrica. Essa seção descreve o modelo de classes de geometria da TerraLib.

A classe base da qual derivam todas as geometrias de TerraLib é chamada de `TeGeometry`. Cada geometria possui uma identificação única, a referência ao seu menor retângulo envolvente e à sua projeção e a identificação do objeto geográfico que representa. As geometrias vetoriais de TerraLib são construídas a partir de coordenadas bi-dimensionais representadas na classe chamada de `TeCoord2D`. Essas geometrias são:

- *Pontos*: representados na classe `TePoint`, implementada como uma instância única de uma `TeCoord2D`;
- *Linhas*: composta de um ou mais segmentos são representadas na classe `TeLine2D`, implementada como um vetor de duas ou mais `TeCoord2D`;
- *Anéis*: representados pela classe `TeLinearRing`, são linhas fechadas, ou seja, a última coordenada é igual a primeira. A classe

`TeLinearRing` é implementada como uma instância única de uma `TeLine2D` que satisfaz a restrição de que a primeira coordenada seja igual a última;

- *Polígonos*: representados pela classe `TePolygon`, são delimitações de áreas que podem conter nenhum, um ou mais buracos, ou filhos. São implementados como um vetor de `TeLinearRing`. O primeiro anel do vetor é sempre o anel externo enquanto que os outros anéis, se existirem, são buracos ou filhos do anel externo.

A fim de otimizar a manutenção das geometrias em memória as classes de geometrias de TerraLib são implementadas segundo o padrão de projeto *handle/body* onde a implementação é separada da interface (Gamma et al., 2000). Além disso, as implementações são referências contadas, ou seja, cada instância de uma classe de geometria guarda o número de referências feitas a ela, inicializado com zero quando a instância é criada. Cada vez que uma cópia dessa instância é solicitada apenas o seu número de referências é incrementado e, cada vez que uma instância é destruída, o número de referências a ela é decrementada. A instância é efetivamente destruída apenas quando esse número chega ao valor zero.

Outro aspecto importante das classes de geometria da TerraLib é que elas são derivadas ou da classe `TeGeomSingle` ou da classe `TeGeomComposite`, representando, respectivamente, que são geometrias com um único elemento menor (como um `TePoint`) ou que podem ser compostas de outros elementos menores (como é o caso de `TePolygon`, `TeLine2D` e `TeLinearRing`). Esse padrão de compostos de elementos menores (Gamma et al., 2000) é aplicado também em classes que formam conjuntos de pontos, linhas, polígonos e são representados nas classes `TePointSet`, `TeLineSet`, e `TePolygonSet`. Os padrões de projeto são implementados com o recurso de classes parametrizadas como mostrado na Figura 12.6, o que torna o código mais reutilizável.

As geometrias matriciais são representadas na classe `TeRaster`, que será descrita por completo no Capítulo 13.

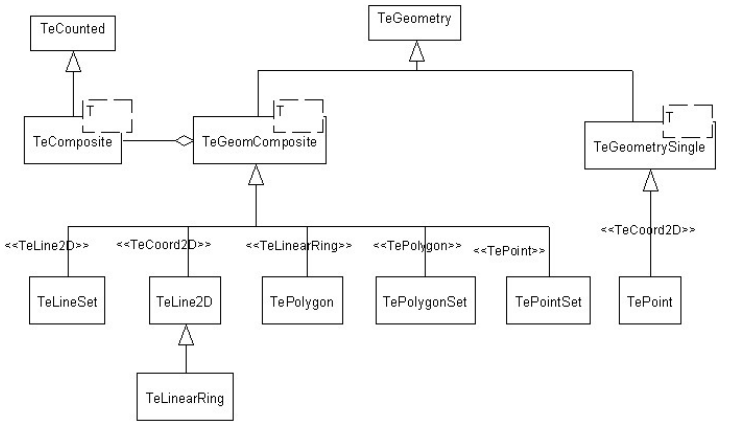


Figura 12.6 – Diagrama das principais classes de geometria da TerraLib (adaptado de Queiroz, 2003)

Um exemplo de criação de geometrias em memória é mostrado no Exemplo 12.8.

```
// Cria um conjunto de linhas
TeLine2D reta;
reta.add(TeCoord2D(500,500));
reta.add(TeCoord2D(600,500));
reta.add(TeCoord2D(700,500));
reta.objectId("reta");

TeLine2D ele;
ele.add(TeCoord2D(700,700));
ele.add(TeCoord2D(800,600));
ele.add(TeCoord2D(900,600));
ele.objectId("ele");

TeLineSet ls;
ls.add(reta);
ls.add(ele);

// Cria um conjunto de polígonos
```

```
// Um polígono simples
TeLine2D line;
line.add(TeCoord2D(900,900));
line.add(TeCoord2D(900,1000));
line.add(TeCoord2D(1000,1000));
line.add(TeCoord2D(1000,900));
line.add(TeCoord2D(900,900));

TeLinearRing r1(line);
TePolygon poly1;
poly1.add(r1);
poly1.objectId("spoli");

// Um polígono com um filho
TeLine2D line2;
line2.add(TeCoord2D(200,200));
line2.add(TeCoord2D(200,400));
line2.add(TeCoord2D(400,400));
line2.add(TeCoord2D(400,200));
line2.add(TeCoord2D(200,200));
TeLinearRing r2(line2);

TeLine2D line3;
line3.add(TeCoord2D(250,250));
line3.add(TeCoord2D(250,300));
line3.add(TeCoord2D(300,300));
line3.add(TeCoord2D(300,250));
line3.add(TeCoord2D(250,250));
TeLinearRing r3(line3);
TePolygon poly2;
poly2.add(r2);
poly2.add(r3);
poly2.objectId("cpoli");
TePolygonSet ps;
ps.add(poly1);
```

```
ps.add(poly2);

// Cria um conjunto de pontos
TePoint p1(40,40);
p1.objectId("ponto1");
TePointSet pos;
pos.add(p1);
```

Exemplo 12.9 – Criação de geometrias vetoriais em memória.

A TerraLib implementa uma estrutura de dados de espaços celulares, que juntamente com o suporte para predicados temporais atende às necessidades de implementação de modelos dinâmicos baseados em espaços celulares. Espaços celulares podem ser vistos ou como uma estrutura matricial generalizada onde cada célula armazena mais que um valor de atributo ou como um conjunto de polígonos que não se interceptam. Essa estrutura traz como uma vantagem a possibilidade de armazenar conjuntamente, numa única estrutura, todo o conjunto de informações necessárias para descrever um fenômeno espacial complexo, o que beneficia aspectos de visualização e interface. Todas as informações podem ser apresentadas da mesma forma que objetos geográficos com representação vetorial. Para atender a essa necessidade, a TerraLib propõe mais uma geometria chamada `TeCell`, que representa uma célula em um espaço celular materializado na classe `TeCellSet`.

O Exemplo 12.10 mostra como criar um espaço celular a partir de um *layer* armazenado em um banco.

```
// Recupera o layer de municípios
TeLayer* municipios = new TeLayer("Municipoios");
db_->loadLayer(municipios);
// Cria um espaço celular sobre a extensão do layer
// de municípios, onde cada célula tem 100 x 100 metros
TeLayer* espaco_cel = TeCreateCells(“CellsMunic”, municipios,
100, 100);
```

Exemplo 12.10 – Criação de um espaço celular.

Cada célula possui uma identificação única e uma referência a sua posição dentro do espaço celular, a qual podem ser associados diferentes atributos conforme o modelo dinâmico sendo construído.

12.6 Modelo de armazenamento de geometrias

A proposta da TerraLib, conforme mostramos na Figura 12.2, é trabalhar em bancos de dados geográficos que podem armazenar tanto atributos descritivos como atributos geométricos (como pontos, linhas, polígonos e dados matriciais). Esses bancos de dados podem ser construídos em SGDB's que possuem extensões espaciais, ou seja, possuem a capacidade de criar tipos espaciais e manipulá-los como tipos básicos e fornecem mecanismos eficientes de indexação e consulta (Shekkar e Chawla, 2002). Podem também ser construídos em SGDB's que oferecerem somente a capacidade de criar tabelas com campos do tipo binário longo. Na TerraLib esses dois tipos de SGDB's são usados de maneira transparente através da classe abstrata `TeDatabase`.

O modelo de armazenamento de geometrias em um banco leva em conta questões relativas à eficiência no seu armazenamento e na sua recuperação, e também a existência ou não de um tipo espacial no SGDB. Todas as tabelas que armazenam as geometrias possuem os campos:

- `geom_id`: do tipo inteiro, que armazena a identificação única da geometria;
- `object_id`: do tipo texto, que armazena a identificação única do objeto geográfico ao qual a geometria está relacionada;
- `spatial_data`: armazena o dado geométrico. O tipo desse campo depende do SGDB onde está armazenado o banco. Para SGDB's com extensão espacial é o tipo fornecido pela extensão. Para SGDB's sem a extensão espacial é um binário longo.

Para os SGDB´s sem extensão espacial as tabelas de geometrias do tipo linhas e polígonos possuem outros campos para armazenar o mínimo retângulo envolvente da geometria (`lower_x`, `lower_y`, `upper_x`, `upper_y` todos do tipo real). Esses campos são indexados pelos mecanismos fornecidos pelo SGBD e serão usados pelas rotinas de recuperação como os indexadores espaciais do dado. Para os SGDB's

com extensão, a coluna com o dado espacial é indexada espacialmente pelo mecanismo oferecido pela extensão.

A Figura 12.7 mostra a diferença entre uma tabela de geometria do tipo polígono criada em um banco sem extensão espacial e em um banco com extensão espacial. No segundo caso o tipo "GEOMETRY" representa o tipo espacial fornecido pela extensão. No caso das geometrias do tipo polígono, o modelo de armazenamento em campos longos, prevê que cada anel do polígono é armazenado em um registro da tabela. O anel externo contém a informação sobre o número de filhos que o polígono possui e os anéis internos guardam a identificação de seu pai, ou seja, do anel externo no qual estão contidos. Essa forma de armazenamento permite a recuperação parcial de polígonos com um grande número de filhos (por exemplo, uma operação de *zoom* em um mapa em uma área grande como a Amazônia legal, em uma escala pequena). Como são armazenados os retângulos envolventes de cada filho é possível recuperar somente o pai e os filhos que estão dentro do retângulo envolvente definido pelo *zoom*. Isso representa uma otimização no processamento desse dado.

Tabela de polígonos em SGDB´s sem extensão espacial

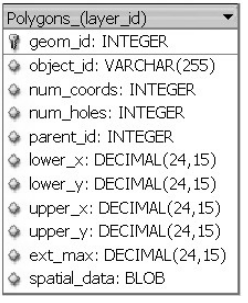


Tabela de polígonos no Oracle Spatial

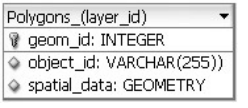


Figura 12.7 – Modelo de armazenamento de geometrias do tipo polígonos (adaptado de Ferreira, 2003).

A Figura 12.8 mostra como são as tabelas que armazenam geometrias do tipo linhas e a Figura 12.9 as tableas para geometrias do tipo pontos.

Tabela de linhas em SGDB´s sem extensão espacial

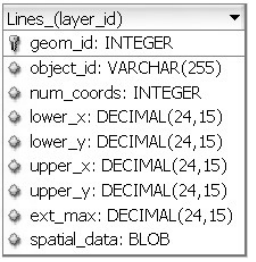


Tabela de linhas no Oracle Spatial

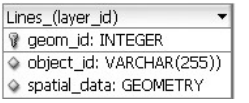


Figura 12.8 – Modelo de armazenamento de geometrias do tipo linhas.

Tabela de pontos em SGDB´s sem extensão espacial

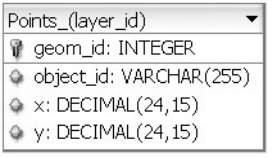


Tabela de pontos no Oracle Spatial

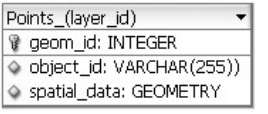


Figura 12.9 – Modelo de armazenamento de geometrias do tipo pontos.

12.7 Atributos descritivos

Os atributos descritivos dos objetos de um *layer* são representados em tabelas relacionais onde cada campo representa um atributo do objeto. Um dos campos deve conter a identificação do objeto e seus valores são repetidos nas tabelas de geometria permitindo assim a ligação entre os atributos descritivos e a geometria do objeto.

Cada *layer* pode ter uma ou mais tabelas de atributos e ao serem inseridos no banco de dados cada tabela de atributo é registrada na tabela de metadados chamada **`te_layer_table`**. Nessa tabela é registrada também o nome do campo que é a chave primária e identificador do objeto. Esse campo serve de ligação entre atributos e geometrias. Ao serem criados, os temas selecionam quais tabelas do *layer* irão usar.

Quando a tabela de atributos não possui nenhuma informação temporal e cada registro representa os atributos de um objeto diferente, a tabela é chamada de **tabela estática**. Essas classificação semântica das tabelas de atributos é uma característica da TerraLib e tem por objetivo definir funções e processamentos dependentes desses tipos. Isso ficará mais claro adiante quando falarmos do processamento de dados espaço-temporais.

Algumas tabelas de atributos, chamadas **tabelas externas**, não representam nenhum objeto definido em um *layer*, mas podem possuir algum campo com valores coincidentes com os valores de um campo de uma tabela estática de um *layer*. Através de uma operação de junção por esses campos coincidentes uma tabela externa pode acrescentar informações aos objetos de um *layer*. Por isso, tabelas externas podem ser incorporadas ao banco e registradas como tal, ficando disponíveis para serem usadas em todos os temas do banco.

Em memória, as tabelas de atributos são instâncias da classe `TeTable`. Essa classe sabe qual o nome, qual o tipo, quais são os campos e pode conter também o seu conteúdo. No Exemplo 12.11 mostramos como criar uma instância da classe `TeTable` em memória e como adicionar alguns registros a essa tabela.

```
// Cria a lista de atributos da tabela
TeAttributeList attList;
TeAttribute at;
at.rep_.type_ = TeSTRING;
at.rep_.numChar_ = 16;
at.rep_.name_ = "IdObjeto";
at.rep_.isPrimaryKey_ = true;
attList.push_back(at);

at.rep_.type_ = TeSTRING;         // um atributo do tipo string
at.rep_.numChar_ = 255;
at.rep_.name_ = "nome";
at.rep_.isPrimaryKey_ = false;
```

```
attList.push_back(at);

at.rep_.type_ = TeREAL;           // um atributo do tipo real
at.rep_.name_ = "val1";
at.rep_.isPrimaryKey_ = false;
attList.push_back(at);

// Cria uma instância da tabela em memória
TeTable attTable("teste2Attr",attList,"IdObjeto","IdObjeto");
row.push_back("reta");
row.push_back("L reta");
row.push_back("11.1");
attTable.add(row);
row.clear();
row.push_back("ele");
row.push_back("L ele");
row.push_back("22.2");
attTable.add(row);
row.clear();
row.push_back("spoli");
row.push_back("Pol simples");
row.push_back("33.3");
attTable.add(row);
row.clear();
row.push_back("ponto1");
row.push_back("Um ponto");
row.push_back("55.5");
attTable.add(row);
row.clear();
row.push_back("cpoli");
row.push_back("Pol buraco");
row.push_back("44.4");
attTable.add(row);
row.clear();
```

Exemplo 12.11 – Criação de uma tabela de atributos em memória.

As seções anteriores tiveram por objetivo explicar o modelo conceitual de um banco de dados geográfico TerraLib e o modelo físico de persistência desse modelo conceitual e dos dados geográficos. As seções seguintes tratarão do acesso ao banco em diferentes níveis de abstração: no nível do *layer* (`TeLayer`), no nível da banco de dados (`TeDatabase`) e no nível de objetos geográfico (`TeQuerier`).

12.8 Acesso ao banco de dados através do *layer*

As rotinas de importação que inserem um arquivo de dados geográfico são escritas usando os métodos da classe `TeLayer` (ver Exemplo 12.2). Essa classe fornece métodos para a inserção de geometrias e atributos. Através desses métodos os registros nas tabelas do modelo conceitual são preenchidos corretamente, fazendo as referências ao identificador do *layer* quando necessário.

O Exemplo 12.12 deve ser lido como uma seqüência do Exemplo 12.10 e do Exemplo 12.11 e mostra como criar um *layer* e como usar seus métodos para inserir no banco de dados as geometrias e atributos criados em memória.

```
// Cria a definição da projeção do layer
TeDatum mDatum = TeDatumFactory::make("SAD69");
TeProjection* pUTM = new TeUtm(mDatum,0.0);
// Cria um novo layer chamado "teste"
// myDb_ é um ponteiro para um banco de dados já conectado
TeLayer *layer1 = new TeLayer("teste",myDb_,pUTM);
layer1->addGeometries(TeLINES,"tabelaLinhas");
// Insere as linhas criadas em memória
layer1->addLines(ls);
// Insere os polígonos criados em memória
layer1->addPolygons(ps);
// Insere os pontos criados em memória
layer1->addPoints(pos);
// cria a tabela de atributos criada em memória
layer1->createAttributeTable(attTable);
// salva a tabela de atributos no aco de dados
layer1->saveAttributeTable(attTable))
```

Exemplo 12.12 – Inserção de geometrias e atributos no banco através da classe `TeLayer`.

O Exemplo 12.12 mostra como controlar a inserção de cada tipo de geometria (pontos, linhas ou polígonos) podendo inclusive especificar o nome para as tabelas de geometrias (como no caso da inserção de linhas).

É interessante verificar o conteúdo do banco de dados após a execução do Exemplo 12.12. Devem ser notados o conteúdo das tabelas de metadados e as referências feitas aos nomes das tabelas de geometrias, aos nomes das tabelas de atributos, ao identificador do *layer*, e o registro dos nomes de colunas usados para relacionar geometrias e atributos. Na Figura 12.10 são representados os conteúdos essas tabelas e seus relacionamentos. Para efeito de clareza nem todos os campos das tabelas são representados. As linhas contínuas representam relacionamentos físicos entre os campos das tabelas e as linhas pontilhadas representam relacionamentos em termos de conteúdo. Por exemplo, o nome da tabela de atributos e o nome do campo que faz o seu relacionamento com as tabelas de geometrias são registrados na tabela `TE_LAYER_TABLE`.

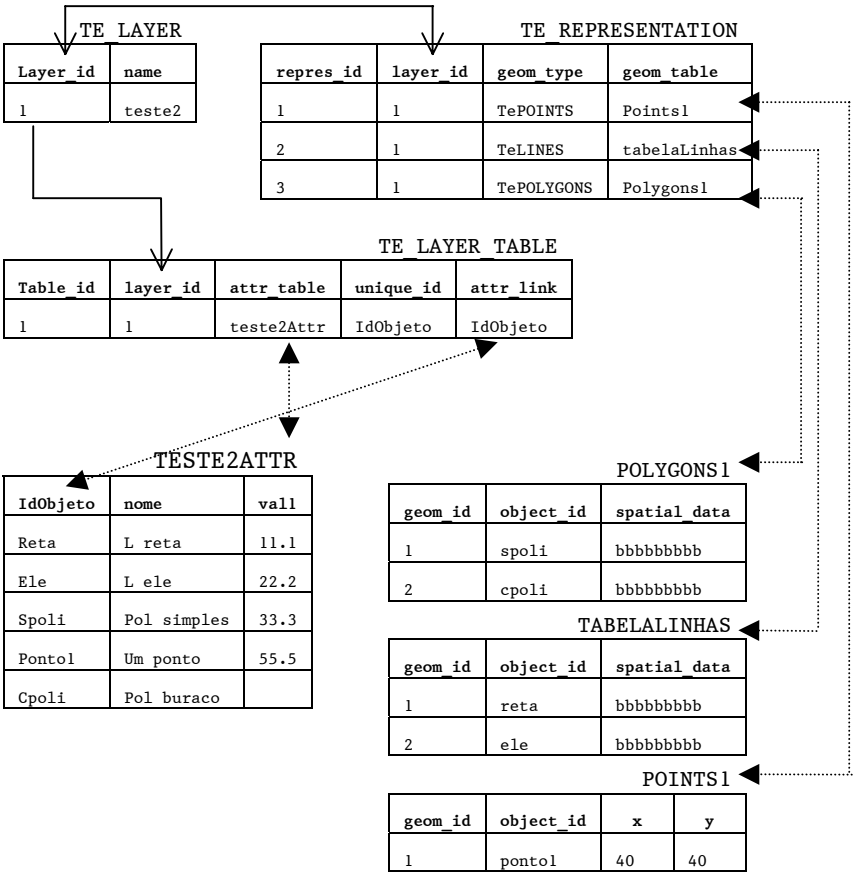


Figura 12.10 – Conteúdo do banco após a execução do Exemplo 10.

A recuperação dos dados armazenados no banco, também pode ser feita diretamente através da classe `TeLayer`. Essa classe fornece alguns métodos para recuperar as geometrias de um determinado tipo como mostrado no Exemplo 12.13.

```
TePolygonSet ps2;
layer1->getPolygons(ps2);
```

```
cout << "Numero de objetos com geometria do tipo poligono: "
<< ps2.size();
ps2.clear();
// recupera somente os polígonos associados ao objeto com o
// identificador “spoli”
layer1->loadGeometrySet("spoli");
```

Exemplo 12.13 – Recuperação de geometrias através do *layer*.

Os métodos para recuperação de geometrias e atributos através da classe `TeLayer` são poucos, mas a classe `TeDatabase` é uma interface completa de acesso ao banco de dados, essa interface será descrita a seguir.

12.9 A interface TeDatabase

A classe `TeDatabase` é uma interface completa de manipulação do banco de dados através da qual é possível inserir, alterar e recuperar as entidades do modelo conceitual e os dados geográficos. Ela contém todos os métodos para criar as tabelas de metadados, as tabelas de geometrias bem como uma tabela de atributos como ilustrado no Exemplo 12.14. Esse exemplo mostra também como recuperar um *layer* do banco e posteriormente como removê-lo. A remoção do *layer* implica que todos os seus dados e metadados são fisicamente removidos do banco bem como todas as outras entidades do modelo conceitual que fazem referência a ele. Por exemplo, ao remover um *layer* do banco todos os temas criados sobre esse *layer* devem ser removidos também.

```
db->createConceptualModel(); // cria todo o modelo conceitual
// cria uma tabela que armazena geometrias do tipo polígono
db->createPolygonGeometry(“TabelaPoligonos”);

// cria uma tabela de atributos descritivos
TeAttributeList meusAttributos;
db->createTable(“TabelaAtributos”, meusAttributos);

// cria uma nova coluna em uma tabela já existente
TeAttributeRep novaColuna;
```

```
novaColuna.type_ = TeSTRING;
novaColuna.name_ = "novaCol";
novaColuna.numChar_ = 10;
db->addColumn(“TabelaAtributos”, novaColuna);
// recupera um layer do banco
TeLayer *layer = new TeLayer("Distritos");
// carrega todas as informações sobre o layer chamado Distritos
db->loadLayer(layer);
// remove o layer
db->deleteLayer(layer->id());
```

Exemplo 12.14 – Métodos de criação e modificação de tabelas da classe `TeDatabase`.

A classe `TeDatabase` também fornece métodos para a inserção de geometrias e atributos nas tabelas do banco de dados como mostra o Exemplo 12.15.

```
TePolygonSet polyset;
db->insertPolygonSet(“tabPoligono”, polyset);

TeLineSet lineset;
db->insertLineSet   (“tabLine”, lineset);

TePointSet pointset;
db->insertPointSet(“tabPoint”, pointset);

TeTable tabAttributes;
insertTable(tabAttributes);
```

Exemplo 12.15 – Inserções no banco através da classe `TeDatabase`.

A classe `TeDatabase` também é capaz de submeter ao banco comandos escritos em SQL – Structured Query Language como mostrado no Exemplo 12.16. Apesar da SQL ser considerada padrão para SGDB's relacionais, podem existir algumas variações em termos da capacidade de execução de um comando SQL. O método `TeDatabase::execute` retorna

verdadeiro ou falso caso o comando foi executado com sucesso ou não, respectivamente. A classe `TeDatabase` armazena o erro retornado pelo SGDB resultante do último comando executado sobre o banco.

```
string sql = "UPDATE TabelaAtributos SET novaCol = 'xxx' ";
if (db->execute(sql))
  cout << "Comando executado com sucesso!";
else
  cout << "Erro: " << db->errorMessage();
```

Exemplo 12.16 – Execução de um comando SQL.

Note que o método `TeDatabase::execute` não deve ser usado para comandos que representam consultas ao banco, ou seja, que retornam valores ou registros, mas somente para comandos que alteram as tabelas e registros do banco.

12.10 A classe TeDatabasePortal

A classe `TeDatabase` fornece um mecanismo mais flexível de consulta ao banco de dados do que apenas através dos métodos da classe `TeDatabase`. Esse mecanismo é implementado pela classe `TeDatabasePortal`. Os portais de consultas são criados a partir de qualquer instância da classe `TeDatabase` que possua uma conexão aberta. Os portais podem submeter consultas SQL ao banco e disponibilizar os registros resultantes da consulta para a aplicação processar. Um banco pode criar quantos portais forem necessários, porém após serem consumidos os resultados todo portal deve ser destruído. O Exemplo 12.17 mostra o uso típico de um portal.

```
// Abre um portal no banco de dados
TeDatabasePortal* portal = db->getPortal();
if (!portal)
  return;

// Submete uma consulta sql a uma tabela
```

```
string sql = "SELEC * FROM TabelaAtributos";
if (!portal->query(sql))
{
  cout << "Não foi possível executar..." << db->errorMessage();
   delete portal;
   return;
}
// Consome todos os registros resultantes
while (portal->fetchRow())
{
  cout << "Atributo 1: " << portal->getData(0) << endl;
  cout << "Atributo 2: " << portal->getData("nome") << endl;
  cout << "Atributo 3: " << portal->getDouble(2) << endl;
}
// Libera os portal para fazer uma nova consulta
portal->freeResult();

// Submete uma nova consulta
sql = "SELECT SUM(val1) FROM TabelaAtributos WHERE val1 > 33.5";
if (portal->query(sql) && portal->fetchRow())
  cout << "Soma: " << portal->getDouble(0);
delete portal;
```

Exemplo 12.17 – Uso do `TeDatabasePortal`.

Analisando o exemplo vemos que após a chamada do método `TeDatabasePortal::query` os registros ficam disponíveis para serem consumidos em seqüência. Um ponteiro interno é posicionado em uma posição *anterior ao primeiro registro*. A cada chamada ao método `TeDatabasePortal::fetchRow` o ponteiro interno é incrementado e o valor verdadeiro é retornado quando o ponteiro está em um registro válido ou falso caso contrário. Dessa forma é possível fazer o *loop* em todos os registros retornados.

Os campos de um registro do portal podem ser acessados de duas formas: pela ordem do campo na seleção expressa na SQL (iniciando em zero), ou diretamente pelo nome do campo. O valor do campo pode ser obtido como uma *string* de caracteres através do método `TeDatabasePortal::getData` independentemente do tipo do campo ou através dos métodos que especificam o tipo de retorno (`getDouble`, `getInt`, `getDate` ou `getBlob`). Na segunda forma é necessário que a aplicação conheça o tipo do campo sendo acessado.

A classe `TeDatabasePortal` também pode executar consultas sobre uma tabela de geometrias e acessar os campos resultantes como os tipos geométricos da TerraLib, como mostrado no Exemplo 12.18.

```
// Submete uma consulta sobre uma tabela de geometrias
string q =" SELECT * FROM Poligonsll WHERE object_id='cpoli';
       q  +=  "  ORDER  BY  parent_id,  num_holes  DESC,  ext_max
ASC";

if (!portal->query(q) || !portal->fetchRow()))
{
  delete portal;
  return false;
}
bool flag = true;
do
{
  TePolygon poly;
  flag = portal->fetchGeometry(poly);
  // usa o polígono retornado
} while (flag);
delete portal;
```

Exemplo 12.18 – Uso de um portal para recuperar geometrias.

Essa seção mostrou como utilizar a interface `TeDatabase` para construir e consultar um banco de dados. As consultas expressas até agora foram sempre baseadas em critérios a serem atendidos sobre atributos descritivos porém, parte importante de um banco de dados geográfico é a

consulta por critérios espaciais. A seção seguinte mostra como as operações espaciais são tratadas na TerraLib.

12.11 Operações espaciais

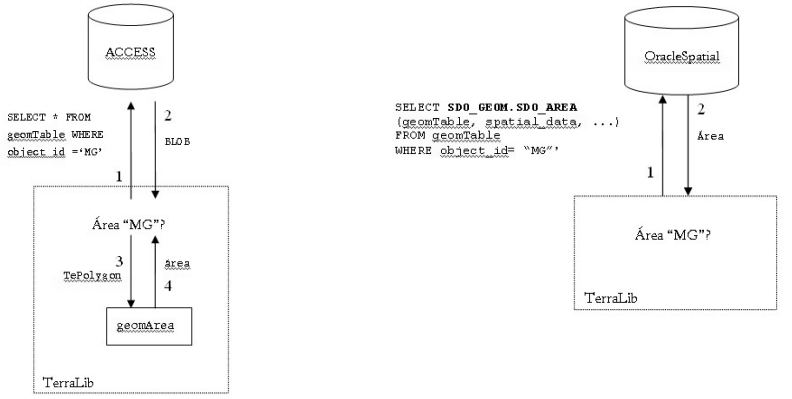
A TerraLib oferece um conjunto de operações espaciais sobre geometrias vetoriais e matriciais (uma revisão da literatura e descrição mais detalhada sobre operações espaciais pode ser encontrada em Ferreira, 2003). As operações espaciais implementadas na TerraLib podem ser divididas em:

- Determinação de relacionamentos topológicos entre geometrias vetoriais: são baseados na matriz de 9-intersecções dimensionalmente estendidas onde são formalizadas relações como toca, intercepta, cobre, disjunto ou é-coberto-por, como mostrado no capítulo 5.
- Funções métricas: como cálculo de áreas e comprimentos e buffers de distância;
- Geração de novas geometrias: como a geração de centróides e geometrias convexas;
- Operações que combinam geometrias: como diferença, união e intersecção e diferença simétrica;
- Operações zonais e focais sobre geometrias matriciais: como a obtenção de medidas estatísticas sobre uma região de um dado matricial e operações de recorde de uma região do dado matricial.

As operações espaciais estão implementados em um conjuntos de métodos da classe `TeDatabase` que funcionam como uma Interface Genérica de Operações Espaciais. Essa interface é genérica porque, através da classe `TeDatabase`, esses métodos podem ser chamados da mesma maneira em *drivers* para SGDB's com extensão espacial ou sem extensão espacial. *Drivers* para SGDB's com extensão espacial implementam a interface de operações especiais usando as funções da extensão espacial. *Drivers* que não possuem a extensão implementam a interface através de funções espaciais básicas escritas sobre as classes de geometria, após a sua recuperação do banco de dados. Na Figura 12.12 mostramos a diferença de execução de uma operação espacial, cálculo de área de um objeto, executada em um *driver* sem extensão espacial (p. ex. ACCESS) e um *driver* com extensão espacial (p. ex. Oracle Spatial).

As operações sobre dados matriciais foram implementadas somente usando as classes da TerraLib, uma vez que as extensões espaciais ainda não

fornecem as operação sobre dados matriciais. Uma descrição completa da interface genérica de operações espaciais em um banco de dados geográfico pode ser encontrada em Ferreira (2003).



(a) Banco sem extensão espacial (b) Banco com extensão espacial

Figura 12.11 – Representação da operação de cálculo de área (adaptado de Ferreira, 2003).

O Exemplo 12.19 mostra a execução de operações espaciais unárias sobre uma tabela de geometrias do tipo polígono, que representam objetos que são municípios do estado de São Paulo, identificados pelo nome do município. Sobre um objeto escolhido (o município de São Paulo) são calculados um valor de área e um *buffer* de distância de 1000 metros do objeto. O resultado da operação é um conjunto com dois polígonos onde o primeiro é o *buffer* externo e o segundo é o *buffer* interno.

```
string polyTable = "Poligons11";  // tabela de polígonos
// Consulta 1: calcular a área do município de São Paulo
// Chama uma função unária sobre a tabela de polígonos
vector<string> spId;              //     identificadores     dos
objetos
spId.push_back("São Paulo");
double area;
db->calculateArea(polyTable, TePOLYGONS, spId, area);
// Consulta 2: calcular um buffer de distância de 1000 metros
do município de São Paulo
// Chama uma função unária sobre a tabela de polígonos
```

```
TePolygonSet& bufferSet;
db->Buffer(polyTable, TePOLYGONS, spId, bufferSet, 1000.0);
```

Exemplo 12.19 – Operações espaciais unárias.

O Exemplo 12.20 mostra a execução de operações espaciais envolvendo mais uma tabela de geometrias, que contém geometrias do tipo linhas representando estradas estaduais. Essas operações são baseadas em consultas sobre relações topológicas entre geometrias, no primeiro caso entre geometrias de uma mesma tabela e no segundo entre geometrias de duas tabelas diferentes. É interessante notar que o resultado é retornado em um portal de forma que a aplicação possa consumir cada geometria resultante uma a uma.

```
string lineTable = "Lines12";     // tabela de linhas
TeDatabasePortal result = db->getPortal();  // portal
// Consulta 3: quais os municípios que tocam São Paulo?
// Chama uma função binária sobre a tabela de polígonos
if (db->spatialRelation(polyTable, TePOLYGONS, spId, result,
TeTOUCHES)) {
  bool flag = true;
  do {
       TePolygon poly;
       flag   =   result->fetchGeometry(poly);   //   polígono
retornado
  } while (flag);
}
result->freeResult();
// Consulta 4: quais as estradas que cruzam São Paulo?
// Função binária entre a tabela de polígonos e de linhas
if (db->spatialRelation(polyTable, TePOLYGONS, spId,
    lineTable, TeLINES, result, TeINTERSECTS)) {
  bool flag = true;
  do {
       TeLine line;
       flag = result->fetchGeometry(line); // linha retornada
  } while (flag);
}
```

Exemplo 12.20 – Operações Espaciais para a consulta de relacionamentos topológicos.

O Exemplo 12.21 mostra as operações Zonal e Recorte de um dado matricial. Na operação zonal, um conjunto de estatísticas (por exemplo soma, valor máximo, valor mínimo, contagem, média, variância, etc.) é calculado sobre a região do dado matricial contida dentro de um polígono que representa a zona de interesse. O resultado é fornecido em uma estrutura própria que armazena as estatísticas calculadas em cada dimensão do dado matricial.

Na operação de recorte o dado matricial é recortado pela zona de interesse, o resultado é um novo *layer* no banco de dados. A implementação dessa função foi feita usando o conceito de *iterador* sobre uma representação matricial, ou seja, um ponteiro que percorre todos os elementos do dado matricial. No caso da operação de recorte, foi usada uma especialização do conceito de *iterador* sobre dados matriciais, de forma que esse percorra somente os elementos que possuam uma certa relação com a região de interesse. As relações possíveis são que a área do elemento esteja toda dentro da região, que a área do elemento esteja toda fora da região, que o centro do elemento esteja dentro da região ou que o centro do elemento esteja fora da área de interesse.

```
string rasterTable = "RasterLayer12";   // tabela de linhas
TePolygon reg;
TeStatisticsDimensionVect result;

// Consulta 5: Calcular as estatísticas do dado matricial
usando como região de interesse
// o polígono defindo em "reg'
db->Zonal(rasterTable, reg, result);

// Consulta 5: Recortar o dado matricial usando como região de
interesse o polígono
// definido em "reg'
TeStrategicIterator st = TeBoxPixelIn;
db->Mask(rasterTable, reg, "Recorte", st);
```

Exemplo 12.21 – Operações sobre dados matriciais.

As rotinas que implementam as operações espaciais sobre as estruturas geométricas também estão disponíveis para serem usadas diretamente pelas aplicações. Essas rotinas são funções parametrizadas de forma a poder serem chamadas com a mesma assinatura, tomando como parâmetros as diferentes classes de geometrias vetoriais, como mostrado no Exemplo 12.22. Uma descrição completa da das operações espaciais da TerraLib está em Queiroz (2003).

```
TeLine line1;
TeLine line2;
bool res = TeOverlaps(line1, line2);
TePolygon pol1;
TePolygon pol2;
res = TeOverlaps(pol1, pol2);
```

Exemplo 12.22 – Chamada de operações espaciais sobre geometrias.

12.12 Dados espaço-temporais

A TerraLib tem uma proposta de trabalhar com dados geográficos espaciais e temporais, ou seja, de considerar a componente temporal dos dados geográficos, quando houver. Como exemplos de dados espaço-temporais podemos identificar:

- ***Eventos*: ocorrências independentes no espaço e no tempo, que possuem um rótulo temporal de quando o evento ocorreu. Esse é o caso de crimes que ocorrem em uma determinada localização de uma cidade em uma determinada data e hora. Cada objeto espacial associado a um associado a um evento terá uma única identificação no banco de dados;**
- ***Objetos Mutáveis*: os quais possuem uma geometria fixa ao longo do tempo, porém seus atributos descritivos podem variar. Um exemplo desse tipo de objeto são os dados manipulados por modelos dinâmicos baseados em espaços celulares, ou estações fixas de coletas de dados;**
- ***Objetos em Movimento*: os quais possuem limites e localizações, ou seja, geometrias que podem variar no tempo assim como seus atributos descritivos. Esse é o caso quando se acompanha a evolução**

de dos lotes de uma cidade, ou dos polígonos de desmatamento de uma região.

No modelo de banco de dados geográfico proposto pela TerraLib os dados são registrados semanticamente em função da sua natureza temporal com o objetivo de possam ser construídas funcionalidades específicas relativas a essa natureza temporal. Entre essas funcionalidades estão incluídos os algoritmos de agrupamento temporal, visualização de animações ou criação de temas com restrições temporais e ferramentas de modelagem dinâmica. Assim, quando uma tabela de atributos, associada a um *layer*, é inserida em um banco de dados, uma das informações registradas na tabela de metadados `TE_LAYER_TABLE` é qual o seu tipo, e dependendo desse tipo outras informações são necessárias. Além das tabelas estáticas e externas descritas anteriormente (ver Seção 12.7) os outros tipos de tabelas são:

- ***Eventos:*** **uma tabela de** atributos de eventos. É preciso registrar qual de seus campos possui a identificação única do evento (e que faz a ligação com sua geometria) e quais de seus campos possuem a informação temporal da ocorrência do evento. A informação temporal normalmente é um intervalo (tempo inicial e tempo final), mas para rótulos temporais pontuais o tempo inicial é igual ao tempo final;
- Atributos Variáveis: uma tabela de atributos relacionada a dados do tipo objetos mutáveis. É preciso **identificar quais o campo possui a identificação única do objeto no banco (e que faz a ligação com sua geometria), qual campo identifica de maneira única cada versão, ou instância de atributos associada ao objeto, e quais campos possuem a informação temporal que determina o intervalo de validade dessa instância.**

O modelo de armazenamento dos objetos em movimento ainda está sendo desenvolvido na TerraLib, uma vez que esse é o caso mais complexo. Além de se considerar a variação no tempo associado aos atributos dos objetos é preciso considerar a variação associada às suas geometrias. Cada geometria tem que armazenar um intervalo (ou instante) de validade e é necessário propor um mecanismo de associação,

em um determinado instante ou intervalo, entre a geometria e os atributos do objeto.

Nas classes da TerraLib o modelo de tratamento de dados espaço-temporais está distribuído nas classes `TeSTInstance`, `TeSTElement`, `TeSTElementSet` e `TeQuerier`.

A classe `TeSTInstance` da representa uma instância, ou versão, no tempo de um elemento ou objeto geográfico. Uma instância contém o identificador do elemento ou objeto ao qual pertence, um intervalo de tempo de validade, suas geometrias e seus de atributos. Todas as instâncias de um mesmo elemento ou objeto geográfico são representadas pela classe `TeSTElement`. A classe `TeQuerier` implementa um mecanismo flexível de consulta ao banco de dados e recuperação de objetos e suas instâncias, dinâmicos ou estático no tempo. No segundo tipo cada objeto tem apenas uma única instância.

```
// Recupera o layer de estações de coletas, ou armadilhas
TeLayer* armadilhas = new TeLayer("LAYER_ARMADILHAS");
db_->loadLayer(armadilhas);
// Preenche os parâmetros da consulta
// carregar todos os atributos dos objetos
bool loadAllAttributes = true;
// não carregar as geometrias dos objetos
bool loadGeometries = false;
TeQuerierParams                    querierParams(loadGeometries,
    loadAllAttributes);

// carregar as instâncias dos objetos do layer
querierParams.setParams(armadilhas);

// Cria uma consulta a partir dos parâmetros
// Carrega as instâncias de objetos do banco que atendem aos
critérios da consulta
TeQuerier  querier(querierParams);
querier.loadInstances();
```

```
// Retorna a lista dos atributos descritivos carregados e
mostra seus nomes
unsigned int i;

TeAttributeList attrList = querier.getAttrList();

for (i=0; i<attrList.size(); ++i)
  cout << attrList[i].rep_.name_ << endl;
// Percorre as instâncias selecionadas uma a uma
TeSTInstance sti;
while(querier.fetchInstance(sti)) {
  cout << " Object: " << sti.objectId() << endl << endl;
   // Mostra o nome e valor de cada atributo da instância
  TePropertyVector vec = sti.getPropertyVector();
  for (i=0; i<vec.size(); ++i) {
     string  attrName = vec[i].attr_.rep_.name_;
     string  attrValue = vec[i].value_;
     cout << attrName  << " : " << attrValue << endl;
   }
}
```

Exemplo 12.23 – Aplicação de um `TeQuerier` para recuperar as instâncias dos objetos de um *layer*.

A classe `TeQuerier` contém uma série de parâmetros que definem quais instâncias de quais objetos e quais atributos desses objetos devem ser recuperados. Uma vez definidos os parâmetros do `TeQuerier` e aplicada a seleção, as aplicações podem processar as instâncias retornadas uma a uma. O Exemplo 12.23 mostra como criar um `TeQuerier` para recuperar todas os objetos de um *layer* e processar todas as instâncias desses objetos.

O *layer* utilizado no exemplo agrega objetos armadilhas para a coleta de ovos de um determinado inseto, causador de uma doença. A cada semana as armadilhas são visitadas e o número de ovos na armadilha e outros parâmetros da são observados e inseridos no banco. Esse dado é do tipo geometrias fixas e atributos variáveis no tempo. A armadilha não muda, mas seu atributo quantidade de ovos varia a cada semana.

Um `TeQuerier` também pode carregar todos objetos definidos em um tema, lembrando que um tema pode definir um subconjunto dos objetos de um *layer*. A seleção expressa no `TeQuerier` vai levar em conta as restrições definidas no tema sendo recuperado. O Exemplo 12.24 mostra uma consulta para recuperar todas as instâncias, atributos e geometrias dos objetos pertencentes às armadilhas do tipo A.

```
// Recupera o tema com somente as armadilhas do tipo A
TeTheme* armadilhasTipoA = new TeTheme("Armadilhas_A");
db_->loadTheme(bairros);
// Preenche os parâmetros da consulta
// carregar todos os atributos dos objetos
bool loadAllAtt = true;
// carregar as geometrias dos objetos
bool loadGeom = true;
// carregar as instâncias dos objetos do tema
TeQuerierParams querierParams(loadGeom, loadAllAttr);
querierParams.setParams(armadilhasTipoA);

// Cria uma consulta e carrega as instâncias
// de objetos do banco que atendem aos critérios da consulta
TeQuerier  querier(querierParams);
querier.loadInstances();
// Percorre as instâncias selecionadas uma a uma
TeSTInstance sti;
while (querier.fetchInstance(sti)) {
  if (sti.hasPoints()) {
     TePointSet ponSet;
     sti.getGeometry(ponSet);
     for (unsigned int i=0; i<ponSet.size(); ++i) {
        cout << " Point : " << ponSet[i].location().x() << ",
";
         cout << ponSet[i].location().y() << endl;
    }
  }
```

```
}
```

Exemplo 12.24 – Aplicação de um `TeQuerier` para recuperar as instâncias dos objetos de um tema.

O Exemplo 12.25 mostra como incluir uma restrição espacial nos parâmetros da consulta a ser expressa no `TeQuerier`, mostra também como selecionar apenas alguns atributos dos objetos. O exemplo seleciona somente as instâncias das armadilhas que estão dentro do polígono que delimita certo bairro, e para cada instância de armadilha seleciona somente o atributo 'nro_ovos'.

```
TePolygonSet limiteBairro;
// Preenche os parâmetros da consulta
vector<string> attributes;
attributes.push_back ("ARMADILHAS. NRO_OVOS");
bool loadGeom = true;      //   carregar   as   geometrias   dos
objetos
TeQuerierParams querierParams(loadGeom, attributes);
querierParams.setParams(armadilhas); // carregar as instâncias
dos objetos do tema
querierParams.setSpatialRest(limiteBairro);
```

Exemplo 12.25 – Definição de uma consulta com restrição espacial e com seleção de somente um atributo.

A classe `TeQuerier` também pode expressar agrupamentos de valores de atributos de as instâncias por objeto, por restrição espacial ou ainda por unidades (chronons) de tempo. O Exemplo 12.16 mostra como agrupar por armadilha todas as instâncias do seu atributo 'nro_ovos' através de uma operação de soma.

```
// Cria uma definição de agrupamento do atributo 'nro_ovos'
// através da operação de soma
TeGroupingAttr attributes;
pair<TeAttributeRep,TeStatisticType>
     attrl(TeAttributeRep("ARMADILHAS.NRO_OVOS"), TeSUM);
attributes.insert(attrl);
// Opção 1: Consulta de instâncias agrupadas por objeto
// Resultado vai ser uma única instância por objeto
```

```
TeQuerierParams querierParams(loadGeom, attributes);
querierParams.setParams(armadilhas);
```

Exemplo 12.26 – Consulta de instâncias agrupadas por objetos.

Os Exemplo 12.25 e 12.26 mostram como usar a classe `TeQuerier` para expressar uma seleção de objetos e suas instâncias no tempo. Uma característica importante é que através desse mecanismo não é necessário armazenar em memória todas as instâncias selecionadas, já que essas podem ser processadas e descartadas uma a uma. Cada instância é carregada para a memória somente quando é requisita através do método `TeQuerier::fetchInstance`. Uma alternativa para carregar para a memória todas as instâncias de todos os objetos de uma seleção é através da classe `TeSTElementSet`. Essa classe funciona como uma estrutura de agregação que armazena em memória as instâncias dos objetos com suas geometrias e atributos descritivos, fornecendo mecanismos para o seu percorrimento. A TerraLib fornece algumas funções de preenchimento da estrutura `TeSTElementSet` como mostramos no Exemplo 12.27.

```
// Recupera o layer de estações de coletas
TeLayer* armadilhas = new TeLayer("LAYER_ARMADILHAS");
db_->loadLayer(armadilhas);
// Cria um STElementSet para os objetos do layer de armadilhas
TeSTElementSet steSet(armadilhas);

// Carrega STElementSet carregando cada instância com todos os
seus atributos e sua geometria
TeSTOSetBuildDB(&steSet, true, true);
cout << "Número de elementos no conjunto: " <<
steSet.numElements() << endl;

// Percorre elementos e suas intâncias através de iteradores
TeSTElementSet::iterator it = steSet.begin();
while ( it != steSet.end()) {
  TeSTInstance st = (*it);
  cout << " Id do objeto :  " << st.objectId() << endl;
  if (sti.hasPoints()) {
     TePointSet ponSet;
```

```
      sti.getGeometry(ponSet);
      for (unsigned int i=0; i<ponSet.size(); ++i) {

         cout << " Point : " << ponSet[i].location().x() << ",
";

          cout << ponSet[i].location().y() << endl;
     }
   }
   ++it;
}
```

Exemplo 12.27 – Criação de um `TeSTElementSet`.

Essa seção mostrou os mecanismos para tratar dados espaço-temporais da TerraLib. Esses mecanismos são baseados no registro da característica temporal das tabelas de atributos no banco de dados e expressão dos conceitos de objetos com uma identidade única e persistente ao longo do tempo e de instâncias no tempo de atributos e geometrias desses objetos.

12.13 Funções de processamento de dados geográficos

Essa Seção descreve resumidamente algumas das funções de processamento oferecidas pela TerraLib, implementadas usando as estruturas e mecanismos descritos nas seções anteriores.

Matriz de Proximidade Generalizada: e uma extensão da matriz de proximidade usada em muitos métodos de análise espacial (Souza et. al, 2003). A TerraLib oferece as funções para gerar matriz de proximidade seguindo diferentes critérios, como por exemplo adjacência ou distância mínima, e para ponderar e fatiar a matriz segundo algum atributo. Além disso, uma matriz de proximidade pode ser gerada a partir de um `TeSTElementSet`.

Algoritmos de análise espacial: incluem algoritmos para geração de superfícies de kernel, índices de correlação espacial como LISA, Moran e Bayesiano local e global. Para saber mais sobre análise espacial de dados geográficos consulte Bayley e Gatrell, 1995.

*Operações Geográfica*s: combinam temas para gerar novos *layers* gerando novos atributos e novas geometrias vetoriais. Essas funções incluem a agregação de objetos por atributos, por exemplo, a agregação

de municípios pelo atributo estado, gerando um *layer* de estados a partir de um *layer* de municípios. Incluem também funções de associação de dados por localização, por exemplo, atribuindo ao *layer* de municípios valores de atributos calculados a partir de todas as ocorrências de crime que acontecem dentro do município.

Geocodificação de endereços: implementa o processo de associar uma coordenada geográfica a um evento tendo por base um *layer* que contenha os endereços que se deseja encontrar. Deste modo, uma vez associado a uma localização, o endereço pode ser usado para visualização dos eventos sobre um mapa ou utilizado para análises.

Atualmente começam a ser integrados na TerraLib funções de processamento de imagens como segmentação e classificação, e interpolação de superfícies a partir de amostras.

Referências

BAILEY, T.; GATRELL, A. **Interactive Spatial Data Analysis.** London: Longman Scientific and Technical,1995.

CÂMARA, G.; SOUZA, R. C. M.; PEDROSA, B.; VINHAS, L.; MONTEIRO, A. M. V.; PAIVA, J. A.; CARVALHO, M. T.; GATTASS, M. TerraLib: Technology in Support of GIS Innovation. In: II **Simpósio Brasileiro em Geoinformática**, GeoInfo2000, 2000, São Paulo.

DAVIS, C.; CÂMARA, G. Arquitetura de Sistemas de Informação Geográfica. In: CÂMARA, G.; DAVIS, C.; MONTEIRO, A. M. V. (Org.). Introdução à ciência da geoinformação. São José dos Campos: INPE, out. 2001. cap. 3.

FERREIRA, K. R. **Interface para Operações Espaciais em Banco de Dados Geográficos**.São José dos Campos: INPE - Instituto Nacional de Pesquisas Espaciais, 2003. Dissertação de Mestrado, Computação Aplicada, 2003.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLÍSSIDES, J. **Design Patterns - Elemens of Reusable Object-Oriented Software**. Reading, MA: Addison-Wesley,1995. 395 p.

INPE-DPI. O aplicativo TerraView. Disponível em: <http://www.dpi.inpe.br/terraview>. Acesso em: maio 2005.

MySQL AB. **MySQL reference manual**. Disponível em: <http://www.mysql.org>.Acesso em: abril 2005.

Open GIS Consortium. Web Map Service Version 1.13. Disponível em: <http://www.opengeospatial.org>. Acesso em: maio 2005.

QUEIROZ, G. R. **Algoritmos Geométricos para Banco de Dados geográficos: da teoria à prática na TerraLib.**São José dos Campos, SP: INPE - Instituto Nacional de Pesquisas Espaciais, 2003. Dissertação de Mestrado, Computação Aplicada, 2003.

SHEKHAR, S.; CHAWLA, S. **Spatial Databases: A Tour.** Prentice Hall,2002.

SNYDER, J. P. **Map Projections – A Working Manual**. Washington, DC, USA: United States Government Printing Office,1987.

SOUZA, R. C.; CÂMARA, G.; AGUIAR, A. P. D. Modeling Spatial Relations by Generalized Proximity Matrices. In: V Simpósio Brasileiro de Geoinformática, 2003, Campos do Jordão. Anais do GeoInfo 2003. São José dos Campos : SBC, 2003.

STROUSTROUP, B. **C++ Programming Language - 3rd Edition.** Reading, MA: Addison-Wesley,1997. 911 p.

13 *Tratamento de dados matriciais na TerraLib*

Lúbia Vinhas
Ricardo Cartaxo Modesto de Souza

13.1 Introdução

Esse capítulo descreve a solução adotada na biblioteca TerraLib (Câmara et al., 2000) para o tratamento de dados matriciais incluindo a decodificação de diferentes formatos e o seu armazenamento e recuperação em bancos de dados objeto-relacionais.

Os bancos de dados geográficos ainda possuem um desafio: o tratamento eficiente de dados matriciais, mais especificamente de dados de imagens de satélites. As imagens de sensoriamento remoto são uma das mais procuradas fontes de informação espacial para pesquisadores interessados em fenômenos geográficos de larga escala. A variedade de resoluções espectrais e espaciais das imagens de sensoriamento remoto é grande, variando desde as imagens pancromáticas do satélite IKONOS, com resoluções espaciais de um metro, até as imagens polarimétricas de radar que em farão parte da nova geração de satélites RADARSAT e JERS. Os recentes avanços da tecnologia de sensoriamento remoto, com o desenvolvimento de novas gerações de sensores, têm trazido consideráveis avanços nas aplicações de monitoramento ambiental e de planejamento urbano (Moreira, 2004).

A construção de bancos de dados geográficos capazes de manipular imagens de satélites tem sido amplamente estudado na literatura e a abordagem principal tem sido o desenvolvimento de servidores de dados especializados, como é o caso do PARADISE (Patel, Yu et al. 1997) e do RASDAMAN (Reiner, Hahn et al. 2002). A vantagem principal dessa abordagem é a capacidade de melhora de desempenho no caso de

grandes bases de dados de imagens. A principal desvantagem dessa abordagem é a necessidade de um servidor específico o que sobrecarrega o SIG – Sistemas de Informação Geográfica.

Além das imagens de satélite os SIG também devem ser capazes de tratar outros tipos de geo-campos, cuja representação computacional mais adequada é a matricial. Esses dados incluem grades numéricas que representam fenômenos qualitativos contínuos, como superfícies de fenômenos físicos (por exemplo, altimetria), ou sociais (por exemplo, exclusão social). A Figura 13.1 mostra dois exemplos de dados geográficos diferentes com representação matricial: uma foto aérea e uma superfície de altimetria.

Figura 13.1 – Exemplos de dados matriciais.

Os SIG's precisam trabalhar de maneira integrada com dados matriciais de diferentes naturezas e na mesma base de dados geográfica, que pode conter também geo-objetos com representação vetorial.

A implementação de um sistema de manipulação de dados matriciais, como grades e imagens de sensoriamento remoto, em um SGDB-OR, juntamente com funções para tratar outros tipos de dado espaciais é de grande importância para os SIG, e traz vantagens sobre sistemas especializados para construir bases de dados de imagens.

A infra-estrutura objeto-relacional pode ser usada para construir um sistema de indexação espacial, o que permite a consulta e recuperação de dados matriciais integrada à interface genérica de manipulação de dados matriciais. Através da indexação adequada, algoritmos de compressão e sistemas de *cache* um desempenho satisfatório pode ser conseguido, para os SIG's manipulando bases de dados compostas de representações matriciais e vetoriais.

13.2 Características dos dados matriciais

Dados matriciais em TerraLib ser entendidos como qualquer dado representado em uma estrutura de matriz retangular com M linhas × N colunas. Exemplos desse tipo de dado são grades regulares com valores de uma determinada grandeza (como uma grade de altimetria) ou imagens de sensoriamento remoto.

Dados matriciais podem possuir outras dimensões além das duas representadas no plano cartesiano das linhas e colunas. Ou seja, a cada elemento da matriz podem estar associados um ou mais valores. Por exemplo, as imagens multi-dimensionais de sensoriamento remoto possuem em cada elemento, ou *pixel*, da imagem o valor relativo a uma banda espectral do instrumento imageador que obteve a imagem. Por esse ser o caso mais típico de dados matriciais multi-dimensionais, em TerraLib dizemos que um dado matricial possui M linhas × M colunas × B bandas, como representado na Figura 13.2.

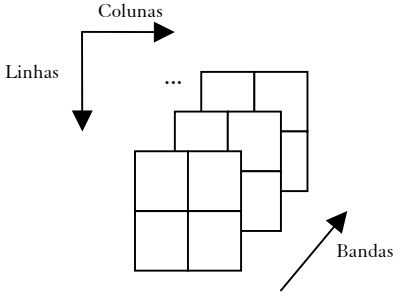


Figura 13.2 – Representação matricial multi-dimensional.

Cada posição da representação matricial é chamada de elemento (equivalente a um *pixel*, no caso de imagens) e costuma ser ter sua posição identificada por um par (linha × coluna), onde a posição (0,0) equivale à linha mais superior e a coluna mais à esquerda da matriz. Cada elemento, em uma dimensão, pode conter um valor dentro de um determinado intervalo de valores possíveis. Esse intervalo dependente do tamanho computacional (ou digital) reservado para seu armazenamento

e todos os elementos possuem o mesmo tamanho. Alguns tamanhos possíveis para cada elemento são:

- Unsigned Char: ocupa 8 bits por elemento. Pode armazenar valores entre 0 a 255;
- Short: ocupa 16 bits por elemento. Pode armazenar valores entre – 32.68 to 32.67
- Float: ocupa 32 bits por elemento. Pode armazenar valores entre 3.4E-38 a 3.4E+38 (7 dígitos);
- Double: ocupa 64 bits por elemento. Pode armazenar valores entre 1.7E- 308 a 1.7E+308 (15 dígitos).

No caso das imagens de sensoriamento remoto o tamanho do elemento relaciona-se com a resolução espectral da imagem e no caso de outros tipos de dados esse valor relaciona-se com a precisão do dado que está sendo representado.

Computacionalmente, a representação matricial não é esparsa, ou seja, todo elemento da matriz possui um valor. Porém muitas vezes é necessário ser possível a criação de uma representação esparsa, ou seja, uma representação onde em alguns pontos da matriz não existe um valor válido. Isso pode ser feito através da escolha de um valor como indicativo da ausência de dado naquele elemento da matriz. Esse valor deve pertencer ao intervalo de valores válidos para os elementos da matriz e é chamado de valor *dummy* ou "*nodata*". A Figura 13.3 (a) mostra um dado matricial, com 3 linhas × 3 colunas × 1 banda. Se o valor 0 for considerado como valor *dummy*, podemos dizer que nas posições (0,1) e (2,2) não existem valores válidos.

Outra característica dos dados matriciais e que os valores em cada elemento podem representar índices para uma tabela de combinações de valores, como no caso típico de imagens do tipo paleta. O valor de cada elemento é uma chave para uma tabela que contém uma tripla de valores R, G e B como mostrado na Figura 13.3(b).

1	**0**	**2**
1	**2**	**3**
2	**1**	**0**

(a) Valores Dummy

Chave	R	G	B
1	255	0	0
2	0	255	0
3	0	0	255

(b) Tabela de cores

Figura 13.3 – Característica da representação matricial.

Finalmente, as representações matriciais são caracterizadas pela resolução de cada elemento da grade, ou seja, a extensão nas direções horizontal e vertical de cada elemento da matriz quando mapeado para a área da superfície da Terra que representa. No caso das imagens de satélite esses parâmetros equivalem à resolução espacial da imagem.

A seção seguinte descreve os mecanismo de tratamento de dados matriciais de TerraLib.

13.3 Tratamento de dados matriciais em TerraLib

Imagens e/ou quaisquer outros tipos de dados com representação matricial são manipulados em TerraLib por três classes básicas:

- Uma classe genérica chamada TeRaster;
- Uma estrutura de dados que representa todos os parâmetros que caracterizam um dado matricial TeRasterParams;
- Uma classe genérica de decodificação de formatos e acesso a dispositivos de armazenamento de dados matriciais chamada TeDecoder.

13.3.1 A classe TeRaster

A classe `TeRaster` é uma interface genérica de acesso aos elementos de um dado matricial. Seus dois métodos básicos são: `setElement` e `getElement`, os quais inserem e recuperam, respectivamente, valores de cada elemento da representação matricial como um valor real de dupla precisão. O Exemplo 13.1 mostra como criar uma representação matricial em memória através da classe `TeRaster`. A representação matricial tem 10

linhas, 10 colunas, 2 bandas e cada elemento (em cada banda) pode conter um valor do tipo *unsigned char*.

```
TeRaster rasterMem(10,20,2,TeUNSIGNEDCHAR);
```

Exemplo 13.1– Criação de uma representação matricial em memória.

Antes que o dado esteja disponível para ser manipulado, todas as inicializações necessárias, como alocações de memória, devem ser executadas e estão concentradas no médodo `TeRaster::init`. Após a chamada desse método cada instância da classe `TeRaster` possui um valor de estado que indica quais as operações (leitura, escrita) podem ser executadas sobre ele. O Exemplo 13.2 mostra a chamada e teste de estado na criação de um dado matricial através da classe `TeRaster`.

```
// 1) cria um objeto raster
TeRaster rasterMem(10,20,2,TeUNSIGNEDCHAR);
// 2) faz inicializações
rasterMem.init();
// 3) verifica resultado
if (rasterMem.status() == TeNOTREADY)
     return false;
```

Exemplo 13.2 – Inicialização de um dado matricial.

13.3.2 A classe TeRasterParams

Um dado matricial é caracterizado por uma série de parâmetros que podem ser divididos nas seguintes categorias:

- *Dimensão*: número de linhas, número de colunas e número de bandas;
- *Geográficos*: projeção cartográfica, resolução horizontal, resolução vertical e menor retângulo envolvente de todo o dado. Como cada elemento de um dado matricial tem área, podemos identificar dois tipos de retângulo envolvente: o que passa pelo centro dos elementos das bordas (chamamos de box) e o que passa pelos extremos dos elementos das bordas da matriz (chamamos de bounding box) como representado na Figura 13.4;

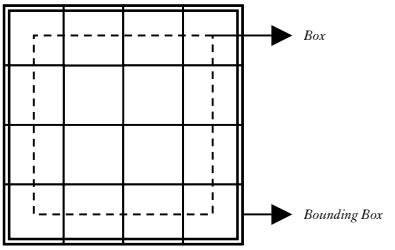


Figura 13.4 – Diferença entre box e bounding box.

- *Características dos elementos*: tamanho digital de cada valor de um elemento e interpretação do valor (paleta ou não);
- *Armazenamento*: nome de arquivo ou tabela de banco de dados onde está armazenado, tipo acesso permitido;
- *Forma de decodificação*: mecanismo usado para ler ou escrever um valor em um elemento.

A classe `TeRasterParams` contém um super conjunto de todos os possíveis parâmetros que descrevem um dado matricial, por isso cada objeto da classe `TeRaster` possui como um dos seus membros um objeto da classe `TeRasterParams`, que pode ser acessado através do método `TeRaster::params` como mostro o Exemplo 13.3.

```
void describe(TeRasterParams& par)
{
  cout << "Nome: " << par.fileName_ << endl;
  cout << "Modo de acesso: " << par.mode_ << endl;
  cout << "Nro bandas: " << par.nBands() << endl;
  cout << "Nro linhas: " << par.nlines_ << endl;
  cout << "Nro colunas: " << par.ncols_ << endl;
  cout << "Resolucao X: " << par.resx_ << endl;
  cout << "Resolucao Y: " << par.resy_ << endl;
  cout << "Identificacao do decodificador: " <<
par.decoderIdentifier_<< endl;
  cout << "Projecao: " << par.projection()->name() << endl;
  cout << "Retangulo envolvente: ";
  TeBox bb = par.boundingBox();
```

```
  cout << bb.x1_ << " " << bb.y1_ << " " << bb.x2_ << " " <<
bb.y2_ << endl;
}
TeRaster rasterMem(10,10,2,TeUNSIGNEDCHAR);
rasterMem.init();
if (rasterMem.status() == TeNOTREAD)
   return 0;
describe(rasterMem.params());
```

Exemplo 13.3 – Parâmetros do dado matricial.

Os diferentes formatos de armazenamento de dados matriciais podem trazer todos ou alguns dos parâmetros citados acima. Vejamos como exemplo:

- *Tiff*: dados matriciais nesses formatos trazem informações sobre o número de Linhas, o número de Colunas, o número de Bandas, o tipo dos elementos e a interpretação do elemento (se é do tipo paleta e sua tabela de cores);
- *GeoTiff*: é uma extensão do formato Tiff específica para o armazenamento de dados geográficos (grades numéricas ou imagens) e traz além das informações do Tiff, a projeção e retângulo envolvente da imagem, o tipo e a resolução espacial dos elementos, e a sua projeção cartográfica. O GeoTiff é o formato de intercâmbio de dados matriciais do consórcio OpenGIS (OPENGIS, 2005);
- *JPEG*: formato para armazenamento de imagens, traz informações sobre o número de Linhas, o número de Colunas e o número de Bandas. O formato JPEG permite armazenar apenas dados com 1 ou 3 bandas e o tipo dos elementos é sempre unsigned char;
- *RAW*: servem para armazenar grades ou imagens como arquivos binários sem nenhuma formatação e não trazem nenhuma informação sobre a representação matricial.

A estrutura `TeRasterParams` armazena também o nível de acesso ao dado, que pode assumir 3 opções:

1. 'r': somente leitura. Caso o dado não exista o método `TeRaster::init` falhará;
2. 'c': criação de um novo. Caso o dado já exista, será apagado e recriado. Será garantido acesso para leitura e escrita;
3. 'w': leitura e escrita. Pressupõe que o dado já exista. Caso contrário o método `TeRaster::init` falhará.

13.3.3 A classe `TeDecoder`

A idéia principal da classe `TeRaster` é poder representar dados matriciais que podem estar em memória, em arquivos em diferentes formatos (como TIFF ou JPEG), ou ainda dentro do um banco de dados geográfico. Com o objetivo de desacoplar a classe `TeRaster` das diferentes alternativas de armazenamento, as requisições de acesso aos elementos da representação matricial são tratadas pela classe `TeDecoder`. Esse mecanismo é um exemplo de uso do padrão de projeto "*Strategy*" (Gamma, Helm et al. 1995). A classe `TeDecoder` é abstrata e TerraLib já fornece um conjunto de decodificadores de imagens e grades. Novos decodificadores podem ser criados através da especialização da classe `TeDecoder` implementando os métodos `init`, `clear`, `setElement` e `getElement`.

A Tabela 13.1 mostra as características das classes de decodificadores já implementadas em TerraLib.

Tabela 13.1 – Decodificadores de Dados Matriciais implementados em TerraLib

Decodificadores	*Identificador*	*Extensão*	*Acesso*	*Descrição*
`TeDecoderMemory`	"MEM"	---	Criação Leitura Escrita	Dados matriciais como uma matriz de valores em memória
`TeDecoderJPEG`	"JPEG"	".jpg" ".jpeg"	Criação Leitura	Dados matriciais armazenados em arquivos no formato

			Escrita	JPEG. Carregam todo o dado para memória
`TeDecoderTIFF`	"TIFF"	".tif" ".tiff"	Criação Leitura	Dados matriciais armazenados em arquivos no formato GeoTIFF ou TIFF
`TeDecoderSPR`	"SPR"	".spr" ".txt"	Criação Leitura Escrita	Dados matriciais armazenados em arquivos no formato ASCII Spring. Carregam todo o dado para memória
`TeDecoderMemoryMap`	"MEMMAP"	".raw"	Criação Leitura Escrita	Dados matriciais armazenados em arquivos binários raw, limitados a aproximadamente 2Gb de tamanho
`TeDecoderFile`	"FILE"	".bin"	Criação Leitura Escrita	Dados matriciais armazenados em arquivos binários *raw*
`TeDecoderDatabase`	"DB"	---	Criação Leitura Escrita	Dados matriciais armazenados em um banco de dados TerraLib

A associação de um decodificador específico a um dado matricial pode ser feita explicitamente, mas também foi implementado um mecanismo mais flexível usando o padrão de projeto "*Factory*" (Gamma, Helm et al. 1995). Existe uma fábrica de decodificadores que constrói a instância de decodificador correta de acordo com um ou mais parâmetros do dado matricial. Um exemplo típico de uso desse mecanismo automático de seleção é associação de extensões de arquivos a decodificadores específicos. Quando possível, a associação de extensões de arquivos a decodificadores é o que permite que um objeto da classe `TeRaster` possa

ser instanciado a partir somente de um nome de arquivo. Essa associação pode ser modificada, conforme interesse do usuário da biblioteca, editando-se a função `TeInitRasterDecoders`. O Exemplo 13.4 mostra o acesso a dois dados matricias, nesse caso arquivos de imagens em dois formatos diferentes, TIFF e JPEG, de forma transparente.

```
// Mostra os valores em um dado matricial
void showRaster(TeRaster& rst)
{
  int i,j,b;
  double val;              // lê e mostra valores
  for (b=0; b<rst.params().nBands();++b)  {
      cout << "\nValores na banda: " << b << \n;
      for (i=0; i<rst.params().nlines_; ++i) {
          for (j=0; j<rst.params().ncols_; ++j) {
            rst.getElement(j,i,val,b);
            cout << val << " ";  }
       cout << endl << endl;
      }
  }
int main()
{
  TeInitRasterDecoders();        // inicializa decodificadores
  TeRaster rasterTIF("d:/Dados/TIFF/brasM.tif");
  if (!rasterTIF.init())
       return 0;
  TeRaster rasterJPEG("d:/Dados/JPEG/brasilia.jpg");
  if (!rasterJPEG.init())
       return 0;
  cout << "Dados no arquivo TIFF: \n";
  showRaster(rasterTIF);
  cout << "\n\nDados no arquivo JPEG: \n";
  showRaster(rasterJPEG);
```

```
}
```

Exemplo 13.4 – Exemplo de uso da classe `TeRaster`.

A Figura 13.5 mostra o relacionamento entre as três classes principais da interface de manipulação de dados matriciais de TerraLib. A classe `TeRaster` é uma geometria como as geometrias vetoriais descritas no Capítulo 12. Cada objeto da classe `TeRaster` possui uma instância de um objeto da classe `TeDecoder`, e ambas possuem um objeto da classe `TeRasterParams`, ou seja, o decodificador tem uma versão dos parâmetros do dado matricial que decodifica.

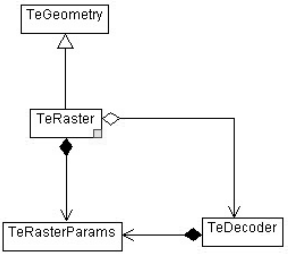


Figura 13.5 – Relacionamento entre as classes da interface de manipulação de dados matriciais.

Esse mecanismo facilita a escrita de algoritmos de manipulação de dados matriciais porque é flexível o bastante para fornecer um acesso uniforme a diferentes formatos de dados matriciais como mostra o Exemplo 13.5.

```
//! Copia os valores em um dado raster para outro
void copiaRaster(TeRaster& rstIn, TeRaster& rstOut)
{
  int i,j,b;
  double val;                              // lê valores
  for (b=0; b<rstIn.nBands();++b) {
      for (i=0; i<rstIn.nLines();++i) {
       for (j=0; j<rstIn.nCols();++j) {
          rstIn.getElement(j,i,val,b);
```

```
            rstOut.setElement(j,i,val,b);
        }
      }
   }
}

int main()
{
   TeInitRasterDecoders();  // inicializa decodificadores

   TeRaster rasterTIF("d:/Dados/TIFF/brasM.tif");
   if (!rasterTIF.init())
        return 1;

   TeRasterParams par = rasterTIF.params();     //  copia  todos
os parâmetros
   par.fileName_ = "d:/Dados/JPEG/bras.jpg";
   par.mode_ = 'c';
   par.decoderIdentifier_ = "JPEG";

   TeRaster rasterJPEG(par);
   if (!rasterJPEG.init())
        return 1;

   copiaRaster(rasterTIF, rasterJPEG);
   return 0;
}
```

Exemplo 13.5 – Criação de um dado matricial a partir de outro.

Pode se observar no exemplo acima que foram aproveitados todos os parâmetros da imagem de origem com exceção de três: o nome do arquivo destino, o modo de criação e a identificação do decodificador. O construtor do objeto `TeRaster` destino recebe uma instância de `TeRasterParams` e não mais um nome de arquivo. Assim a nova imagem será criada no formato JPEG no arquivo especificado. A função de cópia

continua usando apenas os métodos `TeRaster::setElement` e `TeRaster::getElement`.

Quando se deseja instanciar usar a classe `TeRaster` para acessar um dado cujo formato não inclui todas as informações necessárias, existe a possibilidade de se instanciar esse objeto a partir de uma estrutura de parâmetros e previamente preenchida. O Exemplo 13.6 mostra como acessar um dado matricial guardado como uma matriz binária em um arquivo *raw*. Como esse formato não possui nenhum tipo de descritor seus parâmetros devem ser explicitamente informados.

```
TeDatum sad69 = TeDatumFactory::make("SAD69");
TeUtm* proj = new TeUtm(sad69,-45.0*TeCDR);

TeRasterParams par;
par.fileName_ = "D:/Dados/RAW/grade.raw";
par.decoderIdentifier_ = "MEMMAP";
par.nBands(1);
par.mode_ = 'r';
par.setDummy(-9999.9);
par.setDataType(TeFLOAT);
par.topLeftResolutionSize(185350.0,824800.0,1000,500,10,10,tru
e);
par.projection(proj);

TeRaster grade(par);
if (!grade.init())
   return 1;
   //... usa dado
grade.clear();
```

Exemplo 13.6 – Criação de um objeto `TeRaster` a partir de parâmetros.

O método `TeRaster::clear` deve ser chamado ao final da utilização do dado matricial. Esse método libera memória ou quaisquer estruturas que foram alocadas na chamada do método `TeRaster::init`. De maneira geral a ordem de passos para se utilizar um objeto da classe `TeRaster` é a seguinte:

4. Definir parâmetros
5. Instanciar um objeto `TeRaster` a partir dos parâmetros
6. Chamar método `TeRaster::init`
7. Verificar se estado do objeto é o esperado
8. Utilizar objeto
9. Chamar método `TeRaster::clear`

Uma vez executado o método `TeRaster::init` não se deve alterar os parâmetros do dado matricial, pois as inicializações feitas utilizaram os valores então definidos. Qualquer modificação subseqüente pode invalidar a capacidade de decodificação do dado requerendo uma nova chamada ao método `TeRaster::init` como mostrado no Exemplo 13.7.

```
TeRasterParams par;
par.decoderIdentifier_ = "MEM";
par.nBands(1);
par.mode_ = 'c';
par.setDummy(255);
par.setDataType(TeUNSIGNEDCHAR);

TeRaster rMem(par);
if (!rMem.init())                // aloca espaço para 100
elementos
   return 0;

rMem.params().nlines_ = 200;     // altera número de elementos
rMem.params().ncols_ = 200;
rMem.setElement(199,199,0);       // ERRO!

rMem.init()                      //    CERTO!    Reinicializar
raster
rMem.setElement(199,199,0);      // antes de acessar novos
elementos
```

Exemplo 13.7 – Reinicializando um objeto `TeRaster`.

13.3.4 Incluindo informações geográficas

Para que o dado matricial seja geo-referenciado, ou seja, para que se possa identificar para cada linha e coluna uma localização sobre a superfície terrestre é preciso que seu os parâmetros contenham as informações sobre uma projeção cartográfica, as coordenadas de seu retângulo envolvente e a resolução espacial de cada elemento em unidades da projeção. Esses dados devem ser informados nos parâmetros ou obtidos do próprio formato de armazenamento e não devem ser alterados depois da inicialização sob pena de invalidar o geo-referenciamento. O Exemplo 13.8 mostra como incluir as informações geográficas em um dado matricial.

```
TeDatum sad69 = TeDatumFactory::make("SAD69");
TeUtm* proj = new TeUtm(sad69,-45.0*TeCDR);

TeRasterParams par;
par.decoderIdentifier_ = "MEM";
par.nBands(1);
par.mode_ = 'r';
par.setDummy(255);
par.setDataType(TeUNSIGNEDCHAR);

//define o retângulo envolvente do dado e sua resolução
par.lowerLeftResolutionSize(309695,7591957,20,20,3000,2000,tru
e);
// define a projeção do dado
par.projection(proj);

TeRaster grade(par);
if (!par.init())
   return 1;

for (i=0; i< grade.nLines();++i) {          //     retorna     a
coordenada geográfica
   for (j=0; j< grade.nCols();++j) {      // de cada elemento

        char mess[100];
```

```
        TeCoord2D xy = grade.index2Coord(TeCoord2D(j,i));
        sprintf(mess,"[%d,%d]:(%.2f,%.2f)
",i,j,xy.x(),xy.y());
        cout << mess;
    }
    cout << endl;
}
```

Exemplo 13.8 – Associação de parâmetros geográficos.

Os parâmetros geográficos de um dado matricial que permitem sua navegação em coordenadas de uma determinada projeção cartográfica (*bounding box*, *box*, número de linhas, colunas, resoluções horizontais e verticais) são correlacionados e devem ser coerentes. A classe `TeRasterParams` possui os métodos informar parte deles e automaticamente atualizar os outros (`lowerLeftResolutionSize`, `boxResolution`, `upperLeftResolutionSize`, `boundingBoxResolution`, etc.).

13.3.5 Remapeamento de dados raster

Um objeto da classe `TeRaster` propriamente inicializado pode ser parâmetro de funções que usam os seus métodos `getElement` e `setElement` como mostrado anteriormente. A classe `TeRasterRemap` é um exemplo de implementação de uma função que copia um dado matricial para outro, resolvendo diferenças em termos de geometria. Essas diferenças podem ser em relação ao número de linhas, colunas, resoluções diferentes ou retângulos envolventes diferentes, ou ainda, projeções diferentes. A Figura 13.6 mostra o fluxo de operações da classe `TeRasterRemap`.

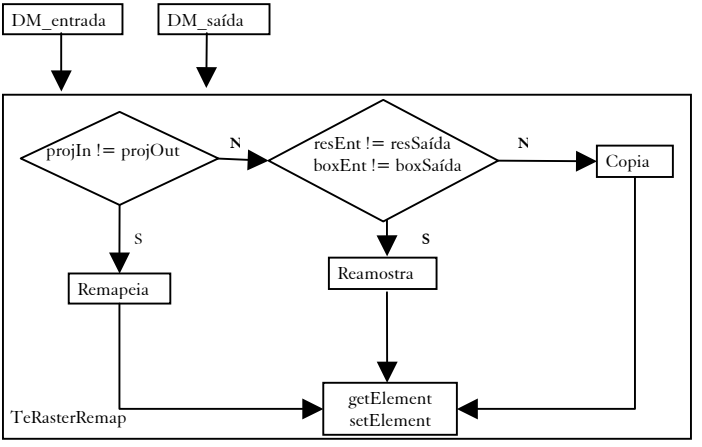


Figura 13.6 – A classe `TeRasterRemap`.

O Exemplo 13.9 mostra três casos de uso dessa função: para degradar a resolução de uma imagem, para recortar um pedaço de uma imagem e para remapear (trocar de projeção) uma imagem.

```
#include "TeRasterRemap.h"
int main()
{
  TeInitRasterDecoders();
  TeRaster rEntrada("d:/Dados/TIFF/brasM.tif");       // raster
em projeção UTM
  if (!rEntrada.init())
  {
     cout << "Erro obtendo dado de entrada!\n";
     return 1;
  }

   TeRasterParams parS = rEntrada.params();
   TeBox bb = parS.boundingBox();              // mantém box
   double resx = parS.resx_*2.0;               //    degrada    a
resolução espacial
```

```
   double resy = parS.resy_*2.0;

parS.boundingBoxResolution(bb.x1_,bb.y1_,bb.x2_,bb.y2_,resx,re
sy);
   parS.fileName_ = "d:/Dados/JPEG/subBras.jpg";
   parS.decoderIdentifier_ = "JPEG";                  // troca
formato de saída
   parS.mode_ = 'c';
  TeRaster rSaida1(parS);
   if (rSaida1.init())
   {
    TeRasterRemap  subamostra(&rEntrada,&rSaida1);     //  faz
reamostragem
    subamostra.apply();
   }
   parS = rEntrada.params();
   parS.fileName_ = "d:/Dados/JPEG/zoomBras.jpg";
   parS.decoderIdentifier_ = "JPEG";    // define box menor

parS.boundingBoxResolution(184000.0,8247000.0,194500.0,8248000
.0, parS.resx_,parS.resy_);
  parS.mode_ = 'c';

   TeRaster rSaida2(parS);
   if (rSaida2.init())
   {
    TeRasterRemap zoom(&rEntrada,&rSaida2);    // faz recorte
    zoom.apply();
  }

  parS = rEntrada.params();
  parS.fileName_ = "d:/Dados/JPEG/latlongBras.jpg";
  parS.decoderIdentifier_ = "JPEG";
  TeDatum sad69 = TeDatumFactory::make("SAD69");
  TeLatLong* latlong = new TeLatLong(sad69);
```

```
  parS.projection(latlong);
  parS.mode_ = 'c';

  TeRaster rSaida3(parS);
  if (rSaida3.init())
  {
       TeRasterRemap remap(&rEntrada,&rSaida3);  // remapeia
       remap.apply();
  }
}
```

Exemplo 13.9 – A classe `TeRasterRemap`.

A funcionalidade da classe `TeRasterRemap` pode ser usada em diversos contextos como a importação de dados para um banco e o apresentação desse dado em uma janela de visualização.

13.3.6 *Iteradores* sobre dados matriciais

Um grande número de algoritmos de processamento de dados geográficos não dependem de uma implementação particular de uma estrutura de dados, mas apenas de alguma propriedade semântica fundamental, como a capacidade de mover entre elementos de uma estrutura de agregação ou comparação entre dois elementos da estrutura. Por exemplo, o algoritmo para calcular o histograma de um dado geográfico não necessita conhecer se o dado está organizado como um conjunto de pontos, um conjunto de polígonos ou um conjunto de *pixels*. Tudo que é necessário é a que a estrutura forneça a capacidade de percorrer uma lista de elementos e obter um valor pra cada um deles. Esse paradigma de desenvolvimento é chamado de programação genérica (Austern 1998). Programação genérica visa o projeto de módulos interoperáveis baseados na separação entre algoritmos e estruturas de dados.

A classe `TeRaster` fornece *iteradores*, ou seja, ponteiros generalizados que permitem o percorrimento de uma representação matricial. Em sua versão mais simples os *iteradores* de dados matriciais começam na primeira linha e na primeira coluna do dado e a cada operação de avançar o *iterador* move-se para a próxima coluna até o fim da linha

quando então se move para a primeira coluna da linha seguinte. *Iteradores* são úteis, pois permitem que sejam escritos algoritmos em função dos *iteradores* e não de uma estrutura de dados em particular. O Exemplo 13.10 mostra como percorrer uma representação matricial através de *iteradores*.

```
TeRaster* img = new TeRaster("./dados/imagem.jpg");
TeRaster::iterator it = img.begin();
while (it != img.end())
{
  cout << (*it);
}
```

Exemplo 13.10 – Percorrimento de um dado matricial através de *iterador*.

Usando esse princípio, a classe `TeRaster` também fornece *iteradores* especializados para percorrimento dos elementos da representação matricial que estão delimitados por uma região de interesse (definida como um `TePolygon`) como mostra o Exemplo 13.11 .

```
TeRaster* img = new TeRaster("./dados/imagem.jpg");
TePolygon region;
//... cria a região de interesse
TeRaster::iteratorPoly itB = img.begin(region);
TeRaster::iteratorPoly itE = img.end(region);
while (it != itE)
{
  cout << (*it);
}
```

Exemplo 13.11 – Percorrimento de região de um dado matricial limitada a uma região.

As seções anteriores mostraram a interface de manipulação de dados matriciais oferecidas por TerraLib. A próxima seção trata das questões

relativas ao armazenamento e recuperação de dados matriciais em um banco de dados TerraLib.

13.4 Dados matriciais em bancos de dados TerraLib

Um banco TerraLib é composto de um conjunto de planos de informação, ou *layers*, onde cada *layer* é formado por um conjunto específico de dado geográfico (como um mapa de solos, uma imagem ou um mapa cadastral). Para o armazenamento desses dados em um SGBDOR – Sistema Gerenciador de Banco de Dados Objeto-Relacional, cada *layer* está associado com um conjunto de tabelas que armazenam a componente descritiva e espacial do dado. Em um banco TerraLib vai existir uma tabela diferente para cada tipo de geometria associada a informação contida no *layer* (pontos, linhas, polígonos, células e representações matriciais). O acesso ao dado espacial armazenado no SGDB-OR é feito através de uma interface que encapsula as diferenças internas de cada SGDB-OR. Essa interface mapeia os tipos espaciais TerraLib em tipos característicos de cada SGDB-OR usando os mecanismos nativos de indexação e otimização se disponíveis. As duas classes abstratas que definem a interface de acesso aos dados espaciais são: `TeDatabase` e `TeDatabasePortal` que permitem: (a) estabelecimento de conexões com um servidor de banco de dados; (b) execução de commandos SQL; (c) defingão de indíces; (d) definição de integridade referencial; (e) execução de consultas espaciais. Para saber mais sobre a interface de acesso a um banco de dados TerraLib o leitor deve recorrer ao capítulo 12.

O consórcio OpenGIS propõe uma especificação de componentes capazes de tratar geo-campos, chamados de *Grid Coverages*, mas, diferentemente de geometrias vetoriais não propõe um modelo de armazenamento desses dados em um banco de dados (OPENGIS, 2005). Trabalhos anteriores (Patel, Yu et al. 1997; Reiner, Hahn et al. 2002) mostram que a estratégia mais apropriada para trabalhar com grandes bases de dados de imagens é uma combinação de divisão por blocos, ou *tiling*, e a criação de uma pirâmide de multi-resolução. Essa combinação tem sido usada também em alguns servidores de mapas via web (DPI/INPE, 2005 e Barclay et al, 2000). O esquema de *tiling* é usado

como indexação espacial, de forma que quando se deseja recuperar uma parte da imagem apenas os blocos relevantes para essa parte serão recuperados. A pirâmide de multi-resolução é útil na visualização de imagens grandes e evita acessos desnecessários, nos níveis da pirâmide onde a resolução é degradada, menos blocos são recuperados. Essa abordagem foi adotada em TerraLib de forma integrada a todo o mecanismo descrito nas seções anteriores de forma que um banco TerraLib possa guardar quaisquer dados com uma representação computacional matricial e não somente imagens.

TerraLib trata uma representação matricial como mais uma tipo de geometria, sendo assim, um *layer* possui um conjunto de objetos cuja componente espacial pode ser uma representação matricial. Muitas vezes, um *layer* representa um geo-campo e nesse caso a geometria se refere ao *layer* e não a objetos discretos que agrega. A tabela de representações geométricas contém um registro para indicar a existência da representação matricial para o *layer*, esse registro, no campo **`geom_table`** informa o nome da tabela que contém as informações sobre a representação matricial de cada objeto do *layer*. Essa tabela de representações matriciais de um *layer* possui um conjunto de informações gerais sobre a representação: número de linhas e colunas, sua resolução espacial, a largura e altura (em número de elementos) dos blocos em que cada banda da representação espacial foi dividida e a indicação da possível técnica de compressão aplicada aos blocos. Finalmente, o campo **`spatial_data`**, aponta para a tabela de geometrias matriciais onde os blocos que formam a representação estão armazenados. A Figura 13.7 mostra os relacionamentos entre as tabelas que compõe o modelo de armazenamento de dados matriciais de TerraLib.

Na tabela de geometrias matriciais, cada registro possui a identificação única do bloco, seu retângulo envolvente e, em um campo binário longo, o bloco comprimido ou não. Todos os blocos de uma tabela de geometrias matriciais possuem a mesma largura e a mesma altura. Esse modelo de armazenamento de dados matriciais pode ser implementado em qualquer SGDB-OR uma vez que não faz uso de nenhum recurso especial do banco.

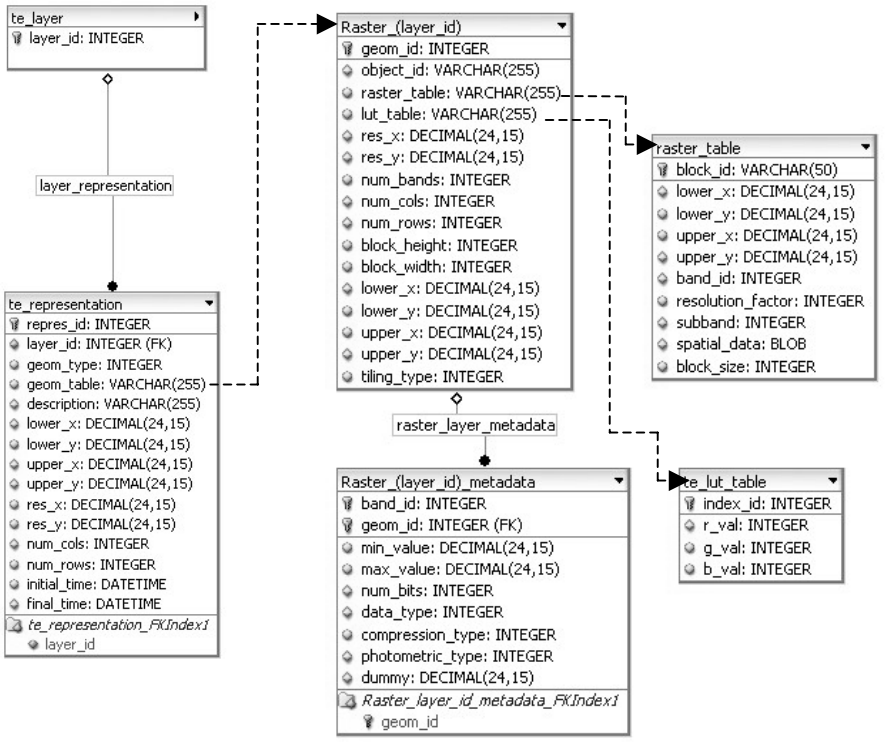


Figura 13.7 – Modelo de armazenamento de representações matriciais de TerraLib.

Poucos SGDB's com extensão espacial propõem um modelo de armazenamento de dados matriciais, um deles é a extensão espacial do Oracle como pode ser visto em ORACLE, 2005.

13.4.1 Modelo de divisão em blocos

O mecanismo de divisão do dado matricial em blocos visa a atender a dois requisitos específicos: (a) eficiência, conseguida pela capacidade de recuperação de parte do dado matricial; (b) flexibilidade, conseguida dada pela capacidade de acrescentar novos dados matriciais a uma representação já armazenada (por exemplo, ao se fazer o mosaico de duas cenas de imagens de satélite).

TerraLib disponibiliza duas formas de divisão de dados matriciais em blocos: não expansível e expansível. Essas duas formas diferentes de divisão implicam em diferentes maneiras de se mapear um elemento da representação matricial como um todo para um elemento dentro de bloco armazenado. A seguir explicamos as duas maneiras.

a) Não expansível

Significa que a divisão dos blocos é feita a partir da linha 0 e coluna 0 da representação matricial. Cada bloco contém $L \times A$ elementos, onde L é a largura do bloco e A a sua altura. Cada bloco é identificado pela sua ordem dentro da divisão total da representação, como mostra a Figura 13.8.

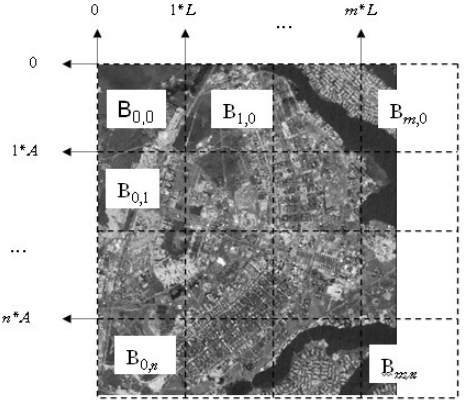


Figura 13.8 – Divisão por blocos de maneira não expansível.

Um elemento que está posição *(j, i)* da representação original está no bloco $B_{m,n}$ tal que $m = \text{INT}(i/L)$, $n = \text{INT}(j/A)$, onde INT(x) representa a parte inteira de x.

Observa-se na figura que a última linha de blocos e a última coluna de blocos ($B_{...,n}$ e $B_{m,..}$) podem não estar completos, ou seja, não existem valores na representação matricial para preencher todo o bloco. Isso acontece se o número de linha da representação não é múltiplo de A ou o

número de colunas da representação não é múltiplo de *L*. Esses elementos do bloco são preenchidos com o valor *dummy*.

Ao se escolher a opção de divisão por blocos não expansível, não será possível acrescentar, ou *mosaicar*, posteriormente novos dados matriciais ao dado armazenado.

b) Expansível

Significa que a divisão dos blocos não é feita na referência das linhas e colunas, mas sim a partir posições arbitrárias calculadas de acordo com as coordenadas geográficas do dado matricial. Isso permite que novos dados matriciais possam ser acrescentados a uma representação matricial já armazenada, desde que seus parâmetros geográficos sejam coerentes. A identificação de cada bloco é dada por seu posicionamento dentro de uma divisão da área geográfica referente ao dado matricial em blocos de tamanho *L*ResX* e *A*ResY*, onde *ResX* e *ResY* são as resoluções espaciais do dado matricial, iniciando na posição 0,0 do sistema de referência geográfica do dado. O esquema de divisão de blocos de maneira expansível é ilustrado na Figura 13.9. É importante notar que na divisão expansível os limites dos blocos estão em uma referência geográfica e não relacionada à linha e coluna de uma matriz. Isso é o que permite áreas de intersecção geográfica entre dois dados arquivos de imagens, por exemplo, sejam armazenados no mesmo bloco, como destacado na figura.

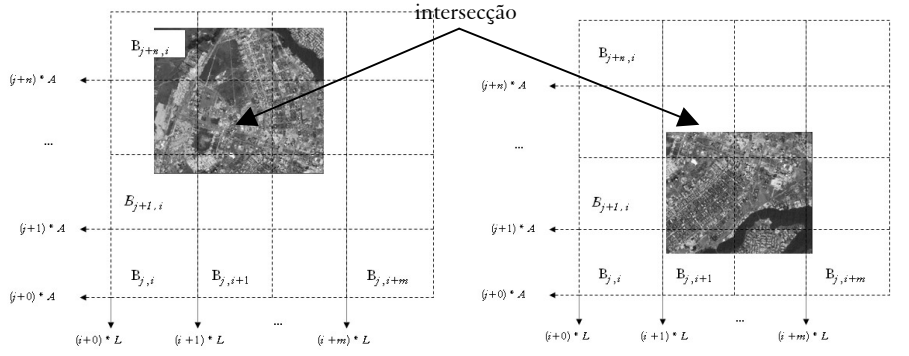


Figura 13.9 – Divisão por blocos de maneira expansível.

Um elemento que está posição *(j, i)* da representação original está no bloco $B_{m,n}$ tal que $m = \text{INT}(x/LG)$ e $n = \text{INT}(y/AG)$ onde (x,y) é a coordenada geográfica de (j,i), $LG = L*resX$ e $AG = A*resY$.

Assim como no caso da divisão não expansível alguns blocos ficam incompletos (observar a Figura 13.9) e são preenchidos com o valor *dummy*.

Através do retângulo envolvente de cada bloco, é possível saber seu retângulo envolvente em termos de linhas e colunas da representação matricial total e assim encontrar a posição do elemento procurado dentro do bloco.

O Exemplo 13.12 mostra o código escrito para executar importar um arquivo de imagens para um banco TerraLib criando um novo *layer*. Depois um segundo arquivo é importado indicando que deve ser armazenado na mesma representação e, portanto executando o mosaico. A Figura 13.10 mostra o resultado visual da operação.

```
TeRaster brasl(“./dados/Brasilial.tif”);
TeRaster brasl(“./dados/Brasilial.tif”);
TeLayer *msc = new TeLayer("Brasilia", db,
brasl.projection());
TeImportRaster(msc,brasl,128,128,TeJPEG,"",255,TeExpansible));
TeImportRaster(msc,bras2,128,128,TeJPEG,"",255,TeExpansible))
```

Exemplo 13.12 – Mosaico de dois arquivos de imagens para uma mesma representação matricial.

Os parâmetros da rotina de importação indicam qual o tamanho dos blocos, qual o algoritmo de compressão, qual o valor a ser usado como *dummy* e qual o método de divisão por blocos.

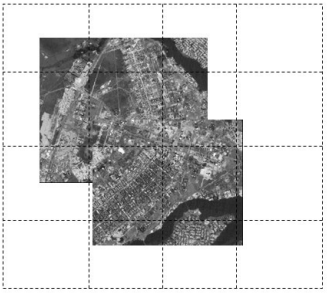


Figura 13.10 – Resultado do mosaico de dois arquivos de imagens.

Os mecanismos de divisão por blocos e de mosaico são válidos tanto para imagens quanto para grades numéricas.

13.4.2 A estrutura de multi-resolução

As operações de visualização de dados matriciais grandes são limitadas pelos dispositivos de saída. Por exemplo, considere uma imagem com 20000 × 20000 *pixels* sendo mostrada em um *display* de 1000 × 1000 pontos. O usuário controlando a aplicação de visualização pode começar obtendo uma visão geral da imagem e depois prosseguir ampliando áreas menores de seu interesse. Em cada situação, ou seja, em cada ampliação, não deveria ser necessário a recuperação simultânea de toda a imagem do banco de dados. Na situação ideal, a recuperação da imagem deveria ser da ordem de magnitude da capacidade do dispositivo de saída. Esse requisito pode ser atendido se a imagem armazenada no banco de dados esteja organizada em uma estrutura de multi-resolução. Combinada com o esquema de divisão por blocos, a pirâmide de multi-resolução armazena, em seu nível mais baixo, os blocos com a resolução total da imagem. Nos níveis mais altos são armazenados blocos com resolução degradada. Dessa forma a aplicação pode controlar, de acordo com a capacidade do dispositivo de saída, em qual nível recuperar os blocos de interesse.

Com essa motivação, TerraLib implementa um esquema pirâmide de multi-resolução. Tradicionalmente são criadas resoluções com fatores multiplicativos em potência de dois, de forma que um nível superior contenha ¼ do número de blocos necessários para se armazenar o dado

no nível inferior (ou seja, de melhor resolução) como mostrado na Figura 13.11.

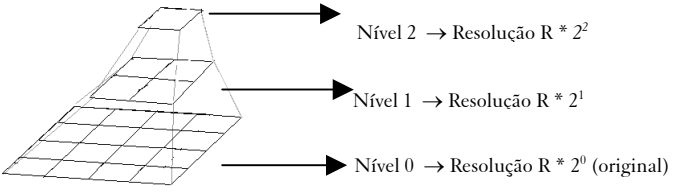


Figura 13.11 – Exemplo de pirâmide com resoluções degradas de fator 2.

TerraLib permite que a aplicação informe qual o fator de degradação, a cada nível, deseja aplicar para construir a pirâmide como mostra o Exemplo 13.13.

```
// importa resolução original res X= 30, res Y=30
 TeImportRaster(nativade,nat,512,512,TeJPEG,"obj1");
 TeBuildLowerResolution(nativade nat, 2, ,"obj1");
 // resX = resY = 60
 TeBuildLowerResolution(nativade nat, 4, ,"obj1");
// resX = resY = 120
 TeBuildLowerResolution(nativade nat, 8, ,"obj1");
// resX = resY = 240
```

Exemplo 13.13 – Importando resoluções extras.

As resoluções extras podem ser criadas nos dois modelos de divisão por blocos: expansível e não expansível. A resolução do bloco também é parte de sua identificação única, por isso os blocos com resoluções degradadas podem ser armazenados na mesma tabela que os blocos na resolução original.

13.4.3 Processamento de consultas e recuperação de dados

Assim como no caso das geometrias vetoriais (ver capítulo 12), existem vários níveis de acesso aos dados matriciais armazenados no banco de dados TerraLib. Através da classe `TeDatabase` e `TaDatabasePortal`, a aplicação pode submeter consultas SQL diretamente sobre a tabela de geometria matricial e pode consumir os registros retornados.

```
TeDatabase portal = db->getPortal();
// seleciona todos os blocos na resolução original
string sql = "SELECT * FROM RasterLayert_01_Raster" WHERE
resolution_factor = 1";
portal->query(sql);
while (portal->fetchRow())
{
  char* blob;
   long size;
   portal-> getRasterBlock(blob, size);
   // usa o blob retornado
}
```

Exemplo 13.14 – Consulta SQL sobre uma tabela de representação matricial.

O problema com essa abordagem é que a aplicação fica responsável por descompactar e decodificar o *blob*. Por isso, em um nível conceitual mais alto, a classe `TeRaster` possui um esquema de seleção dos blocos com um dado fator de resolução e que interceptam um certo retângulo envolvente. O resultado da seleção é consumido como se fosse um portal, porém o retorno é uma instância de decodificador já inicializado, que pode ser usado para construir uma representação matricial independente em memória como mostrado no Exemplo 13.15.

```
TeBox bb(183557, 8246310, 194987, 8258940);
int resFac = 2;
TeRasterParams parBlock;
// seleciona os blocos com um certo fator de resolução e que
// interceptam um determinado retângulo envolvente
if (raster->selectBlocks(bb, resFac,parBlock))
{
   // Processa cada bloco como um dado matricial independente
em memoria
  TeRaster* block = new TeRaster;
  TeDecoderMemory* decMem = new TeDecoderMemory(parBlock);
```

```
   decMem->init();
   do
   {
        flag = raster->fetchRasterBlock(decMem);
        block->setDecoder(decMem);
   } while (flag);
    raster->clearBlockSelection();
}
```

Exemplo 13.15 – Esquema de seleção de blocos de uma determinada representação matricial.

Aumentando um pouco mais o nível de abstração, através da classe `TeLayer` é possível se obter a sua representação matricial como uma instância da classe `TeRaster` que pode então acessar seus elementos individualmente através dos métodos `setElement` e `getElement`, ou através de *iteradores* como mostrado nos exemplos das seções anteriores. O Exemplo 13.16 mostra como pode ser feito esse acesso.

```
TeLayer img("Brasilia");
db->loadLayer(img);
TeRaster* bsbImg = img->raster();
//… acessa elementos individuais da imagem
double val;
bsbImg->getElement(0,0,val,0);
```

Exemplo 13.16 – Acessando uma representação matricial de um *layer*.

13.4.4 A classe `TeDecoderDatabase`

No Exemplo 13.16, internamente é criado um decodificador para um dado matricial armazenado em uma banco de dados TerraLib (representado pela classe `TeDecoderDatabase`), escondendo todos os detalhes de tabelas e compressões associados ao dado armazenado. O decodificador de dados matriciais em um banco de dados TerraLib levou em conta algumas questões de eficiência para permitir o acesso a elementos individuais da representação. Existe um mecanismo de *cache* de blocos em memória a fim de reduzir o número de acessos ao banco de dados. O identificador único de cada bloco é usado como chave no

armazenamento no banco de dados e também no sistema de *cache*. Cada vez que a aplicação requisita o valor de um elemento com as coordenadas (i,j) na banda b, em uma resolução r, se o bloco que contém o elemento não está na memória, é recuperado e colocado no sistema de *cache*. No próximo acesso a um elemento, o sistema inicialmente verifica se o bloco que o contém está no sistema de *cache*. Se estiver no o valor do elemento é recuperado diretamente da memória sem a necessidade de se acessar o banco novamente. O número máximo de blocos mantido no sistema de *cache* é controlado pela aplicação. Quando é necessário trazer um bloco para o *cache* e não existe mais espaço, o bloco menos recentemente usado é liberado do *cache*, caso tenha sido modificado seu conteúdo é atualizado no banco de dados. O sistema de cache é implementado na classe `TeDecoderVirtualMemory` da qual a classe `TeDecoderDatabase` é derivada. O sistema de troca de blocos no *cache* atual é simples, onde os blocos mais usados são mantidos por mais tempo. No entanto, esse é um ponto onde podem ser desenvolvidas outras estratégias mais adaptadas a cada caso.

A classe `TeDecoderDatabase` permite que dados matriciais armazenados em um banco de dados TerraLib possam ser usados da mesma maneira que dados matriciais armazenados em memória ou em arquivos em formatos proprietários. Juntamente com a classe `TeRasterRemap` aplicações como a importação de um dado matricial para o banco de dados ou a visualização de um dado matricial em um dispositivo de saída é facilmente implementada. A Figura 13.12 mostra o esquema usado para importar um dado matricial para um banco TerraLib.

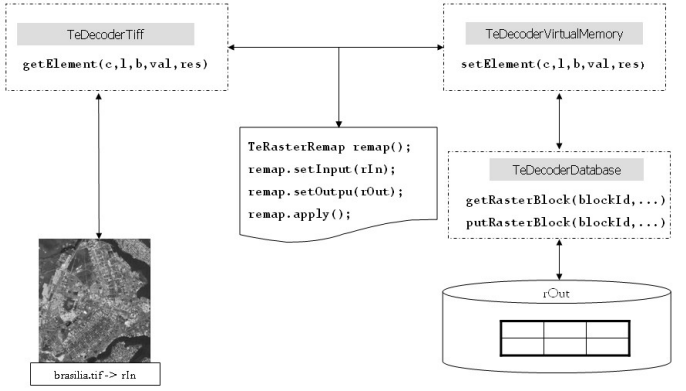


Figura 13.12 – Esquema da importação de um dado matricial para o banco.

13.5 Um exemplo de aplicação

Na TerraLib, o esquema de manipulação de dados matriciais foi funciona em todos os sistemas gerenciadores de banco de dados que implementam a interface `TeDatabase` como ORACLE, MySQL, PostgreSQL e Access. Um exemplo de aplicação é o SIG TerraView, que possui as funções para importar para um banco de dados TerraLib, visualizar e exportar dados matriciais em diversos formatos. A Figura 13.13 mostra um a tela do TerraView mostrando um *layer* com representação matricial, no caso uma imagem, sobreposta por uma dado vetorial, o limite dos estados da região norte.

A representação matricial do *layer* mostrado foi construída como um mosaico de duas imagens vindas de arquivos em formato GeoTiff. A primeira imagem possui 7020 × 7984 *pixels* e 3 bandas, segunda imagem 7414 × 8239 *pixels* e também 3 bandas. Os dois arquivos GeoTiff juntos ocupavam 335Mb.

As imagens foram divididas em blocos de 512 × 512 *pixels* e foram construídos cinco níveis de resolução, cada nível degradando a resolução do nível anterior de um fator de dois. Os blocos foram armazenados em um banco TerraLib armazenado no SGDB ACCESS, e comprimidos pelo formato JPEG com um nível de qualidade de 75%. A tabela de geometrias matriciais ficou com um total de 1500 blocos ocupando um

espaço de armazenamento de 472 Mb. Para visualizar a imagem total foram decodificados cinco blocos com um fator de resolução de 64 vezes a resolução original.

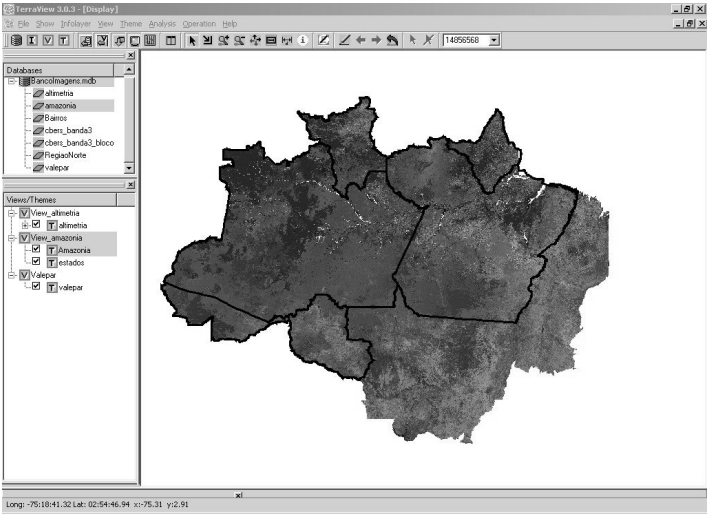


Figura 13.13 – Tela principal do TerraView.

A Figura 13.14 mostra o resultado da ampliação de aproximadamente um quarto da imagem e a aplicação precisou descomprimir 16 blocos com um fator de resolução de oito vezes a resolução original.

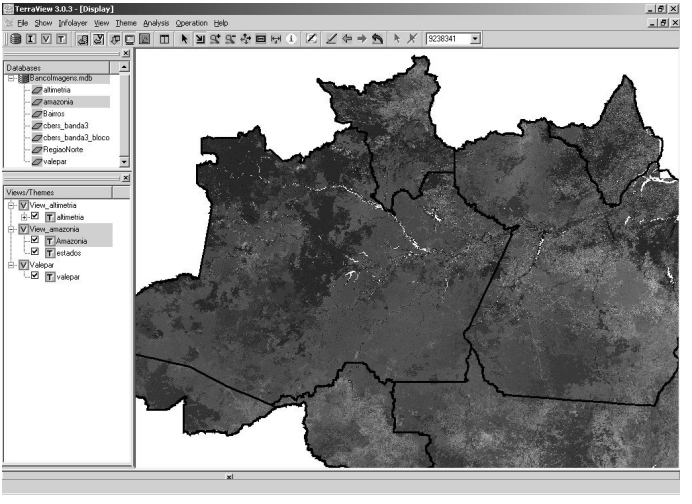


Figura 13.14 - Primeira ampliação.

Finalmente, a Figura 13.15 mostra uma ampliação detalhada, para a qual a aplicação descomprimiu seis blocos na resolução original.

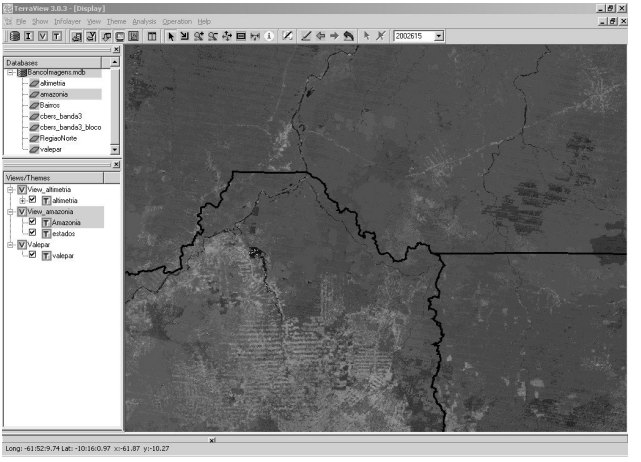


Figura 13.15 – Segunda ampliação.

A Figura 13.16 mostra a visualização de uma grade de altimetria, armazenada segundo o mesmo esquema do armazenamento da imagem. A grade é mostrada como se fosse uma imagem em tons de cinza onde o valor mais baixo da grade está mapeado para o preto e o valor mais alto está mapeado para o branco.

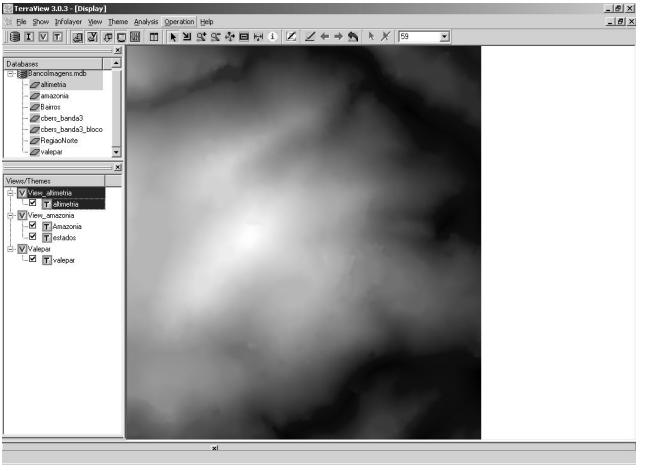


Figura 13.16 – Visualização da grade de altimetria.

Referências

AUSTERN, M. (1998). **Generic Programming and the STL : Using and Extending the C++ Standard Template Library**. Reading, MA, Addison-Wesley.

BARCLAY, T.; GRAY, J.; SLUTZ, D. Microsoft TerraServer: A Spatial Data Warehouse. In: **ACM SIGMOD International Conference on Management of Data**,2000,Dallas, Texas, USA. ACM Press, p. 307-318.

CÂMARA, G.; SOUZA, R. C. M.; PEDROSA, B.; VINHAS, L.; MONTEIRO, A. M. V.; PAIVA, J. A.; CARVALHO, M. T.; GATTASS, M. TerraLib: Technology in Support of GIS Innovation. In: II **Simpósio Brasileiro em Geoinformática**, GeoInfo2000, 2000, São Paulo.

GAMMA, E.; HELM, R.; JOHNSON, R.; VLÍSSIDES, J. **Design Patterns - Elemens of Reusable Object-Oriented Software.** Reading, MA: Addison-Wesley,1995. 395 p.

MOREIRA, M. A. **Fundamentos do Sensoriamento Remoto e Metodologias de Aplicação**. UFV, 2003.

DPI/INPE. **Mosaico do Brasil**. Disponível em:

<http://www.dpi.inpe.br/mosaico>. Acesso em: jan. 2005.

Open GIS Consortium. OpenGis Implementation Especification: Grid Coverages. Revision 1.0. Disponível em: <http://www.opengeospatial.org>. Acesso em: abril 2005.

ORACLE Corporation. Managing Geographic Raster Data Using GeoRaster. Disponível em <http://www.oracle.com>. Acesso em: abril 2005.

PATEL, J., J. YU, et al. (1997). Building a Scalable Geo-Spatial DBMS: Technology, Implementation, and Evaluation. SIGMOD Conference, Tucson, Arizona.

REINER, B., K. HAHN, et al. (2002). Hierarchical Storage Support and Management for Large-Scale Multidimensional Array Database Management Systems. **3th International Conference on Database and Expert Systems Applications (DEXA)**, Aix en Provence, France.

14 *Desenvolvimento de aplicativos com a TerraLib*

Marcelo Tílio de Carvalho
Mário de Sá Vera

14.1 Introdução

Este capítulo descreve o *TerraLib Development Kit - Tdk*, cujo objetivo principal é facilitar o desenvolvimento de aplicativos geográficos que utilizem a TerraLib.

A motivação do desenvolvimento do Tdk decorre de uma análise dos objetivos da TerraLib. Recorde do Capítulo 12 que um dos principais objetivos do projeto TerraLib (www.terralib.org) é oferecer suporte, através de uma biblioteca (modelos, estrutura de dados e algoritmos), para a pesquisa e o desenvolvimento de tecnologias inovadoras na área da ciência da Geoinformação (ver Capítulo 12). Um segundo objetivo importante é estabelecer um ambiente colaborativo para o desenvolvimento de novas ferramentas e aplicativos que componham uma nova geração de Sistemas de Informações Geográfica (SIGs).

Para que este objetivo seja atingido de modo satisfatório, precisamos tratar três questões :

- A complexidade intrínseca do código da TerraLib demanda um tempo grande de aprendizado e requer experiência de programação (ver Capítulo 12).
- A TerraLib não oferecer suporte direto para o desenvolvimento de alguns componentes presentes em um SIG básico. Em uma visão abrangente, pode-se indicar que este tipo de aplicação deve apresentar os seguintes componentes, apesar da sua implementação poder variar para cada sistema: interface gráfica-interativa com o usuário, entrada e integração de dados,

armazenamento e recuperação de dados (organizados sob a forma de um banco de dados geográficos), funções de consulta, funções de análise espacial e visualização e plotagem. Destes componentes, a TerraLib não oferece suporte direto para a interface gráfica-interativa com o usuário e para a visualização e plotagem.

- A possibilidade de interoperar dados com outras aplicações. Além de uma base de código aberto, uma comunidade de desenvolvimento de *software* deve poder utilizar padrões da indústria para troca de dados e serviços.

14.2 Motivação e estruturação de um SDK

14.2.1 Motivação para um SDK

Um SDK (*Software Development Kit*) é um conjunto de ferramentas que permite o desenvolvimento de aplicativos utilizando-se funcionalidades de um *software* ou de uma biblioteca referente a um domínio de aplicação específico. Um SDK é justificado quando queremos oferecer as funcionalidades de uma ferramenta, criada por especialistas em um domínio específico, para acelerar o desenvolvimento de aplicativos que dependam de funcionalidades deste domínio. Normalmente, um SDK oferece APIs (*Application Programming Interfaces*) ou *plugin*, além de utilitários que agilizam a programação, como ferramentas para ajudar na depuração e, documentação detalhada. Um exemplo de um SDK para SIGs são as extensões do ArcGIS da ESRI (www.esri.com/software/arcgis).

Uma API é um conjunto de definições que permitem acesso às funcionalidades e serviços de um determinado *software* ou biblioteca. Este conjunto pode ser disponibilizado como funções de uso comum (ex. funções para desenhar janelas e ícones), permitindo que programadores não precisem programar tudo do início. Uma boa API oferece um alto nível de abstração escondendo detalhes da implementação.

Um *plugin* é um pedaço de programa desenvolvido para oferecer uma funcionalidade específica. É um *mini-programa*, que roda embutido e utilizando a interface de um programa principal. Normalmente, os *plugins* possuem uma definição dos limites de suas funcionalidades Um exemplo de *plugin* é o SVG da Adobe (www.adobe.com/svg).

De uma forma resumida, as principais vantagens de um SDK são:

- Permitir que os aplicativos possam ser desenvolvidos em menos tempo e em menos passos.
- Oferecer facilidades para construção de aplicações sem a necessidade de muita experiência em programação.
- Oferecer suporte a várias linguagens de programação.
- Facilitar a formação e manutenção de uma comunidade. Através de um SDK, a comunidade pode contribuir com sua criatividade.

14.2.2 Estruturação de um SDK

Para poder oferecer as vantagens citadas acima, um SDK deve ser estruturado respeitando alguns requisitos:

- Modularidade – o acoplamento entre os componentes oferecidos por um SDK deve ser pequeno, para que o programador tenha liberdade para utilizar somente os componentes que ele necessite.
- Reuso – os componentes de um SDK devem ser extensíveis, para que o programador possa, com alguma adaptação, reutilizá-los, de acordo com as necessidades específicas da sua aplicação. Além disto, devem poder ser utilizados em diversos contextos e realizar diferentes tarefas.
- Interface com múltiplas linguagens de programação – é desejável que um SDK disponibilize APIs para mais de uma linguagem de programação. Para isto, O SDK deve ser pensado como uma especificação genérica, a partir da qual as APIs devem ser implementadas. Isto garante a compatibilidade entre as implementações, ao mesmo tempo que mantém a independência.

14.2.3 O TerraLib Development Kit

Conforme o exposto acima, fica evidente que seria desejável a TerraLib possuir um SDK. Deste modo, iniciamos o projeto Tdk (*TerraLib Development Kit*), cujo objetivo principal é facilitar o desenvolvimento de aplicativos geográficos que utilizem a TerraLib.

Para atingir este objetivo, projetamos o Tdk levando em consideração os seguintes requerimentos:

- API Simplificada – prover, para usuários que não sejam proficientes em programação com a TerraLib, uma API simplificada para acesso às suas funcionalidades mais comuns. O Tdk, no entanto, não oferece a mesma flexibilidade que o código da TerraLib, de forma que deve ser permitido o acesso direto à TerraLib.
- Arquitetura genérica – a arquitetura do Tdk não deve depender de nenhuma tecnologia específica (aberta ou proprietária). Idealmente o programador deve poder implementar a arquitetura proposta com a tecnologia mais apropriada ao contexto da sua aplicação (Web ou Desktop, JAVA ou C++). De uma forma simplista, a arquitetura do Tdk consiste em uma especificação de como implementar um ambiente de desenvolvimento para SIGs.
- Modelos reutilizáveis e extensíveis – sugerir modelos funcionais que sejam úteis para o desenvolvimento de SIGs (modelos para autenticação de usuários, acesso ao banco de dados através de múltiplas conexões, modelos para edição e impressão de mapas, entre outros). As soluções propostas pelos modelos sugeridos devem poder ser usadas isoladamente e poder ser estendidas, para acomodar funcionalidades específicas da aplicação dos usuários.
- Compatibilidade com padrões – oferecer uma interface para o acesso à TerraLib, compatível com os padrões publicados pelo Open GIS Consortium (OGC) (www.opengis.org). Isto permite que os programadores acostumados com o vocabulário e a arquitetura do OGC, utilizem as funcionalidades do Terralib no desenvolvimento de suas aplicações. Além disto, para promover a interoperabilidade com aplicativos desenvolvidos com a TerraLib, o Tdk deve oferecer serviços compatíveis com as especificações do OGC (como os serviços WMS, WFS e WCS, de publicação de dados na Web).

Com isto, esperamos resolver as três questões levantadas acima e facilitar o desenvolvimento de um ambiente colaborativo para o desenvolvimento de SIGs avançados.

14.3 Fundamentos conceituais da arquitetura

Desenvolvemos o Tdk como uma extensão modular da TerraLib, direcionado a oferecer suporte para a implementação de funcionalidades de aplicativos no domínio de SIG. Isto se reflete no modelo de dados utilizado, concebido a partir do modelo de dados original da TerraLib. Embora seja desenvolvido em cima da biblioteca TerraLib, o Tdk não restrinje o acesso direto às suas funcionalidades, conforme mostrado na Figura 14.1.

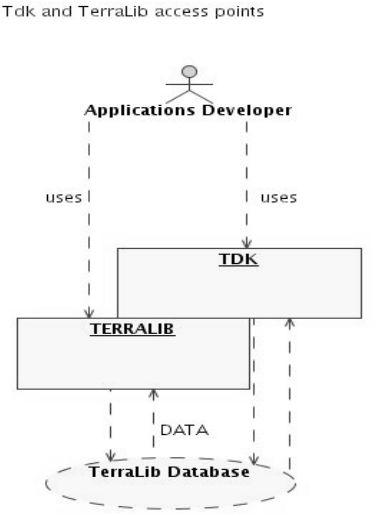


Figura 14.1 – O Tdk como extensão modular da TerraLib.

A arquitetura do Tdk especifica interfaces de programação (APIs) bem definidas e sugere modelos funcionais referentes à implementação de funcionalidades comuns a aplicativos gráfico-interativos de SIGs. Os elementos da arquitetura são descritos a seguir.

14.3.1 Modelo de dados Tdk

Definimos o modelo de dados conceitual do Tdk a partir do modelo original da TerraLib (tema, layer, view, etc. Ver Capítulo 12). O modelo do Tdk segue as especificações de um *composite* (Gamma et al., 1995) e pode, portanto, ser estendido para acomodar novos componentes, conforme mostrado na Figura 14.2.

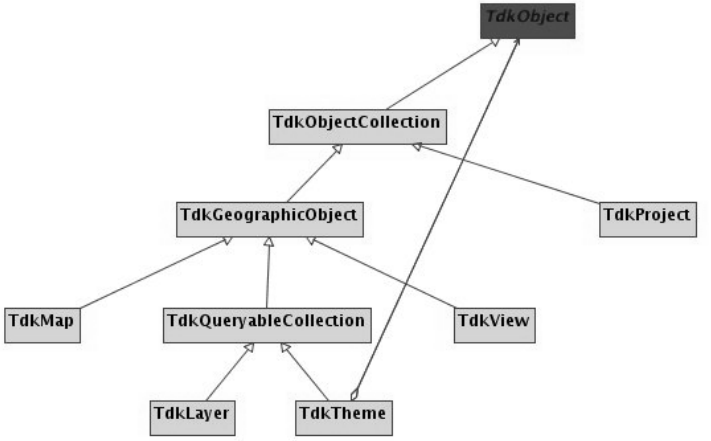


Figura 14.2 – Diagrama hierárquico com o modelo de dados Tdk.

O elemento que define um objeto Tdk (*TdkObject*) é a raiz do modelo e pode representar três tipos básicos de elementos:

- Geometria – Uma representação geométrica simples (um ponto, linha, polígono, imagem, ou qualquer outra representação geométrica).
- Objeto Simples – Um objeto geográfico composto por n geometrias em atributos.
- Coleção – Um conjunto de objetos simples ou de outras coleções.

Esses elementos são organizados hierarquicamente segundo as regras de que um objeto simples agrupa um conjunto de geometrias e uma coleção agrupa objetos simples ou outras coleções. Uma vez definida uma interface única e comum a objetos e coleções, torna-se possível o tratamento uniforme entre estes elementos.

14.3.2 Interfaces de programação

Como concebemos o Tdk dentro do paradigma de orientação a objetos (Rumbaugh et al., 1991), a presença de componentes na arquitetura é naturalmente esperada. Neste contexto, componentes são classes projetadas a partir do conceito de reuso e que são acessados através de métodos e operações especificadas pela interface do componente. Os

componentes devem ser extensíveis. O Tdk oferece um conjunto de componentes agrupados em uma interface de programação.

Notamos, no entanto, que em muitos casos, o uso devido de componentes requer um conhecimento grande sobre como utilizá-los, o que pode levar o programador a um aprofundamento em documentos técnicos. Como alternativa, visando diminuir o tempo necessário para se obter proficiência na utilização do Tdk, projetamos uma segunda interface de programação, com características de programação procedural, através de métodos que se comportam como funções. Esta interface oferece um conjunto de funcionalidades bem definidas, com um alto nível de abstração, escondendo a complexidade da implementação. Os métodos oferecidos, chamados de serviços, foram agrupados segundo seu contexto semântico, de forma que as funcionalidades possam ser facilmente localizadas e acessadas diretamente. Diferente dos componentes, os serviços não podem ser estendidos.

Em resumo, o Tdk oferece duas interfaces de programação (ver Figura 14.3), conforme descrevemos a seguir :

- uma API de componentes, através da qual se pode acessar um conjunto de classes que encapsulam dados e operações definindo comportamento consistente a ser utilizado ou estendido;
- uma API de Serviços, na qual as funcionalidades mais comuns são agrupadas de acordo com o seu contexto e a sua semântica e são acessíveis através de uma interface simplificada. Idealmente, a API de serviços deve referenciar tipos primitivos nas assinaturas de seus métodos, de modo a não ser necessário conhecimento prévio sobre componentes do Tdk, embora internamente os serviços devam utilizá-los.

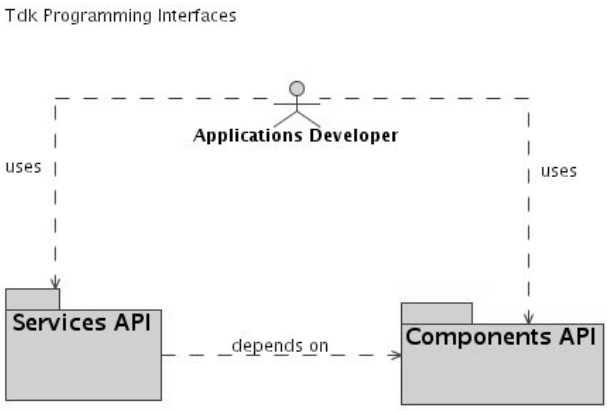


Figura 14.3 – As interfaces de programação Tdk.

14.3.3 Modelos funcionais

Os modelos funcionais sugerem estratégias de implementação para as funcionalidades de um sistema. No Tdk, projetamos modelos para atender às principais funcionalidades presentes em um tipo de aplicativo comum dentro do domínio de um SIG. Neste sentido, estamos falando de aplicações gráfico-interativas, que persistem seus dados em um banco de dados e que oferecem operações básicas de visualização, impressão, consulta e edição.

Para atender a este tipo de aplicação, o Tdk oferece três modelos principais: i) um modelo de persistência, que gerencia a comunicação com o banco de dados, ii) um modelo de interação, que estrutura a comunicação entre os elementos do sistema e possibilita a implementação das funcionalidades de visualização, impressão, consulta e edição de uma forma integrada e iii) um modelo de apresentação, que define um conjunto de componentes GUI, implementadas sobre um pacote gráfico-interativo externo. Existe ainda a intenção de oferecer um modelo temporal, para suporte à aplicações dinâmicas.

A concepção dos modelos funcionais no Tdk visa atender a um requisito de extensibilidade e reuso, oferecendo a possibilidade do programador adaptá-lo às suas necessidades específicas.

Em termos de implementação, os modelos funcionais serão construídos utilizando-se as interfaces de programação Tdk, conforme ilustrado na Figura 14.4.

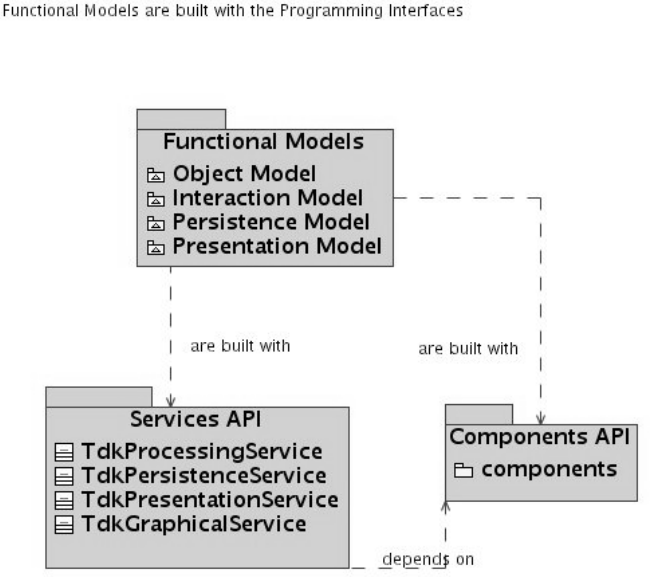


Figura 14.4 – Os Modelos funcionais são implementados utilizando-se as interfaces de programação Tdk.

14.4 Descrição técnica dos elementos da arquitetura

Nesta seção estaremos refinando as definições apresentadas como fundamentos da arquitetura Tdk. Descrevemos, em detalhes, os principais elementos referenciados até aqui. Com objetivo didático, estaremos exemplificando, através de codificação, o uso de alguns destes elementos.

14.4.1 API de componentes

Alguns dos principais componentes oferecidos pelo Tdk são descritos abaixo:

Componente referente a um Objeto Tdk (TdkObject) – este componente consiste na raiz da árvore hierárquica de dados Tdk. Ele representa a

agregação das interfaces que impõe comportamento a todos os elementos do modelo de dados Tdk (ex: TdkEventHandler, TdkPersistenceObject).

Componente para interface de persistência (TdkPersistenceObject) – define o comportamento de uma classe aderente ao modelo de persistência Tdk. As classes de aplicação deverão implementar esta interface direta ou indiretamente. As classes pertencentes ao modelo dados Tdk implementam indiretamente esta interface através do TdkObject.

Componente de identificação de objetos (TdkGlobalID) – a idéia de tratar todos os elementos, coleção ou folha, de uma base de dados geográfica da mesma forma abre espaço para a criação de um identificador único para cada TdkObject. O Global ID (GID) foi criado com esse propósito. A idéia do GID é representar TkdObjects da mesma forma que o sistema de DNS representa URL's, dessa forma podemos ter um GID único para cada TdkObject do sistema (inclusive sistemas que englobam vários bancos de dados). O GID é composto de: [Database Descriptor].[Object Type].[Object ID]

Componente para fabricação de objetos (TdkObjectFactory) – responsável por criar instâncias de TdkObjects dinamicamente. Todos os tipos criados por uma aplicação, que possuírem uma classe representante definida pela própria aplicação, devem ter métodos construtores registrados nesta classe. Ela contém uma estrutura de dados que tem o nome da classe como chave e um ponteiro para uma função que cria uma instância da classe correspondente ao tipo.

Componente Canvas Genérico (TdkCanvas) – representa uma abstração, independente de pacotes gráficos, de um canvas (painel) para desenho que tem formas primitivas para a TerraLib. Em outras palavras, o TdkCanvas provê uma interface específica para o desenho de geometrias TerraLib, que deve ser estendida para ter acesso às suas funcionalidades (ex. plotPoint, plotLine, plotPolygon, plotText, plotRaster).

Como exemplo de implementação de um *canvas* genérico mostramos aqui o *TdkCDCanvas*. Este componente consiste na versão CD (www.tecgraf.puc-rio.br/cd) de um *TdkCanvas*. No código apresentado a seguir mostramos como desenhar um ponto simplesmente :

```
/* declarações omitidas :
   • qtParent -  é a janela Qt principal, na qual,este canvas
     estará associado.
   • object – um objeto Tdk com geometria representada por um
     conjunto de polígonos.
   */
//…
TdkQtCanvas* canvas = new TdkQtCanvas(qtParent);
//Armazena a geometria num container.
TePolygonSet& polys=
((TdkGeographicObject*)object)->getPolygonsGeometry();

for(int j = 0; j < polys.size(); j++)
   canvas->plotPolygon(polys[j]);
//…
```

Codificação 14.1 – Exemplo de implementação em C++ do componente *TdkCanvas* com o pacote gráfico CD.

Componente de acesso ao banco de dados (TdkDatabase) – os dados e metadados são armazenados no banco TerraLib em tabelas relacionais e refletidos nas classes definidas. As tabelas são divididas em dois grupos; tabelas de metadados, que são usadas para guardar o conceito Terralib e possuem formato pré-definido e as tabelas de dados, que são usadas para armazenar os dados geográficos (componente espacial + descritiva). A TerraLib implementa um primeiro nível de abstração destes dados com a interface `TeDatabase` (ver Capítulo 12), onde temos métodos para manipular geometrias, layers, temas, etc.. Porém, é a interface *TdkDatabase* que oferece um nível mais alto de abstração, introduzindo tipos como *TdkMap* e *TdkProject*, e escondendo do programador detalhes do modelo de dados TerraLib.

Como exemplo de implementação do componente *TdkDatabase* apresentamos, a seguir, um código de acesso a uma tabela existente em um banco de dados Oracle :

```
//...
TdkOracleConDescriptor desc("oraserver",
                            "tdkuser",
                            "tdkl23" ,
                            "gisdev");
TdkDatabase* oracleDriver = new TdkOracleImpl(desc);
String tableName = "myTable";
oracleDriver->loadTable(tableName);
//…
```

Codificação 14.2 – Exemplo de implementação em C++ de *TdkDatabase* para acesso a um banco Oracle.

Abstração de uma aplicação básica (TdkAplication) – provê métodos que oferecem funcionalidades básicas para agilizar o processo de desenvolvimento de um SIG básico, como a criação de um novo modelo de dados, de um novo projeto, funções para visualização e consulta, impressão, etc.. O TdkApplication não provê métodos de interface gráfica, de modo que o programador pode escolher o *toolkit* para interface de sua preferência.

Como exemplo de implementação do componente *TdkApplication* apresentamos o código a seguir, que configura o modo de interação de uma aplicação que utiliza o pacote gráfico IUP (www.tecgraf.puc-rio.br/iup) para responder à interação com o usuário de forma a criar geometrias – linhas no caso – em uma base de dados TerraLib :

```
//…
Ihandle* btn = IupButton("", "CreateLineCb");
IupSetAttributes(btn,
                 "TIP = \"Criar linha\",
                 IMAGE = lines_image,
                 IMPRESS = lines_press_image,
                 IMINACTIVE = lines_inactive_image,
                 PRESSED = 0");
IupSetFunction("CreateLineCb",
              (Icallback)TdkIupApplication::CreateLine);
//…
```

Codificação 14.3 – Exemplo de implementação em C++ do componente *TdkApplication* sob o pacote gráfico IUP.

Componente de controle de interação (TdkController) – componente raiz da hierarquia de componentes responsáveis pela implementação da estratégia de resposta às ações do usuário. Todas as controladoras descritas adiante, no modelo de interação, irão implementar esta interface (*TdkMapController,TdkInteractController etc...)*.

14.4.2 API de serviços

Alguns dos principais serviços oferecidos pelo Tdk são descritos abaixo:

Serviço de persistência – disponibiliza funcionalidades de alto nível que permitem persistir, consultar e atualizar as informações em um banco de dados TerraLib.

O código a seguir ilustra o que é necessário para criar uma base de dados TerraLib completa, utilizando o Tdk. Modificar o código para receber outro banco de dados, como o Oracle por exemplo, seria um trabalho muito simples, bastando trocar o descritor de conexão para o descritor Oracle (*TdkOracleConDescriptor*) de acordo com os parâmetros necessários para esta conexão (nome do servidor etc...).

```
//…
TdkAccessConDesc desc(C:/“mydb.mdb”);
TdkPersistenceService.createTableModel(desc);
//…
```

Codificação 14.4 – Exemplo de uso do Serviço de persistência em Java na criação de um banco de dados TerraLib.

Serviço de processamento – provê funcionalidades que auxiliam as tarefas de calcular, converter dados e selecionar áreas georeferenciadas.

Serviço gráfico – oferece uma série de funcionalidades para interface gráfica baseadas no *TdkCanvas*.

Serviço de apresentação – oferece métodos para criação de diálogos de comunicação com o usuário.

14.4.3 Modelo de persistência

Conforme mencionado na Seção 1.2.3, o Tdk oferece um modelo funcional para a implementação do processo de gerência da persistência de dados em uma base TerraLib. Este modelo explora, principalmente, os conceitos de *transparência de localidade* e *automação da persistência*.

- Transparência de localidade – este conceito consiste em poder tratar dados de maneira uniforme, independente da sua origem.
- Automação da persistência – este conceito consiste em assumir a responsabilidade de controlar o acesso e atualização dos dados a serem persistidos.

Para implementar o conceito de transparência de localidade, o modelo define o uso de um identificador global (*TdkGlobalID*) por todas as classes aderentes ao modelo de persistência. Nesta identificação, a classe carrega a informação referente à sua base TerraLib de origem como descrito na especificação funcional do Tdk (www.terralib.org/tdk). No domínio de SIGs, podemos imaginar uma aplicação deste conceito quando se deseja visualizar dados oriundos de bancos de dados distintos.

Para implementar o conceito de automação da persistência, foi criado um componente *TdkPersistenceObject,* que define uma interface única

com os métodos a serem implementados por uma classe nova interessada em aderir ao modelo de persistência. Este componente é representado por um novo elemento, que foi acrescentado ao modelo de dados original (ver Figura 14.4), conforme mostrado na Figura 14.5.

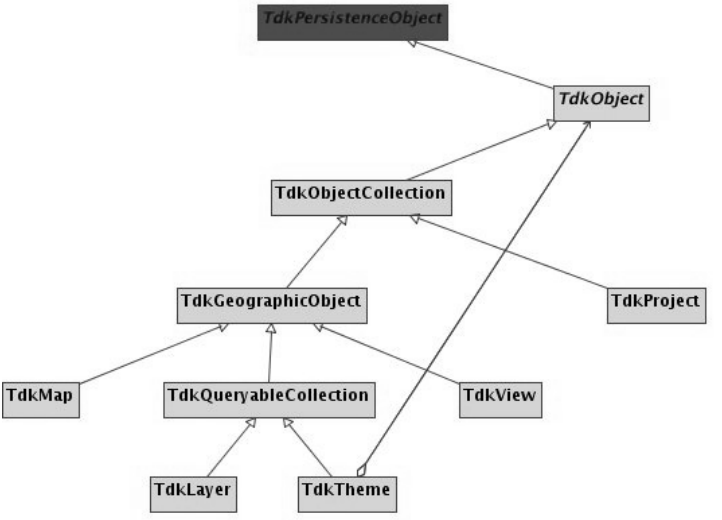


Figura 14.5 – O modelo de dados Tdk incluindo a classe *TdkPersistenceObject*.

Através da implementação deste último conceito, a tarefa de persistência dos dados da aplicação é muito simplificada. Duas situações podem ocorrer: i) o conjunto de elementos que definem o modelo conceitual Tdk é suficiente para atender às demandas do aplicativo a ser desenvolvido. Neste caso, o programador não precisa se preocupar em implementar nenhum código relacionado à persistência de seus dados numa base TerraLib. ii) o modelo original do Tdk é estendido através de novos elementos necessários ao domínio da aplicação. Neste caso, os componentes que representam os novos elementos devem implementar a interface única definida pelo *TdkPersistenceObject*. Desta forma, estes componentes passam a aderir ao modelo de persistência, ganhando a possibilidade de explicitar as particularidades referentes à persistência de

seus atributos na base TerraLib. Sem a utilização do conceito de automação, seria necessário estender o componente da TerraLib (*TeDatabase*) e definir os métodos de persistência para os novos elementos[1], que representa uma tarefa bem mais complicada.

Uma outra questão tratada pelo modelo de persistência, se refere ao controle do ciclo de vida de componentes provenientes da extensão do modelo de dados do Tdk. Como fazer, por exemplo, para que ao carregar um tema TerraLib, composto por objetos definidos fora do modelo de dados Tdk, seja possível instanciar esses objetos de forma automática?

Para endereçar a situação descrita acima, o Tdk implementa o conceito de uma fábrica de objetos, que possibilita às classes aderentes ao modelo de persistência registrar seus atributos simples e tê-los, desta forma, gerenciados pelo Tdk.

Apresentamos a seguir um exemplo da utilização do modelo. Na codificação ilustrada abaixo realizamos a carga de um tema persistido para a manipulação em memória :

```
TdkAccessConDescriptor desc( "C:/mydb.mdb" );
TdkObjectGID tdkObjGID(1,
                       "1",
                      "TDK_ THEME",
                      desc.getDbKey());
TdkTheme myTheme( tdkObjGID );
TdkPersistenceService::loadObject( &mTema );
```

Codificação 14.5 – Carregando um objeto persistido para memória.

14.4.4 Modelo de interação

Conforme mencionado na Seção 1.2.3, pretendemos atender à demandas que surgem no desenvolvimento de aplicativos gráfico-interativos. Para isto, projetamos um modelo, que sugere estratégias de implementação

[1] Como foi feito com *TdkProject*.

para o suporte às operações que requeiram interação com o usuário. O modelo de interação do Tdk apresenta as seguintes características:

- Flexibilidade - permite uma implementação genérica e independente de soluções proprietárias. É independente também do pacote gráfico a ser utilizado;
- Independência modular – através da utilização de um modelo de eventos, permite um desacoplamento entre os seus componentes;
- Para garantir estas características, concebemos o modelo com base no padrão MVC de arquitetura (ver Figura 14.6) (Krasner e Pope, 1988).

MVC (Model,View,Controller) – este padrão é bastante difundindo no domínio de aplicações gráfico-interativas e tem como principal objetivo organizar a interação entre os elementos participantes da implementação de uma determinada funcionalidade, garantindo o desacoplamento entre a interação do usuário e a representação do dado.

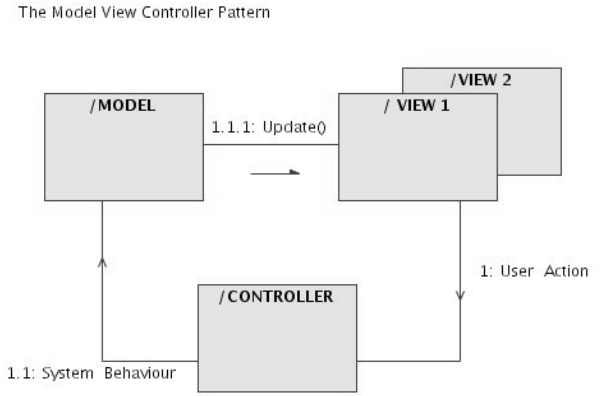


Figura 14.6 – O padrão MVC para desenvolvimento de aplicativos de visualização.

Uma vez aplicado, o padrão *MVC* resulta em uma divisão em camadas de contextos – Modelo, Visão e Controle – onde cada camada possui um papel bem definido na interação. No Modelo estarão os dados do domínio representado (ex: temas e objetos). A Visão representa a

interface direta com o usuário – como uma metáfora de um conceito do domínio (ex: legenda, mapa). Finalmente, o Controle determina a reação à uma ação do usuário sobre os dados do Modelo (ex: *zoom, pan*).

Como exemplo de aplicação do *MVC,* podemos imaginar, a visualização de diversos mapas em uma aplicação de visualização de dados geográficos e, no entanto, todas essas visualizações fazerem referência ao mesmo dado representado na camada de dados que, por sua vez, representa o dado persistido no banco de dados.

A Figura 14.7 apresenta a divisão das camadas de classes adotada pelo modelo de interação do Tdk, que são descritas em seguida.

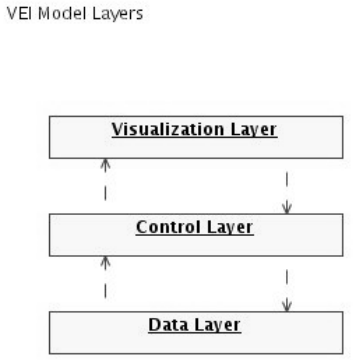


Figura 14.7 – As camadas referentes ao MVC do modelo de interação

Camada de Visualização (Visualization Layer) – nesta camada apresentamos um conjunto de metáforas do domínio da aplicação, responsáveis pela interação direta com o usuário. Não nos referimos aqui ao controle desta interação, mas sim a contextualização das possíveis ações que faz sentido realizar em cada entidade representada nesta camada (ex: uma legenda pode ter a ordem de seus temas trocada). Seguindo o padrão *MVC,* nesta camada, estariam as visões (*View*) dos modelos (*Model*) pertencentes à camada de dados.

Como exemplos de componentes desta camada podemos citar o componente de visualização de um tema TerraLib responsável por encapsular atributos de visualização como estilos de objetos selecionados.

Camada de Controle (Control Layer) – nesta camada acontece a mediação da interação entre usuário e aplicação. As ações do usuário, realizadas via elementos da camada de visualização, realizam algum tipo de operação sobre os dados representados, logicamente, pela camada de dados. A estratégia de reação, por parte do sistema, às ações do usuário são implementadas nesta camada. Estas estratégias são, inclusive, configuráveis de acordo com o tipo de comportamento desejado na resposta do sistema. Exemplificando, um *mouse click* pode causar uma reação de criação de ponto ou seleção de objeto, dependendo da configuração da camada de controle. Seguindo o padrão *MVC,* nesta camada estão as controladoras (*Controllers*) que definem o comportamento do sistema em reposta às acoes do usuario sob as visões (*Views*) dos dados (*Model*).

Como exemplos de componentes desta camada podemos citar dentre outros :

- A controladora de interação com o Mapa visualizado
- A controladora de interação com a Legenda

Uma observação final quanto a camada de controle do modelo de interação é a adoção de um paradigma de programação orientada a eventos. Apesar do *MVC* não depender deste paradigma, já que seria possível implementá-lo utilizando chamadas de métodos simples (*callback methods*), esta estratégia nos trouxe uma flexibilidade interessante. Com o uso de eventos, o código responsável pela notificação de uma determinada ocorrência (ex: a troca de estilo de um objeto), assim como o código que implementa a resposta à uma ação do usuário (ex: *mouse move*), ficam completamente separados do código funcional (*business code*). O modelo de eventos adotado foi sugerido pela arquitetura VIX (Santos, 2005) (ver Figura 14.8).

A arquitetura VIX – A maioria dos sistemas gráfico-interativos adota o modelo de orientação a eventos, difundido a partir de Smalltalk (Goldberg e Robson, 1983). Tipicamente, o fluxo de informações neste tipo de aplicação segue um padrão. A cada ação do usuário, são gerados eventos do sistema de interface que passam o controle para a aplicação.

Com o intuito de facilitar o desenvolvimento de aplicativos com este tipo de interação, é desejável tratar os eventos de maneira uniforme,

independente do tipo de ação que ele irá ocasionar. Isto levou à adoção do conceito de objetos visuais interativos (VOs).

Pode-se entender um objeto visual como qualquer objeto que possua uma representação e um comportamento.

Outro conceito importante é o de espaço visual (VS), que mapeia entidades que transmitem eventos para VOs e fornecem uma superfície de visualização onde eles são posicionados. Os VSs devem ser encarados como elementos de ligação entre o usuário e o VO que ele está manipulando. No processo de comunicação entre estes objetos, são utilizados dois mecanismos:

- Mensagens - através deste mecanismo os objetos podem trocar dados uns com outros (o VO deve saber responder às requisições do VS).
- Filtros - servem para modelar objetos que fazem a interface entre um VS e um VO. Um filtro repassa para o seu VO associado os eventos gerados em seu VS, podendo dar algum tratamento especial a esses eventos.

Na implementação Tdk da arquitetura VIX, como parte estrutural do modelo de interação definimos dois novos elementos:

- uma interface única para encapsular o código referente a troca de mensagens entre os componentes (TdkEventHandler).
- uma abstração de um objeto de edição (TdkLayoutObject) que servirá de componente raiz para todos os elementos do modelo de dados Tdk presentes na implementação de funcionalidades de edição.

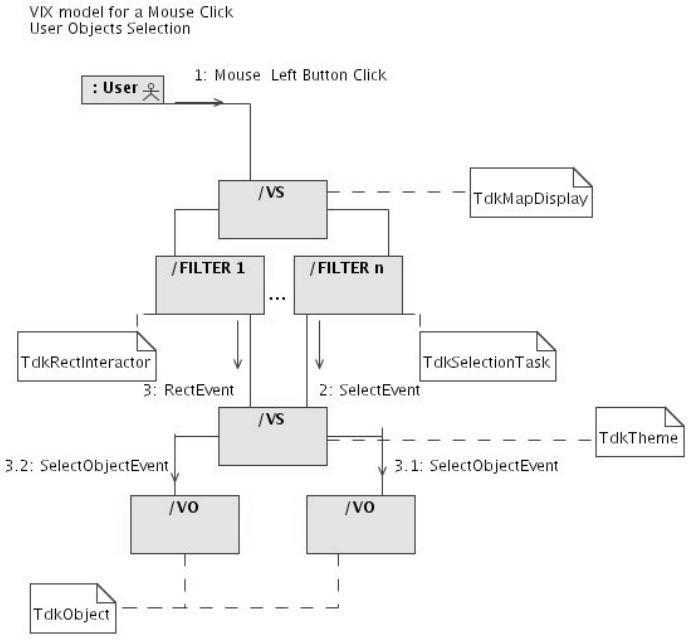


Figura 14.8 – O modelo de eventos baseado no VIX

Apresentamos a seguir o modelo de dados do Tdk, com estes novos elementos.

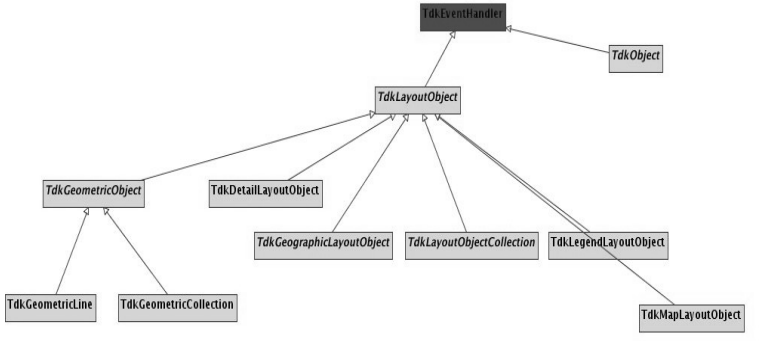


Figura 14.9 – O modelo de dados Tdk acrescido dos elementos referentes ao modelo de interação.

Para efeitos de ilustração, mostramos abaixo uma implementação de resposta ao evento de *scrolling* (rolagem da barra de janela da interface com o usuário) do usuário, tratado pela controladora do mapa visualizado .

```
void TdkMapController::
handleVSEvent(TdkScrollEvent& event)
{
   float x = event.getX();
   float y = event.getY();
   double dx = event.getDX();
   double dy = event.getDY();
   double xc = x + dx / 2.0;
   double yc = -y - dy / 2.0;
   mapDisplay_->center(xc, yc);
   mapDisplay_->draw();
}
```

Codificação 14.6 – Tratamento de um *scroll* no visualizador de mapas pela classe controladora *TdkMapController*.

Camada de Dados (Data Layer) – o principal objetivo da camada de dados é o armazenamento lógico dos dados persistidos.Em termos de interação com as outras camadas, ela é manipulada pela camada de visualização e controle através do modelo de eventos.Seguindo o padrão *MVC* esta camada armazena os componentes de modelo (*Model*).

Como exemplo de um componente desta camada podemos citar a representação de um tema, armazenando em memória seus objetos e atributos.

14.4.5 Modelo de apresentação

Este modelo apresenta um conjunto de componentes GUI, implementadas sobre um pacote gráfico-interativo externo. Os componentes GUI são implementados da forma mais fina possível, tendo comportamento e estado independentes de recursos específicos do pacote gráfico. Desta forma, pretende-se minimizar o esforço diante de uma

mudança de pacote gráfico. Além disto, tendo estado e comportamento descritos na camada funcional, a aplicação passa a ter mais flexibilidade sobre o comportamento do modelo. Como exemplos de componentes desta camada podemos citar:

- Diálogo para apresentação de atributos de um objeto.
- Menu de interação com o usuário.
- Diálogo para importação de dados.

14.5 Compatibilização com o OGC

Conforme mencionado na introdução, um dos requerimentos do Tdk é oferecer suporte a padrões do OGC. Este suporte é oferecido de duas formas:

- Uma camada de interface para o modelo de dados do OGC, que permite os programadores acostumados com o vocabulário e a arquitetura do OGC utilizarem as funcionalidades do Terralib.
- Serviços do OGC. Um exemplo disto é o serviço WMS para publicação de dados na web.

14.5.1 Interface para programação com o OGC

O Tdk oferece uma API que mapeia o modelo de dados do Tdk para o modelo de dados do OGC (www.opengeospatial.org). Sendo assim, é possível o desenvolvimento de aplicativos que utilizem as funcionalidades oferecidas pela TerraLib e que sejam compatíveis com os padrões publicados pelo OGC (Figura 14.10).

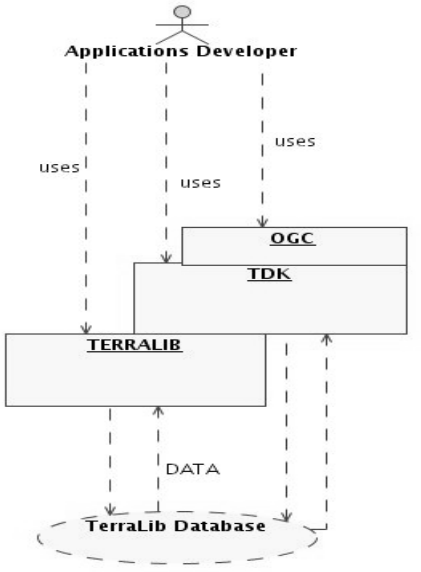


Figura 14.10 – A interface do Tdk com o padrão OGC.

14.5.2 Serviço WMS

O serviço WMS (*Web Map Servise*) permite a publicação de mapas na Web, produzidos a partir de dados georeferenciados. O serviço especifica um padrão de como o cliente deve requisitar as informações para o servidor e como este deve responder. Para esta comunicação são definidas três operações: *GetCapabilities*, *GetMap* e *GetFeatureInfo* (www.opengeospatial.org).

- *GetCapabilities*: permite ao cliente solicitar todas as metainformações sobre a base de dados, tais como o nome dos mapas contidos, projeção, escala, estilo possíveis dos mapas, formatos possíveis dos mapas (GIF, JPEG, PNG), formato das informações descritivas (FeatureInfo). Os parâmetros da operação são recebidos via HTTP e a resposta deve ser um arquivo XML (text/xml) no mesmo protocolo. Segundo o protocolo WMS, esta operação deve ser implementada obrigatoriamente.
- *GetMap*: retorna o mapa propriamente dito. Com esta operação, o cliente do Web Service pode requisitar um mapa em particular. Para isto basta usar as informações que foram retornadas pelo

GetCapabilities. Os parâmetros da operação são recebidos via HTTP e a resposta deve ser um arquivo de imagem no formato solicitado, que pode ser um das opções: GIF, PNG ou JPEG. Esta operação também é obrigatória.

- *GetFeatureInfo*: permite ao cliente realizar consultas nos mapas da base de dados. Os parâmetros da operação são recebidos via HTTP e a resposta deve ser um arquivo no formato solicitado que pode ser um dos seguintes: texto, XML ou GML. Esta operação não é obrigatória segundo o protocolo WMS.

14.6 Exemplos

A especificação definida pelo Tdk pode ser implementada em diversos contextos. Mostramos abaixo alguns exemplos, onde ilustramos o uso dos elementos de arquitetura Tdk na confecção de diferentes aplicativos.

14.6.1 Um sistema para visualização, impressão, consulta e edição

Ilustramos a seguir aspectos do processo de desenvolvimento de um aplicativo gráfico-interativo, que possui operações de visualização, impressão, consulta e edição de dados geográficos, a partir do Tdk. O processo consiste em estender as classes definidas pelo modelo de interação (ver Figura 14.11) para atender às funcionalidades da aplicação (zoom, impressão, criação de pontos e linhas, seleção de objetos, etc.) e a utilização dos modelos de Persistência e Apresentação.

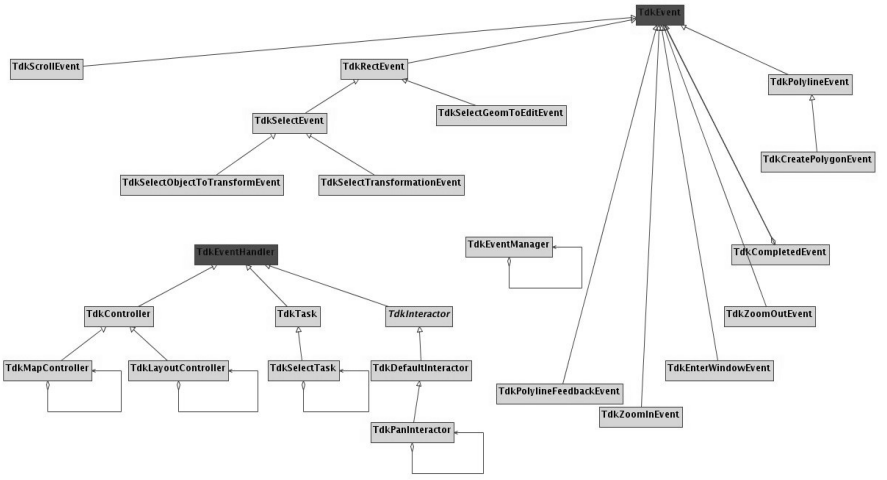


Figura 14.11 – A extensão do modelo de interação para o sistema VICE, onde capacitamos a aplicação com funcionalidades de seleção*, zoom*, criação de geometrias.

O aplicativo VICE - Sistema para Visualização, Impressão, Consulta e Edição – oferece a noção de modos de operações de acordo com o contexto de utilização do aplicativo (Visualização e Consulta, *Layout* e *Preview* de Impressão. A utilização do modelo funcional de interação, foi fundamental para a implementação destes modos de operações. A seguir, apresentamos as telas (*printscreen*) referentes a cada um desses modos em funcionamento (Figuras 14.12, 14.13 e 14.14).

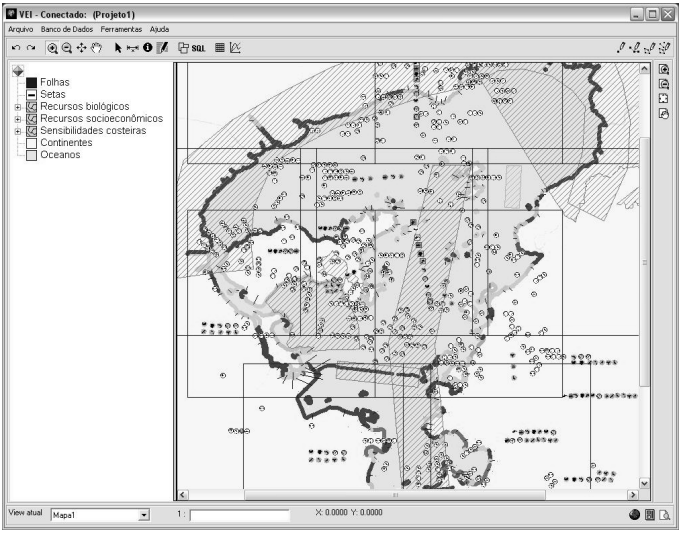


Figura 14.12 – VICE - modo de Visualização e Consulta.

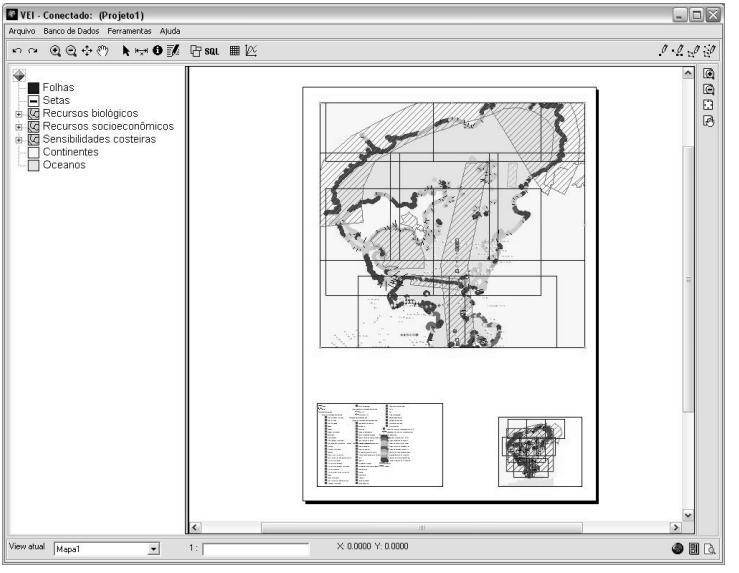


Figura 14.13 – VICE - modo de *Layout*.

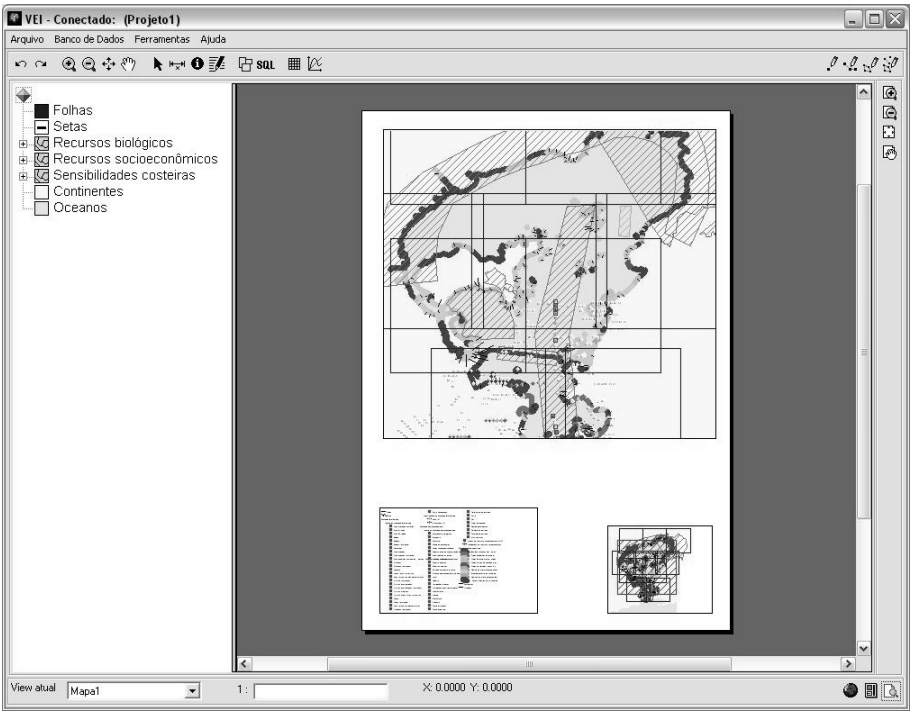


Figura 14.14 – VICE - modo de *Preview* de Impressão.

14.6.2 Aplicativos Java Desktop

Como destacado na seção de requisitos do Tdk, sua arquitetura deveria ser independente de linguagens. No caso de Java, as implementações necessárias para se ter o aplicativo descrito no exemplo anterior seriam bastante análogas ao caso de C++. No entanto, devido à impedância sintática entre as linguagens, devemos adicionar uma camada responsável pela troca de dados entre o mundo C++ e o mundo Java, conforme mostramos na Figura 14.15.

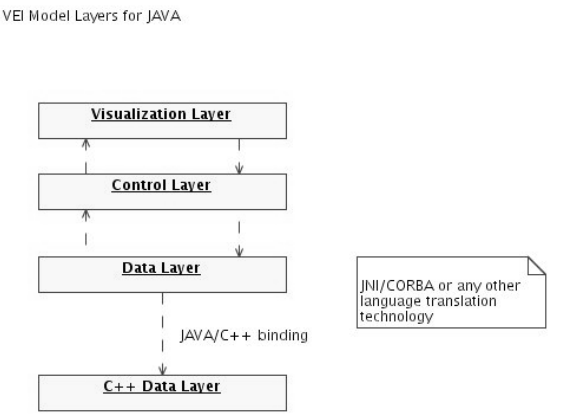


Figura 14.15 – A camada de acesso para abstração da troca de dados entre Java e C++.

14.6.3 Aplicativos Web

Um terceiro exemplo de produto desenvolvido com o Tdk é o aplicativo *Tdk Web Client* (ver Figura 14.16), que consiste em um visualizador de mapas na Web, compatível com o padrão WMS. Este aplicativo acessa o serviço WMS do Tdk (ver Seção 1.4).

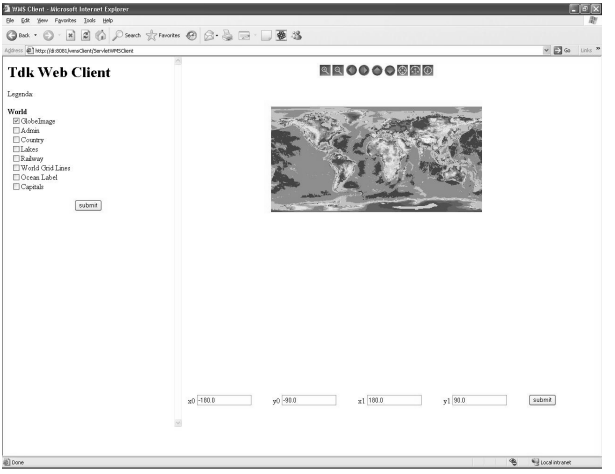


Figura 14.16 – Um produto Tdk de visualização de Mapas na Web.

Referências

GAMMA, E.; HELM, R.; JOHNSON, R.; VLÍSSIDES, J. Design Patterns - Elemens of Reusable Object-Oriented Software. Reading, MA: Addison-Wesley,1995. 395 p.

GOLDBERG, A.; ROBSON. D. Smalltalk-80 : The Language and its Implementa-tion. Addison-Wesley, 1983.

KRASNER, E., G.; POPE, S. T.; A cookbook for using the model-view controller user interface paradigm in Smalltalk-80. Journal of Object-Oriented Programming, v. 1, n. 3, p. 26-49, 1988.

RUMBAUGH, J.; BLAHA, M.; PERMERLANI, W.; EDDY, F.; LORENSON, W.; *Object-Oriented Modeling and Design*. Prentice Hall , Englewood Cliffs , NJ, 1991.

SANTOS, A. L. S. C. VIX - Um Framework para Suporte a Objetos Visuais Interativos. Rio de Janeiro: Pontifícia Universidade Católica do Rio de Janeiro, 2005.