UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

INSTITUTO DE INFORMÁTICA

PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

# Incorporando Suporte a Restrições Espaciais de Caráter Topológico ao *Modelo Abstrato* do Consórcio *Open* GIS

por

VANIA BOGORNY

Dissertação submetida à avaliação, como requisito parcial para a obtenção do grau de Mestre em Ciência da Computação

Prof. Dr. Cirano Iochpe

Orientador

Porto Alegre, julho de 2001.

CIP – CATALOGAÇÃO NA PUBLICAÇÃO

Bogorny, Vania

Incorporando suporte a restrições espaciais de caráter topológico ao Modelo Abstrato do consórcio Open GIS / por Vania Bogorny. - Porto Alegre: PPGC da UFRGS, 2001.

134 p.:il.

Dissertação (mestrado) – Universidade Federal do Rio Grande do Sul. Programa de Pós-Graduação em Computação, Porto Alegre, BR-RS, 2001. Orientador: Iochpe, Cirano.

1. *OpenGIS*. 2. Sistemas de Informação Geográfica. 3. Restrições Espaciais. 4. Relacionamentos Topológicos. I. Iochpe, Cirano. II. Título.

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Reitora: Prof[a]. Wrana Panizzi

Pró-Reitor de Ensino: Prof. José Carlos Ferraz Hennemann

Pró-Reitor Adjunto de Pós-Graduação: Prof. Philippe Olivier Alexandre Navaux

Diretor do Instituto de Informática: Prof. Philippe Olivier Alexandre Navaux

Coordenador do PPGC: Prof. Carlos Alberto Heuser

Bibliotecária-Chefe do Instituto de Informática: Beatriz Regina Bastos Haro

# Agradecimentos

Agradeço a Deus por te me concedido a oportunidade de estar nesta vida, por ter me dado este corpo perfeito e saudável para que pudesse desenvolver-me e de alguma forma contribuir para a humanidade.

Agradeço ao Jucemar, amigo e companheiro, pelo incentivo, conselhos e pela paciência em saber esperar.

Aos amigos, Alessandra e Felipe, pela amizade, compreensão e apoio nas horas difíceis.

Agradeço à Universidade Federal do Rio Grande do Sul pela oportunidade concedida para a realização deste trabalho.

Ao meu orientador Dr. Cirano Iochpe, pela confiança e por todo conhecimento transferido. Agradeço também pela oportunidade e experiência adquirida ao trabalhar em seus projetos de pesquisa.

Aos funcionários, pelo bom atendimento e pela colaboração com informações e serviços prestados.

Ao CNPQ, pelo auxilio financeiro concedido para a conclusão do curso.

À Primeira Divisão de Levantamento do Exército Brasileiro, pela oportunidade em transformar parte desta pesquisa em realidade prática.

# Sumário

# Lista de Figuras

# Lista de Tabelas

# Resumo

Os Sistemas de Informação Geográfica (SIG) são construídos, especificamente, para armazenar, analisar e manipular dados geográficos, ou seja, dados que representam objetos e fenômenos do mundo real, cuja localização em relação à superfície da Terra seja considerada. A interoperabilidade desses sistemas, que constitui-se na capacidade de compartilhar e trocar informações e processos entre ambientes computacionais heterogêneos, se faz necessária, pois, devido ao elevado custo de aquisição dos dados geográficos, as comunidades de informação precisam compartilhar dados de fontes existentes, sem a necessidade de fazer conversões. Porém, pela complexidade e incompatibilidades de representação, de estrutura e de semântica das informações geográficas, a maioria dos softwares de SIG, hoje, não são interoperáveis. Existe também, além do problema da não interoperabilidade, uma crescente preocupação com relação à qualidade e à integridade espacial dos dados geográficos. Contudo, alguns modelos conceituais de dados geográficos e os softwares de SIG não oferecem, ainda, os meios adequados para representar e garantir a integridade espacial das informações. As restrições de integridade definidas durante a fase de projeto conceitual, normalmente, são implementadas durante o projeto físico, seja de forma implícita ou explícita, podendo ser incorporadas diretamente no modelo de implementação do SIG, de forma que o usuário da aplicação apenas mencione a regra e o sistema a implemente e a garanta automaticamente.Este trabalho de pesquisa propõe uma extensão ao *Modelo Abstrato OpenGIS*, modelo este que deve ser um padrão de interoperabilidade de software para SIG. A extensão proposta incorpora ao mesmo um subconjunto de tipos de restrição espacial, buscando com isso oferecer melhor suporte às regras da realidade geográfica expressáveis na modelagem conceitual do sistema.

**Palavras-Chave:** *OpenGIS*, Sistemas de Informação Geográfica, Restrições Espaciais, Relacionamentos Topológicos.

**TITLE:** "INCORPORATING SUPPORT TO SPATIAL CONSTRAINTS BASED ON TOPOLOGICAL RELATIONS IN THE OPEN GIS ABSTRACT MODEL"

# Abstract

Geographic Information Systems (GIS) are especially build to store, analyze and manipulate geographic data, that is, data that represent real world phenomenon and objects whose location in relation to earth surface is considered. The interoperability of these systems, which is the ability of sharing and exchanging information and processes between heterogeneous computational environments, is necessary due to the high cost of the acquisition of geographic data, allowing the user to share existent data without making conversions. However, by the complexity and incompatibilities of representation, structure and semantics of geographic information, the most of systems actually are not interoperable. There is also, beyond the non interoperability problem, an increasing concern with the geographic data quality and spatial integrity. However, some geographic conceptual data models and GIS software do not provide a way of representing and warranting the information's spatial integrity yet. The integrity constraints defined during the conceptual project phase are commonly implemented during the physical project, explicitly or implicitly, becoming to be directly incorporated in the GIS implementation, so that the application´s user just mention the rule and the system will automatically implement and warrant it.This research work proposes an extension to the OpenGIS Abstract Model, a model that should be a standard for GIS software interoperability. The extension proposal incorporates a subset of spatial constraints types, aiming at offering better support to the geographical reality rules expressed in the conceptual modeling system.

**Keywords:** *OpenGIS*, Geographic Information Systems, Spatial Constraints, Topological Relationships.

# 1 Introdução

O propósito deste trabalho é estender a arquitetura de software aberta e interoperável para Sistemas de Informação Geográfica proposta pelo consórcio *Open* GIS (OGC), incorporando a ela um subconjunto de tipos de restrição espacial, visando ao provimento de melhor suporte às regras da realidade geográfica que precisam ser mantidas no sistema.

A implementação de um SIG requer a integração de conhecimentos de diversas áreas da Ciência da Computação e de disciplinas relacionadas ao processamento de tipos específicos de dados [CAM 96]. Como o trabalho busca estender um modelo de implementação de software para SIG, esse Capítulo apresenta, inicialmente, alguns conceitos básicos de geoprocessamento. Apresenta também alguns recursos de banco de dados como transações, por exemplo, necessários para a realização deste trabalho.

A Seção 1.1 aborda conceitos, características e alguns aspectos básicos que envolvem os SIG. A Seção 1.2 introduz o funcionamento de um ambiente de produção do SIG. A Seção 1.3 apresenta a motivação que levou a essa pesquisa. A Seção 1.4 enumera os objetivos atingidos durante o trabalho e a Seção 1.5 apresenta a estrutura da dissertação.

## 1.1 Sistemas de Informação Geográfica

Os SIG são sistemas especialmente construídos para realizar o tratamento computacional de dados geográficos, pois armazenam e manipulam, além dos atributos descritivos, a espacialidade dos dados georreferenciados [GUT 94]. Através dos SIG, pode-se armazenar, analisar e manipular dados que representam objetos e fenômenos cuja localização em relação à superfície terrestre é uma característica inerente e indispensável para tratá-los [CAM 96].

Uma das vantagens dos SIG é que eles podem manipular dados *convencionais* (descritivos) e *espaciais* de forma integrada, provendo uma estrutura consistente para análises e consultas envolvendo dados geográficos. Com relação aos dados espaciais, os SIG permitem o armazenamento de informações de localização geográfica relativas a projeções cartográficas e escalas específicas, características estruturais, geométricas e topológicas de entidades pertencentes a um domínio de aplicação [LIS 96].

Existem outros sistemas que também manipulam dados espaciais como, por exemplo, sistemas de CAD. Porém, os SIG caracterizam-se por permitir, ao usuário, a realização de complexas operações de consulta e de análise sobre dados georreferenciados.

### 1.1.1 Conceitos Básicos

Segundo alguns autores [LIS 97], os SIG e seus modelos de dados geográficos devem refletir a forma como as pessoas observam o mundo. Um dos princípios

filosóficos da percepção humana dos fenômenos geográficos é que a realidade é composta de entidades individuais e de superfícies contínuas.

Em SIG, os elementos da realidade são abstraídos e modelados em um banco de dados geográficos[1] (BDG). De acordo com o Padrão Nacional de Cartografia Digital dos Estados Unidos [GAR 98], o elemento da realidade é denominado *entidade* ou *fenômeno,* e o elemento representado no banco de dados é tratado como *objeto.*

Denomina-se *entidade* qualquer fenômeno da natureza, ou resultante da ação direta do homem, que é de interesse para o domínio da aplicação. Do ponto de vista do SIG, uma entidade é denominada *entidade geográfica* ou *feição geográfica* quando possui atributos de localização sobre a superfície da Terra num determinado período de tempo [CAM 96].

*Objeto* é a representação espacial de uma entidade, ou parte dela. Em SIG, um objeto é denominado *objeto geográfico* [GAR 98]. Assim, as representações de entidades ou fenômenos geográficos do mundo real, em SIG, são tratadas como *objetos geográficos.* Exemplos incluem ferrovias, florestas, rios, edificações, etc.

Os objetos geográficos estão fortemente relacionados. Por exemplo, ferrovias podem cruzar rios; rios podem atravessar florestas; estados são formados por municípios, e assim por diante. Tais associações, chamadas relacionamentos espaciais, são apresentadas no Capítulo 2.

### 1.1.2 Objetos e Campos Geográficos

Entidades e fenômenos geográficos são percebidos no mundo real, segundo as visões dicotômicas de campo e de objeto [LIS 96], acarretando em diferentes formas de modelagem dos mesmos fenômenos geográficos.

Na visão de campo, a realidade é modelada por variáveis distribuídas no espaço através de funções de cobertura. Toda posição no espaço geográfico pode ser caracterizada através de um conjunto de atributos como, por exemplo, a temperatura, o tipo de solo e o relevo, medidos em um conjunto de coordenadas geográficas (apresentadas na Seção 1.1.3). Campos geográficos, normalmente, são modelados como funções sobre variáveis. Segundo Pires [PIR 97], alguns deles referem-se a variáveis distribuídas de forma contínua sobre a superfície, como é o caso da temperatura e a cobertura do solo. Outros referem-se a variáveis distribuídas de forma discreta como, por exemplo, a população, ocorrências epidemiológicas, pontos de captação de água, etc.

Os aspectos espaciais de um campo geográfico são abstraídos de forma diferente dos aspectos espaciais de um objeto geográfico. Eles podem ser representados por grades de células, polígonos adjacentes, isolinhas, grades de pontos, redes

---

[1] Banco de dados geográficos é o termo utilizado, neste trabalho, para referenciar o componente responsável pelo armazenamento dos dados de um SIG.

triangulares irregulares e pontos amostrados irregularmente. Detalhes sobre a forma de representação de fenômenos da realidade geográfica, percebidos na visão de campo, podem ser encontrados em [LIS 97, CAM 96].

Na visão de objetos, a realidade consiste de entidades individuais bem definidas e identificáveis [LIS 99]. Cada entidade tem suas propriedades e ocupa um determinado lugar no espaço, sendo que duas ou mais entidades podem estar situadas sobre a mesma posição geográfica.

Em SIG, o objeto espacial representa o aspecto espacial do objeto geográfico, e pode ser descrito por duas características: a geometria que define a forma do objeto e que é expressa em termos de pontos, linhas, polígonos, volume e combinações destes; e o referenciamento espacial, que é definido pelo estabelecimento de um sistema de coordenadas relativas a superfície terrestre, associado ao objeto [GAR 98].

Em um BDG, cada objeto geográfico é representado de acordo com um tipo de objeto espacial apropriado em um determinado sistema de coordenadas. Os tipos básicos de objetos espaciais são classificados segundo suas dimensões espaciais [GAR98], conforme ilustra a tabela 1.1.

TABELA 1.1 – Tipos básicos de objetos espaciais

| **Dimensão** | **Tipo** | **Descrição** |
|---|---|---|
| **0D** | Ponto | Objeto que ocupa uma posição no espaço mas não possui dimensão espacial. Exemplos de entidades geográficas que podem ser representadas por objetos do tipo ponto são: pontos de captação de água, hidrantes, etc. |
| **1D** | Linha | Objeto que tem um certo comprimento. É formado por dois ou mais objetos 0D e possui distribuição linear. Exemplos de entidades que podem ter esse tipo de representação incluem rios, rodovias, ferrovias, ruas, etc. |
| **2D** | Polígono | Objeto com comprimento e largura, sendo limitado por no mínimo, três objetos unidimensionais. Um objeto do tipo polígono representa uma área no espaço, delineada por uma zona limítrofe composta por segmentos de linha [GAR 98]. Cidades, edificações, estados, etc, podem ser representados por esse tipo de objeto espacial. |
| **3D** | Sólido | É um objeto de comprimento, largura e altura, sendo limitado por, pelo menos, quatro objetos bidimensionais. Projeções tridimensionais de superfícies contínuas podem ser usadas, por exemplo, para permitir uma melhor visualização do relevo da área observada. |

O tipo de representação do objeto espacial pode variar de acordo com a escala na qual a entidade geográfica foi abstraída da realidade (escala de origem).

A escala é a relação entre a dimensão dos elementos representados em um mapa e sua grandeza correspondente medida sobre a superfície da Terra [GAR 98]. Por exemplo, em um mapa na escala 1:50.000, um centímetro no mapa corresponde a 50.000 cm (ou 500m) na superfície terrestre. Uma escala grande, como a de 1:10.000 (1 cm no mapa corresponde a 100 m na superfície terrestre), é suficiente para representar o traçado urbano de ruas de uma cidade. Porém, é insuficiente caso a aplicação necessite manipular informações como, por exemplo, lotes urbanos. A escala é uma informação obrigatória e fundamental em qualquer mapa e, consequentemente, para qualquer SIG, pois ambos necessitam conhecer a escala e a projeção em que o dado foi capturado para realizar consultas e análises espaciais com precisão.

Projeção é o método matemático através do qual a superfície curva da terra é representada sobre uma superfície plana [LIS 96]. Existem diferentes projeções utilizadas na confecção de mapas, as quais atendem a objetivos distintos como, por exemplo, preservar a área dos objetos representados, sua forma, ou mesmo a distância entre pontos num mapa.

Do ponto de vista do usuário que utiliza dados espaciais, os elementos do mundo real são abstraídos em elementos cartográficos, cada um em uma escala apropriada com seu sistema de coordenadas e sua projeção, criando diversas camadas sobrepostas, cada uma contendo informações distintas sobre um determinado assunto como, por exemplo, hidrografia, transporte, altimetria, infra-estrutura, etc.

### 1.1.3 Sistemas de Coordenadas

Os sistemas de coordenadas dividem-se em dois grandes grupos: *sistemas de coordenadas geográficas* ou *terrestres* e *sistemas de coordenadas planas* ou *cartesianas* [CAM 96].

No *sistema de coordenadas geográficas* ou *terrestres,* cada ponto da superfície da terra é localizado na interseção de um meridiano com um paralelo. *Meridianos* são círculos máximos da esfera cujos planos contêm o eixo dos pólos. *Paralelos* são círculos da esfera cujos planos são perpendiculares ao eixo dos pólos [CAM 96].

Um ponto na superfície terrestre é representado por um valor de latitude e longitude. *Longitude* é a distância angular entre um ponto qualquer da superfície terrestre e o meridiano de origem. *Latitude* é a distância angular entre um ponto qualquer da superfície da Terra e a linha do Equador [CAM 96].

Como o sistema de coordenadas geográficas considera desvios angulares a partir do centro da Terra, não é um sistema conveniente para aplicações em que se busca distâncias ou áreas. Para esses casos, utilizam-se outros sistemas de coordenadas mais adequados como, por exemplo, o sistema de coordenadas planas descrito a seguir.

O *sistema de coordenadas planas* baseia-se na escolha de dois eixos perpendiculares, usualmente denominados eixo horizontal e eixo vertical, cuja interseção é denominada origem, estabelecida como base para a localização de qualquer ponto do plano. Nesse sistema de coordenadas, um ponto é representado por dois números: um correspondente à projeção sobre o eixo *x* (horizontal), associado principalmente à longitude; e, outro correspondente à projeção sobre o eixo *y* (vertical), associado à latitude. Essas coordenadas estão, matematicamente, relacionadas às

coordenadas geográficas, de maneira que umas podem ser convertidas nas outras [CAM96].

### 1.1.4 Qualidade dos Dados

A qualidade dos dados geográficos é um fator de grande importância em SIG, pois informações inconsistentes podem gerar problemas na tomada de decisão baseada em consultas e análises espaciais.

Uma aplicação de SIG diferencia-se das demais aplicações de bancos de dados por fazer uso de grande volume de informações, normalmente importadas de outros sistemas, muitas vezes de outras organizações [LIS 99]. Esse fenômeno gera, freqüentemente, certos problemas em relação a confiabilidade dos dados geográficos.

Os SIG podem conter dados com erros, que precisam ser identificados e tratados. Os erros podem ser introduzidos no banco de dados de diversas formas [LIS 96]: decorrentes de erros na captura e armazenamento dos dados; adicionados durante processos de conversão; gerados durante a exibição ou impressão dos dados; ou, ainda, surgirem a partir de resultados equivocados em operações de análise espacial.

Embora muitos dados espaciais estejam associados a um grau de erro, eles são representados, computacionalmente, com alta precisão. A *precisão* pode ser definida como o número de casas decimais ou dígitos significativos em uma determinada medida [LIS 96]. Se um objeto espacial possui precisão de posicionamento com vários dígitos significativos após a vírgula, isso não implica que essa informação seja acurada.

*Acurácia* é uma estimativa de valores serem verdadeiros ou com a probabilidade de uma predição estar correta [LIS 96]. O objetivo de identificar os erros não é eliminá-los completamente, mas buscar gerenciá-los e reduzi-los. A acurácia dos dados é crucial para que os usuários do SIG confiem no sistema.

A qualidade dos dados pode ser medida a partir da análise de características como acurácia posicional, acurácia dos atributos (descritivos e espaciais), consistência lógica (ex.: relacionamentos topológicos), resolução da imagem, temporalidade e histórico do processo de obtenção dos dados [LIS 96].

## 1.2 Ambiente de Produção de um SIG

Na fase de produção do SIG ocorre a entrada dos dados geográficos no banco de dados. Câmara em [CAM 96] define quatro principais formas para dar entrada de dados em um SIG: digitalização em mesa, digitalização ótica, entrada de dados via caderneta e a leitura de dados na forma digital, incluindo a importação em outros formatos.

Uma vez populado o banco de dados geográficos, o usuário do SIG manipula esses dados realizando operações de inserção, alteração e remoção. Para gerenciar essas operações, o SIG utiliza o mesmo recurso adotado em bancos de dados convencionais, um gerenciador de transações (abordado na próxima seção).

As restrições espaciais, tratadas no Capítulo 2, precisam ser testadas durante a população do banco de dados geográficos e a cada operação de atualização realizada pelo usuário. Esses testes são realizados usando alguns recursos do gerenciador de transações do banco de dados. Para fazer uso desses recursos são necessários alguns conceitos básicos sobre transações, os quais são apresentados na Seção 1.2.1. A Seção 1.2.1 desvia-se do escopo do trabalho, mas seu conteúdo é fundamental para o entendimento do processo de controle de restrições espaciais no modelo de implementação de software para SIG do consórcio *Open* GIS.

### 1.2.1 Conceitos e Modelos de Transações de Banco de Dados

Segundo Silberchatz [SIL 95], transação é uma coleção de instruções de computador que executa uma única função lógica do ponto de vista da aplicação do banco de dados.

A transferência de fundos de um conta bancária para outra, processo no qual uma conta é debitada e outra creditada, representa um exemplo típico de transação. Claramente, é essencial à consistência do banco de dados que ambos, o crédito e o débito, ocorram ou que nenhum dos dois aconteça. Isto é, a transferência de fundos deve ocorrer por inteiro ou não deve ocorrer. Este requerimento *tudo* ou *nada* é denominado *atomicidade.*

A transação é uma unidade de atomicidade. A preservação da atomicidade é uma questão importante no processamento de transações do banco de dados, independentemente da possibilidade de falhas dentro do sistema do computador [SIL 95].

Uma transação corresponde a uma unidade de programa que faz o acesso e, possivelmente, atualiza vários itens de dados. As transações não podem violar a consistência do banco de dados, ou seja, se ele era consistente quando a transação iniciou, deverá continuar assim quando a transação terminar.

As transações, em geral, possuem quatro propriedades que garantem a correção de sua execução concorrente [MAN 95]: a atomicidade, a consistência, o isolamento e a durabilidade. Transações que possuem as quatro propriedades são conhecidas como transações ACID.

A *atomicidade*, do ponto de vista da aplicação, faz com que o SGBD (Sistema Gerenciador de Banco de Dados) garanta que, se algum erro ocorrer durante a execução da transação, todas as ações efetuadas por ela serão desfeitas. Em outras palavras, ou todas as operações da transação são efetuadas ou nenhuma delas deve persistir no banco de dados.

A transação, por definição, quando executada em sua totalidade, deve levar o banco de dados de um estado *consistente* para outro, também consistente [MAN 95].

A propriedade de *isolamento* determina que os efeitos parciais de transações em execução não sejam percebidos por outras transações em execução.

A *durabilidade* de uma transação concluída com sucesso garante que as alterações, por ela efetuadas, permaneçam inalteradas até que outra transação as modifique.

Uma transação longa é aquela cuja execução, mesmo sem a interferência de outras transações, dura uma substancial quantia de tempo, podendo levar horas ou dias para terminar. Aplicações como CAD, SIG e *WorkFlow* exigem o uso de transações longas, uma vez que o processamento das informações pode demorar um longo período de tempo.

A transação tem uma duração longa quando [SIL 95]:

- processa muitos objetos da base de dados;
- faz cálculos demorados; ou
- quando fica aguardando enquanto o usuário efetua muitas inserções.

A transação longa possui maior probabilidade de falhas que transações mais curtas. Exemplos de transações longas incluem cálculos mensais em agências bancárias e coletas de estatísticas sobre uma base de dados inteira.

Na literatura, podem ser encontrados vários modelos de transações longas de banco de dados, entre os quais destacam-se: SAGAS [MOL 87], *ConTracts* [MAN 95], ACTA [ELM 90], *Nested Transactions* [HAE 87], DOM [ELM 90], S-*Transaction* [ELM 90], entre outros. Dentre esses modelos de transação, apenas SAGAS é descrito (resumidamente) abaixo, pois o objetivo em introduzir transações longas é tão somente o de indicar, ao leitor, o contexto no qual a extensão ao *Modelo Abstrato* será utilizada.

O modelo SAGAS foi escolhido porque é de fácil entendimento e, basicamente, oferece todos os recursos necessários para gerenciar o processo de verificação e garantia de restrições espaciais. Apenas pequenas adaptações foram feitas a este modelo, principalmente para permitir o armazenamento temporário, no banco de dados, de feições inconsistentes. Detalhes sobre tais adaptações estão descritas na Seção 4.2.3.1.

SAGAS é um modelo de transações longas que podem ser particionadas em uma coleção de subtransações ACID e intercaladas, de qualquer forma, com outras transações.

Numa aplicação de reserva de passagens aéreas, por exemplo, uma transação longa T pode ser responsável pela reserva de um número *x* de poltronas de um vôo. Essa transação não precisa bloquear o acesso a todas as poltronas até que a reserva atual esteja concluída. Após reservar uma poltrona $P_x$ para o vôo $V_1$, a transação pode permitir que outras transações reservem acentos no mesmo vôo. Em outras palavras, a transação T pode ser vista como uma coleção de subtransações ACID $T_1$, $T_2$, $T_3$,...$T_n$, que reservam as poltronas individuais. A transação T é responsável por controlar todas as reservas ou cancelá-las, caso o vôo seja suspenso.

Em uma SAGA, cada subtransação $T_i$ está associada a uma transação de compensação $C_i$, utilizada para desfazer a subtransação após seu término. No exemplo da aplicação de reserva de vôos, se a subtransação $T_i$ reserva uma poltrona em um vôo, a subtransação $C_i$ desfaz essa reserva, ou seja, reduz um do número total de reservas de poltronas. Contudo, $C_i$ não pode simplesmente armazenar o número de poltronas

existentes na base de dados quando $T_i$ iniciou, pois outras transações podem ter sido executadas no tempo em que $T_i$ reservava a poltrona e $C_i$ cancelava a reserva, sendo que o número de reservas para esse vôo pode ter sido modificado nesse período. Assim, um sistema de transações SAGA garante a seguinte seqüência de subtransações:

$T_1, T_{2,...}T_j\ C_{j,...}C_2, C_1$ , onde $0 \leq j < n$

Cada uma das subtransações que compõem uma SAGA tem indicações de início e fim. Caso ocorra alguma falha enquanto uma subtransação estiver aberta, a mesma é desfeita, mas a SAGA permanece. Cada SAGA também possui uma indicação de início e fim.

## 1.3 Motivação

A qualidade dos dados é uma característica fundamental em qualquer sistema de informação. Em SIG, a forma geométrica dos objetos espaciais torna o gerenciamento da qualidade dos dados bastante complexa. As formas de obtenção desses dados (digitalização de mapas, aerofotogrametria, sensoriamento remoto, etc.) resultam em conjuntos de informações que precisam ser suficientemente consistentes, para serem usadas em análises espaciais e assegurarem confiança na tomada de decisão como, por exemplo, no caso do planejamento urbano.

Entretanto, muitas bases de dados geográficos existentes hoje apresentam uma série de estruturas geométricas e topológicas inconsistentes. Por isso, a credibilidade de qualquer resultado de consulta ou análise espacial não pode ser assegurada pelo SIG [SER 2000].

Outra característica importante dos dados geográficos é a diversidade de fontes geradoras e de formatos apresentados [CAM 96a]. O elevado custo de aquisição dos dados georreferenciados tem obrigado seus usuários a compartilharem dados já existentes de terceiros. Porém, a utilização de soluções inadequadas para proporcionar a troca de dados entre bases heterogêneas gera erros e ocasiona eventual perda de informações [BOG 2000]. Esses problemas evidenciaram a necessidade de sistemas abertos e interoperáveis, para compartilhar dados geográficos sem a necessidade de realizar conversões.

A interoperabilidade constitui-se na capacidade de compartilhar e trocar informações e processos entre ambientes computacionais heterogêneos, autônomos e distribuídos [THO 98]. Porém, pela complexidade e incompatibilidade de representação, de estrutura e de semântica das informações geográficas, a maioria dos softwares de SIG, hoje, não são interoperáveis.

O consórcio *Open* GIS (*Open GIS Consortium* - OGC) [BUE 98] é uma organização internacional que está criando novas padronizações técnicas e comerciais para garantir interoperabilidade em SIG. Através de seu *Modelo Abstrato*, orientado a objetos e baseado na notação UML (*Unified Modeling Language*) [BOO 98], o OGC tenta criar um padrão, de direito, de arquitetura de software para SIG, que garanta interoperabilidade de dados e de operações.

A especificação de software proposta pelo OGC é uma arquitetura usada na criação de software de SIG que apresenta um modelo de implementação com um conjunto de funcionalidades necessárias para que o *OpenGIS* seja a base para o desenvolvimento de um banco de dados geográficos totalmente interoperável.

Em SIG existe, além do problema da não interoperabilidade, uma grande preocupação com relação à qualidade e à integridade espacial dos dados georreferenciados. No escopo das aplicações geográficas, surgem problemas com os dados espaciais, devido a seus aspectos de localização na superfície da Terra com relação a outros objetos. Porém, a maioria dos softwares de SIG não oferece os meios adequados para representar e garantir a integridade espacial das informações. Muitos erros durante a entrada dos dados no banco de dados geográficos poderiam ser evitados, se o SIG fosse capaz de gerenciar a definição e a garantia de regras da realidade geográfica que precisam ser preservadas no sistema.

Restrições de integridade espacial podem expressar a obrigatoriedade ou a proibição de determinados relacionamentos espaciais entre tipos específicos de objetos, dentro de um banco de dados geográficos [SER 2000]. Elas são responsáveis por assegurar a integridade e a consistência da forma de representação espacial dos dados georreferenciados.

Na realidade geográfica, existem diversos tipos de restrições espaciais que podem ser considerados mais ou menos importantes, dependendo do contexto e do objetivo da aplicação. Grande parte das restrições espaciais é baseada em relacionamentos topológicos binários. Assim, a relação topológica entre dois objetos é a principal parte de uma restrição espacial [SER 2000]. Considerando a forma e a localização dos objetos, é possível computar todos os possíveis relacionamentos topológicos entre dois objetos espaciais. Considerando a semântica de cada um deles, é possível definir quais relacionamentos são consistentes e quais são inconsistentes.

Apesar de sua importância para a qualidade dos dados geográficos, a arquitetura de software proposta pelo OGC ainda não oferece um submodelo para a definição e a garantia de restrições espaciais, mesmo sendo este um aspecto tratado com grande interesse por autores de modelos conceituais para SIG.

Um SIG que suporta a definição e a garantia de regras da realidade geográfica que precisam ser conservadas no sistema, reduz, significativamente, o número de erros referentes à geometria e à topologia dos dados georreferenciados durante a fase de construção do banco de dados. Erros como, por exemplo, o armazenamento de duas edificações com as mesmas coordenadas, ou uma estrada atravessando uma edificação, ou ainda um rio e uma estrada localizados sobre as mesmas coordenadas, representam casos, como os apresentados no Anexo 2, que tendem a ser eliminados, aumentando assim, além da precisão e da confiabilidade, a qualidade e a integridade dos dados armazenados e das análises espaciais feitas a partir deles.

Com base em algumas das necessidades dos SIG contemporâneos e na busca do aumento da qualidade e da integridade dos dados geográficos, decidiu-se, então, pela extensão da arquitetura proposta pelo OGC, a fim de permitir a definição e a garantia de restrições que assegurem a correta forma geométrica e as relações topológicas na representação, em banco de dados, das entidades da realidade geográfica.

Este trabalho propõe um submodelo para a definição de restrições de integridade de caráter topológico, podendo ser estendido, em trabalhos futuros, incorporando suporte a outros tipos de restrição espacial encontrados no mundo real.

## 1.4 Objetivos da Dissertação

O principal objetivo da pesquisa realizada foi estender o padrão de interoperabilidade para software de geoprocessamento *OpenGIS*, dotando-o de suporte a restrições de integridade espacial. Mais precisamente, o trabalho buscou:

- a partir dos estudos de caso realizados na Primeira Divisão de Levantamento do Exército Brasileiro e no contexto do Programa Pró-Guaíba [GAR 98], identificar uma série de restrições espaciais relacionadas à geometria e à topologia das entidades geográficas, as quais precisam ser conservadas no sistema;
- definir, com base na literatura e nos estudos de caso, um subconjunto, suficientemente genérico, de tipos de restrição topológica, buscando expressar o maior número possível de regras da realidade geográfica;
- estudar o *Modelo Abstrato OpenGIS* e estendê-lo, de forma que suporte a definição, a verificação e a garantia de restrições de integridade espacial. Essa extensão envolve, desde a definição de novas classes, atributos e métodos, até a descrição detalhada, em inglês estruturado, dos métodos que garantem cada regra;
- validar o *Modelo Abstrato OpenGIS Estendido*.

## 1.5 Estrutura da Dissertação

O texto da dissertação está estruturado em quatro capítulos e quatro anexos, além das referências bibliográficas.

O Capítulo 2 apresenta uma introdução a restrições espaciais e os relacionamentos necessários para suportar a definição e a garantia das mesmas ao longo do processamento. Trata também de alguns métodos que definem, formalmente, um conjunto significativo de relacionamentos topológicos.

O Capítulo 3 tem por finalidade apresentar uma visão geral da arquitetura proposta pelo OGC e uma descrição detalhada de alguns submodelos do *Modelo Abstrato*, os quais foram a base para a incorporação de restrições topológicas. Neste capítulo também é apresentado o conjunto genérico necessário de tipos de restrições espaciais.

No Capítulo 4, é apresentada a proposta de extensão do *Modelo Abstrato OpenGIS*, oferecendo um subconjunto de regras topológicas que suportam o maior número possível de restrições da realidade geográfica. É apresentada, também, a forma como foi validado o conjunto de restrições incorporadas ao *Modelo Abstrato OpenGIS Estendido*.

O Capítulo 5 apresenta as considerações finais do trabalho, expondo suas contribuições e os resultados obtidos durante a pesquisa e implementação feitas neste trabalho. O texto aborda, também, questões que ficaram em aberto, indicando possíveis trabalhos futuros.

O Anexo 1 apresenta, com base na teoria dos conjuntos, uma série de combinações entre o limite e o interior dos objetos espaciais, através das quais é possível identificar alguns tipos de relacionamentos topológicos.

O Anexo 2 introduz alguns conceitos de cartografia básica e apresenta dois estudos de caso realizados para identificar uma série de restrições espaciais.

O Anexo 3 apresenta a estrutura de implementação (algoritmos) da extensão proposta nesta pesquisa.

No Anexo 4, estão algumas funções e comportamentos equivalentes aos das restrições espaciais incorporadas ao *Modelo Abstrato OpenGIS*, os quais foram implementados em um sistema de gerência de banco de dados geográficos real para validar o conjunto genérico de restrições espaciais apresentadas nesse trabalho.

# 2 Relacionamentos e Restrições Espaciais

A espacialidade da realidade geográfica é capturada na forma e na localização de entidades e fenômenos do mundo real (também denominados feições) e pelos relacionamentos espaciais que entre si apresentam [PAR 98].

Segundo Parent [PAR 98], os relacionamentos entre objetos espaciais são uma parte essencial da informação necessária para aplicações de gerenciamento de dados geográficos, onde grande parte deles podem ser deduzidos a partir da espacialidade dos objetos, que é representada pelas suas coordenadas. Dessa forma, os relacionamentos permanecem implícitos e podem ser acessados através de funções do SIG. No entanto, é importante que as relações espaciais possam ser explicitamente descritas nos modelos conceituais [CAM 96], para facultar a adição de atributos e métodos aos relacionamentos.

Certos tipos de relacionamento só são possíveis entre determinados objetos espaciais [BOR 97], pois são dependentes da sua forma geométrica. Por exemplo, a existência da relação *atravessa* pressupõe que, pelo menos, um dos objetos relacionados tenha a forma de uma linha.

Vários autores classificam os relacionamentos espaciais. Borges [BOR 97] agrupa-os em topológicos, métricos, de ordem e *fuzzy*. Parent [PAR 98] categoriza-os em topológicos, orientados (ou de ordem), métricos e de agregação. Faria [FAR 98] classifica-os em métricos, topológicos, de orientação e de localização.

Os relacionamentos de orientação ou direcionais [FAR 98, BOR 97] verificam se existe alguma relação de orientação entre dois objetos geométricos. Eles descrevem a ordem como os objetos estão posicionados uns em relação aos outros [CAM 96], partindo, normalmente, de um marco de referência, que determina a direção na qual o objeto está localizado. Exemplos desse tipo de relacionamento [FAR 98] são: *acima, abaixo, ao norte, ao sul, a leste, a oeste, à esquerda* e *à direita.*

Métricos são os relacionamentos de distância que descrevem quão afastado um objeto está em relação a outro ou a um marco de referência [CLE 94]. Eles dependem de definições métricas no sentido de parametrizar o *quanto* é perto ou longe, o que dependerá das circunstâncias e das entidades geográficas relacionadas [BOR 97]. Exemplos de relacionamentos métricos são: *distância* e *comprimento.*

Os relacionamentos *fuzzy* são aqueles que tratam de entidades geográficas que não possuem limites bem definidos. Um lago, por exemplo, pode ter sua área bastante modificada entre uma estação de seca e uma estação de chuvas. Muitos riachos chegam a desaparecer durante períodos de seca [LIS 97]. Pesquisas estão sendo realizadas nesta área, mas nenhum dos modelos para dados geográficos abordados na Seção 2.3 consideram esse tipo de problema.

Relacionamentos topológicos são aqueles que se referem à posição relativa dos objetos no espaço onde estão contidos. Eles descrevem se dois objetos interceptam-se ou não, e qual a forma de interseção existente [CLE 94]. Esse tipo de relacionamento espacial é preservado sob transformações como rotação, escala e translação [EGE 94].

Na literatura, podem ser encontrados vários métodos que buscam definir um conjunto significativo de relacionamentos topológicos.

Relacionamentos topológicos binários são aqueles que tratam de relações topológicas entre dois objetos espaciais. Exemplos incluem *disjunção, adjacência* e *continência.*

Este capítulo apresenta, na Seção 2.1, alguns métodos que buscam definir um conjunto de relacionamentos topológicos binários. A Seção 2.2 introduz restrições espaciais. A Seção 2.3 apresenta alguns modelos de dados que suportam relacionamentos e restrições espaciais. A Seção 2.4 aborda os estudos de caso realizados e a Seção 2.5 apresenta a proposta de um conjunto genérico necessário dos tipos de restrição espacial que são considerados neste trabalho.

## 2.1 Métodos de Representação de Relacionamentos Topológicos Binários

Na literatura podem ser encontradas várias tentativas para descrever um conjunto significativo de relacionamentos topológicos, porém é difícil encontrar uma definição formal para eles. As próximas subseções descrevem, sucintamente, quatro métodos utilizados para descrever formalmente um conjunto de relações espaciais. Esses métodos apresentam relacionamentos topológicos binários e são baseados na teoria dos conjuntos.

Os quatro métodos tratam pontos, linhas e polígonos como objetos simples. Cada feição (seja ela representada por um ponto, uma linha ou um polígono) é tratada como um conjunto fechado contendo todos os pontos que formam a sua própria fronteira/limite. Um polígono, por exemplo, não é definido por sua área, mas pelo conjunto de todos os pontos que formam o seu limite e o seu interior. Por isso, os seguintes aspectos devem ser considerados [CLE 93]:

- feições representadas por polígonos/áreas/regiões/superfícies têm o limite conectado, não podendo ser a união de conjuntos disjuntos de pontos (mais de uma feição) e nem conter orifícios;
- feições representadas por linhas não se auto-interceptam, podendo ser circulares, onde o ponto inicial e o final são o mesmo, ou ter apenas dois pontos terminais (um inicial e um final). O interior de uma linha é formado por todos os pontos contidos entre seus pontos inicial e final;
- feições na forma de ponto podem conter apenas um ponto e não possuem interior.

### 2.1.1 O Método 4-*Intersection*

Max Egenhofer, em [EGE 91, EGE 93, EGE 94, EGE 95], define uma série de relacionamentos topológicos binários entre objetos espaciais bidimensionais (áreas). Baseado em uma característica particular desses relacionamentos, que é a relação entre o limite e o interior dos objetos, Egenhofer [EGE 91, EGE 93] define o relacionamento

topológico entre dois objetos espaciais A e B, através da interseção do limite e do interior de A com o limite e o interior de B.

O limite e o interior dos objetos podem ser combinados para formar os quatro critérios fundamentais dos relacionamentos topológicos entre dois objetos espaciais [EGE 93]. As combinações possíveis são representadas pela interseção entre o limite e o interior de ambos os objetos, originando uma *matriz M de* quatro interseções, denominada *4-Intersection,* conforme ilustra a figura 2.1. O limite do objeto é representado pelo símbolo ∂; o interior, pelo símbolo ° e a interseção, pelo símbolo ∩.

$$M = \begin{pmatrix} \partial A \cap \partial B & \partial A \cap B^{\circ} \\ A^{\circ} \cap \partial B & A^{\circ} \cap B^{\circ} \end{pmatrix}$$

FIGURA 2.1 – Matriz de quatro interseções

Para constatar se existe interseção entre dois objetos espaciais, é necessário verificar se o conjunto definido em cada entrada da matriz é vazio ou não. Utilizando os valores binários vazio (∅) e não vazio (¬∅), são obtidos um total de dezesseis combinações diferentes, as quais fornecem a base para a definição formal de oito relacionamentos topológicos possíveis[EGE 91]. Veja tabela 2.1.

TABELA 2.1 – Dezesseis especificações dos relacionamentos topológicos binários baseadas na interseção vazia ou não entre o limite e o interior de dois objetos

| Caso | ¶A Ç ¶B | A° Ç °B | ¶A Ç °B | A° Ç ¶B | Nome do Relacionamento |
|---|---|---|---|---|---|
| **1** | ∅ | ∅ | ∅ | ∅ | A disjoint B |
| **2** | ∅ | ¬∅ | ∅ | ∅ | ____ |
| **3** | ¬∅ | ∅ | ∅ | ∅ | A touches B |
| **4** | ¬∅ | ¬∅ | ¬∅ | ¬∅ | A overlaps B |
| **5** | ∅ | ∅ | ¬∅ | ¬∅ | ____ |
| **6** | ¬∅ | ¬∅ | ∅ | ∅ | A equal B |
| **7** | ∅ | ¬∅ | ∅ | ¬∅ | A contains B |
| **8** | ∅ | ∅ | ¬∅ | ∅ | ____ |
| **9** | ∅ | ¬∅ | ¬∅ | ∅ | A inside B |
| **10** | ∅ | ¬∅ | ¬∅ | ¬∅ | ____ |
| **11** | ∅ | ∅ | ∅ | ¬∅ | ____ |
| **12** | ¬∅ | ∅ | ∅ | ¬∅ | ____ |
| **13** | ¬∅ | ¬∅ | ∅ | ¬∅ | A covers B |
| **14** | ¬∅ | ∅ | ¬∅ | ∅ | ____ |
| **15** | ¬∅ | ¬∅ | ¬∅ | ∅ | A covered by B |
| **16** | ¬∅ | ∅ | ¬∅ | ¬∅ | ____ |

A representação gráfica de cada um dos dezesseis casos apresentados na tabela 2.1 é ilustrada no Anexo 1, buscando facilitar a visualização dos possíveis tipos de relacionamentos topológicos entre dois objetos representados na forma de polígonos.

Dentre as dezesseis especificações entre duas regiões, nem todas são consistentes, sendo que algumas violam propriedades das regiões simples [CAM 96, EGE 91]. A combinação na segunda linha da tabela 2.1, por exemplo, é desconsiderada,

pois, se o interior de A e B se interceptam, o limite de A deveria interceptar o interior de B, ou vice-versa, porém isso não acontece, tornando a especificação inválida. Dessa forma, apenas oito combinações podem ser consideradas, resultando em oito relacionamentos válidos entre duas regiões: *disjoint, touches, overlaps, equal, inside, contains, covers* e *coveredBy*. Esses relacionamentos são ilustrados, graficamente, na figura 2.2 e descritos detalhadamente, a seguir, com base nas definições de Egenhofer:

- *disjoint* – duas regiões estão espacialmente separadas/disjuntas quando todas as quatro interseções entre o limite e o interior de ambas resultarem num conjunto vazio;
- *touches* – se o resultado da interseção entre os limites das duas regiões não for vazio e as outras três forem, então os dois objetos se encontram/tocam;
- *overlaps* – duas regiões estão sobrepostas quando as quatro interseções entre o limite e o interior de ambas resultarem num conjunto não vazio, ou seja, ambas compartilham parte de seu interior e seu limite, bem como, o limite de A intercepta o interior de B e vice-versa;
- *equal* – duas regiões são iguais quando ocorrer interseção entre o limite e entre o interior das mesmas;
- *inside* – uma região A está dentro de uma região B quando: A e B compartilham o mesmo interior mas não o limite, e o limite de A é um subconjunto do interior de B, e o limite de B não intercepta qualquer parte do interior de A;
- *contains* – uma região A contém uma região B, se A e B compartilham o mesmo interior, mas não têm limites em comum, e, se o limite de B é um subconjunto do interior de A, e, nenhuma parte do limite de A intercepta qualquer parte do interior de B;
- *covers* – uma região A cobre uma região B, se ambas compartilham parte de seu limite e interior, e, se o interior de B intercepta o limite de A, e, nenhuma parte do interior de A intercepta o limite de B;
- *coveredBy* – uma região A é coberta por uma região B, se ambas tiverem parte de seus limites e interior em comum, e, se parte do interior de A intercepta parte do limite de B, e, se o interior de B não intercepta o limite de A.

Clementini [CLE 93] trata apenas seis, dos relacionamentos topológicos binários entre regiões definidos por Egenhofer [EGE 91], pois considera as relações *inside*/*contains* e *covers*/*coveredBy* simétricas, nomeando-as, respectivamente, como *in* e *covers.*

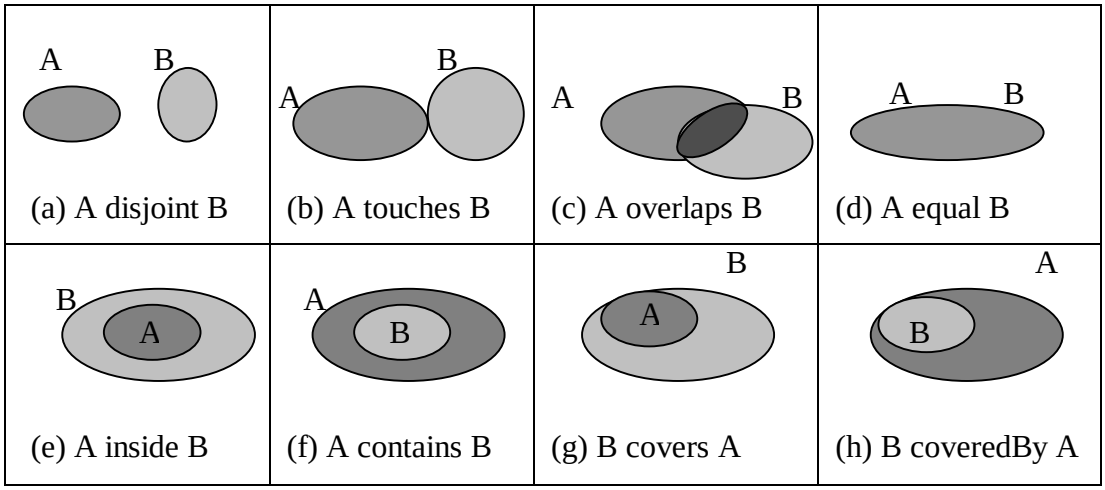



FIGURA 2.2 – Representação gráfica de relacionamentos entre duas regiões A e B num espaço bidimensional

Em [HAD 92], as combinações entre o limite e o interior de dois objetos do tipo área, especificadas pelo método *4-Intersection*, são aplicadas em relações espaciais de objetos com a forma geométrica de ponto, linha e área, resultando em seis grupos de relacionamentos topológicos binários: *área/área*, *linha/área*, *ponto/área*, *linha/linha*, *linha/ponto* e *ponto/ponto*, conforme ilustra a tabela 2.2.

Por exemplo, duas *áreas* que têm um *ponto* em comum, duas *áreas* que têm uma *linha* inteira em comum e uma *área* e uma *linha* que têm um *ponto* em comum, resultam no mesmo relacionamento *touches*, pois a interseção do limite dos objetos é um conjunto não vazio e as demais combinações do limite e do interior dos objetos envolvidos formam um conjunto vazio.

As oito relações válidas encontradas para as combinações entre o limite e o interior de dois objetos espaciais do tipo ponto, linha e polígono, são praticamente as mesmas definidas por Egenhofer [EGE 91] entre duas regiões (veja figura 2.2). A diferença é a relação entre dois objetos *linha/linha* e *linha/área*, que originou um novo relacionamento topológico [HAD 92], denominado *crosses*, em que uma linha pode cruzar outra linha ou uma área (veja tabela 2.2). A tabela 2.2 apresenta todas as combinações entre o limite e o interior de pontos (P), linhas (L) e regiões (R) encontradas por Hadzilacos [HAD 92], através do método 4-*Intersection*.

As contribuições de Hadzilacos para o método *4-Intersection*, criado por Egenhofer, permitem que o *4-Intersection* seja utilizado para verificar o tipo de relação topológica existente entre dois objetos espaciais, tenham eles a forma geométrica de um ponto, uma linha ou um polígono.

TABELA 2.2 – Relacionamentos topológicos binários entre pontos, linhas e áreas (fonte obtida em [HAD 92])

| | δδ oo δo oδ | R-R | R-L | R-P | L-L | L-P | P-P |
|---|---|---|---|---|---|---|---|
| r1 | ∅ ∅ ∅ ∅ |  |  |  |  |  |  |
| r2 | ¬∅ ∅ ∅ ∅ |  |  |  |  |  |  |
| r3 | ∅ ¬∅ ∅ ∅ | n | n | - |  | - | - |
| r4 | ¬∅ ¬∅ ∅ ∅ |  |  | - |  | - | - |
| r5 | ∅ ∅ ¬∅ ∅ | n | n | - |  | - | - |
| r6 | ¬∅ ∅ ¬∅ ∅ | n | n | - |  | - | - |
| r7 | ∅ ¬∅ ¬∅ ∅ |  |  | - |  | - | - |
| r8 | ¬∅ ¬∅ ¬ ∅ ∅ |  |  | - |  | - | - |
| r9 | ∅ ∅ ∅ ¬ ∅ | n |  |  |  | | - |
| r10 | ¬∅ ∅ ∅ ¬ ∅ | n |  | n |  | n | - |
| $r_{11}$ | ∅ ¬∅ ∅ ¬ ∅ |  |  | - |  | - | - |
| $r_{12}$ | ¬∅ ¬∅ ∅ ¬ ∅ |  |  | - |  | - | - |
| $r_{13}$ | ∅ ∅ ¬∅ ¬ ∅ | n | n | - |  | - | - |
| $r_{14}$ | ¬∅ ∅ ¬∅ ¬∅ | n | n | - |  | - | - |
| $r_{15}$ | ∅ ¬∅ ¬∅ ¬∅ | n |  | - |  | - | - |
| $r_{16}$ | ¬∅ ¬∅ ¬∅ ¬∅ |  |  | - |  | - | - |

n: impossível no espaço bidimensional; -: não é aplicável porque um ponto não tem interior

### 2.1.2 O Método DEM (*Dimension Extended Method*)

O método DEM, segundo Clementini [CLE 93], pode ser considerado uma extensão do *4-Intersection*, pois a classificação dos relacionamentos topológicos binários em ambos os métodos é baseada na interseção do limite e do interior de duas feições.

A partir da extensão proposta em [HAD 92], onde o método *4-Intersection* é utilizado para combinar feições representadas por pontos, linhas e áreas, Clementini [CLE 93] apresenta uma segunda extensão, na qual considera a dimensão *dim* da interseção ao invés de apenas distinguir o resultado da combinação entre dois objetos como um conjunto vazio ou não. Para ilustrá-la, Clementini [CLE 93] apresenta o relacionamento topológico *linha/área*, onde a linha é representada por L e a área por A.

Num espaço bidimensional, a interseção pode resultar num conjunto S igual a ∅ (vazio), 0D (ponto), 1D (linha) ou 2D (área). Essas quatro possibilidades podem resultar em 256 ($4^4$) combinações diferentes, mas apenas as citadas abaixo são aplicáveis:

S1 = $\partial A \cap \partial L : \varnothing$, ou 0D ( 2 casos )
S2 = $\partial A \cap L^\circ : \varnothing$, ou 0D, ou 1D ( 3 casos )
S3 = $A^\circ \cap \partial L : \varnothing$, ou 0D ( 2 casos )
S4 = $A^\circ \cap L^\circ : \varnothing$, ou 1D ( 2 casos )

A matriz usada para representar a interseção entre o limite e o interior de dois objetos, conforme ilustra a equação abaixo, apenas indica se o resultado da interseção é vazio ou não. Ela não expressa a natureza da interseção, o que pode causar problemas na caracterização do fenômeno [CIL 96]. No exemplo ilustrado pela figura 2.3, existem dois tipos de interseção entre dois objetos do tipo linha (L) e área (A). Ambos têm a mesma matriz de 4 interseções. No entanto, no caso à esquerda, de acordo com o conceito de [EGE 91], trata-se de um relacionamento de cobertura (*coveredBy*), sendo que o objeto linha é coberto pelo objeto área. Já no caso à direita, de acordo com [HAD 92], o relacionamento é de cruzamento (*crosses*), pois o objeto linha atravessa o limite do objeto área. Para eliminar tal ambiguidade, Clementini [CLE 93] criou o método da dimensão estendida, onde aprimorou a matriz de 4 interseções para considerar a dimensão da interseção entre dois objetos quaisquer de uma, duas ou três dimensões. No caso da figura 2.3, a dimensão *dim* é igual a *1* para o caso à esquerda e *dim* é igual a *0* para o caso à direita.

$$\begin{pmatrix} \partial A \cap \partial L & \partial A \cap L^\circ \\ A^\circ \cap \partial L & A^\circ \cap L^\circ \end{pmatrix} = \begin{pmatrix} \neg 0 & \neg\ 0 \\ 0 & \neg 0 \end{pmatrix}$$

A dimensão do limite e do interior de uma área A e uma linha L correspondem, respectivamente [CLE 93]: *dim*(∂A) = 1, *dim*(A°) = 2, *dim*(∂L) = 0, *dim*(L°) = 1. A dimensão da interseção não pode ser maior que a menor dimensão dos dois objetos envolvidos na combinação.

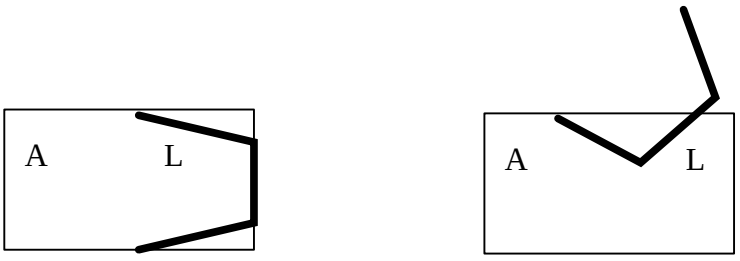



FIGURA 2.3 – Relacionamentos *coveredBy* e *crosses* relativos à mesma matriz

Detalhes sobre o método DEM podem ser encontrados em [CLE 93, CLE 94].

### 2.1.3 O Método CBM (*Calculus-Based Method*)

Em [CLE 93], é introduzido o método CBM, que é baseado em cálculo. Através desse método, Clementini [CLE 93, CLE 94] busca provar que os relacionamentos *disjoint, touches, overlaps, in e crosses* são mutuamente exclusivos, constituem uma cobertura completa de todas as situações topológicas obtidas através do método DEM e podem ser expressos de forma mais simplificada, facilitando a compreensão por parte do usuário.

Os relacionamentos definidos por Clementini [CLE 93, CLE 94] através do método CBM são detalhados abaixo. A expressão ao lado esquerdo do sinal de equivalência ($\Leftrightarrow$) representa o relacionamento topológico, e sua definição é dada ao lado direito, sob a forma de uma expressão equivalente a um conjunto de pontos. O símbolo $\lambda$ é usado para representar uma feição com qualquer forma geométrica simples (ponto, linha ou área). A notação ($\lambda_1$, r, $\lambda_2$), no cálculo, indica que as feições $\lambda_1$ e $\lambda_2$ estão envolvidas no relacionamento r. Ela pode ser combinada com os operadores boleanos *and* ($\wedge$) e *or* ($\vee$). A notação $\lambda^\circ$ corresponde ao interior da feição geográfica. A notação $\lambda$ representa o limite da feição quando a expressão envolve testes com o interior da mesma. Quando o interior não é testado na expressão, a notação $\lambda$ representa ambos, o limite e o interior da feição geográfica (como é o caso do relacionamento *disjoint*).

- *Disjoint*: duas feições são disjuntas quando não existe nenhum ponto de interseção entre elas, ou seja, a interseção entre ambas é um conjunto vazio. O relacionamento de disjunção pode ser aplicado em todas as combinações entre pontos, linhas e polígonos. A definição desse relacionamento, no cálculo, é dada por:

$$(\lambda_1, \textit{disjoint}, \lambda_2) \Leftrightarrow \lambda_1 \cap \lambda_2 = \varnothing$$

- *Touches*: duas feições tocam-se quando a única coisa em comum entre elas é a união de parte de seus limites. Esse relacionamento é aplicado às combinações de polígono/polígono, linha/linha, linha/polígono, ponto/polígono e ponto/linha. A definição desse relacionamento, no cálculo, é dada por:

$$(\lambda_1, \textit{touches}, \lambda_2) \Leftrightarrow (\lambda^\circ_1 \cap \lambda^\circ_2 = \varnothing) \wedge (\lambda_1 \cap \lambda_2 \neq \varnothing).$$

- *Overlaps*: segundo Clementini [CLE 93], duas feições estão sobrepostas se o resultado da interseção entre ambas for uma terceira feição com a mesma dimensão, porém diferente das duas sobrepostas. Isso ocorre porque o relacionamento de sobreposição é aplicado somente em combinações de objetos geométricos homogêneos [CLE 93] de linha/linha e polígono/polígono. A definição desse relacionamento, no cálculo, é dada por:

$$(\lambda_1, \textit{overlaps}, \lambda_2) \Leftrightarrow (\textit{dim}(\lambda^\circ_1) = \textit{dim}(\lambda^\circ_2) = \textit{dim}(\lambda^\circ_1 \cap \lambda^\circ_2)) \wedge (\lambda_1 \cap \lambda_2 \neq \lambda_1) \wedge (\lambda_1 \cap \lambda_2 \neq \lambda_2).$$

- *In/Inside*: uma feição está dentro de outra se forem de tamanhos diferentes e uma estiver totalmente contida dentro da outra. Esse tipo de relacionamento é aplicado em todas as combinações dos elementos geométricos, exceto à combinação ponto/ponto. Sua definição, no cálculo, é dada como:

$$(\lambda_1, \textit{in}, \lambda_2) \Leftrightarrow (\lambda^\circ_1 \cap \lambda^\circ_2 \neq \varnothing) \wedge (\lambda_1 \cap \lambda_2 = \lambda_1).$$

- *Crosses*: este tipo de relacionamento é aplicável em situações linha/linha e linha/polígono [CLE 93]. Uma linha cruza outra quando elas têm um ponto de seu interior em comum. Uma linha atravessa ou cruza um polígono se ela estiver parcialmente dentro e parcialmente fora da área do polígono. Esse tipo de relacionamento difere do *touches*, porque, no último, apenas os pontos que formam o limite da geometria interceptam-se. Sua definição, no cálculo, é dada por:

$$(\lambda_1, \textit{crosses}, \lambda_2) \Leftrightarrow \textit{dim}(\lambda^\circ_1 \cap \lambda^\circ_2) = (\textit{max}(\textit{dim}(\lambda^\circ_1), \textit{dim}(\lambda^\circ_2)) - 1) \wedge (\lambda_1 \cap \lambda_2 \neq \lambda_1) \wedge (\lambda_1 \cap \lambda_2 \neq \lambda_2).$$

A representação dos relacionamentos topológicos, através do cálculo, torna o método mais expressivo que os demais, pois, para cada relação é possível definir uma equivalência correspondente a cada termo básico do método DEM.

Em geral, o cálculo de relacionamentos permite definir as relações topológicas com precisão, sem introduzir um extenso número de conceitos. Porém, Clementini [CLE 93] não apresenta, no método CBM, todos os relacionamentos possíveis e definidos através do método 4-*Intersection*, como é o caso dos relacionamentos *equal* e *covers*.

### 2.1.4 O Método 9-*Intersection*

O método 9-*Intersection* [CLE 94] também é uma extensão do 4-*Intersection*, considerando, entretanto, além do limite e do interior, o exterior das feições, originando um modelo amplo para relacionamentos topológicos binários entre pontos, linhas e polígonos.

O relacionamento topológico binário entre dois objetos, A e B, é baseado na interseção ($\cap$) do interior ($A^\circ$), limite ($\partial A$) e exterior ($A^-$) de A com o interior ($B^\circ$), limite ($\partial B$) e exterior ($B^-$) de B [CLE 94]. As nove interseções entre as três partes dos objetos descrevem relacionamentos topológicos e podem ser, concisamente, representadas por uma matriz M de 3x3, denominada *9-Intersection*, conforme ilustra a figura 2.4.

$$M = \begin{pmatrix} A^\circ \cap B^\circ & A^\circ \cap \partial B & A^\circ \cap B^- \\ \partial A \cap B^\circ & \partial A \cap \partial B & \partial A \cap B^- \\ A^- \cap B^\circ & A^- \cap \partial B & A^- \cap B^- \end{pmatrix}$$

FIGURA 2.4 – Matriz de nove interseções

Considerando os valores vazio ($\varnothing$) e não vazio ($\neg\varnothing$), podem ser feitas 512 ($2^9$) combinações entre o limite, o interior e o exterior de pontos, linhas e áreas, onde da mesma forma como no método *4-Intersection,* nem todas as combinações originam uma relação topológica válida. Mas por considerar além do limite e do interior, o exterior das feições relacionadas, o método *9-Intersection* permite obter maiores detalhes sobre a forma como os objetos estão interligados, permitido, por exemplo, conhecer todas as formas como um objeto do tipo *linha*, atravessa um objeto do tipo *área*.

Os métodos *4-Intersection* e *9-Intersection* originam o mesmo conjunto de oito tipos de relação topológica entre dois objetos do tipo *área* (veja figura 2.2). Contudo, o método *9-Intersection* permite identificar maiores detalhes do relacionamento. Para as combinações *linha/área*, por exemplo, o método *4-Intersection* distingue apenas onze ligações válidas (veja tabela 2.2), enquanto o *9-Intersection* apresenta dezenove ligações diferentes em combinações *linha/área*, conforme ilustra a figura 2.5.

O relacionamento *linha/linha,* baseado no método 9-*Intersection*, por exemplo, também apresenta um conjunto bem maior de combinações ou interpretações geométricas entre duas linhas simples, ilustrando assim, as diversas formas como uma

linha pode interceptar outra [EGE 95a]. A representação gráfica desse exemplo e detalhes sobre esse método estão em [EGE 93, EGE 93a, EGE 95a, CLE 94].

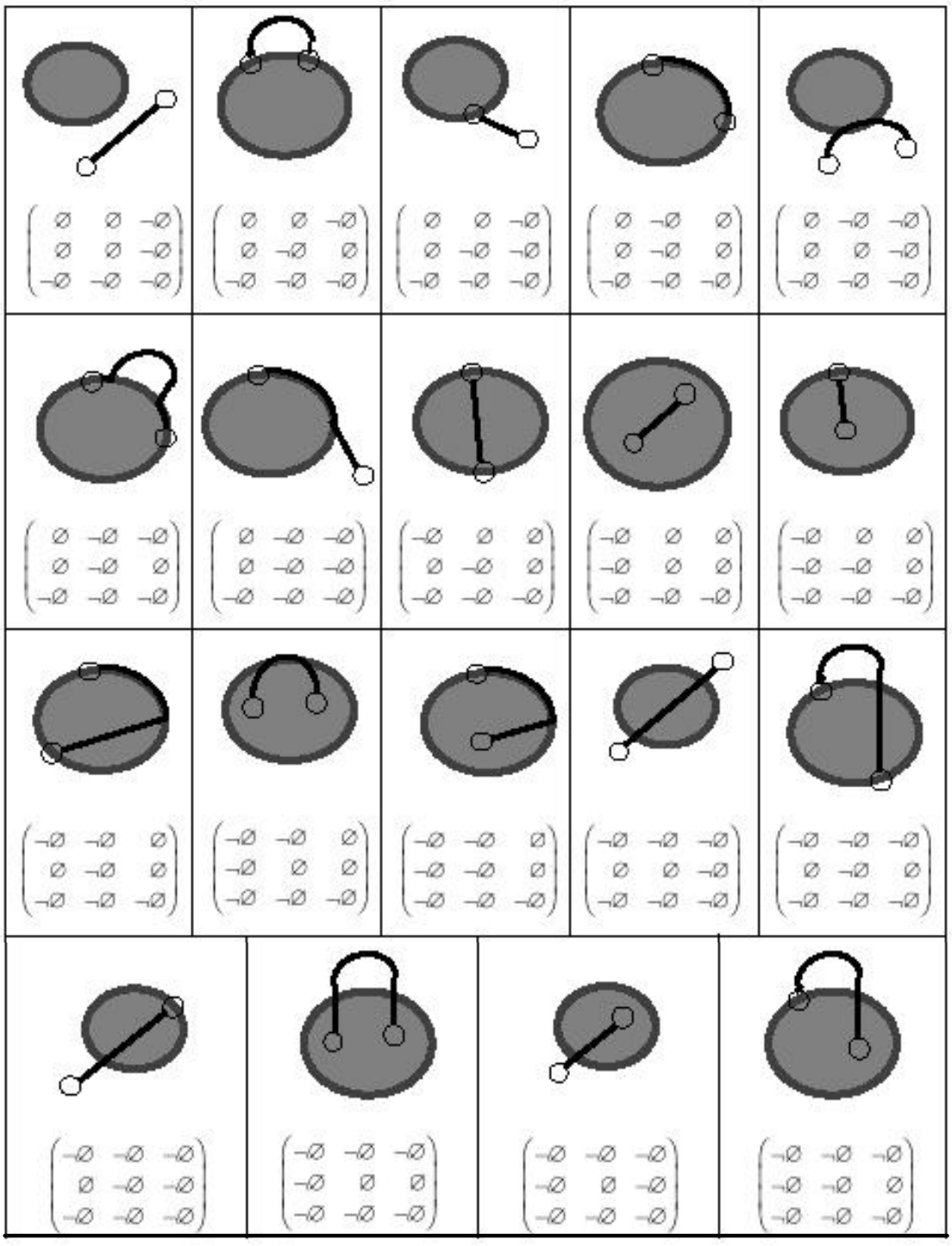


FIGURA 2.5 - Interpretação geométrica de dezenove relações *linha/área* que podem ser obtidas pelo método *9-Intersection*

Fonte obtida em [EGE 95a]

## 2.2 Restrições Espaciais

Segundo Borges [BOR 97], muitas aplicações geográficas manipulam dados dependentes de relacionamentos topológicos, os quais precisam ser respeitados e representados, explicitamente, no banco de dados. Nesses casos, cuidados especiais devem ser tomados para que a consistência espacial seja mantida. Esses cuidados interferem não só na entrada inicial dos dados geográficos, como na manutenção da integridade semântica do banco de dados durante sua fase de produção.

Algumas restrições de integridade espacial podem ser dependentes desses relacionamentos topológicos. A restrição de que *todo estado deve estar contido dentro de um e somente um país,* por exemplo, está diretamente associada à obrigatoriedade do relacionamento de continência (*contains*) entre objetos de tipo *estado* e *país*. Já, a regra *uma estrada não pode atravessar uma construção* está diretamente associada à proibição do relacionamento *crosses* entre objetos de tipo *estrada* e *construção.* Assim, pode-se dizer que relacionamentos espaciais são a base das restrições espaciais [SER 2000], pois podem representar a obrigatoriedade ou a proibição de determinados relacionamentos entre tipos específicos de objetos dentro de um banco de dados geográficos.

A especificação de restrições espaciais deve ocorrer na fase de projeto do banco de dados [COC 97]. Sua verificação e garantia deve ocorrer na primeira carga dos dados no sistema assim como em operações de atualização da base de dados.

Os principais tipos de restrição de integridade que ocorrem na modelagem de dados são convencionais, como, por exemplo, restrições para o domínio de atributos, restrições para a estrutura de entidades, relacionamentos e restrições de integridade semântica em geral [BOR 99]. Cockcroft, em [COC 97], estende essa classificação para atender às peculiaridades dos dados espaciais. A nova classificação obtida está baseada na distinção entre regras topológicas, semânticas e de usuários, todas referentes à espacialidade dos dados geográficos.

Restrições de integridade topológica referem-se às propriedades geométricas e aos relacionamentos espaciais de feições geográficas. Elas podem ser aplicadas a entidades e relacionamentos específicos da aplicação, fornecendo a base para o controle da integridade. A divisão política dos estados pode ser usada como exemplo desse tipo de restrição.

As restrições de integridade semântica preocupam-se com o significado das feições geográficas. Um exemplo desse tipo de restrição pode ser uma regra, declarando que o usuário não pode inserir uma feição do tipo *estrada* e outra do tipo *rio,* localizadas nas mesmas coordenadas, já que elas não podem estar espacialmente localizadas uma sobre a outra;

Restrições de integridade definidas pelo usuário permitem que a consistência da base de dados seja mantida de acordo com as necessidades das aplicações, e podem ou não, ser baseadas em semântica. Esse tipo de restrição pode, por exemplo, exigir que, por razões legais, uma estação de energia nuclear esteja à uma determinada distância de zonas residenciais. Na tentativa de inserir, na base de dados, um caso em que isso não ocorra, uma ação programada pelo usuário pode ser ativada.

Na realidade geográfica, podem ser encontrados muitos outros tipos de restrição espacial, além dos citados por Cockcroft. Essas restrições podem ser consideradas mais ou menos importantes, dependendo do contexto e do objetivo da aplicação. Entre elas podem ser citadas:

- restrições métricas (baseadas em relacionamentos métricos);
- de orientação (fundamentadas em relacionamentos de ordem ou direção);
- restrições relacionadas à geometria dos objetos como, por exemplo, *um polígono precisa ser fechado* ou *isolinhas devem ser concêntricas*;
- restrições *fuzzy* relacionadas a objetos que não possuem uma forma espacial bem definida, como é o caso de rios intermitentes ou lagos que podem até desaparecer em épocas de seca;
- restrições associadas a objetos com orifícios internos.

## 2.3 Modelos de Dados que Suportam Relacionamentos e Restrições Espaciais

Um *modelo de dados* é um conjunto de ferramentas conceituais utilizado para descrever como a realidade geográfica será representada no sistema computacional [CAM 96a]. Ele é um aspecto fundamental no projeto de um SIG, pois nenhuma outra decisão limita tanto a abrangência e o crescimento futuro do sistema quanto a escolha do modelo de dados.

A modelagem conceitual oferece importantes vantagens para aplicações que manipulam dados espaço-temporais. Através dela, o usuário pode expressar seu conhecimento sobre a aplicação, usando definições que estão mais próximas da sua realidade, independentemente dos conceitos do computador. Assim, a modelagem conceitual é independente da ferramenta de software que será utilizada na implementação do banco de dados, de forma que a modelagem do projeto continua válida, mesmo havendo uma troca de tecnologia (somente a transformação do modelo conceitual para o lógico é afetada).

A maioria dos modelos conceituais de dados geográficos prevêem a definição de relacionamentos espaciais, tanto os métricos como os de orientação e os topológicos. Entre estes modelos estão o OMT-G [BOR 99], o MADS [PAR 98], o GeoOOA [KOS 95] e o GMOD [PIR 97].

Os softwares de SIG, normalmente, oferecem funções que armazenam ou calculam determinados tipos de relacionamentos espaciais entre feições geográficas, com a finalidade de facilitar o processamento de consultas e análises espaciais sobre os dados georreferenciados. Exemplos de softwares de SIG que incluem essas informações são o Oracle 8i [HEB 99] e o Gothic [ROA 99]. Tais softwares, porém, não permitem a definição e não garantem automaticamente a manutenção de restrições espaciais baseadas nos relacionamentos que armazenam.

A garantia de restrições espaciais através do SIG pode reduzir, significativamente, o número de erros e inconsistências referentes à geometria e à topologia dos dados georreferenciados durante a construção e a atualização da base de

dados geográficos. Porém, a maioria das atuais arquiteturas de software para SIG não oferecem os meios adequados para traduzir as restrições espaciais, definidas na modelagem conceitual da aplicação, para seus respectivos modelos lógicos ou físicos.

No *ORACLE 8i* e no *GOTHIC*, por exemplo, restrições espaciais podem ser definidas através da programação de *triggers* ou métodos de classes. Contudo, essa tarefa pode ser muito trabalhosa para o projetista do banco de dados, devido ao grande número de regras da realidade geográfica que precisam ser garantidas.

Diversos trabalhos de pesquisa vêm sendo desenvolvidos com o objetivo de obter um modelo conceitual capaz de representar a complexa realidade geográfica das aplicações de SIG [HAD 92, PAR 98, BOR 97]. Outros, buscam incorporar, aos modelos conceituais existentes, um conjunto de relacionamentos topológicos. Modelos de dados lógicos também são propostos. As Seções 2.3.1, 2.3.2 e 2.3.3 abordam, resumidamente e a título de ilustração, algumas propostas de modelos conceituais e lógicos para aplicações geográficas, os quais suportam relacionamentos e/ou restrições espaciais.

### 2.3.1 O Modelo Conceitual MADS

O *Modeling of Aplication Data with Spatio-Temporal features* (MADS) [PAR98] é uma proposta de modelo de dados conceitual desenvolvida pela *University of Lausanne* e o *Laboratoriere de Base de Données,* da Escola Politécnica Federal de Lausanne na Suíça, proposta esta elaborada com base nas características encontradas em aplicações práticas estudadas.

Dentre as quatro classificações de relacionamentos espaciais definidas por Parent [PAR 98] (conforme apresentado no início deste capítulo), o MADS oferece apenas duas categorias: os relacionamentos topológicos e as agregações espaciais, as quais, segundo a autora, correspondem às exigências mais comuns das aplicações geográficas.

Os relacionamentos topológicos oferecidos pelo modelo incluem *disjoint, touches, crosses, overlaps* e *contains.*

Segundo Parent, a agregação espacial é muito comum entre aplicações espaço-temporais. No MADS, a agregação é uma ligação binária direcionada do tipo de objeto composto para o tipo componente. É comum que alguns atributos do objeto composto ou componente estejam relacionados [PAR 98]. Essas ligações podem ser representadas por atributos derivados como, por exemplo, *a população de um município é igual a soma da população dos distritos deste município*. Podem também ser expressas por restrições de integridade como, por exemplo, *as áreas dos distritos de um município precisam estar conectadas,* ou *a área de uma casa não pode ser maior que a área do lote onde ela se localiza*.

### 2.3.2 O Modelo *OMT-G*

O *Geo-OMT* é um modelo de dados geográficos conceitual, proposto por Borges em 1997 [BOR 97], com base no modelo de objetos *OMT*. A opção de estender o modelo *OMT* ocorreu devido a sua capacidade de representar aspectos semânticos de

uma aplicação, oferecer a abordagem de orientação a objetos e, também, ao seu amplo uso na modelagem de aplicações geográficas.

Em 1999, o modelo conceitual *Geo-OMT* foi aprimorado em alguns aspectos [BOR 99] e sofreu uma mudança de nome. O modelo passou a ser chamado de *OMT-G*.

O modelo *OMT-G* oferece suporte a um grande conjunto de relacionamentos espaciais. Esse conjunto inclui, além dos relacionamentos topológicos originados pelo método *4-Intersection* (*disjoint, touches, overlaps, equal, inside, contains, covers, coveredBy e crosses*), alguns tipos de relacionamentos métricos e de ordem (perto de, acima/abaixo, sobre/sob, entre, coincide, em frente a, à esquerda e à direita).

Em [BOR 99], o modelo *OMT-G* oferece suporte a uma série de tipos de restrição espacial como: regras de dependência espacial, regras de disjunção e regras de conectividade.

As regras de dependência espacial são restrições impostas pela existência de objetos agregados, onde a natureza do objeto agregado depende da existência de outro objeto geométrico (um sub-objeto). Segundo Borges [BOR 99], essas regras correspondem às primitivas espaciais de subdivisão, união e continência. Por exemplo, a geometria do objeto que contém (*país*) deve conter a geometria dos objetos contidos (*estados*).

As regras de disjunção são importantes na manutenção da integridade em relação à entrada de dados [BOR 97]. Por exemplo, um *trecho de rua* é disjunto de uma *edificação*. Isso implica que não pode existir nenhum trecho que cruze uma edificação. No momento da criação do trecho ou da edificação, esta regra deve ser observada.

As regras de conectividade normalmente são garantidas pelo SIG. No caso de uma rede de esgoto, por exemplo, a conexão entre o nó e o segmento é garantida automaticamente pelo sistema [BOR 97].

Maiores detalhes sobre os tipos de restrições espaciais oferecidos pelo modelo *OMT-G* podem ser encontrados em [BOR 97, BOR 99].

### 2.3.3 O Modelo GRDM

O *GeoRelational Data Model* (GRDM) é um modelo de dados, baseado no modelo Entidade-Relacionamento (ER), criado por Hadzilacos e Tryfona [HAD 96] para oferecer suporte à fase de projeto lógico do sistema de banco de dados para informações geográficas.

O projeto lógico [HAD 96] é o estágio intermediário da tradução entre o modelo conceitual e o de implementação (físico). Nesta fase são definidos os atributos espaciais, incluindo seus tipos geométricos, os atributos convencionais, as restrições de integridade espacial e conceitos de temporalidade e versionamento das entidades.

Segundo [HAD 96], é na fase da modelagem lógica do sistema que o projetista descreve as propriedades estáticas e as restrições de integridade da aplicação. Por essa razão, o GRDM oferece uma série de funcionalidades para expressar a realidade geográfica no sistema computacional. Entre elas estão: um modelo para a sobreposição

de camadas de dados; a definição de relacionamentos topológicos; e a criação de restrições espaciais.

O GRDM oferece uma linguagem para a definição de restrições e relacionamentos espaciais. Detalhes sobre o modelo e a sintaxe formal para a definição do modelo lógico e o modelo externo (visão do usuário) podem ser obtidos em [HAD 96, COC 96].

## 2.4 Restrições Espaciais Identificadas em Estudos de Caso

Para incorporar suporte a restrições espaciais na arquitetura da OGC (Capítulo 4) é necessário conhecer um número significativo de regras da realidade geográfica que devem ser tratadas pelo sistema. Por essa razão, foram realizados dois estudos de caso em organizações diferentes: um na Primeira Divisão de Levantamento do Exército Brasileiro (1ªDL) - responsável pela produção da Base Cartográfica Brasileira; outro na Secretaria Executiva do Programa Pró-Guaíba, subordinada à Secretaria da Administração e Planejamento do Estado do Rio Grande do Sul, onde é realizada a análise e a manipulação de dados geográficos de interesse do Estado. As regras encontradas nos estudos de caso estão detalhadas no Anexo 2.

Com base nas restrições espaciais encontradas na literatura e no estudo de caso realizado no contexto do Programa Pró-Guaíba, foi possível constatar que a maioria das regras da realidade geográfica são baseadas em relacionamentos espaciais, principalmente nos topológicos. Grande parte dessas restrições está baseada na obrigatoriedade ou na proibição do relacionamento topológico de continência. São regras do mundo real traduzidas para o sistema como restrições de integridade espacial.

No estudo de caso realizado na 1ªDL, também foi identificado um grande número de restrições. No entanto, a maioria delas não é baseada em relacionamentos espaciais, mas na correta forma de representação geométrica de cada objeto do mundo real, já que aquele órgão do Exército está mais preocupado com restrições que devem ser respeitadas no processo de construção da cartografia digital.

A maioria dos SIG já incorpora regras de geometria e regras ligadas ao tipo de estrutura usada na representação do aspecto espacial. Por exemplo, *o limite de um polígono deve ser fechado* ou *curvas de nível não devem se interceptar e precisam ser concêntricas*, são exemplos de restrições mínimas que um SIG deve garantir para a correta representação da forma do objeto em relação ao seu formato no mundo real. Esse tipo de restrição não é considerado neste trabalho, já que a arquitetura *OpenGIS*, abordada no próximo capítulo, já prevê restrições dessa natureza.

Segundo Hadzilacos [HAD 92], a evolução das pesquisas na área de modelagem de topologia em SIG mostra claramente que não há necessidade de se criar novos modelos ou métodos formais para especificar relacionamentos topológicos com diferentes operadores básicos (sobreposição, adjacência, disjunção, etc..), mas necessita-se de uma teoria para garantir que os relacionamentos já definidos sejam completos, seguros e capazes de atender os mais variados casos da realidade, sustentando as necessidades da aplicação.

## 2.5 Definição de um Conjunto Básico de Tipos de Restrição Topológica

Com base nos estudos de caso e nos quatro métodos já apresentados, os quais definem formalmente os relacionamentos topológicos binários, foi definido um sub-conjunto básico dos tipos de relacionamento necessários para garantir a integridade espacial dos dados geográficos para um conjunto significativo de aplicações. Com base nos relacionamentos definidos pelos quatro métodos estudados, foi especificado um conjunto de nove relacionamentos topológicos binários. São eles: *disjoint, touches, overlaps, equal, inside, contains, covers, coveredBy e crosses.* Entretanto, de acordo com o método utilizado na implementação do SIG (*4-Intersection, 9-Intersection, CBM* ou *DEM*), os tipos de relacionamento topológico podem ser especializados. No caso do *9-Intersection*, por exemplo, é possível definir as diversas formas como um objeto linha pode cruzar outro objeto do tipo linha ou área.

A partir dos nove relacionamentos topológicos encontrados, pode-se definir nove tipos básicos de restrição espacial de natureza topológica. Cada tipo de restrição é formado por dois tipos de feição, o relacionamento topológico entre eles, e o número mínimo e máximo de vezes que o primeiro tipo de feição pode estar relacionado ao segundo em cada tipo de relacionamento imposto pela restrição. Os números mínimo e máximo são denominados, respectivamente, *cardinalidade mínima* e *máxima*.

As restrições que *obrigam* certo relacionamento topológico podem ser expressas pela cardinalidade mínima igual a *0* ou *1*, e máxima igual a *1* ou *n*. Já as restrições de *proibição* de determinado tipo de relacionamento são expressas pelas cardinalidades mínima e máxima iguais a *0*.

Com base nas definições acima, restrições de integridade topológica podem ser expressas na seguinte sintaxe, baseada na EBNF:

<restrição> ::= <tipoFeição1> <tipoFeição2> <predicado>

<tipoFeição1>::=featureType1
<tipoFeição2>::=featureType2

<predicado> ::= <relObrig> | <relProib>

<relObrig> ::= <tipoRelac> {OR <tipoRelac>} <cardMin$_1$> <cardMax$_1$>

<tipoRelac> ::= 'covers'|'coveredBy'|'touches'|'overlaps'|'equals'|'inside'|'contains'|'crosses'
<cardMin$_1$> ::= 0 | 1
<cardMax$_1$> ::= 1 | n % (0,n) não expressa restrição %

<relProib> ::= <tipoRelac> {AND <tipoRelac>} <cardMin$_2$> <cardMax$_2$>

<cardMin$_2$> ::= 0 <cardMax$_2$> ::= 0

As restrições espaciais que envolvem o relacionamento topológico *disjoint* são expressas na sintaxe acima da seguinte forma:

- se o *disjoint* for proibido, ao menos um dos demais oito relacionamentos deve existir para cada par dos dois tipos de feição. Então, o predicado usado na definição da restrição de disjunção será <relObrig> e o tipo de relacionamento (<tipoRelac>) deve incluir todos os oito tipos de relacionamento ligados pelo operador lógico OR. A cardinalidade mínima (cardMin1) deve ser igual a *1* e a máxima (cardMax1) deve ser igual a *n*;
- se o *disjoint* for obrigatório, ou seja, os dois tipos de feição precisam ser disjuntos, o predicado usado na definição da restrição de disjunção será <relProib> e o tipo de relacionamento (<tipoRelac>) será igual a todos os demais oito tipos de relacionamento topológico ligados pelo operador lógico AND. As cardinalidades <cardMin2> e <cardMax2> serão iguais a *0* em todos os relacionamentos, já que a obrigatoriedade do relacionamento de disjunção indica a proibição de qualquer um dos demais oito relacionamentos topológicos.

Este Capítulo abordou o tema relacionamentos e restrições espaciais onde, com base na literatura e na realização de dois estudos de caso, foi identificado um subconjunto de restrições espaciais de natureza topológica. Esse conjunto de restrições foi estruturado e será utilizado, no Capítulo 4, na extensão ao *Modelo Abstrato OpenGIS.*

O próximo Capítulo apresenta um estudo do modelo de objetos do consórcio *Open GIS.*

# 3 O *Modelo Abstrato* Proposto pelo OGC

O *Open GIS Consortium* (OGC) [BUE 98, BOG 2000] é uma organização internacional que está criando novas padronizações técnicas e comerciais para garantir interoperabilidade em SIG. Fundada em 1994 por fornecedores de software, companhias de telecomunicações, universidades, provedores de informação e órgãos governamentais, entre outros,o OGC busca criar uma especificação de software e novas estratégias empresariais, a fim de tornar os sistemas de geoprocessamento abertos e integrar completamente os dados geográficos e as operações necessárias para manipulá-los [BUE 98].

Parte dos membros da OGC têm uma visão positiva de uma infra-estrutura de informação global em que dados geográficos e recursos de geoprocessamento são distribuídos gratuitamente. Esses recursos estão completamente integrados com as mais recentes tecnologias de computação distribuída, acessíveis para o mundo todo, habilitando uma grande variedade de atividades que, atualmente, estão fora do domínio de geoprocessamento. Entre alguns dos membros desse consórcio estão empresas como a Microsoft e a Oracle.

Este Capítulo está organizado conforme segue. A Seção 3.1 introduz a arquitetura proposta pelo OGC. A Seção 3.2 apresenta uma visão geral do *Modelo Abstrato* proposto pelo consórcio *OpenGIS*. A Seção 3.3 especifica o submodelo *Feições do OpenGIS,* incluindo suas formas de representação espacial. A Seção 3.4 detalha o submodelo de *Relacionamentos entre Feições* e a Seção 3.5 apresenta o tratamento da topologia de feições no *OpenGIS*.

O trabalho apresenta um estudo aprofundado sobre o submodelo de feições, relacionamentos entre feições e a topologia no *OpenGIS* porque são a base para a extensão do *Modelo Abstrato*. Maiores informações sobre este tema podem ser obtidas em [BOG 99, BOG 99a].

## 3.1 A Arquitetura Proposta pelo OGC

O OGC divide a especificação de software em dois modelos [OGC 99]: o *Modelo Essencial* e o *Modelo Abstrato*.

O *Modelo Essencial OpenGIS* é resultado de um processo de abstração dos elementos observados na realidade, descrevendo seus conceitos e comportamentos da forma como são. Na abordagem clássica do projeto de banco de dados, o modelo essencial pode ser comparado ao modelo conceitual.

O *Modelo Abstrato OpenGIS* especifica como os conceitos, definidos no *Modelo Essencial*, são representados na implementação de software. Também denominado *especificação de implementação OpenGIS***,** ele determina todas as funcionalidades e serviços que devem ser oferecidos pelo software de SIG para localizar, recuperar, armazenar e manipular as informações geográficas contidas no banco de dados [BUE 98]. O *Modelo Abstrato OpenGIS* foi criado para oferecer, aos produtores de software, um padrão de interface comum e detalhado, para que desenvolvam softwares de SIG

capazes de interoperar uns com os outros, todos baseados no *OpenGIS*, mas implementados por diferentes fabricantes.

O OGC definiu uma arquitetura para SIG, visando a padronização dos seguintes aspectos:

- um meio comum para representar digitalmente a terra e seus fenômenos, conceitualmente baseados na matemática;
- um modelo comum para implementar serviços de acesso, administração, manipulação, representação e compartilhamento de dados geográficos entre comunidades de informação[2];
- um *framework* para usar o modelo de dados e o modelo de serviços para resolver os problemas técnicos e institucionais de não-interoperabilidade em SIG.

## 3.2 O *Modelo Abstrato OpenGIS*

O *Modelo Abstrato* é composto por dezesseis submodelos, cada um dos quais é uma hierarquia de classes baseada na notação *UML (Unified Modeling Language*). Por ser um padrão em desenvolvimento, novos submodelos podem ser criados, bem como os já existentes podem sofrer alterações. A figura 3.1 apresenta algumas dependências entre os submodelos que, até o momento, compõem o *Modelo Abstrato OpenGIS* [OGC 99].

Os submodelos *Arquitetura de Serviços, Serviços do Catálogo, Serviços de Transformação de Coordenadas* e o de *Exploração de Imagens* são responsáveis pela definição dos serviços geoespaciais fundamentais que um SIG deve oferecer. O primeiro citado especifica a arquitetura geral de um amplo conjunto de serviços necessários para manipular um banco de dados geográficos. Essa arquitetura é dividida em três categorias: *Serviços do Catálogo, Serviços de Transformação de Coordenadas* e os de *Exploração de Imagens.*

- *Catálogo*: são os serviços responsáveis pela localização, recuperação, armazenamento, organização, administração e otimização das informações geográficas no banco de dados.
- *Exploração de Imagens*: são os serviços necessários para recuperar, interpretar e projetar os objetos geográficos na forma de imagens, garantindo a acurácia na representação.
- *Transformação de Coordenadas de Imagens*: são os serviços responsáveis por identificar e converter imagens com diferentes coordenadas para o mesmo sistema de coordenadas geográficas.

---

[2] Conjunto de pessoas que compartilham definições comuns no contexto da informação geográfica. São usuários de SIG que possuem a mesma visão do mundo, ou seja, utilizam a mesma forma de abstração para a representação de feições, metadados, funções de geoprocessamento e coleções de feições.

Os submodelos *Feições do OpenGIS, O Tipo Coverage* e *Imagens da Terra* estão fortemente interligados. O submodelo de *Feições* trata da definição de todos os objetos e os fenômenos do mundo real que precisam ser introduzidos e manipulados pelo sistema. O tipo *Coverage* é uma forma especial para representar, espacialmente, feições geográficas na visão de campo como, por exemplo, a temperatura e o relevo. O submodelo *Imagens da Terra* trata de casos especiais de c*overages* como, por exemplo, imagens de satélite.

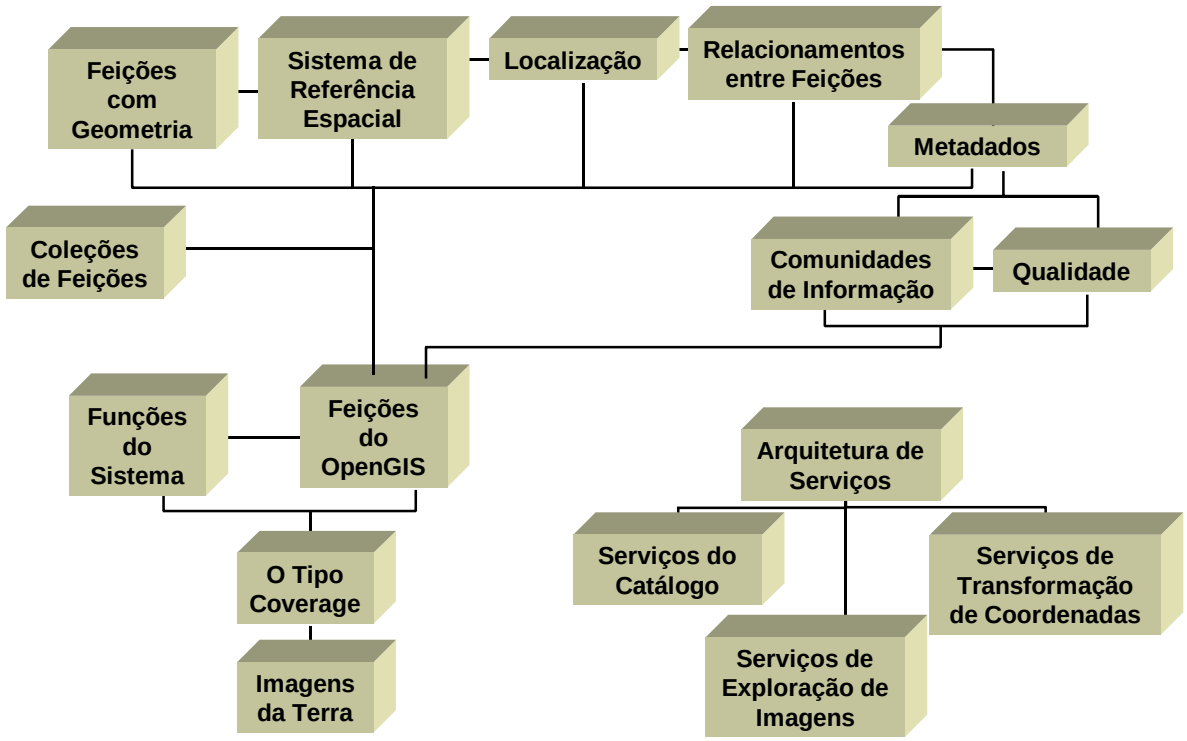



FIGURA 3.1 – Dependência dos submodelos da especificação abstrata

*Feições com Geometria* é um modelo padrão para tratar a representação espacial de feições geográficas percebidas na realidade na visão de objeto.

Os submodelos *Localização* e *Sistema de Referência Espacial* tratam da localização geográfica de feições relacionadas à superfície da Terra, caracterizando as ferramentas usadas na obtenção das referências geo-espaciais dessas feições.

*Relacionamentos entre Feições* é uma modelo padrão para representar a ligação existente entre feições. Ele permite descrever detalhadamente, através dos atributos do relacionamento, todos os aspectos que caracterizam uma relação (espacial ou não) entre feições geográficas.

Os *Metadados* tratam da modelagem dos dados adicionais que caracterizam as feições ou coleções de feições.

O submodelo *Funções do Sistema* trata do armazenamento das funções necessárias para manipular os dados representados espacialmente na forma de

*coverages*. Essas funções são utilizadas para calcular o valor ou interpolar pontos da *coverage.*

Os submodelos *Coleções de Feições, Comunidades de Informação* e *Qualidade* apresentam alguns conceitos necessários para compreender melhor a forma de manipulação das informações geo-espaciais e a tecnologia *OpenGIS*.

## 3.3 Feições no *OpenGIS*

Os elementos geográficos do mundo real dividem-se em entidades e fenômenos [OGC 99b].

No *OpenGIS,* entidades e fenômenos são tratados como feições, podendo ser diferenciadas, entre si, através da forma de representação espacial. Uma feição pode ser definida como sendo um átomo de informação geográfica [BUE 98]. É a representação de uma entidade ou fenômeno do mundo real, sendo caracterizada por ter um domínio espacial, temporal ou espaço-temporal. Exemplos de feições incluem quase todos os objetos que podem ser localizados no tempo e no espaço, como edifícios, árvores, florestas, cidades, ecossistemas e assim por diante.

Toda feição tem localização relativa à superfície da Terra. Ela somente poderá ser localizada se for possível descrevê-la em relação a outras feições cujas posições sejam previamente conhecidas, ou se tiver sua localização determinada em um sistema de coordenadas. Quando se dispõe de um sistema de coordenadas fixas, pode-se definir a localização de qualquer ponto na superfície terrestre.

No modelo *OpenGIS* [OGC 99b], todas as feições são representadas pela classe *FT_Feature*, conforme ilustra a figura 3.2. Na modelagem de feições, as mesmas são tratadas como instâncias de *FT_Feature* e não como especializações. Elas são diferenciadas entre si através da relação entre *FT_Feature* e *FT_FeatureType* que permite agrupá-las por tipo.

Segundo Buehler [BUE 98], a feição é formada por três elementos básicos:

- *geometria*: é a representação interna, em computador, da forma da feição;
- *propriedades semânticas***:** semântica é a visão e a forma de abstração da feição. Diferentes da geometria, que não varia de uma comunidade de informação para outra, semânticas podem variar de uma aplicação para outra;
- *metadados*: informações adicionais podem ser necessárias para posicionar a feição no contexto da aplicação ou para uma comunidade de usuários. Essas informações adicionais sobre os dados são denominadas *metadados* [OGC 99e]. Elas podem representar exigências ou padrões que são, geralmente, definidos por profissionais e usados para garantir a qualidade das informações ao usuário.

*Coleção de Feições* é um conjunto de feições agrupadas que são tratadas como sendo um único objeto feição. Ela é tratada, no *Modelo Abstrato*, como uma instância da classe *FT_Feature*, a qual representa as feições geográficas. Por isso, a coleção apresenta as mesmas características de uma feição simples: um tipo, um identificador unívoco e um subconjunto próprio de valores de atributos. O que diferencia a *coleção*

*de feições* de uma feição simples é sua forma de representação espacial. Sua geometria será formada pela agregação da geometria das feições que dão origem a coleção. Por exemplo: um arquipélago é uma coleção de feições. O arquipélago é uma instância de *FT_Feature* porque é tratado como uma feição, mas sua geometria é formada pela agregação da geometria de cada ilha que compõe o arquipélago.

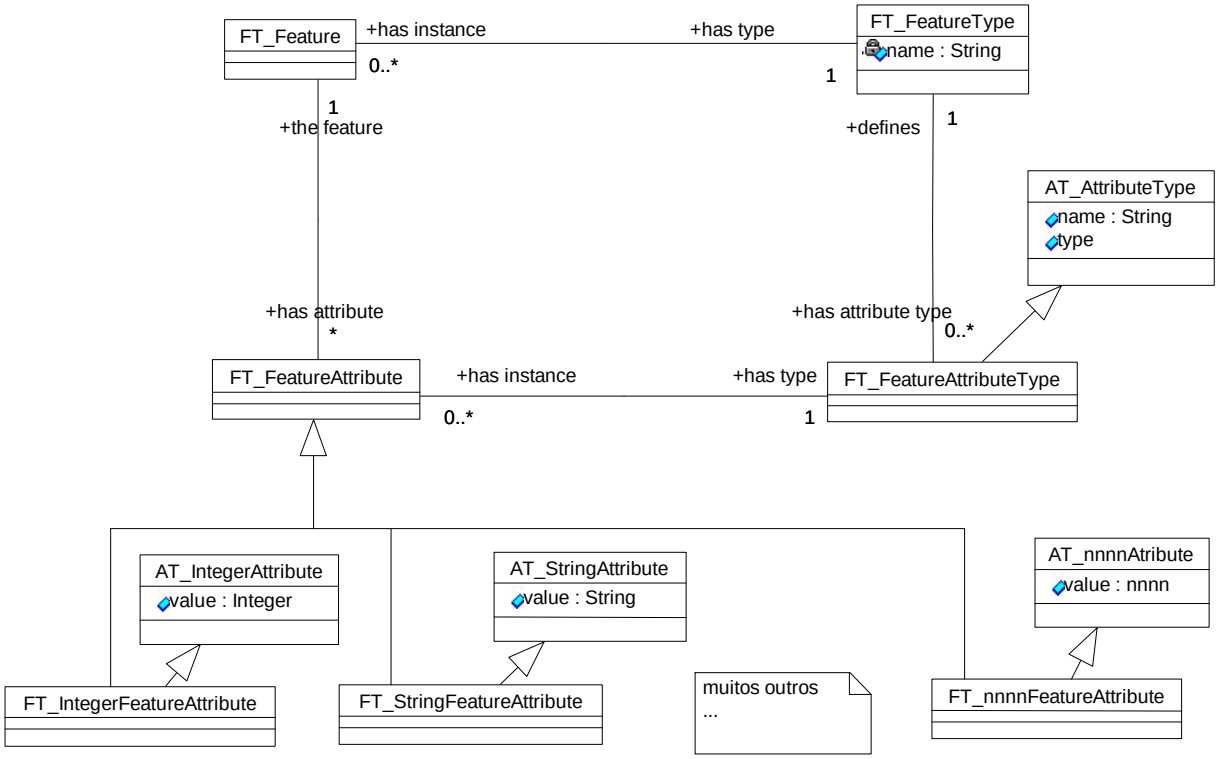



FIGURA 3.2 – Tipos e atributos de feições

### 3.3.1 Tipos e Atributos

O objeto *feição*, representado no *Modelo Abstrato* pela classe *FT_Feature*, representa ocorrências dos mais diversos tipos de entidades e fenômenos do mundo real como, por exemplo, hospitais, postos de combustível, rodovias e tudo o que pode ser localizado geograficamente [OGC 99b]. Cada hospital, cada posto de combustível e cada rodovia será uma instância de feição. Cada instância é de um determinado tipo, o qual tem um subconjunto próprio de características.

As feições do mundo real como, por exemplo, o rio Jacuí, a *BR-116* e o hospital *Mãe de Deus*, pertencem aos respectivos tipos: *rio, rodovia* e *hospital*. O *tipo*, ao qual a feição está associada, é representado pela classe *FT_FeatureType* e pelo relacionamento que o liga à classe das feições. Um tipo de feição é definido pelo conjunto de atributos que o caracterizam. O domínio de cada atributo pode ser inteiro, string, real, entre outros, ou uma geometria. Todos os nomes de atributos de um ou mais *tipos de feição* são instâncias de *FT_FeatureAttributeType.*

Assim como cada *tipo de feição* está associado a um conjunto de *nomes* de atributos, uma *feição* tem um conjunto próprio de *valores* para esses atributos. Esse conjunto é representado como uma superclasse, denominada *FT_FeatureAttribute*, que

generaliza os possíveis valores de atributos. De acordo com o seu domínio, uma das subclasses de *FT_FeatureAttribute* é instanciada.

Considere o seguinte exemplo: a feição *Jacuí* (instância de *FT_Feature*) está associada a um tipo denominado *rio* (instância de *FT_FeatureType*). Esse tipo de feição tem um subconjunto próprio de nomes de atributos. São eles: nome do rio, pontos de lançamento de líquido e comprimento. Seus respectivos domínios são: string, integer e real. O nome dos atributos e seus domínios são instâncias da classe *FT_FeatureAttributeType*, definidos, respectivamente, por *name* e *type*, herdados da superclasse *AT_AttributeType*. O valor de cada atributo, de acordo com o seu domínio, é instância de uma das subclasses de *FT_FeatureAttribute*. Considerando que o valor de cada atributo seja: nome = *Jacuí*, pontos de lançamento = *20* e comprimento = 210, o valor *Jacuí* será uma instância de *FT_StringFeatureAttribute*, especificado através do atributo *value* (herdado de *AT_IntegerAttribute*). Os valores *20* e *210* são, respectivamente, instâncias de *FT_IntegerFeatureAttribute* e *FT_RealFeatureAttribute*.

### 3.3.2 Formas de Representação Espacial

Conforme mencionado no Capítulo 1, uma feição pode ser percebida na realidade geográfica, na visão de campo ou na visão de objetos [LIS 99]. Para representar espacialmente as duas formas de interpretação de feições,o OGC dividiu a forma de representação espacial de uma feição em dois submodelos: feições com geometria (visão de objetos), tratadas como *GM_Object*, e o tipo *coverage* (visão de campo). Feições podem, ainda, não estar diretamente associadas com qualquer geometria como, por exemplo, objetos que não possuem localização em relação à superfície da Terra (fenômenos ou objetos não-geográficos) [OGC 99c], conforme ilustra a figura 3.3.

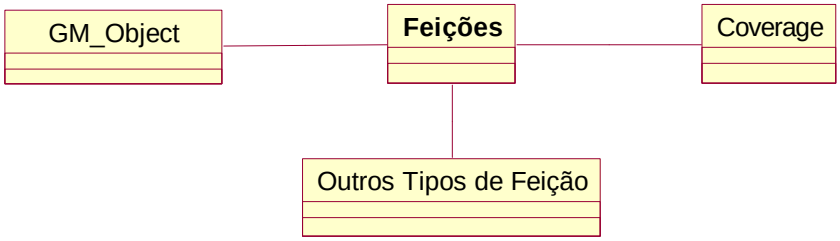



FIGURA 3.3 – Subtipos de feição

O submodelo de *Feições com Geometria* apresenta um conjunto de classes básicas, através das quais é possível representar, no computador, todos os objetos que possuem uma localização espacial bem definida. Uma visão geral do diagrama de classes que compõem esse submodelo é ilustrada na figura 3.4.

O submodelo é formado por uma superclasse que generaliza as possíveis formas de representação espacial de um objeto geográfico. Ela contém as operações necessárias para manipular qualquer objeto espacial com a forma de uma geometria primitiva do tipo ponto, linha, polígono, sólido, uma geometria complexa ou uma agregada. Essa superclasse é denominada *GM_Object* [OGC 99a].

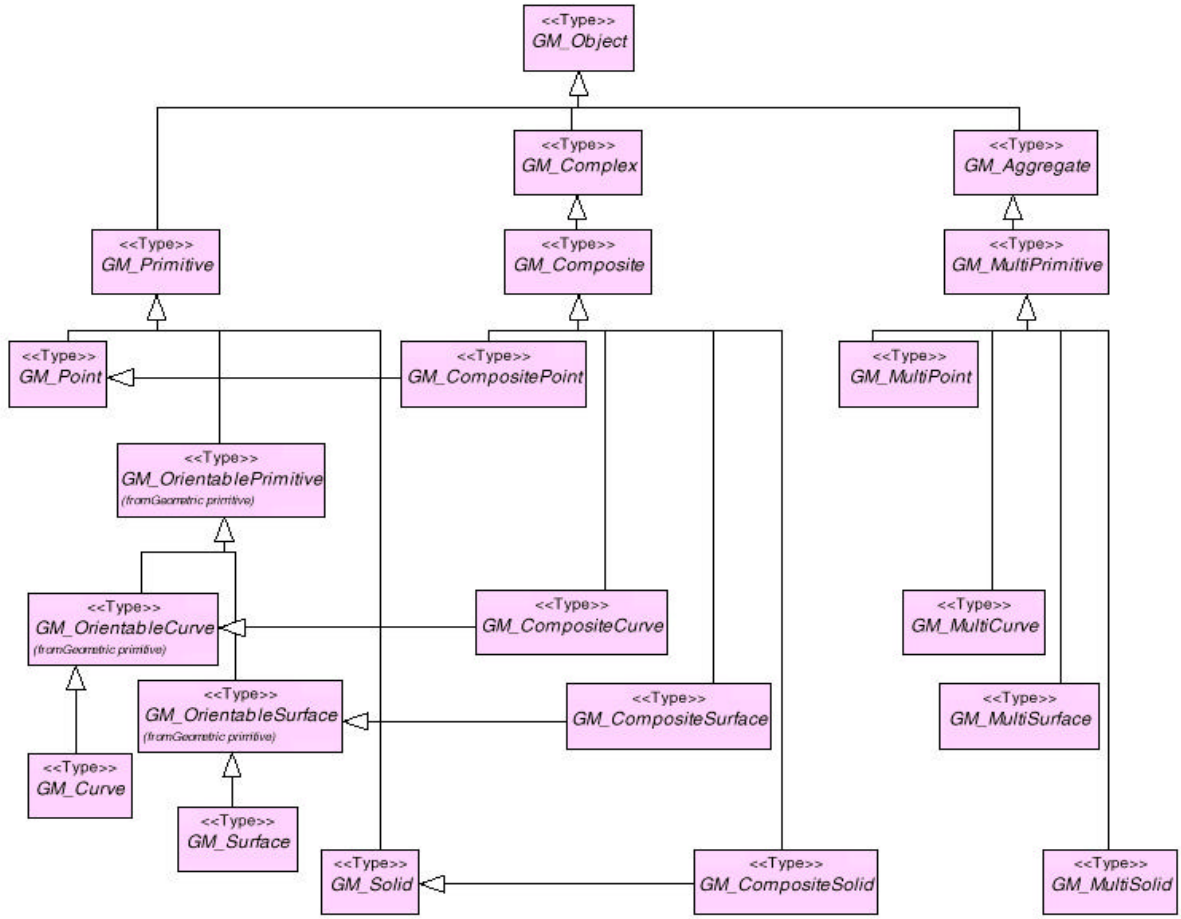



FIGURA 3.4 – Diagrama das classes básicas para feições com geometria

A classe *GM_Primitive* generaliza as possíveis formas de representação das geometrias primitivas. São elas o ponto, a linha/curva, o polígono/superfície/área e o sólido, representadas no diagrama da figura 3.4 pelas respectivas classes: *GM_Point*, *GM_Curve, GM_Surface* e *GM_Solid*.

A geometria complexa é uma coleção de primitivas, geometricamente ligadas umas às outras, formando uma geometria maior, que é a complexa. Um exemplo de objeto complexo é a união dos estados formando um país. O país é um objeto complexo porque é composto de um conjunto de geometrias primitivas, os estados (representados como superfícies), os quais estão ligados, uns aos outros, por suas fronteiras. Pode-se dizer que a geometria complexa é formada por um conjunto de duas ou mais geometrias primitivas. Ela é representada, no modelo em questão, pela classe *GM_Complex.* Se a geometria complexa for composta de pontos, a classe *GM_CompositePoint* é instanciada. Se for composta de linhas, *GM_CompositeCurve* é instanciada. Se for composta por polígonos ou sólidos, as classes *GM_CompositeSurface* ou *GM_CompositeSolid* são instanciadas, respectivamente.

Além da composição, existe também a agregação de primitivas. A agregação diferencia-se da composição no sentido de que as geometrias que formam a agregação não estão ligadas pelo seu limite. A classe *GM_Aggregate* generaliza todas as formas de agregação de pontos, linhas, polígonos ou sólidos. Um exemplo de objeto agregado é um arquipélago.

Os diagramas de classe, apresentados anteriormente, referem-se à geometria das feições geográficas [OGC 99a, OGC 2000]. A determinação da topologia existente entre as feições requer diferentes tipos de cálculos e definições, os quais são apresentados, posteriormente, na Seção 3.5.

## 3.4 Relacionamentos entre Feições

Entidades do mundo real não existem isoladas. Elas estão relacionadas a outras entidades de várias formas como, por exemplo, a vizinhança, a distância entre duas feições, os pontos de interseção entre feições, etc.

Todo tipo de relacionamento [OGC 99d] envolve, ao menos, dois tipos de feição, e cada um deles desempenha um determinado tipo de papel no relacionamento. Por exemplo, o relacionamento entre o estado e seus municípios envolve os tipos de feição *estado* e *município*, cada qual desempenhando um papel: *contém* – papel do estado, e *contido* – papel do município. Quando o relacionamento define somente dois tipos de papel ele é denominado binário.

Esta seção introduz o submodelo que retrata relacionamentos entre feições do mundo real. Esse submodelo é suficientemente abstrato para representar todos os tipos de relação entre feições, sejam essas espaciais ou não. O submodelo de relacionamentos é ilustrado na figura 3.5.

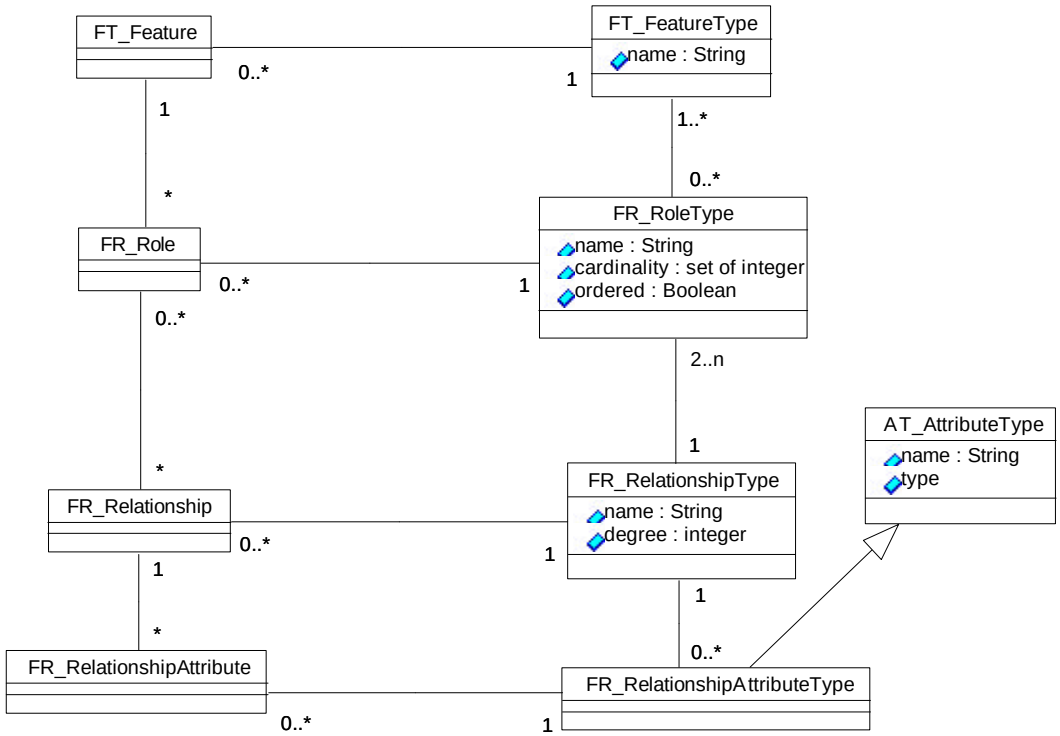



FIGURA 3.5 – Diagrama de classes de relacionamentos entre feições

Os relacionamentos espaciais podem ser agrupados em tipos. A classe *FR_RelatioshipType* retrata todos os tipos de relacionamento existentes entre dois tipos de feição [OGC 99d]. Os tipos de relacionamento são caracterizados por, ao menos, dois atributos:

- *name:* caracteriza, descritivamente, o tipo de relacionamento;
- *degree:* especifica o número de tipos de papel de cada tipo de relacionamento, podendo ser igual a dois ou maior. Para relacionamentos que expressam restrições espaciais de caráter topológico, o valor do atributo *degree* será sempre igual a dois, uma vez que tais restrições são baseadas em relacionamentos topológicos binários, os quais apresentam somente dois papéis.

Através da classe *FR_RelationshipType,* o *Modelo Abstrato OpenGIS* permite definir quaisquer tipos de relacionamento, inclusive os não espaciais. Convém lembrar, entretanto, que os relacionamentos considerados nesse trabalho são os topológicos.

Por exemplo, o tipo de relacionamento *casa contida em lote* tem dois tipos de papel: o tipo de papel da casa que está *contida* no lote e o tipo de papel do lote que *contém* a casa. Toda casa precisa, obrigatoriamente, estar dentro de um e somente um lote. Já o lote não precisa necessariamente conter qualquer casa, assim como pode ter mais de uma.

Os tipos de papel do tipo de relacionamento são representados pela classe *FR_RoleType* e são caracterizados pelos seguintes atributos [OGC 99d]:

- *name:* caracteriza, descritivamente, o tipo de papel de um determinado tipo de relacionamento;
- *cardinality:* determina o número mínimo e máximo de vezes que o papel de um certo tipo de feição pode ser desempenhado em relação ao outro tipo de feição do relacionamento. Esse atributo tem um par de dois valores onde o primeiro corresponde ao número mínimo e o segundo ao máximo. No tipo de relacionamento *casamento* entre um homem e uma mulher, por exemplo, o homem cumpre o papel de *marido* e está ligado a no mínimo e no máximo uma mulher, que da mesma forma cumpre o papel de *esposa;*
- *ordered:* determina a seqüência dos tipos de papel dentro do tipo de relacionamento. Por exemplo, trechos de rua estão relacionados uns aos outros, formando uma rota de ônibus. A ordem como esses trechos estão ligados é necessária para saber a sequência das ruas que o ônibus deve percorrer.

A associação entre as classes *FR_RoleType* e *FR_RelationshipType* liga cada tipo de papel a um tipo de relacionamento, ou seja, determina quais os papéis que fazem parte do mesmo tipo de relacionamento. No exemplo considerado anteriormente, os dois tipos de papel *casa contida* e *lote contém* estão associados ao mesmo tipo de relacionamento *casa&lote.*

A associação entre as classes *FT_FeatureType* e *FR_RoleType* liga o tipo de feição ao tipo de papel que ela desempenha no relacionamento. As classes *FT_FeatureType, FR_RoleType, FR_RelationshipType* e *FR_RelationshipAttributeType* são instanciadas na fase de projeto do banco de dados.

As classes *FT_Feature, FR_Role, FR_Relationship* e *FR_RelationshipAttribute* representam, respectivamente, as feições do mundo real, os papéis de cada feição no relacionamento, o relacionamento propriamente dito e seus atributos. Essas classes são instanciadas na fase de produção do SIG, ou seja, quando o usuário final da aplicação manipula os dados geográficos.

Considere, por exemplo, o tipo de relacionamento *país&estados.* Os tipos de papel desse tipo de relacionamento são: *país contém [1,n]* e *estado contido[1,1]*. Com base nesse exemplo, as seguintes feições, papéis e relacionamentos podem ser definidos em suas respectivas classes:

- as feições *Brasil* e *Paraná* instanciam a classe *FT_Feature* e estão associadas aos respectivos tipos, *país* e *estado* (instâncias de *FT_FeatureType*) *;*
- o relacionamento *Brasil* contém *Paraná* instancia a classe *FR_Relationship;*
- os papéis *Brasil contém* e *Paraná contido* instanciam a classe *FR_Role* e estão associados aos respectivos tipos de papel *país contém* e *estado contido* (objetos de *FR_RoleType*).

## 3.5 Topologia de Feições

Topologia é a parte da geometria que estuda as propriedades geométricas que são invariantes sob transformações como: rotação, escala e translação [OGC 2000].

O objeto topológico é diferente do objeto geométrico, pois tem apenas os pontos que são comuns a qualquer outro objeto geométrico. Mais especificamente, o objeto topológico é formado pelas coordenadas que originam um relacionamento topológico. Por isso, o objeto topológico requer operações diferentes das atribuídas ao objeto geométrico, razão pela qual o OGC criou, junto ao submodelo de feições com geometria, um submodelo para a topologia.

As classes que compõem o diagrama da topologia no *Modelo Abstrato OpenGIS* são equivalentes mas não subclasses da geometria (veja figura 3.6).

A definição de procedimentos, funções e operações no submodelo da topologia pode auxiliar na resolução de problemas no domínio da geometria. O problema pode ser traduzido para o âmbito algébrico no domínio da topologia, resolvido e, posteriormente, traduzido novamente para o domínio da geometria. Isso permite tratar situações como: um *estado* armazenado com as coordenadas geográficas erradas pode pertencer a um país incorreto. Associando a geometria do objeto *estado* à topologia, onde um estado deve estar geograficamente contido dentro do *país* a que pertence, é possível detectar se o *estado* está ou não com valores de coordenadas errados.

Uma única geometria primitiva pode fazer parte de muitas geometrias complexas independentes, podendo ser usada para determinar diferentes tipos de topologia complexa. Um mesmo objeto ilha, por exemplo, pode fazer parte de um arquipélago e pode ser um estado que compõe um país.

O objeto topológico é representado, no submodelo, como uma classe abstrata, denominada *TP_Object,* oferecendo suporte às topologias primitiva e complexa. Ele possui operadores, alguns dos quais são responsáveis por determinar a dimensão, o limite, o interior e o exterior do objeto geométrico. Esses operadores são necessários porque, conforme abordado no Capítulo 2, a topologia é obtida através da interseção dos pontos que formam a fronteira, a parte interna e a parte externa dos objetos.

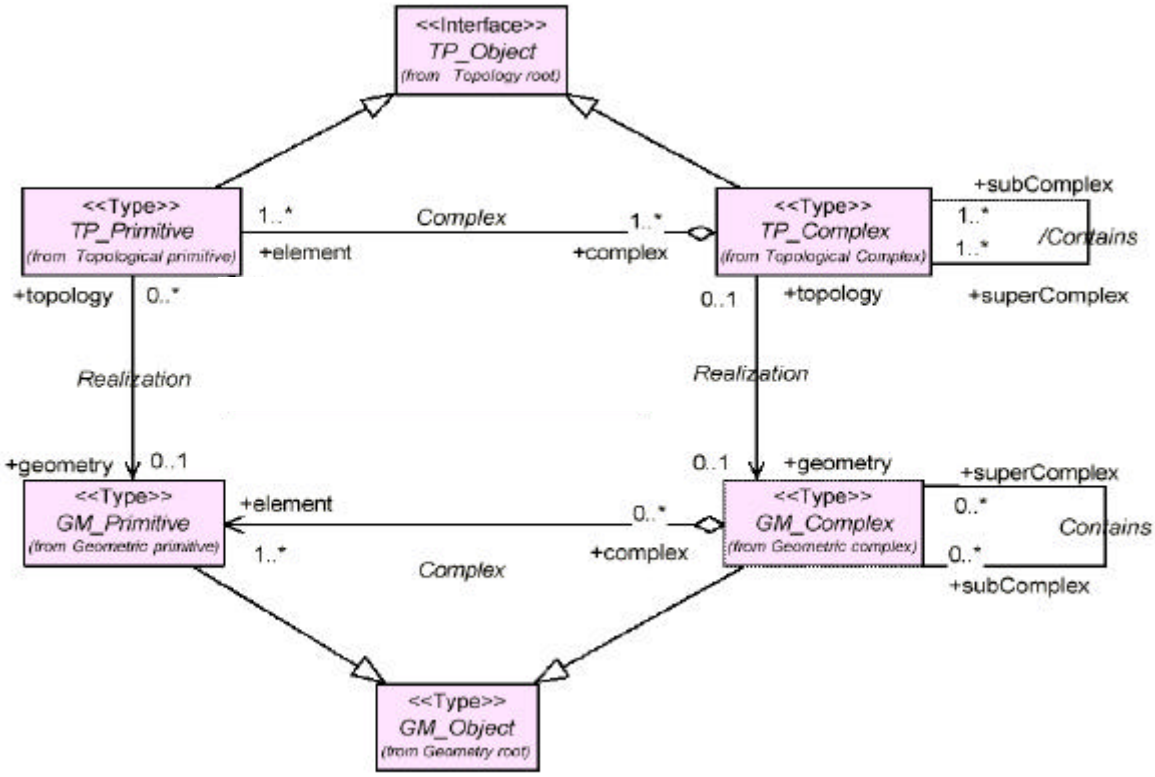



FIGURA 3.6 - Relação entre geometria e topologia

Por representar uma arquitetura de software, o *Modelo Abstrato OpenGIS* busca definir padrões suficientemente genéricos, para que o produtor do SIG determine sua própria forma de implementação. Para isso, o modelo de topologia de feições não apresenta operadores específicos que verificam se dois objetos estão disjuntos, adjacentes, sobrepostos, etc. No entanto, ele oferece operações como *boundary(), interior()* e *exterior()* que recuperam, respectivamente, o limite, o interior e o exterior dos objetos relacionados. Essas operações permitem que um ou mais métodos (*4-Intersection, 9-Intersection e DEM*) apresentados no Capítulo 2, sejam utilizados para implementar relacionamentos topológicos binários, ficando a critério do produtor do SIG a escolha do(s) método(s) a ser(em) utilizado(s).

Uma das partes que compõem o diagrama de classes para a topologia de feições é ilustrada na figura 3.7.

Conforme abordado anteriormente, o submodelo de Topologia de Feições oferece a base para três dos métodos que determinam, formalmente, um conjunto de relações topológicas: *4-Intersection*, *9-Intersection* e *DEM*. Para verificar o tipo de relacionamento topológico existente entre duas feições, o OGC oferece três funções, uma para cada método utilizado. Tais funções são descritas nas próximas subseções.

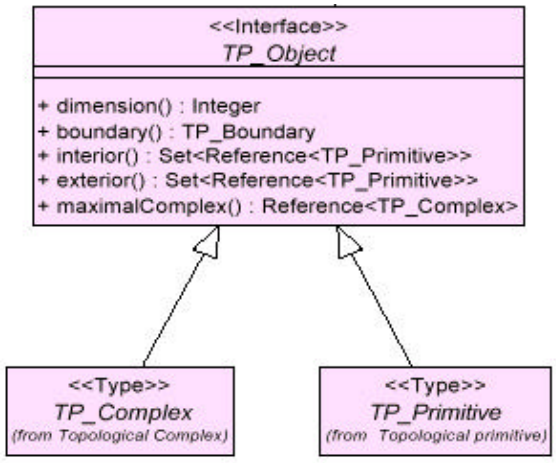



FIGURA 3.7 – Submodelo do diagrama de topologia

### 3.5.1 Relacionamentos Baseados no Método *4-Intersection*

Para representar os relacionamentos topológicos obtidos pelo método *4-Intersection, o OGC* define uma função denominada *bRelate*(). Ela retorna um valor lógico indicando se existe ou não, um determinado relacionamento entre dois objetos espaciais. A função tem a seguinte estrutura:

boolean *bRelate* (TP_Object, TP_Object, intersectionPatternMatrix)

Os dois primeiros argumentos correspondem aos objetos espaciais que estão sendo comparados. O argumento *intersectionPatternMatrix* é um valor do tipo string, com quatro caracteres, cada um dos quais corresponde ao resultado lógico da interseção entre o limite e o interior dos dois objetos, de acordo com a matriz de quatro interseções apresentada no Capítulo 2. A interpretação do resultado de cada elemento da matriz é dada na tabela 3.1.

TABELA 3.1 - Semântica de cada caracter da string que compõe o argumento *intersectionPatternMatrix* para o método *4-Intersection* e o *9-Intersection*

| Símbolo | Não Vazio | Semântica |
|---|---|---|
| T | TRUE | A interseção nessa posição da matriz é não-vazia ($\neg\phi$) |
| F | FALSE | A interseção nessa posição da matriz é vazia ($\phi$) |
| N | NULL | Não testa a interseção nesse ponto da matriz. O resultado da interseção nessa posição também é considerado vazio ($\phi$). |

Cada caractere que compõe a string correspondente ao argumento *intersectionPatternMatrix*, pode ter um valor igual a F, T ou N, onde os dois primeiros caracteres correspondem à primeira linha da matriz, seguidos pelos da segunda linha.

A função *bRelate()* pode ser utilizada para testar, por exemplo, o relacionamento *contém* para um objeto composto, conforme ilustrado abaixo.

C : GM_Composite, G : GM_Object;

C.contains (G) = bRelate (C, G, "TFNF");

### 3.5.2 Relacionamentos Baseados no Método 9-*Intersection*

Os relacionamentos topológicos originados pelo método *9-Intersection* são recuperados através da função *eRelate*(). Ela retorna um valor boleano indicando se existe um determinado relacionamento topológico entre dois objetos espaciais. A função tem a seguinte estrutura:

boolean *eRelate* (TP_Object, TP_Object, intersectionPatternMatrix)

Os dois primeiros argumentos correspondem aos objetos espaciais que estão sendo comparados, podendo ser tanto o objeto *TP_Object* como o *GM_Object*. O argumento *intersectionPatternMatrix* é um valor do tipo string com nove caracteres, cada um corresponde ao resultado lógico da interseção entre o limite, o interior e o exterior dos dois objetos, de acordo com a matriz de nove interseções do método *9-Intersection* apresentada no Capítulo 2 (Seção 2.1.4). A interpretação do resultado de cada elemento da matriz utiliza os mesmos valores da tabela 3.1.

Da mesma forma como para o método *4-Intersection,* a função *eRelate* pode ser utilizada para implementar, por exemplo, o relacionamento *contém* para um objeto composto, considerando, porém, além do interior e do limite, o exterior de cada objeto. Veja o exemplo abaixo, onde é testado se o objeto G está contido no objeto C:

C : GM_Composite, G : GM_Object;

C.contains (G) = *eRelate* (C, G, "FFNTTNFFT");

### 3.5.3 Relacionamentos Baseados no Método DEM

O método DEM (*Dimension Extended Method*) apresenta operadores semelhantes aos do método *4-Intersection,* porém com uma distinção mais detalhada entre os possíveis resultados da interseção. Ele permite obter, além do tipo de relacionamento topológico entre dois objetos espaciais, a dimensão do relacionamento existente. A função que recupera os relacionamentos baseados no método DEM denomina-se *cRelate* e possui a mesma estrutura das anteriores, conforme ilustrado abaixo.

boolean *cRelate* (TP_Object, TP_Object, intersectionPatternMatrix)

A diferença entre as funções apresentadas é o valor do parâmetro *intersectionPatternMatrix,* cuja interpretação dos possíveis resultados para o método DEM, é dada pela tabela 3.2.

Através das funções *eRelate, cRelate* e *bRelate, o OGC* oferece os meios para que o produtor do SIG escolha os relacionamentos topológicos e o método que deseja implementar.

TABELA 3.2 - Semântica de cada caracter que compõe o argumento *intersectionPatternMatrix* para o método DEM

| Símbolo | Não Vazio | Semântica |
|---|---|---|
| 0 | TRUE | A interseção nessa posição da matriz contém apenas pontos. |
| 1 | TRUE | A interseção nessa posição da matriz contém pontos e curvas. |
| 2 | TRUE | A interseção nessa posição da matriz contém pontos, curvas e áreas. |
| 3 | TRUE | A interseção nessa posição da matriz contém pontos, curvas, áreas e sólidos. |
| F | FALSE | A interseção nessa posição da matriz é vazia. |
| N | NULL | A interseção nessa posição da matriz não é testada. |

Este Capítulo apresentou um estudo sobre o *Modelo Abstrato* do consórcio *OpenGIS*. O próximo Capítulo apresenta uma proposta de extensão a esse modelo.

# 4 Proposta de Extensão ao *Modelo Abstrato OpenGIS*

A arquitetura de um software de SIG é formada por diversos módulos, cada um dos quais executa diferentes tipos de serviço. O controle de restrições espaciais pode ser um desses módulos, pois apresenta funcionalidades e serviços específicos que garantem a integridade espacial dos dados geográficos (veja figura 4.1).

O módulo de controle de restrições espaciais interage com o módulo gerenciador de transações, pois as restrições são gerenciadas dentro do contexto de transações.

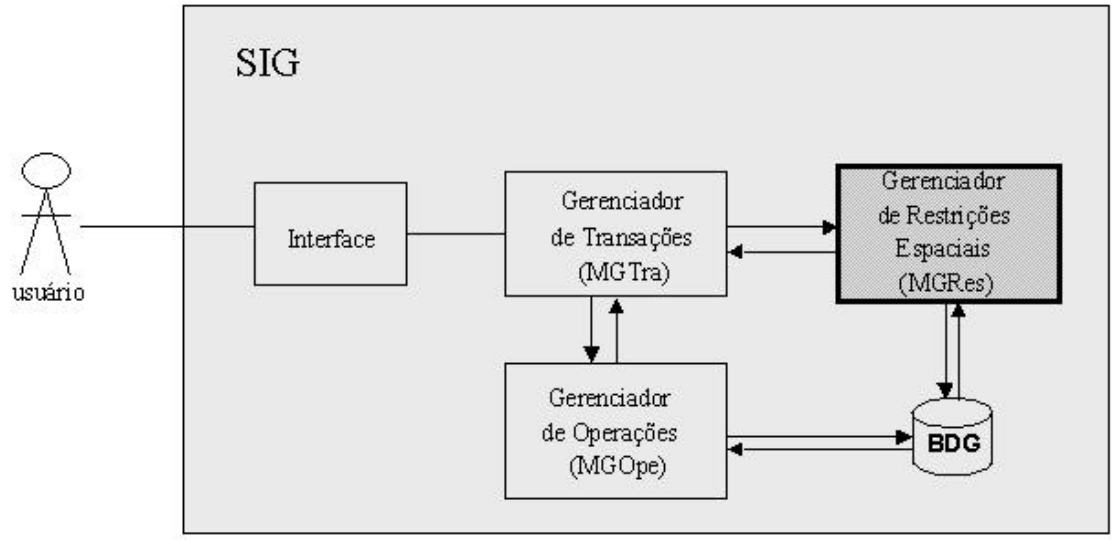



FIGURA 4.1 – Arquitetura de software de SIG

O módulo em destaque na figura 4.1 é a parte central deste trabalho, sendo aqui denominado Módulo Gerenciador de Restrições Espaciais (MGRes). É o módulo que está sendo proposto com base no *Modelo Abstrato OpenGIS.*

O MGRes inicia suas atividades quando o usuário da aplicação solicita ao MGTra, através da interface do SIG, uma operação de inclusão, alteração ou exclusão de uma feição geográfica. O MGTra comunica, então, ao MGRes que uma nova operação foi solicitada. Na seqüência, o MGRes verifica e garante as restrições espaciais, acessando informações do BDG. Ao finalizar suas atividades, o MGRes comunica ao MGTra que seu trabalho foi concluído. Por fim, o MGOpe realiza a operação submetida pelo usuário, no banco de dados geográficos.

Para verificar e garantir as restrições espaciais entre feições geográficas, o MGRes considera as seguintes possibilidades:

- a feição atende a todas as restrições espaciais associadas ao tipo da feição. Neste caso a feição está consistente;
- a feição viola algum tipo de restrição. Neste caso a feição está inconsistente;
- a feição é uma exceção à regra, ou seja, foi definida pelo usuário como um caso excepcional, podendo não atender a um conjunto específico de restrições. Nesta situação, se a feição atender a todas as restrições, para as quais não é exceção, a

feição está consistente. Se for exceção em todas as regras ela também será consistente.

Com base na notação UML, a figura 4.2 ilustra os diferentes estados que uma feição *A* pode assumir.

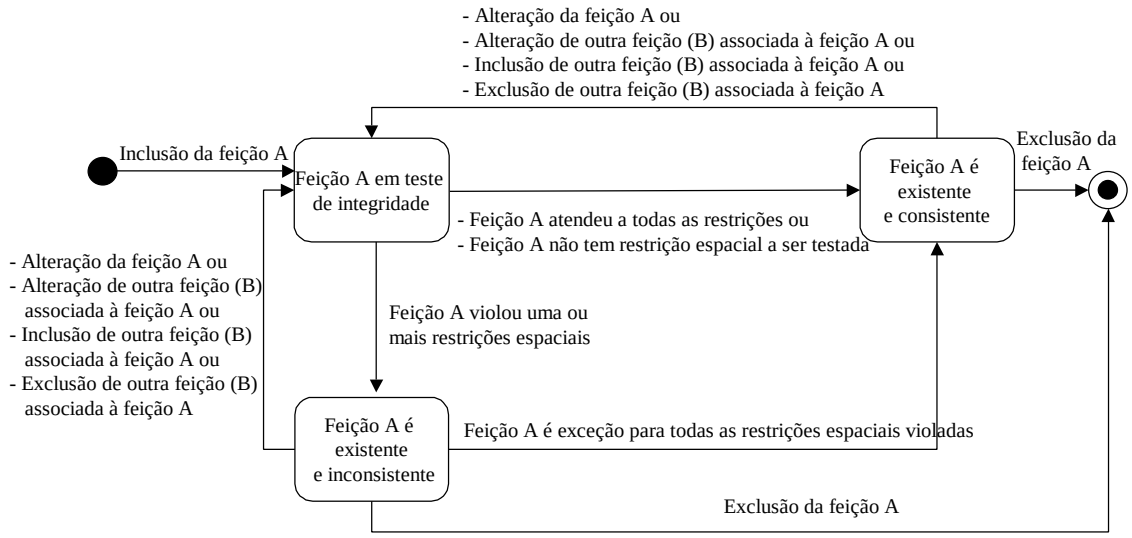



FIGURA 4.2 – Diagrama de estados da feição A

Conforme ilustra a figura 4.2, uma feição geográfica pode mudar de estado toda vez que o usuário solicitar alguma operação de atualização sobre a mesma.

O processo de existência de uma feição *A* inicia quando o usuário a insere no BDG. A feição *A*, então, passa pelo estado de teste de integridade, onde são verificadas todas as restrições espaciais associadas a seu tipo.

Durante a fase de teste de integridade, se a feição *A* atender a todas as restrições espaciais, ou não tiver qualquer restrição associada, ela passa do estado de teste de integridade para o estado existente/consistente. Se ao contrário, uma ou mais restrições foram violadas, a feição *A* passa para o estado existente/inconsistente.

Uma feição *A*, no estado inconsistente, pode transitar para outro estado quando ocorrer algum dos seguintes eventos ou atividades:

- *exclusão da feição A:* quando a feição *A* é removida do banco de dados ela passa a não existir, ou seja, a feição chega ao estado final;
- *a feição A é declarada, pelo usuário, exceção para todas as suas restrições topológicas violadas*: quando o usuário indicar que a feição *A* é uma exceção para cada uma das restrições que está violando, a feição *A* passa para o estado consistente;
- *alteração da feição A:* quando a geometria da feição *A* é modificada, ela sai do estado de inconsistente e volta para o teste de integridade;
- *alteração de uma feição B associada à feição A:* a feição *A* pode ter sua(s) restrição(ões) atendida(s) quando a geometria de uma feição *B*, envolvida no

relacionamento, for modificada. Então quando a feição *B* for alterada, *A* volta ao estado de teste de integridade;

- *inclusão de uma feição B associada à feição A:* quando uma nova feição *B* é inserida no banco de dados, a feição *A* que está inconsistente pode ter todas as suas restrições satisfeitas. Então *A* volta ao estado de teste de integridade;
- *exclusão de uma feição B associada à feição A*: quando uma feição *B* é excluída do banco de dados, todas as restrições de *A* podem ser atendidas. Então *A* entra em estado de teste de integridade novamente.

Uma feição *A* que está consistente pode transitar para o estado de teste de integridade ou para seu estado final. Quando o evento de *exclusão da feição A* ocorre, *A* passa do estado consistente para o estado final. Contudo, a feição *A* também pode transitar do estado consistente para o estado de teste de integridade quando ocorrer algum dos seguintes eventos:

- alteração da feição *A;*
- alteração de uma feição *B* associada à feição *A;*
- inclusão de uma feição *B* associada à feição *A;* e
- exclusão de uma feição *B* associada à feição *A.*

Os diferentes estados que uma feição pode assumir são importantes para compreender o processo de verificação e garantia de restrições espaciais no *Modelo Abstrato OpenGIS Estendido.*

Duas alternativas de extensão diferentes ao *Modelo Abstrato* foram desenvolvidas. Ambas são baseadas no Submodelo de *Relacionamentos entre Feições.* Para destacar os novos elementos incorporados ao diagrama de classes de *Relacionamentos*, as novas classes estão sombreadas e os novos métodos e atributos são indicados em fonte *negrito itálica*.

A primeira extensão, apresentada na Seção 4.1, é uma proposta preliminar que representa em três classes distintas, respectivamente, as feições consistentes, as feições que violaram algum tipo de restrição espacial (inconsistentes) e aquelas que são exceção em um ou mais tipos de restrição topológica. Essa extensão foi avaliada com base em alguns critérios estipulados e originou o desenvolvimento de uma segunda proposta.

Na Segunda e definitiva extensão, apresentada na Seção 4.2, o controle das restrições é feito através do papel que a feição desempenha no relacionamento topológico que serve de base à restrição espacial, e todas as feições, independentemente de seu estado, são representadas por uma única classe.

## 4.1 Proposta Inicial de Extensão ao *Modelo Abstrato OpenGIS*

A extensão do *Modelo Abstrato OpenGIS,* mais especificamente do submodelo de Relacionamentos, é ilustrada na figura 4.3. A semântica e a funcionalidade dos novos elementos do modelo é apresentada em duas partes: as novas classes e relacionamentos são detalhados na Seção 4.1.1; e os novos métodos e atributos são tratados na Seção 4.1.2.

Os procedimentos necessários para manipular os novos elementos e fazer uso adequado da extensão proposta ocorrem em diferentes fases do ciclo de vida de um SIG. A Seção 4.1.3 descreve o processo de definição das restrições espaciais na fase de projeto do banco de dados. A verificação e a garantia das restrições acontecem na fase de produção, ou seja, quando o usuário final do SIG atualiza dados geográficos. Essa fase é descrita na Seção 4.1.4. A Seção 4.1.5 apresenta alguns critérios utilizados na avaliação da extensão proposta e que serviram de base para o desenvolvimento de uma segunda extensão.

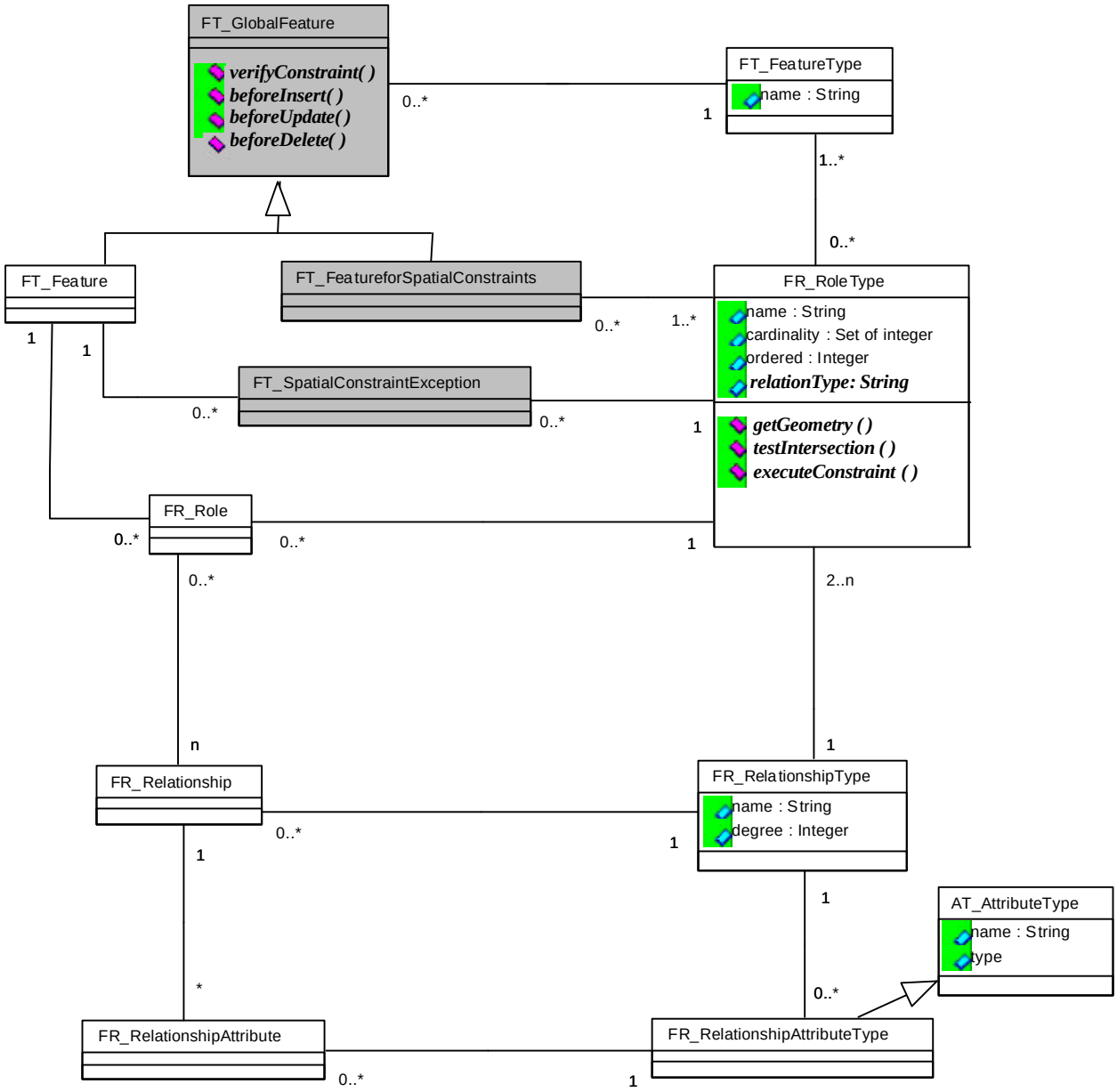



FIGURA 4.3 – Primeira Proposta de extensão ao *Modelo Abstrato OpenGIS*

### 4.1.1 Classes e Associações

As feições geográficas que satisfazem todas as restrições topológicas, ou são de um tipo para o qual não há restrição definida, são armazenadas no banco de dados quando termina o processo de verificação e garantia das restrições. Essas feições são consistentes e instanciam a classe *FT_Feature*.

As feições que não atenderam a algum tipo de restrição são consideradas inconsistentes e representadas na classe *FT_FeatureforSpatialConstraints*, onde permanecem, temporariamente, até receberem o tratamento adequado. Essa classe tem as mesmas características de *FT_Feature*, porém suas instâncias permanecem armazenadas por um limitado período de tempo.

A classe *FT_FeatureforSpatialConstraints* está associada à classe *FR_RoleType*. Essa associação liga cada feição que infringiu alguma restrição, ao tipo de papel/restrição que foi violado/a. O processo para tratar as feições inconsistentes é apresentado mais adiante.

A classe *FT_GlobalFeature* generaliza as feições consistentes e inconsistentes.

Após receber o tratamento adequado, as feições inconsistentes que, então, atendem a todas as restrições topológicas são armazenadas, definitivamente, em *FT_Feature* e removidas da classe de apoio (*FT_FeatureforSpatialConstraints*).

As feições que ainda violam uma ou mais restrições, mas que, por decisão do usuário, constituem uma exceção nas restrições violadas, também são armazenadas definitivamente em *FT_Feature* e eliminadas da classe de apoio. Contudo, o identificador da feição e o identificador do tipo de restrição violada são armazenados na classe *FT_SpatialConstraintException*, a fim de controlar as feições que constituem exceções às regras.

### 4.1.2 Métodos e Atributos

O método *verifyConstraint( )*, definido na classe *FT_GlobalFeature*, verifica se a feição é de um tipo que possui ou não alguma restrição espacial associada. Todas as restrições identificadas são recuperadas e inseridas numa lista (coleção de objetos) que é retornada para o método *beforeInsert(), beforeUpdate()* ou *beforeDelete()* para serem validadas.

O método *beforeInsert()* executa todo processo de controle das restrições topológicas dos objetos espaciais submetidos à inserção no banco de dados geográficos antes da inserção ser efetivada pelo MGOpe. Este método é responsável por garantir as restrições. Se todas as restrições da feição forem atendidas, o MGOpe insere a feição em *FT_Feature*, caso contrário, ele o faz em *FT_FeatureforSpatialConstraints.*

O método *beforeUpdate()* executa todo processo de controle das restrições topológicas dos objetos espaciais submetidos à atualização no banco de dados geográficos. Ele verifica se as restrições estão sendo respeitadas pelas respectivas feições, antes da atualização ser efetivada no banco de dados.

O método *beforeDelete()* executa todo processo de controle das restrições topológicas quando um ou mais objetos espaciais são submetidos à remoção do banco de dados geográficos.

O método *getGeometry(),* definido na classe *FR_RoleType*, recupera a geometria de cada uma das feições envolvidas na restrição. Através da associação entre a classe *FT_Feature* e *GM_Object* (veja Capítulo 3) é possível identificar a forma geométrica da feição.

O método *testIntersection()* verifica, através da função *brelate()*, se o tipo de relacionamento topológico imposto pela restrição é verdadeiro ou não para cada feição específica. Se o relacionamento existe, o método *testIntersection()* retorna verdadeiro (*TRUE*); caso contrário, retorna falso (*FALSE*).

A função *brelate()* é definida no *Modelo Abstrato* e verifica a existência de relacionamentos topológicos com base no método *4-Intersection.* Esta função foi escolhida devido à popularidade e facilidade de uso do método *4-Intersection*. Entretanto, se o fabricante do SIG optar pela implementação do método *DEM* ou *9-Intersection* através das funções *cRelate()* ou *eRelate()*, respectivamente, poderá fazê-lo sem maiores dificuldades. Ele deverá, portanto, substituir a função *bRelate()* por uma das outras funções e aplicar os testes necessários para o método (*9-Intersection, DEM*) utilizado a partir do método *testIntersection()*.

A verificação, propriamente dita, da consistência da feição é feita pelo método *executeConstraint()*. Ele compara o valor retornado pelo método *testIntersection()* e as cardinalidades de cada restrição. Com base no resultado dessas combinações a restrição é satisfeita ou não.

O atributo *relationType*, definido na classe *FR_RoleType*, indica o tipo de relacionamento do tipo de papel e determina qual o tipo de relacionamento topológico que precisa ser testado pelo método *testIntersection()*. Nesta proposta, *relationType* pode assumir os seguintes valores: *Disjoint(D)*, *Touches(T)*, *Inside(I)*, *croSses(S)*, *coVers(V)*, *coveredBy(B)*, *Overlaps(O)*, *Contains(C)* ou *Equal(E)*, de acordo com o conjunto de restrições topológicas definidas no Capítulo 2.

### 4.1.3 Procedimentos na Fase de Projeto do Banco de Dados

Na fase de projeto do banco de dados, as entidades do mundo real precisam ser modeladas. Nessa modelagem devem ser representados todos os elementos importantes à aplicação, inclusive os tipos de relacionamento e os tipos de restrição espacial.

Para definir as restrições topológicas no *Modelo Abstrato Estendido*, os elementos envolvidos nas restrições precisam ser declarados dentro de uma certa ordem. Primeiramente, todos os tipos de feição que farão parte da aplicação geográfica precisam ser definidos (objetos de *FT_FeatureType*).

Uma vez criados os tipos de feição, os tipos de relacionamento topológico que serão considerados devem ser especificados (objetos de *FR_RelationshipType*). Esses tipos de relacionamento devem ter os seguintes atributos: o nome do tipo de relacionamento (*name*) e o grau (*degree*) que deve ser igual a dois, por se tratar de relacionamentos binários. Se o tipo do relacionamento exige tipos de atributo para caracterizar o relacionamento, os mesmos devem ser criados com seus respectivos domínios (objetos de *FR_RelationshipAttributeType*).

Uma vez definidos os tipos de feição e os tipos de relacionamento, os tipos de papel precisam ser especificados, os quais associam cada tipo de relacionamento aos respectivos tipos de papel. Para definir um tipo de papel, o projetista deve especificar o valor dos seguintes atributos, alguns dos quais estão implícitos no *Modelo Abstrato*:

- *featureType_id*: tipo de feição que desempenha o tipo de papel;

- *relationshipType_id*: o tipo de relacionamento do qual o tipo de papel faz parte;
- *name*: nome do tipo de papel;
- *cardinality*: cardinalidade do tipo de papel. Esse é o atributo mais importante na definição do tipo de papel. É ele que define a restrição de integridade, ou seja, o número mínimo e máximo de vezes que uma instância de feição pode estar relacionada a outras pelo relacionamento topológico em questão;
- *ordered*: ordem de cada papel dentro do relacionamento;
- *relationType*: tipo de relação topológica que o tipo de feição desempenha no relacionamento (ex.: crosses).

### 4.1.4 Procedimentos na Fase de Produção

Na fase de produção do SIG, as restrições topológicas definidas na fase de projeto do banco de dados são verificadas e garantidas. Nessa fase, os usuários inserem, alteram e removem dados espaciais. Tais operações são importantes nesse trabalho porque podem modificar o estado de uma feição (veja figura 4.2) com relação a uma ou mais restrições espaciais. Isso exige que, a cada operação de atualização no banco de dados, as restrições de integridade topológica sejam checadas e garantidas.

Os procedimentos necessários para fazer uso adequado do *Modelo Abstrato Estendido*, na fase de produção do SIG, variam de acordo com o tipo de operação executada pelo usuário. Para gerenciar tais procedimentos são utilizados os recursos de um modelo de *transações.* Nesse trabalho, o modelo SAGAS [ELM 90, MOL 87] é utilizado como exemplo.

Inicialmente, quando o usuário abre uma base de dados, ainda antes de iniciar qualquer operação sobre os dados, o MGTra deve iniciar uma transação longa (SAGA). Essa transação ficará aberta até que o usuário feche a base de dados que está sendo manipulada. Cada operação de inclusão, alteração e remoção será implementada, nos níveis mais baixos do SIG, como uma subtransação da SAGA, com todas as características de uma transação ACID [ELM 90, SIL 95].

Durante as operações de inserção, atualização ou remoção de feições do banco de dados, feições que violaram uma ou mais restrições topológicas são criadas na classe de apoio, denominada *FT_FeatureforSpatialConstraints.* O tratamento dessas feições é realizado quando o usuário solicita o final da transação longa, através da submissão de um comando para fechar a base de dados em uso. O tratamento de cada feição em *FT_FeatureforSpatialConstraints* também é gerenciado por uma subtransação. Ao final, quando o usuário encerrar o tratamento das feições inconsistentes, a SAGA será concluída. Caso ocorra alguma falha no sistema, durante a execução das subtransações, a SAGA não será desfeita, podendo recomeçar do início da subtransação que falhou. O sistema garante, ainda, que os resultados de todas as subtransações são desfeitas no banco de dados, caso o usuário aborte a SAGA.

#### 4.1.4.1 Procedimentos para Inserção de Feições

Ao ser submetida a operação de inserção de uma feição geográfica *A*, o MGRes testa todas as restrições espaciais associadas ao tipo dessa feição. Primeiramente, ele recupera em *FR_RoleType* todas as restrições da feição *A.* Em seguida, realiza uma

pesquisa nas classes *FT_Feature e FT_FeatureforSpatialConstraints,* procurando por uma ou mais feições *B* que, em combinação com a feição *A*, podem atender ou violar as restrições espaciais associadas ao tipo de *A*.

A inserção de uma feição *A* no banco de dados pode mudar o estado de uma ou mais feições *B* envolvidas no relacionamento com *A*. Por isso, o estado das feições *B* também precisa ser avaliado.

Uma vez testadas todas as restrições topológicas de *A*, o MGRes executa os seguintes procedimentos:

a) se uma ou mais restrições da feição *A* foram violadas, a feição *A* é instanciada na classe de apoio *FT_FeatureforSpatialConstraints.* Em seguida, as restrições das feições *B* envolvidas em relacionamentos com *A* são avaliadas. Então, para cada feição *B* é realizado o seguinte teste:

    a.1) se *B* é consistente, então *B* é removida de *FT_Feature* e inserida na classe de apoio, pois infringiu uma ou mais restrições por causa de *A*. Os papéis de *B* também são removidos de *FR_Role;*

b) se ao contrário, todas as restrições da feição *A* foram respeitadas, *A* é inserida como objeto de *FT_Feature* e seus papéis como objetos de *FR_Role.* Em seguida, o estado de todas as feições *B* envolvidas no relacionamento é analisado. Então para cada feição *B* é feito o seguinte teste:

    b.1) se *B* está inconsistente, ou seja, é objeto de *FT_FeatureforSpatialConstraints,* então é necessário verificar se todas as restrições pendentes de *B* foram satisfeitas com a inserção de *A.* Em caso positivo, *B* também é inserida em *FT_Feature* e removida da classe de apoio, pois tornou-se consistente. Todos os papéis de *B* também são inseridos em *FR_Role.*

Somente as feições que estão consistentes têm instâncias de papel em *FR_Role.* Para cada restrição respeitada, há um papel em *FR_Role*. As feições que instanciam a classe *FT_FeatureforSpatialConstraints* não estão associadas a papéis em *FR_Role.* Entretanto, quando a feição muda do estado inconsistente para consistente, ao ser eliminada de *FT_FeatureforSpatialConstraints* para ser inserida em *FT_Feature*, todos os papéis que ela desempenhou para mudar de estado são inseridos em *FR_Role.*

#### 4.1.4.2 Procedimentos para Atualização de Feições

O controle das restrições espaciais na atualização de feições é mais complexo do que na inserção das mesmas. O MGRes inicia o processo quando o usuário submete uma operação de atualização da geometria de uma feição *A*.

A alteração de uma feição *A*, no banco de dados, pode mudar o estado de uma ou mais feições *B* envolvidas em relacionamentos com *A* (veja figura 4.2). Por isso, a situação das feições *B* também precisa ser avaliada.

Inicialmente, o MGRes verifica em *FR_RoleType* se a feição *A* tem restrições topológicas associadas a seu tipo. Se não encontrar restrições, *A* é atualizada no banco

de dados em *FT_Feature*. Mas se, ao contrário, uma ou mais restrições forem encontradas, as mesmas são recuperadas e inseridas numa lista.

Uma vez recuperada a lista de restrições da feição *A*, o MGRes verifica se *A* está consistente (em *FT_Feature*) ou inconsistente (em *FT_FeatureforSpatialConstraints*). Em ambos os casos, todas as restrições da lista serão testadas com feições de *FT_Feature* e *FT_FeatureforSpatialConstraints* que, combinadas com a feição *A*, podem atender ou violar as restrições de *A*. As feições testadas com a feição *A* são tratadas como *B*. Entretanto, dependendo do estado de *A*, diferentes tipos de teste são realizados pelo MGRes.

Caso *A* esteja em *FT_FeatureforSpatialConstraints*, após checar todas as restrições de *A*, o MGRes executa os seguintes procedimentos:

a) se, com a nova geometria, a feição *A* ainda tiver alguma restrição violada, ela continua inconsistente em *FT_FeatureforSpatialConstraints*;

b) se, com a nova geometria da feição *A*, todas as suas restrições foram respeitadas, então *A* é migrada da classe de apoio para *FT_Feature*. Os papéis para cada restrição atendida são criados em *FR_Role*. Além disso, os estados de todas as feições *B* envolvidas nos relacionamentos com *A* precisam ser avaliados. Para cada feição *B* que estiver inconsistente, o MGRes realiza o seguinte procedimento:

b.1) testa todas as restrições pendentes de *B*. Caso todas elas estejam atendidas, *B* passa ao estado consistente. Então, *B* é removida da classe de apoio e inserida em *FT_Feature*. Além disso, os papéis de *B* são inseridos em *FR_Role*.

Por outro lado, se a feição *A* sendo modificada já está em *FT_Feature* e a lista de restrições não é vazia, então as restrições da lista são testadas com a nova geometria de *A*. Com base no resultado final dos testes, o MGRes executa os seguintes procedimentos:

a) se todas as restrições de *A* continuam satisfeitas com sua nova geometria, então *A* continua consistente em *FT_Feature*. Neste caso, as restrições das feições *B*, envolvidas em relacionamentos com *A*, precisam ser testadas. Para cada feição *B* inconsistente, o MGRes realiza o seguinte procedimento:

a.1) se todas as restrições de *B* foram respeitadas, *B* é removido da classe de apoio e inserida em *FT_Feature*, pois tornou-se consistente. Seus papéis para cada restrição atendida são inseridos em *FR_Role;*

b) se ao menos uma restrição de *A* foi violada, *A* é removida de *FT_Feature* e inserida na classe de apoio, passando para o estado inconsistente. Todos os seus papéis também são removidos de *FR_Role*. Neste caso, os estados das feições *B* que, combinadas com *A*, violaram uma ou mais restrições de *A* precisam ser avaliados. Para cada feição *B* consistente, o MGRes realiza o seguinte procedimento:

b.1) testa todas as restrições de *B*. Se ao menos uma restrição de *B* foi violada, devido a modificação da geometria de *A*, *B* passa a ser inconsistente. Em seguida, *B* é removida de *FT_Feature* e inserida na classe de apoio. Seus papéis também são removidos de *FR_Role.*

#### 4.1.4.3 Procedimentos para Remoção de Feições

O teste das restrições espaciais é realizado, também, quando a geometria de uma feição geográfica *A* for submetida a exclusão no banco de dados.

Inicialmente, o MGRes verifica em *FR_RoleType* se a feição *A* tem restrições espaciais associadas ao seu tipo. Caso *A* não tenha restrições espaciais, a mesma é removida do banco de dados sem que o sistema realize testes adicionais. Mas se ao contrário, *A* possuir uma ou mais restrições espaciais, essas são recuperadas e inseridas numa lista.

As restrições da lista não precisam ser checadas para a feição *A*, pois a mesma será excluída. Contudo, a exclusão de *A* pode mudar o estado de uma ou mais feições *B* envolvidas em relacionamentos topológicos com a feição *A.* Então, para cada restrição da lista, o segundo tipo de feição e suas restrições precisam ser encontrados.

Para cada feição *B* em *FT_Feature* ou *FT_FeatureforSpatialConstraints* cujo tipo é igual ao segundo tipo de feição envolvido em restrições de *A*, as restrições de *B* precisam ser testadas.

Após checar todas as restrições de cada feição *B* envolvida em relacionamentos topológicos com *A*, o MGRes avalia o resultado com base no estado atual de cada feição *B* e executa os seguintes procedimentos:

- *B* é consistente: se *B* está em *FT_Feature* e com a exclusão de *A* alguma de suas restrições foi violada, então *B* é removida de *FT_Feature* e inserida na classe de apoio. Seus papéis também são removidos de *FR_Role;*
- *B* é inconsistente: neste caso *B* está em *FT_FeatureforSpatialConstraints.* Então, se com a exclusão de *A* todas as restrições de *B* foram respeitadas, *B* é removida da classe de apoio e inserida em *FT_Feature.* Os papéis de *B* para todas as restrições são inseridos em *FR_Role.*

#### 4.1.4.4 Tratamento de Feições que Violaram alguma Restrição

Esta fase é responsável pelo tratamento das feições que, por alguma razão, não atenderam uma ou mais restrições espaciais até o final da transação longa. Ela é iniciada quando o usuário solicitar o fechamento do banco de dados em uso. Antes de concluir a SAGA, iniciada quando o usuário abriu a base de dados, o MGRes deve verificar se alguma feição permanece em *FT_FeatureforSpatialConstraints.* Em caso positivo, o MGRes deve apresentar, através de uma interface, o conjunto de feições que violaram uma ou mais restrições espaciais e quais as regras violadas. Assim, o usuário da aplicação pode interagir com o sistema e decidir sobre o futuro de cada uma das feições remanescentes. A cada feição tratada, o MGTra inicia uma nova subtransação.

Existem duas formas para uma feição *A* sair do estado inconsistente nesta fase: sendo eliminada pelo usuário; ou sendo definida como uma exceção a todas as regras

(supostamente violadas), passando para o estado consistente. Para isso, a interface apresentada ao usuário para realizar o tratamento de feições inconsistentes deve oferecer alguns requisitos básicos:

- permitir que a feição *A* seja exceção em todas as regras. Neste caso, o sistema remove *A* da classe de apoio e insere-a em *FT_Feature.* O identificador de *A* e o identificador das restrições onde *A* é exceção são armazenados em *FT_SpatialConstraintException.* Neste caso, nenhum papel é associado a feição;
- permitir que a feição *A* seja eliminada de *FT_FeatureforSpatialConstraints*, para que o usuário insira uma nova feição no banco de dados, visando a correção da geometria de um ou mais objetos envolvidos na restrição;
- permitir ao usuário encerrar a SAGA atual, mantendo as informações na classe de apoio para serem tratadas no futuro, em uma nova transação SAGA.

### 4.1.5 Critérios de Avaliação do *Modelo Abstrato Estendido*

Ao final do desenvolvimento desta primeira extensão ao *Modelo Abstrato OpenGIS,* foram identificadas algumas limitações suas, as quais estão descritas abaixo:

- *desempenho do sistema*: a representação de feições consistentes e inconsistentes em duas classes distintas pode comprometer o desempenho do sistema em virtude dos objetos, quando mudam de estado, migrarem de uma classe para outra;
- *complexidade na implementação*: a troca de objetos entre as classes torna a implementação dos algoritmos dessa estrutura bastante complexa, principalmente os procedimentos de atualização de feições;
- *invasão à estrutura original do Modelo Abstrato*: a criação das novas classes afeta os demais submodelos que compõem o *Modelo Abstrato,* uma vez que as feições são parte fundamental nos demais submodelos.

## 4.2 Segunda Extensão ao *Modelo Abstrato OpenGIS* (Definitiva)

Na busca de simplificar a extensão ao *Modelo Abstrato*, alterando o mínimo possível a estrutura original do Submodelo de Relacionamentos do *OpenGIS,* e buscando reduzir a complexidade de implementação, foi desenvolvido um segundo modelo, diferente do apresentado na Seção 4.1.

Este modelo apresenta, como principal vantagem, o fato de as feições que violam restrições, durante a atualização de objetos no banco de dados, pertencerem à mesma classe das feições que atendem a todas as restrições espaciais.

A semântica e a funcionalidade dos elementos incorporados ao modelo é apresentada na Seção 4.2.1. Os procedimentos necessários para manipular esses elementos e fazer uso adequado da extensão ocorrem em diferentes fases durante a criação de um SIG. A Seção 4.2.2 apresenta o processo de definição das restrições na fase do projeto do banco de dados. A verificação e a garantia das restrições acontecem

na fase de produção, ou seja, quando o usuário final do SIG manipular os dados geográficos. Essa fase é descrita na Seção 4.2.3.

### 4.2.1 Semântica do Modelo

A principal diferença entre as duas extensões está no controle das feições que passam por diferentes estados. Neste modelo, todas as feições (consistentes e inconsistentes) instanciam a classe *FT_Feature*, não havendo a necessidade de uma classe de apoio para representar as feições que violaram uma ou mais restrições espaciais.

O controle do estado das feições no banco de dados é realizado com base em um novo atributo, denominado *status*, criado na classe *FR_Role*, conforme ilustra a figura 4.4.

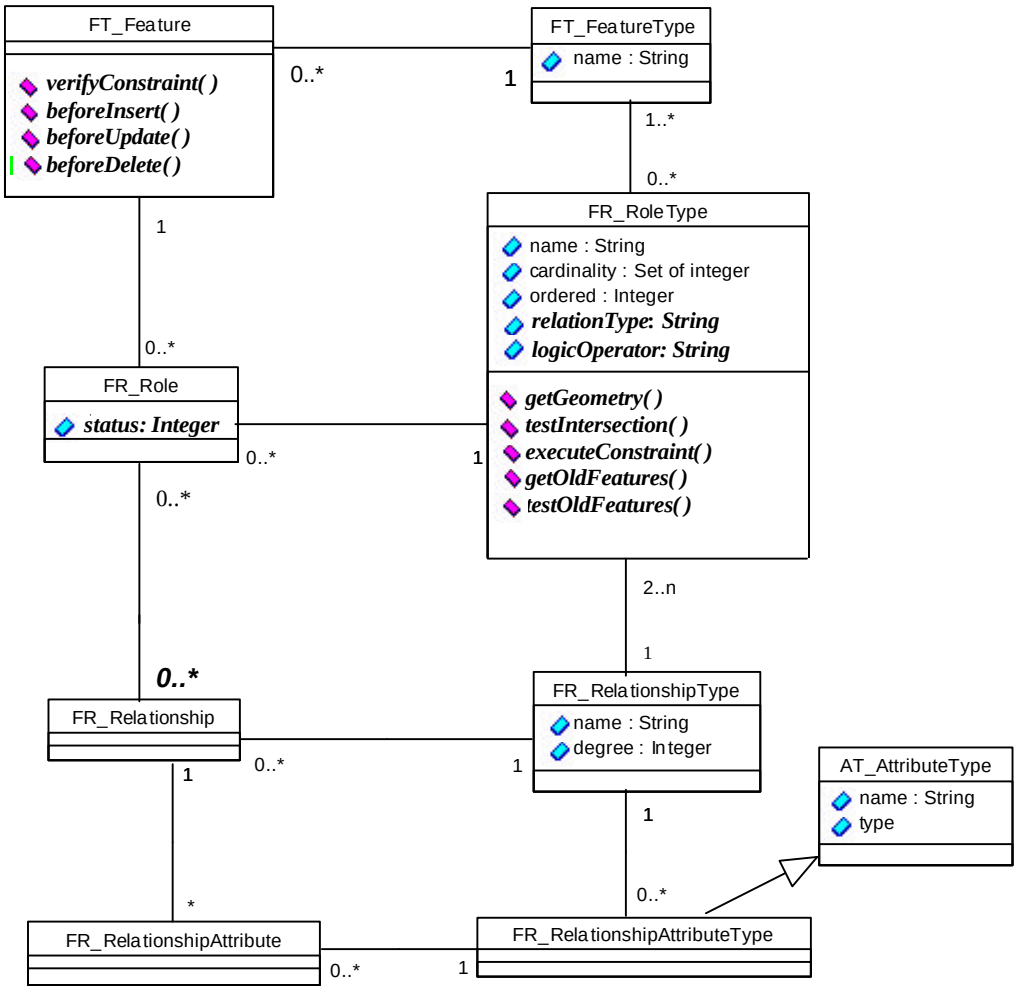



FIGURA 4.4 – Segunda extensão ao *Modelo Abstrato OpenGIS*

A classe *FR_Role* representa os papéis de cada feição armazenada no banco de dados de acordo com o tipo de papel que ela desempenha em cada relacionamento do qual faz parte. Pelo atributo *status* é possível controlar a situação de cada um desses papéis, e a partir dele, conhecer o estado de uma feição no banco de dados. O atributo *status* possui um conjunto de valores pré-definidos. Sua semântica é apresentada a seguir:

- 0 : indica que a feição desempenhou o papel, satisfazendo a restrição topológica definida pelo tipo de papel. Se todos os papéis da feição têm esse valor no atributo *status*, a feição é consistente;
- -1 : a feição encontra-se num estado de pendência, pois não desempenhou o papel para o tipo de restrição associado. Nesse caso, a feição está inconsistente e deve ser tratada futuramente. Esse estado deve mudar para 1 ou 0 após o tratamento adequado da feição;
- 1 : a feição não desempenhou o papel, mas deve ser considerada uma exceção à regra. Nesta situação, a feição é considerada consistente, mesmo transgredindo a respectiva regra associada a esta instância específica de papel.

À medida que o usuário insere, modifica ou remove feições do banco de dados, o MGRes instancia, automaticamente, as classes *FR_Role* e *FR_Relationship*, de acordo com a existência ou não de um relacionamento topológico.

Nesta extensão, a classe *FR_Role* é instanciada toda vez que uma nova feição *A* é inserida no banco de dados e seu tipo estiver associado a um ou mais tipos de papel. Então, para cada tipo de papel (tipo de restrição) do tipo da feição *A*, uma instância é criada em *FR_Role.* Quando a restrição é atendida o atributo *status* do papel da feição *A*, no relacionamento em questão, recebe o valor *0;* caso contrário, o valor *-1*.

Quando o papel foi corretamente desempenhado pela feição *A*, ou seja, o valor do atributo *status* for igual a *0*, uma instância de relacionamento é criada em *FR_Relationship.* Esse relacionamento estará associado a dois papéis em *FR_Role* por tratar-se de relacionamento topológico binário. Os papéis do relacionamento criado em *FR_Relationship* são o papel da feição *A* e o papel da feição *B*, a qual combinada com *A*, satisfêz aquela restrição de integridade espacial.

Nos casos em que uma feição passa a não mais desempenhar um papel no relacionamento, ou seja, quando o valor do atributo *status* de seu respectivo papel for alterado de *0* para *–1*, o relacionamento, do qual o papel dessa feição participa, deve ser removido de *FR_Relationship.*

Para controlar as restrições satisfeitas e as violadas, de cada feição, foi modificada a cardinalidade da associação entre as classes *FR_Role* e *FR_Relationship*. A cardinalidade ***, associada a classe *FR_Relationship*, foi substituída para *0..**, conforme pode ser observado no diagrama de classes da figura 4.4. Esta mudança permite criar uma instância em *FR_Role* sem relacioná-la a qualquer instância de *FR_Relationship.* A instância em *FR_Relationship* será criada somente quando o relacionamento espacial, imposto pela restrição, existir efetivamente entre os dois objetos espaciais, ou seja, quando o atributo *status* dos dois papéis envolvidos no relacionamento for igual a *0.*

Convém lembrar que, no caso da restrição ser de *proibição*, ou seja, o atributo *cardinality* do tipo de papel ser igual a [0,0], não haverá instância em *FR_Relationship*, pois a cardinalidade indica que o relacionamento topológico não deve existir.

Uma feição pode ter um ou mais papéis em *FR_Role* de acordo com o número de tipos de papel associados a seu tipo de feição. Alguns desses papéis podem ter o valor do atributo *status* igual a *0* e outros igual a *–1,* indicando, respectivamente, que algumas restrições foram satisfeitas e outras ainda não. Caso, em determinado

momento, a feição apresente, ao menos, um papel com o valor do atributo *status* igual a *–1*, ela é considerada, naquele momento, uma feição inconsistente no banco de dados.

O atributo *relationType* tem a mesma semântica da primeira proposta de extensão, mas neste contexto, permite expressar mais de um relacionamento topológico binário em uma única restrição de integridade espacial de caráter topológico. Esses tipos de relacionamento podem ser combinados pelos operadores lógicos AND ou OR, especificados pelo novo atributo *logicOperator*, de forma que, a partir do resultado da combinação dos possíveis tipos de relação topológica, uma restrição é atendida ou violada. Todavia, os dois operadores lógicos nunca aparecem na definição da mesma restrição, uma vez que os relacionamentos topológicos entre dois tipos de feição são mutuamente exclusivos.

Observe, por exemplo, a restrição: *todo estado é coberto ou está contido em um e somente um país.* Para definir esse tipo de restrição, o atributo *relationType* assume um conjunto de valores, dado por [B,I], que correspondem, respectivamente, aos relacionamentos topológicos *coveredBy* e *Inside* (veja Seção 4.1.2). O novo atributo *logicOperator*, neste exemplo, assume o operador lógico OR de forma que se um estado for *coberto* ou estiver *contido* em um, e somente um país, a restrição é considerada atendida.

Um outro caso a ser considerado é uma restrição de integridade espacial que expressa a obrigatoriedade do relacionamento topológico de disjunção. Por exemplo, *todo rio é disjunto de todas as edificações.* Como o Modelo Abstrato não permite as cardinalidades no mínimo *TODOS* e no máximo *TODOS*, esse tipo de restrição é definido através da proibição de todos os demais relacionamentos topológicos. Para isso, o valor do atributo *relationType* assume um conjunto com todos os relacionamentos topológicos ([T,I,S,V,B,O,C,E]), exceto o *Disjoint.* Esses relacionamentos são ligados pelo operador lógico AND e as cardinalidades mínima e máxima em todos os relacionamentos serão iguais a zero, expressando a proibição de qualquer relacionamento topológico que não seja o *disjoint*.

Quando o atributo *relationType* expressa apenas um tipo de relacionamento topológico, o operador lógico (*logicOperator*) não precisa ser especificado na definição da restrição.

Os métodos definidos no diagrama de classes da figura 4.4 são usados no processo de verificação e garantia das restrições topológicas. A ordem como esses métodos são executados durante o processo depende da ação realizada pelo usuário.

A figura 4.5 ilustra, em três diagramas de seqüência baseados na notação UML [FOW 96, FUR 98, BOO 98], a ordem como os métodos do *Modelo Abstrato OpenGIS Estendido* são executados. O diagrama 1 ilustra a seqüência de execução dos métodos quando uma nova feição é submetida à inserção no banco de dados. O diagrama 2 retrata a ordem de invocação dos métodos quando o usuário altera a geometria de uma feição no banco de dados. Por último, o diagrama 3 ilustra a ordem de execução dos métodos necessários para gerenciar restrições espaciais quando uma feição é removida do banco de dados.

Diagrama 1

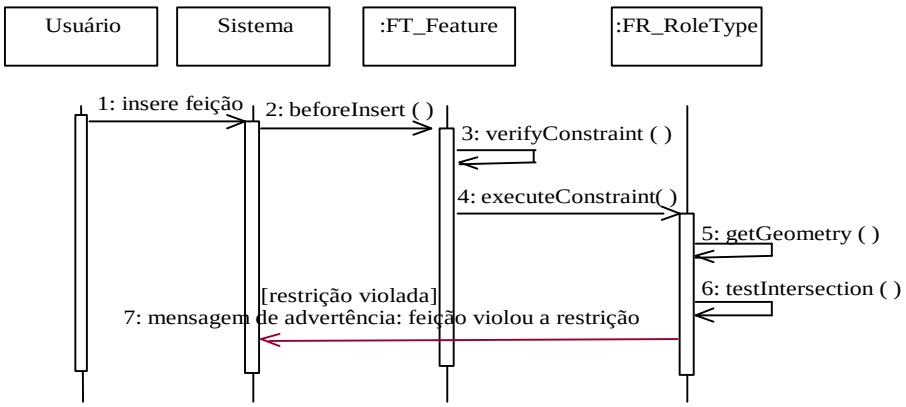



Diagrama 2

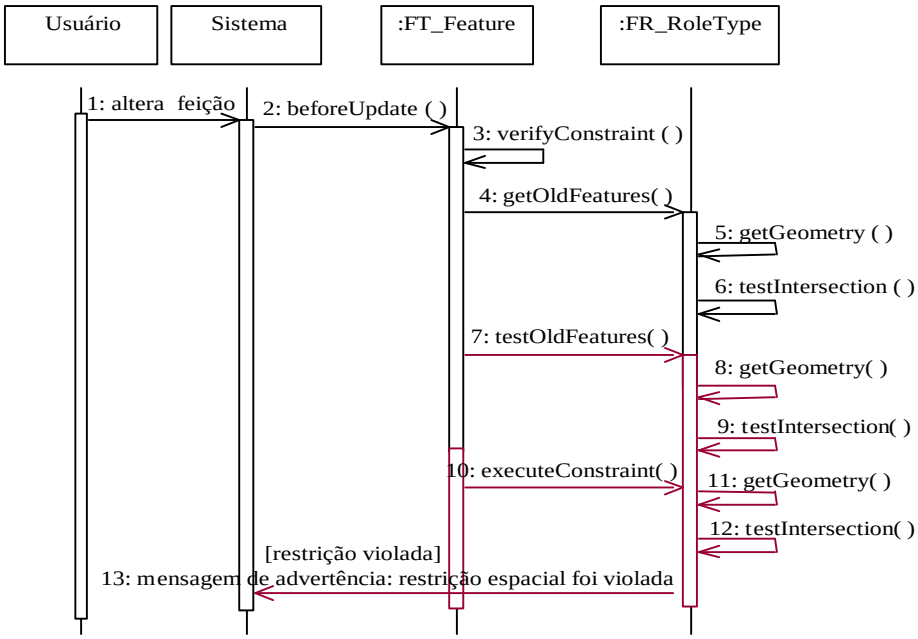



Diagrama 3

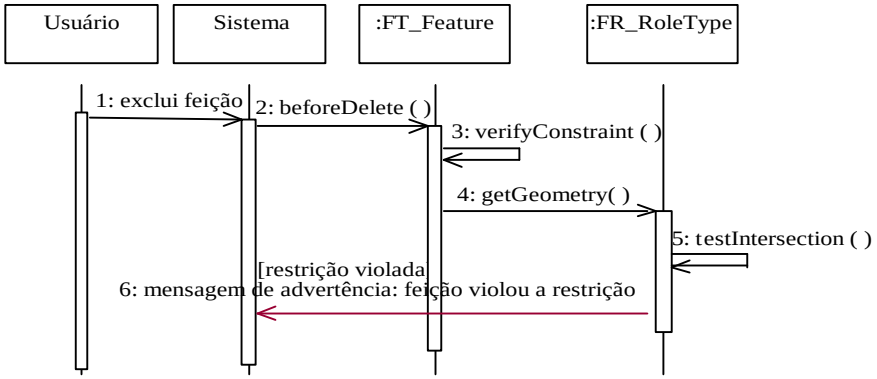



FIGURA 4.5 – Diagramas de seqüência para inserção, atualização e remoção de feições

Cada um dos métodos executa uma série de atividades, as quais são representadas, nesta dissertação, por diagramas de atividade da UML [OMG 99, FOW96, BOO 98, BOO 2000]. O algoritmo que implementa cada método, ou seja, seu diagrama de atividades, é especificado, em inglês estruturado, no Anexo 3.

As atividades com um ícone do tipo  , no canto inferior direito de sua representação gráfica, são formadas por um conjunto de sub-atividades [OMG 99].

O método *getGeometry*() tem a mesma semântica do primeiro modelo estendido. Os demais métodos incorporados ao modelo e suas respectivas atividades são descritos nas próximas subseções.

#### 4.2.1.1 BeforeInsert

O método *beforeInsert()* executa o controle das restrições topológicas dos objetos espaciais inseridos no banco de dados geográficos.

As atividades desempenhadas pelo método *beforeInsert()* são apresentadas em dois diagramas de atividade, os quais são ilustrados pela figura 4.6. O início das atividades se dá pelo diagrama 1, quando acontece a verificação da existência de uma ou mais restrições espaciais associadas ao tipo da feição *A* que está sendo inserida no banco de dados geográficos.

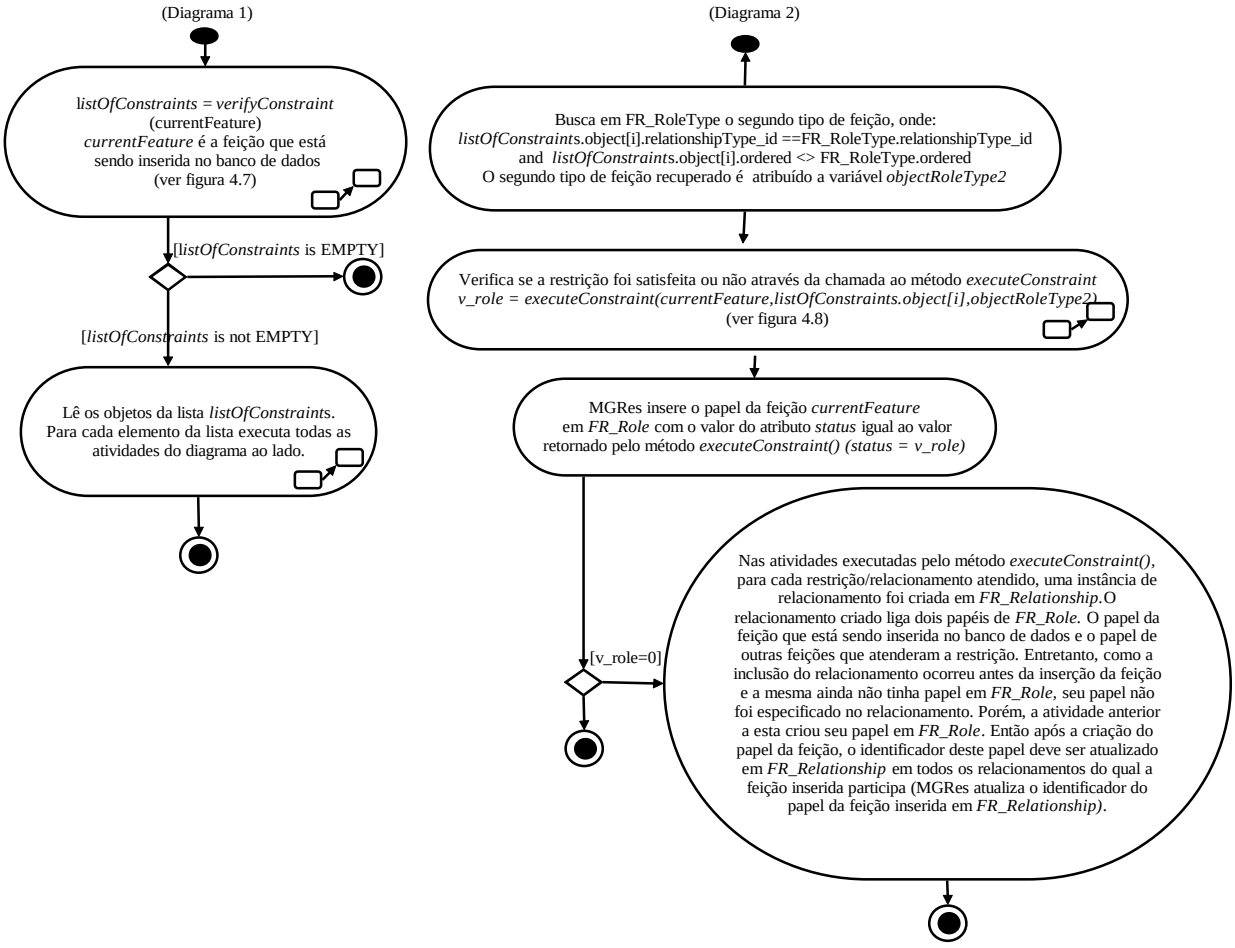



FIGURA 4.6 – Diagramas de atividades do método *beforeInsert()*

A busca propriamente dita, das restrições espaciais do tipo da feição *A* é realizada pelo método *verifyConstraint()*. Ele recupera todas as restrições topológicas do tipo da feição *A* e as armazena numa variável do tipo lista ou coleção de objetos, retornando-a para o método *beforeInsert()*. Se a lista é vazia, nenhuma restrição foi encontrada. Se a lista retornada é não vazia, cada uma das restrições nela contidas, deve ser testada.

Para testar cada restrição da lista, as atividades do diagrama 2 são executadas. As mesmas são descritas a seguir:

a) encontrar o segundo tipo de feição envolvido na restrição da lista. A partir desse tipo é possível obter as feições *B*, que combinadas com *A*, podem atender ou não à restrição;

b) após encontrar o segundo tipo de feição, o método *executeConstraint()* é invocado. Ele executa uma série de atividades próprias, ilustradas pelo diagrama da figura 4.8, e retorna o valor que o atributo *status* do papel da feição *A* deve receber, de acordo com o cumprimento ou não da restrição de *A* sendo testada. Uma das sub-atividades do método *executeConstraint()* é criar uma instância de relacionamento em *FR_Relationship* para cada feição *B* que atender a restrição de *A*. Esse relacionamento está associado a dois papéis: ao papel da feição *A;* e ao papel de *B*. Mas, como a feição *A* ainda está em fase de inserção e ainda não tem papel em *FR_Role*, o mesmo não é informado na criação do relacionamento;

c) para cada restrição testada pelo método *executeConstraint()*, o papel da feição *A*, que está sendo incluída, é armazenado em *FR_Role* com o valor do atributo *status* igual a *0* ou *–1*, de acordo com o valor retornado por *executeConstraint();*

d) se o papel da feição *A*, criado pela atividade anterior, for igual a *0*, o identificador do papel da feição *A* precisa ser atualizado em *FR_Relationship*, pois o relacionamento entre *A* e *B* foi criado na atividade *(b)*, acima, sem o identificador do papel da feição *A*.

#### 4.2.1.2 VerifyConstraint

O método *verifyConstraint()*, definido na classe *FT_Feature*, é responsável por averiguar se uma determinada feição possui algum tipo de restrição topológica. Ele avalia todas as instâncias de *FR_RoleType*. As restrições encontradas são recuperadas e inseridas numa lista, para posteriormente serem validadas. Caso nenhuma restrição seja encontrada, a lista retorna vazia.

A figura 4.7 apresenta o diagrama das atividades executadas pelo método *verifyConstraint()*.

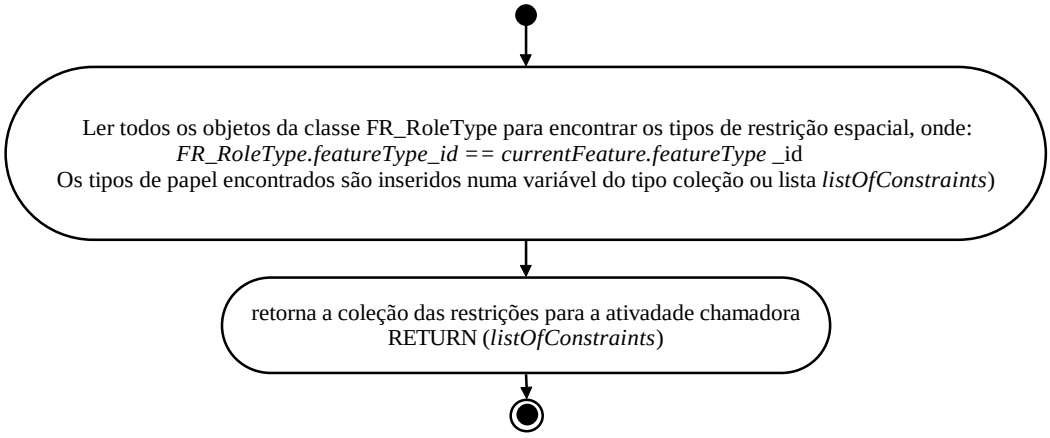



FIGURA 4.7 – Diagrama de atividades do método *verifyConstraint()*

#### 4.2.1.3 ExecuteConstraint

A consistência da feição *A* em relação a todas as suas restrições é determinada pelo método *executeConstraint()*. Para cada restrição topológica associada ao tipo da feição *A*, este método compara o valor retornado pelo método *testIntersection()*, o qual verifica se o relacionamento topológico entre duas feições existe ou não, e as cardinalidades da restrição. Com base no resultado dessas combinações, a restrição é considerada satisfeita ou não. Se a restrição é violada, o método *executeConstraint()* retorna o valor –1, caso contrário, retorna o valor 0.

Para cada tipo de restrição ou tipo de papel da feição *A*, o método *executeConstraint()* precisa encontrar as feições *B* que, combinadas com *A*, atendem ou violam as restrições da feição *A*.

As feições *B* são localizadas através de *FR_Role*, não havendo necessidade de buscá-las em *FT_Feature*, já que esta segunda extensão obriga todas as feições que têm restrições espaciais, a terem um papel em *FR_Role* para cada uma destas restrições, estando ela satisfeita ou violada.

Através do identificador do tipo de papel do segundo tipo de feição, o sistema recupera, em *FR_Role*, o papel da feição cujo atributo *roleType_id* é igual ao tipo de papel do segundo tipo de feição. Para cada papel de *FR_Role*, pode-se obter a feição *B* a ser testada, através do atributo *feature_id*.

A figura 4.8 ilustra dois diagramas de atividade que representam os procedimentos executados pelo método *executeConstraint()* para testar cada uma das restrições de *A*. Nos diagramas, a feição *A* corresponde a *currentFeature* e a feição *B* a *objectRole2.feature_id*. Estas mesmas variáveis também são utilizadas nos algoritmos do Anexo 3.

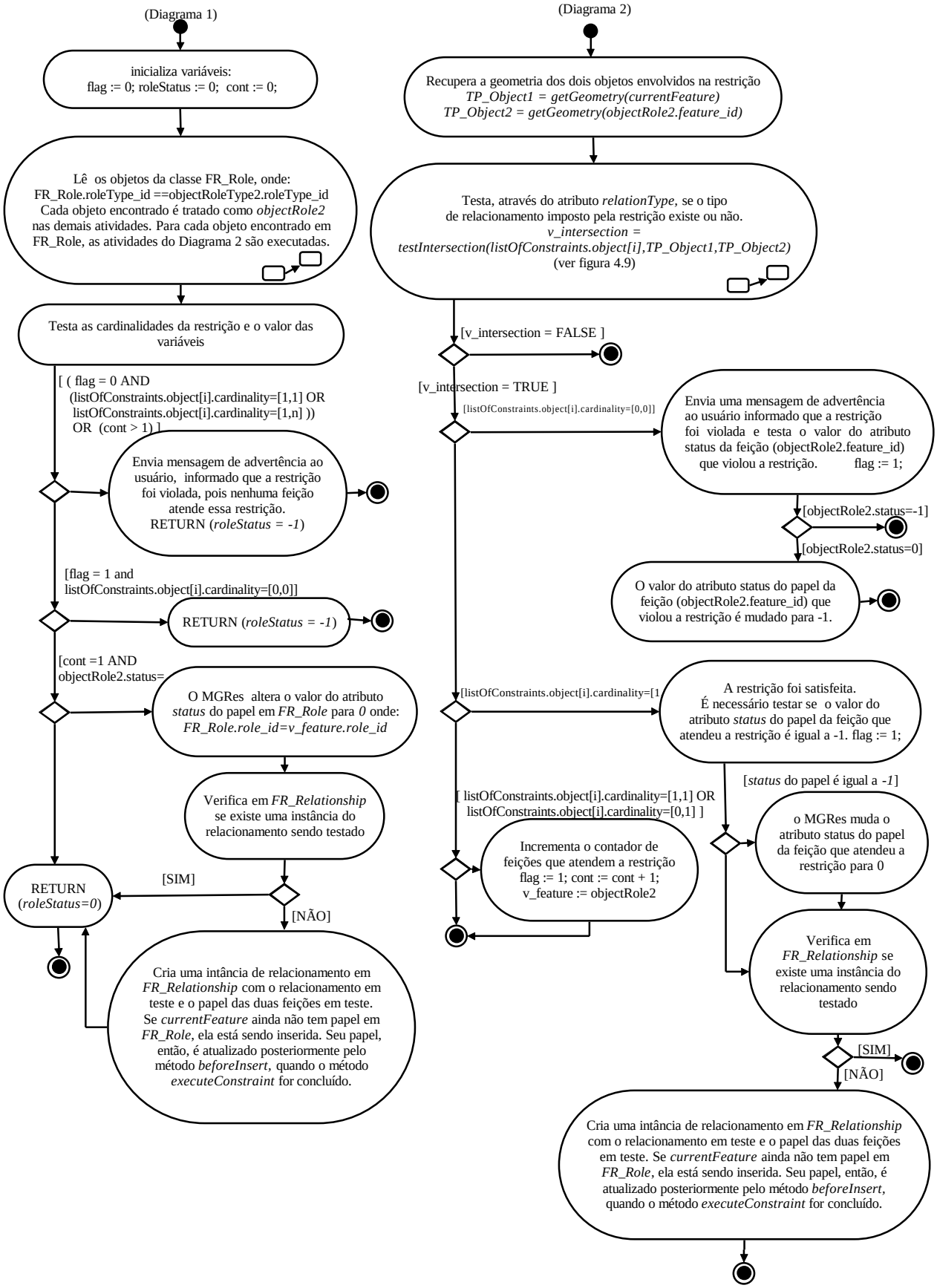



FIGURA 4.8 – Diagramas de atividades do método *executeConstraint()*

As atividades que testam cada restrição de *A* são detalhadas no primeiro diagrama da figura 4.8 e descritas a seguir:

a) lê os papéis em *FR_Role* onde o atributo *roleType_id* é igual ao tipo de papel do segundo tipo de feição envolvido na restrição. Para cada papel encontrado nesta atividade, recupera o atributo *feature_id* (que corresponde à feição *B*) e executa as atividades representadas pelo segundo diagrama de atividades da figura 4.8. As atividades do diagrama 2 determinam, para cada restrição, se o papel de *B* é consistente ou não, conforme descrito a seguir:

a.1) recupera a geometria das duas feições envolvidas no relacionamento;

a.2) testa, através do método *testIntersection()*, se o relacionamento topológico imposto pela restrição existe ou não entre os objetos espaciais recuperados em (*a.1*). Se o relacionamento não existe, ou seja, *testIntersection()* retornou *FALSE*, o teste da restrição em questão é concluído, pois se não há ligação espacial entre a geometria das feições *A* e *B*, então *B* não pode atender à restrição de *A*, a não ser que esta se baseie na relação topológica de disjunção (*disjoint)*. Caso contrário, se *testIntersection()* retornou *TRUE*, as cardinalidades da restrição são testadas para determinar o valor do atributo *status* do papel da feição *B*. Esses testes são feitos pelas atividades (*a.3),( a.4) e (a.5);*

a.3) se a cardinalidade da restrição de *A* é igual a [0,0], então uma mensagem de advertência é enviada ao usuário, pois a feição *B* violou a restrição. Neste caso, o atributo *status* do papel da feição *B* é testado. Se o mesmo for igual a *0*, este é alterado para *–1;*

a.4) se a cardinalidade da restrição de *A* é igual a [1,n] a restrição foi satisfeita. Então é necessário testar se o valor do atributo *status* do papel da feição *B* que atendeu a restrição é igual a *–1*. Em caso positivo, este deve ser mudado para *0.* Neste caso, o relacionamento topológico entre *A e B* existe, então, se ainda não existir a instância desse relacionamento, a mesma deve ser criada em *FR_Relationship;*

a.5) se a cardinalidade da restrição de *A* é igual a [0,1] ou [1,1] é necessário contar quantas feições mais atendem à restrição. Se mais de uma feição *B* atender à restrição, então a mesma foi violada, pois a cardinalidade indica que é um o número máximo de feições permitido para atender este tipo de restrição. O resultado da contagem dessas feições é testado posteriormente, pela atividade (*b.3)* descrita abaixo;

b) esta atividade é executada depois que todas as feições *B* foram testadas para a restrição de *A* em questão. Após testar todas as feições *B* para a restrição atual, através do diagrama 2 da figura 4.8, as cardinalidades da restrição são novamente testadas, agora para determinar o valor do atributo *status* do papel da feição *A* (continuação das atividades do diagrama 1)*:*

b.1) se nenhuma feição *B* atendeu à restrição de *A* e a cardinalidade da restrição é igual a [1,1] ou [1,n], então a restrição de *A* foi violada. Neste

caso, uma mensagem de advertência é enviada ao usuário e o valor *–1* é retornado pelo método *executeConstraint()*;

b.2) se uma ou mais feições *B* atenderam à restrição e a cardinalidade da mesma é igual a [0,0], então a restrição foi violada. Neste caso, o valor *–1* é retornado pelo método *executeConstraint()*;

b.3) se mais de uma feição *B* atendeu à restrição de *A* e a cardinalidade da mesma é igual a [1,1] ou [0,1], então a restrição foi violada e o valor *–1* é retornado;

b.4) se após testar todas as feições cujo tipo está envolvido numa restrição, apenas uma feição *B* atende a restrição de *A* em questão e a cardinalidade da mesma for igual a [1,1] ou [0,1] e o valor do atributo *status* do papel dessa feição *B* para a restrição é *-1*, o mesmo deve ser alterado para *0*. Neste caso, o relacionamento topológico entre *A e B* existe. Então deve ser criada uma instância desse relacionamento em *FR_Relationship*, caso a mesma ainda não exista. Em seguida, o valor *0* é retornado pelo método *executeConstraint();*

b.5) se nenhuma das atividades (*b.1), (b.2), (b.3)* ou *(b.4)* foi executada, então a restrição espacial de *A* foi atendida. Neste caso, o valor *0* é retornado pelo método *executeConstraint()*.

#### 4.2.1.4 TestIntersection

O método *testIntersection()* testa se o tipo de relacionamento topológico imposto pela restrição é verdadeiro ou não. As atividades executadas por esse método são representadas em três diagramas de atividade, conforme ilustra a figura 4.9.

Iniciando pelo diagrama 1, o *testIntersection()* executa as seguintes atividades:

a) contar o número de caracteres do atributo *relationType*. Esse número pode ser maior que um quando a restrição envolve dois ou mais relacionamentos topológicos;

b) para cada caractere do atributo *relationType*, as atividades do diagrama 2 da figura 4.9 são executadas. Essas atividades realizam os seguintes testes:

b.1) o caractere do atributo *relationType* em questão é avaliado. De acordo com o valor do caractere, ou seja, o relacionamento topológico imposto pelo mesmo, um argumento diferente é passado para a função *brelate()*, definida pelo *Modelo Abstrato* para testar o relacionamento topológico*;*

b.2) para cada resultado retornado pela função *bRelate()* é testado o número total de caracteres que compõem o atributo *relationType*, número este obtido na atividade (*a*) acima. Se esse número for igual a um, o operador lógico representado pelo atributo *logicOperator* não precisa ser testado. Então o valor retornado pela função *bRelate()* é devolvido para a função chamadora do método *testIntersection()*. Caso haja mais de um

relacionamento topológico em questão, o operador lógico é testado pelas atividades do diagrama 3, conforme descrito a seguir:

b.2.1) se a função *bRelate()* retornou *TRUE* e o operador lógico for igual a AND é necessário continuar testando os demais tipos de relacionamento impostos pelo atributo *relationType*. Caso contrário, se o operador lógico for igual a OR, o valor *TRUE* é retornado para a função chamadora do método *testIntersection();*

b.2.2) se a função *bRelate(),* retornou *FALSE* e o operador lógico for igual a AND, o valor *FALSE* é retornado para a função chamadora do método *testIntersection()*. Caso contrário, se o operador for igual a OR, é necessário continuar testando os demais tipos de relacionamento impostos pelo atributo *relationType*.

c) continuando as atividades do diagrama 1 da figura 4.9, após testar todos os caracteres de *relationType,* o valor do operador lógico é novamente testado, pois se esta atividade é executada, significa que os relacionamentos topológicos de cada restrição, combinados com o respectivo operador lógico, existem no banco de dados. Então, se o operador lógico for igual a AND, o valor *TRUE* é retornado para a função chamadora do método *testIntersection()*. Caso contrário, o valor *FALSE* é retornado.

Assim como a função *brelate(),* baseada no método *4-Intersection,* as funções *erelate()* ou *crelate()* poderiam ter sido utilizadas. A função *bRelate()* foi escolhida, nesta extensão do Modelo Abstrato, pela popularidade e facilidade de uso do método *4-Intersection.*

Caso na implementação do método *testIntersection(),* o fabricante do software preferir as duas funções não tratadas nesta extensão, o mesmo deve substituir a função *brelate()* da figura 4.9 por uma das outras funções. É necessário, também, de acordo com a função escolhida, modificar o terceiro argumento passado para a função, já que este determina o método (*4-Intersection, 9-Intersection ou DEM*) utilizado no teste da existência ou não de relacionamento topológico entre duas feições.

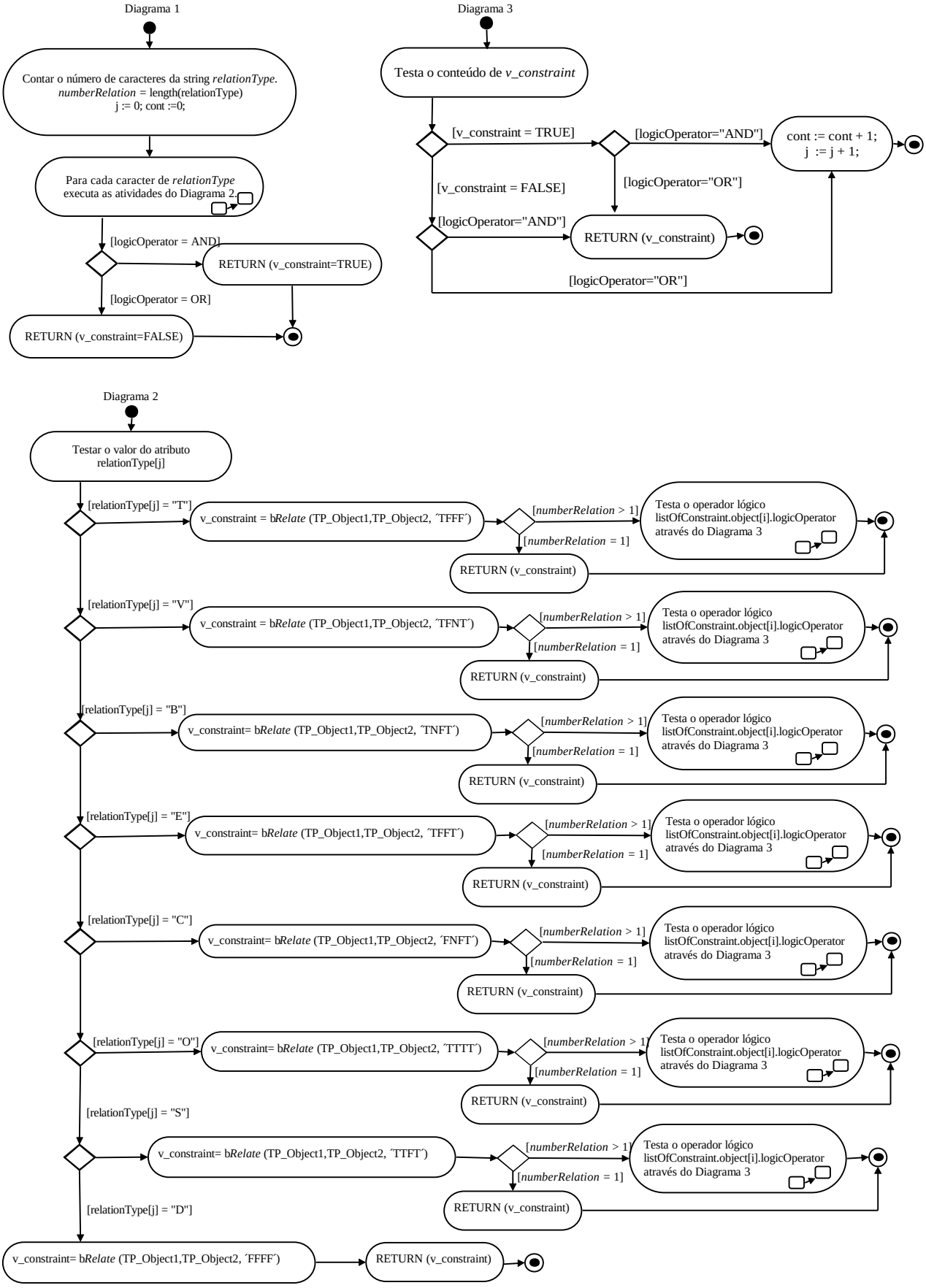



FIGURA 4.9 – Diagramas de atividades do método *testIntersection()*

#### 4.2.1.5 BeforeUpdate

O método *beforeUpdate()* executa todo processo de controle das restrições topológicas dos objetos atualizados no banco de dados geográficos. Ele verifica essas restrições através dos métodos *verifyConstraint(), executeConstraint(), getOldFeatures()* e *testOldFeatures().*

As atividades realizadas por este método são apresentadas em dois diagramas de atividade conforme ilustra a figura 4.10. Iniciando pelo diagrama 1, a primeira atividade executada pelo método *beforeUpdate()* é verificar se a feição *A*, a qual está sendo modificada no banco de dados, tem alguma restrição espacial associada a seu tipo.

A busca, propriamente dita, das restrições associadas à feição *A* é realizada através da chamada ao método *verifyConstraint()*, que retorna uma lista com todas as restrições de *A*. Se nenhuma restrição foi encontrada, o método *beforeUpdate()* é finalizado. Caso contrário, se uma ou mais restrições foram encontradas, cada uma delas precisa ser testada.

O MGRes armazena a antiga versão da feição *A* na variável *oldFeature* e a nova versão em *currentFeature.* Depois, para cada restrição da feição *A*, as seguintes atividades, ilustradas pelo diagrama 2 da figura 4.10, são executadas:

a) recupera o segundo tipo de feição envolvido na restrição de *A*, ou seja, o segundo tipo de papel;

b) recupera o papel da feição *A* sendo modificada;

c) testa o valor do atributo *status* do papel de *A* para a restrição sendo testada. Se o valor for igual a *1* ou *–1*, nenhuma feição *B* atendia essa restrição de *A* antes de sua modificação. Contudo, todas as feições *B* que, combinadas com *A*, podem atender ou transgredir as restrições de *A* precisam ser testadas com a nova geometria da feição modificada (*currentFeature*). Para isso, o método *executeConstraint()* é invocado;

d) se o valor do atributo *status* do papel da feição *A* cuja restrição está em teste for igual a *0*, as seguintes atividades são executadas:

d.1) recupera as feições *B* que atendem à restrição de *A* (*oldFeature*) antes da geometria de *A* ser modificada, e as insere numa lista de feições (*listOfFeatures*). Esse teste é feito pelo método *getOldFeatures();*

d.2) para cada feição *B* recuperada na atividade anterior é necessário testar a restrição de *B*, ou seja, se a restrição de *B* continua atendida após a alteração de *A*. Esse teste é realizado pelo método *testOldFeatures();*

d.3) com a nova geometria de *A*, novas feições *B* podem ter suas restrições atendidas ou violadas, bem como, atender ou transgredir restrições de *A*. Então, o método *executeConstraint()* é invocado para testar essas restrições.

Ao final das atividades que verificam e garantem cada restrição da feição *A*, o valor do atributo *status* do papel de *A* é atualizado em *FR_Role* com o valor retornado pelo método *executeConstraint()*.

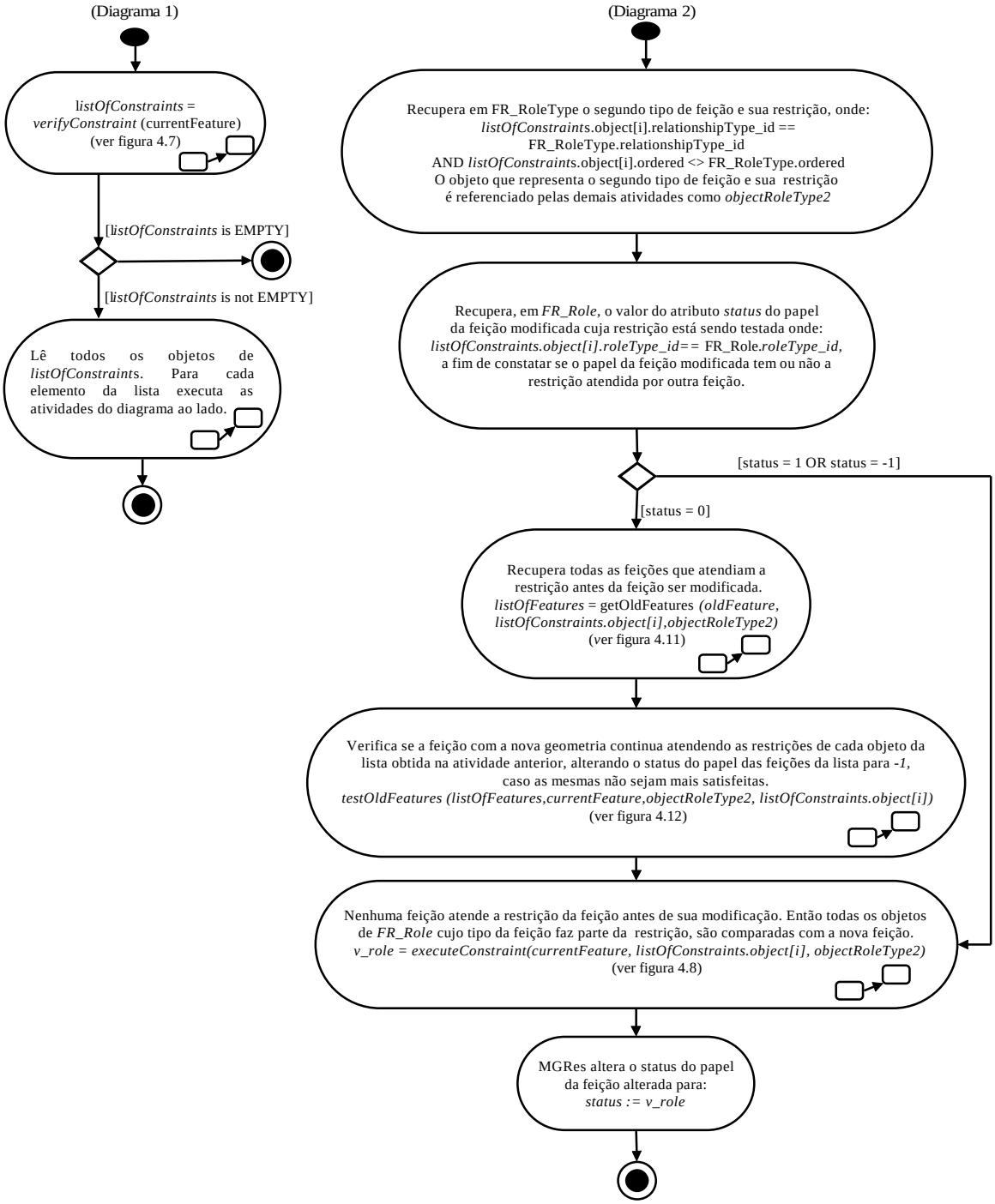



FIGURA 4.10 – Diagramas de atividades do método *beforeUpdate()*

### 4.2.1.6 GetOldFeatures

O método *getOldFeatures()* recupera todas as feições *B* que atendiam às restrições da feição *A* antes de sua modificação. Ele procura em *FR_Role* apenas os

papéis em que o valor do atributo *status* é igual a 0, pois só interessam as feições *B* cujo papel foi satisfeito pela feição *A*.

Este método só é utilizado quando uma feição é submetida a uma alteração no banco de dados. As atividades por ele executadas, para obter as feições *B* que atendem as restrições de *A*, são ilustradas pela figura 4.11 em dois diagramas de atividade.

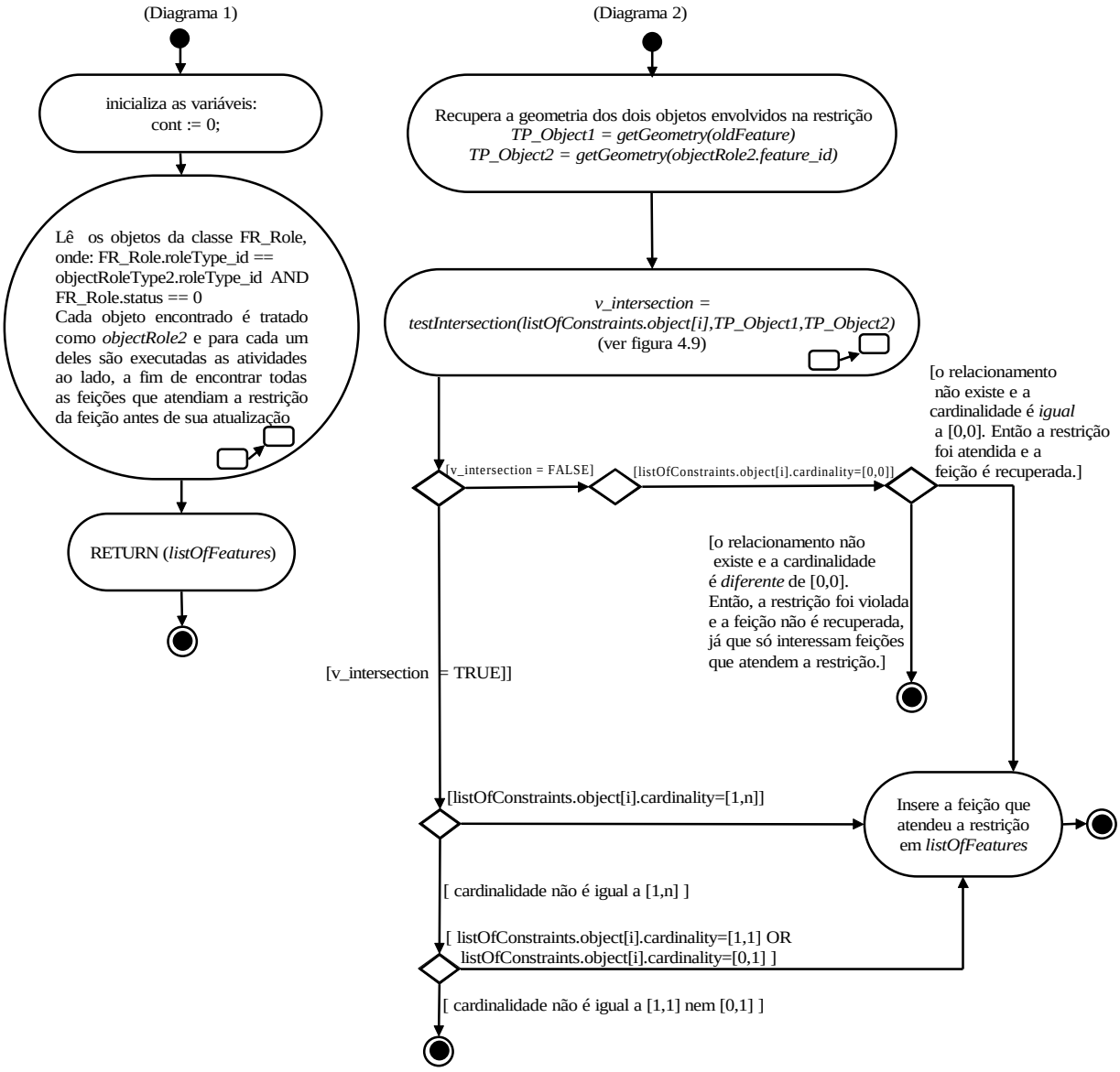



FIGURA 4.11 – Diagramas de atividades do método *getOldFeatures()*

Na segunda atividade do diagrama 1, todas as feições *B* envolvidas em algum relacionamento com *A* são recuperadas através de *FR_Role*. Entretanto, apenas as feições *B*, cujo *status* seja igual a *0* são recuperadas. Para cada um desses objetos, as atividades do diagrama 2 são executadas. Sua descrição é apresentada a seguir:

a) recupera a geometria das duas feições, *A* e *B*, envolvidas no relacionamento;

b) testa se o relacionamento topológico entre ambas as geometrias existe. Se o relacionamento não existir e a cardinalidade da restrição for igual a [0,0], então a restrição é satisfeita e a feição *B* é inserida numa lista de feições

(*listOfFeatures*). Se ao contrário, o relacionamento topológico entre *A* e *B* existe, as seguintes cardinalidades são testadas:

b.1) se a cardinalidade da restrição for igual a [1,n], então *B* atende a restrição de *A*. Neste caso, *B* é inserida na lista;

b.2) se a cardinalidade da restrição for igual a [1,1] ou [0,1], *B* atende à restrição de *A*. Como somente as feições *B* com *status* igual a *0* são testadas em cada restrição, não é necessário contar se *A* é a única feição que atende a restrição de *B*, pois o *status* do papel de *B* indica que *A* era a única feição que atendia a restrição. Então *B* é inserida na lista;

c) uma vez recuperada a lista de feições *B*, cujas restrições eram satisfeitas por *A* antes da geometria desta ser modificada, a lista de feições *B* é retornada para o método *beforeUpdate()*.

#### 4.2.1.7 TestOldFeatures

O método *testOldFeatures()* verifica se as restrições de todos os objetos *B*, recuperados pelo método *getOldFeatures()*, continuam sendo atendidas pelo objeto modificado *A*.

Iniciando as atividades pelo diagrama 1 da figura 4.12, o método *testOldFeatures()* avalia cada um dos objetos *B* da lista recuperada pelo método *getOldFeatures()*. Para cada objeto *B* da lista, as atividades do diagrama 2 da figura 4.12 são executadas. Essas atividades estão descritas abaixo:

a) recupera a geometria de *B* e a compara com a nova geometria de *A*;

b) as duas geometrias encontradas são comparadas através do método *testIntersection()*, que verifica se o relacionamento topológico imposto pela restrição das feições da lista (tratadas como *B*) existe, ou não, retornando verdadeiro ou falso, respectivamente. Caso o resultado da interseção seja *FALSE*, as seguintes cardinalidades da feição da lista são testadas:

b.1) se e a cardinalidade da restrição for igual a [1,1], a restrição de *B* foi violada. Nesse caso, o MGRes altera o valor do atributo *status* do papel de *B* para *–1*. Após, remove o relacionamento entre *A* e *B* de *FR_Relationship* e os atributos do relacionamento de *FR_RelationshipAttribute*;

b.2) se a cardinalidade da restrição de *B* for igual [1,n] é necessário verificar se a feição *A* era a única que atendia essa restrição. Caso positivo, o MGRes altera o valor do atributo *status* do papel da feição *B* para *–1* e remove o relacionamento e seus atributos de *FR_Relationship* e *FR_RelationshipAttribute, respectivamente.*

Caso o resultado da interseção, obtido na atividade (*b*), seja *TRUE*, apenas a cardinalidade [0,0] precisa ser testada. A cardinalidade [0,0] indica que o relacionamento topológico entre dois objetos espaciais é proibido. Então o MGRes altera o valor do atributo *status* do papel da feição *B* para *–1*, pois a restrição foi

violada, e remove o relacionamento entre *A* e *B* de *FR_Relationship* e os atributos do relacionamento de *FR_RelationshipAttribute.*

Este método garante que os respectivos papéis das feições *B,* cujas restrições eram satisfeitas pela feição *A* antes de sua modificação, continuem válidas, somente se a feição *A* ainda atender essas restrições após ter sido modificada. As feições *B* que não mais tiverem suas restrições atendidas, terão o valor do atributo *status* do papel da feição alterado para *–1* e o relacionamento e os atributos deste, removidos de *FR_Relationship* e *FR_RelationshipAttribute,* respectivamente,.

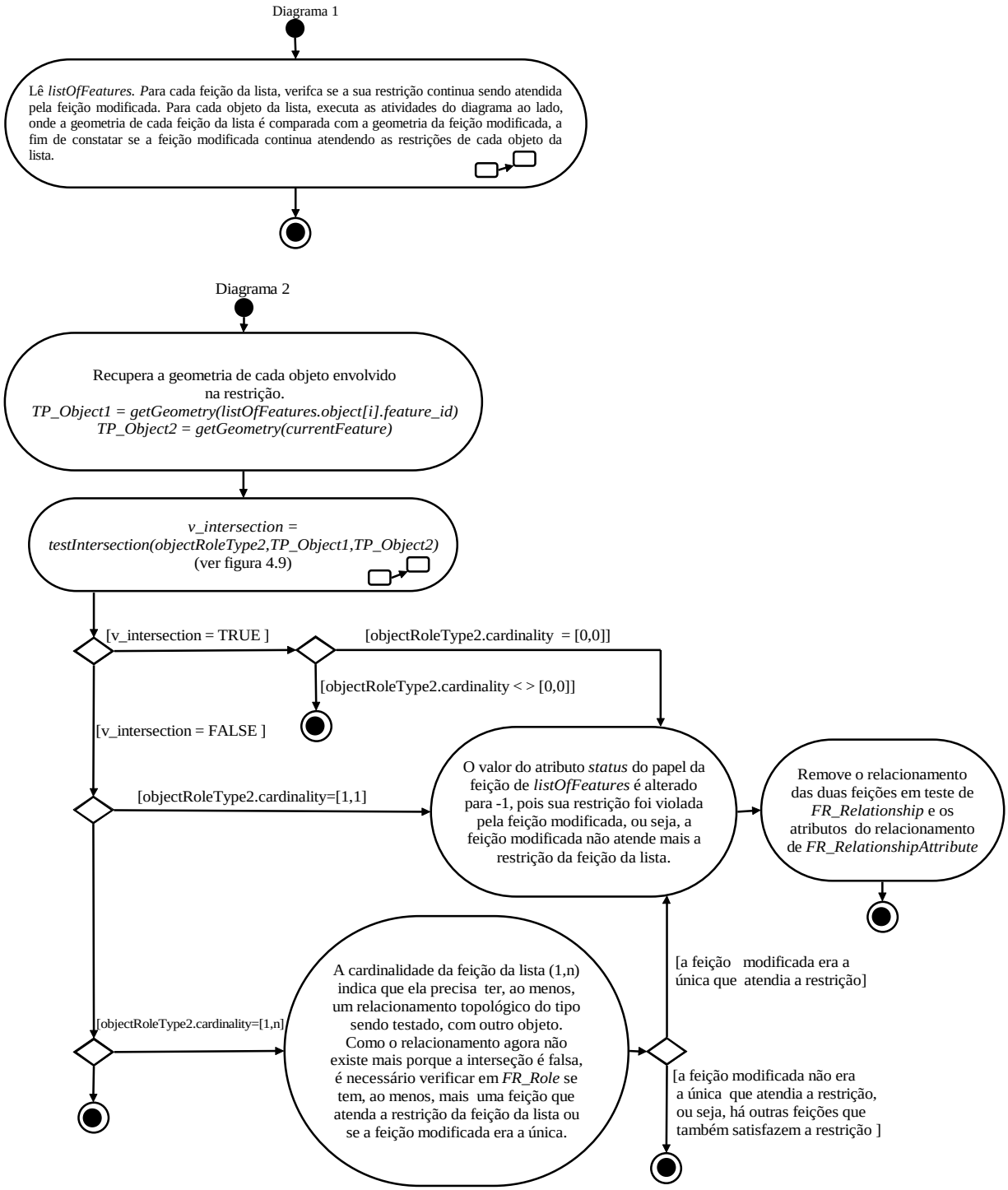



FIGURA 4.12 – Diagrama de atividades do método *testOldFeatures()*

#### 4.2.1.8 BeforeDelete

O método *beforeDelete()*, conforme já mencionado, é invocado quando o usuário submete uma operação de exclusão de qualquer feição *A* do banco de dados.

A exclusão da feição *A* não implica no teste de integridade das restrições espaciais associadas ao tipo da feição *A*, mas no teste das restrições de uma ou mais feições *B* que, combinadas com *A*, podem ter sua integridade espacial violada ou mantida.

As atividades realizadas pelo método *beforeDelete()* são representadas por três diagramas de atividade, os quais são ilustrados pela figura 4.13. Nesses diagramas, a feição *A*, sendo excluída, é representada por *currentFeature* e as feições *B* comparadas com *A*, são tratadas como *objectRole2.feature_id.*

Iniciando as atividades pelo diagrama 1 da figura 4.13, o método *verifyConstraint()* recupera as restrições espaciais da feição *A* que será removida. Se nenhuma restrição for encontrada, o método *beforeDelete()* é encerrado, permitindo a exclusão de *A* sem realizar outros testes. Caso contrário, se uma ou mais restrições forem encontradas, as mesmas são recuperadas e inseridas numa lista de objetos, pois são necessárias para obter as restrições do segundo tipo de feição envolvido em relacionamentos topológicos com a feição *A.*

A partir da lista de restrições do tipo da feição *A*, as atividades do diagrama 2 são realizadas. Essas atividades são executadas para cada restrição da lista e estão descritas abaixo:

a) recupera o segundo tipo de feição e sua restrição, ou seja, o segundo tipo de papel do relacionamento imposto pela restrição de *A* na lista;

b) para cada tipo de feição encontrado na atividade (*a*), todos os papéis em *FR_Role,* onde o atributo *roleType_id* for igual ao atributo *roleType_id* do tipo de feição encontrado na atividade (*a*), são recuperados, pois através do papel é possível obter o identificador da feição *B* a ser testada. Para cada feição *B* (atributo *feature_id* do papel de *FR_Role*), a restrição encontrada na atividade (*a*) é testada. Ela é verificada e garantida pelas atividades especificadas no diagrama 3 da figura 4.13, conforme descrito abaixo:

b.1) recupera a geometria das feições *A* e *B;*

b.2) verifica através do método *testIntersection()* se o(s) relacionamento(s) imposto(s) pela restrição de *B* existe(m) entre *A* e *B.* Se o método retornar *FALSE*, nenhuma cardinalidade precisa ser testada, pois se o relacionamento não existe, a exclusão de *A* não afeta o estado de *B.* Caso contrário, se o método *testIntersection()* retornar *TRUE*, as cardinalidades da restrição de *B* precisam ser testadas, a fim de definir o valor do atributo *status* do papel de *B*, afetado pela exclusão de *A.* As atividades (*b.3*), (*b.4*) e (*b.5*) fazem esses testes;

b.3) se a cardinalidade da restrição de *B* for igual a [0,0], então *A* viola a restrição de *B.* Entretanto, a exclusão de *A* pode tornar a restrição de *B* consistente se nenhuma outra feição violar essa restrição. Então é

verificado se *A* era a única feição que violava a restrição de *B.* Se era a única, o valor do atributo *status* do papel de *B* para essa restrição é mudado para *0*, pois com a exclusão de *A*, a restrição de *B* é atendida;

b.4) se a cardinalidade da restrição de *B* for igual a [1,n], então *A* atende a restrição de *B.* Neste caso, é verificado se *A* é a única feição que atende esta restrição de *B.* Caso *A* seja a única, com a exclusão desta, a restrição de *B* fica violada. Então, o valor do atributo *status* do papel de *B* para essa restrição é mudado para *–1;*

b.5) se a cardinalidade da restrição de *B* for igual a [1,1] ou [0,1], então a feição *A* atende a restrição de *B.* Neste caso, no máximo uma feição pode atender a restrição, razão pela qual é verificado se outras feições também atendem a esta restrição de *B.* Caso *A* seja a única e a cardinalidade da restrição é igual a [1,1], então com a exclusão de *A*, a restrição de *B* fica violada e o valor do atributo *status* do papel de *B* para esta restrição é mudado para *–1*. Se ao contrário, mais feições além de *A* satisfazem a restrição, então é verificado quantas feições mais atendem a restrição. Se apenas mais uma feição além de *A* atende a restrição, então com a exclusão de *A*, a restrição será atendida. Assim sendo, o valor do atributo *status* do papel da feição *B*, para a restrição em questão, é mudado para *0.*

Com a exclusão da feição *A*, independentemente da restrição de *B* ter sido atendida ou não, o relacionamento e os atributos deste devem ser removidos de *FR_Relationship e FR_RelationshipAttribute*, respectivamente.

Após o término das atividades acima descritas, o método *beforeDelete()* é concluído e a operação de exclusão da feição *A*, solicitada pelo usuário, pode ser efetivada no banco de dados pelo MGOpe.

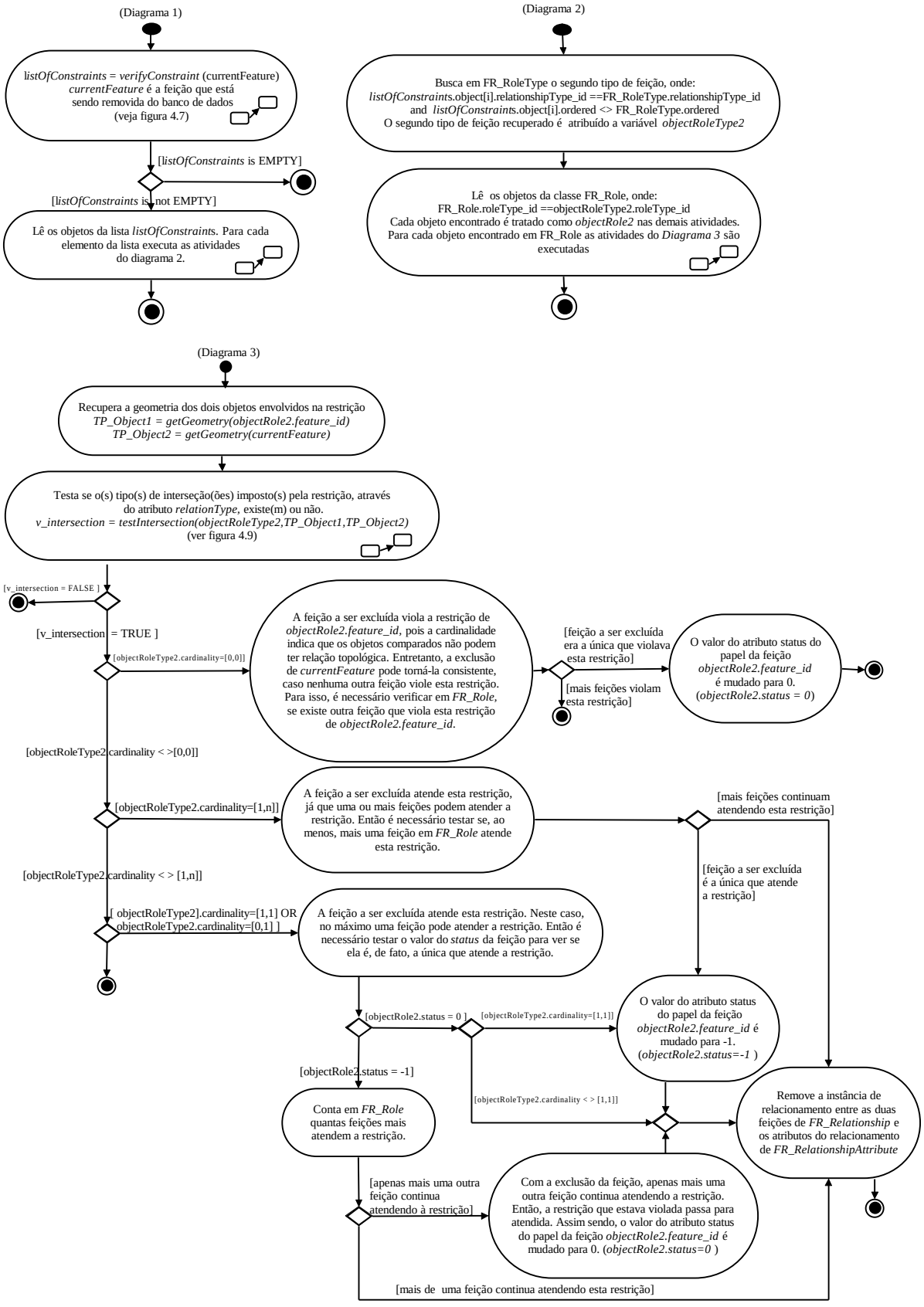



FIGURA 4.13 – Diagramas de atividades do método *beforeDelete()*

### 4.2.2 Procedimentos na Fase de Projeto

Nesta fase, todos os elementos que compõem a coluna direita do diagrama de classes ilustrado na figura 4.4 são definidos pelo projetista da aplicação. Esses elementos são os mesmos do primeiro modelo e recebem o mesmo tratamento. Por essa razão, os procedimentos necessários para a definição das restrições não são descritos nesta Seção.

### 4.2.3 Procedimentos na Fase de Produção

Nesta fase, as restrições topológicas definidas durante o projeto do banco de dados são verificadas. De acordo com a operação efetuada pelo usuário, diferentes procedimentos são executados. São estes procedimentos que executam os métodos apresentados nas Seções anteriores. Os procedimentos para inserção são especificados na Seção 4.2.3.1. Os procedimentos a serem realizados na atualização de feições são descritos na Seção 4.2.3.2. O processo de exclusão de feições é apresentado na Seção 4.2.3.3.

Operações realizadas sobre os dados geográficos podem tornar feições inconsistentes (*status=-1*). Por essa razão, quando o usuário submete o pedido de encerramento da SAGA, o MGRes e o usuário devem decidir o futuro dessas feições. O tratamento das feições inconsistentes é descrito na Seção 4.2.3.4.

Ao encerrar o tratamento das feições inconsistentes (que têm ao menos um papel em *FR_Role* onde o valor do atributo *status* seja igual a *–1*), se ainda existirem feições neste estado, o MGRes comunica o usuário para que ele possa decidir se as feições permanecerão armazenadas até o início da próxima transação SAGA, ou se devem ser eliminadas.

Os procedimentos executados pelo MGRes para garantir a integridade espacial das feições submetidas à inserção, atualização ou remoção no banco de dados são ilustradas através dos fluxogramas que seguem. Conforme já mencionado, a notação usada é a dos diagramas de atividades da UML.

A figura 4.14 ilustra o processo onde o usuário solicita a abertura de uma base de dados. O pedido passa pelo MGTra, que solicita ao MGOpe para verificar o estado da base de dados. Se ela está fechada, o MGOpe abre a base de dados e o MGTra inicia uma transação SAGA, a qual deverá encapsular o trabalho do usuário.

Caso a base de dados esteja aberta, o MGOpe envia uma mensagem ao usuário, informando que a base de dados já está aberta.

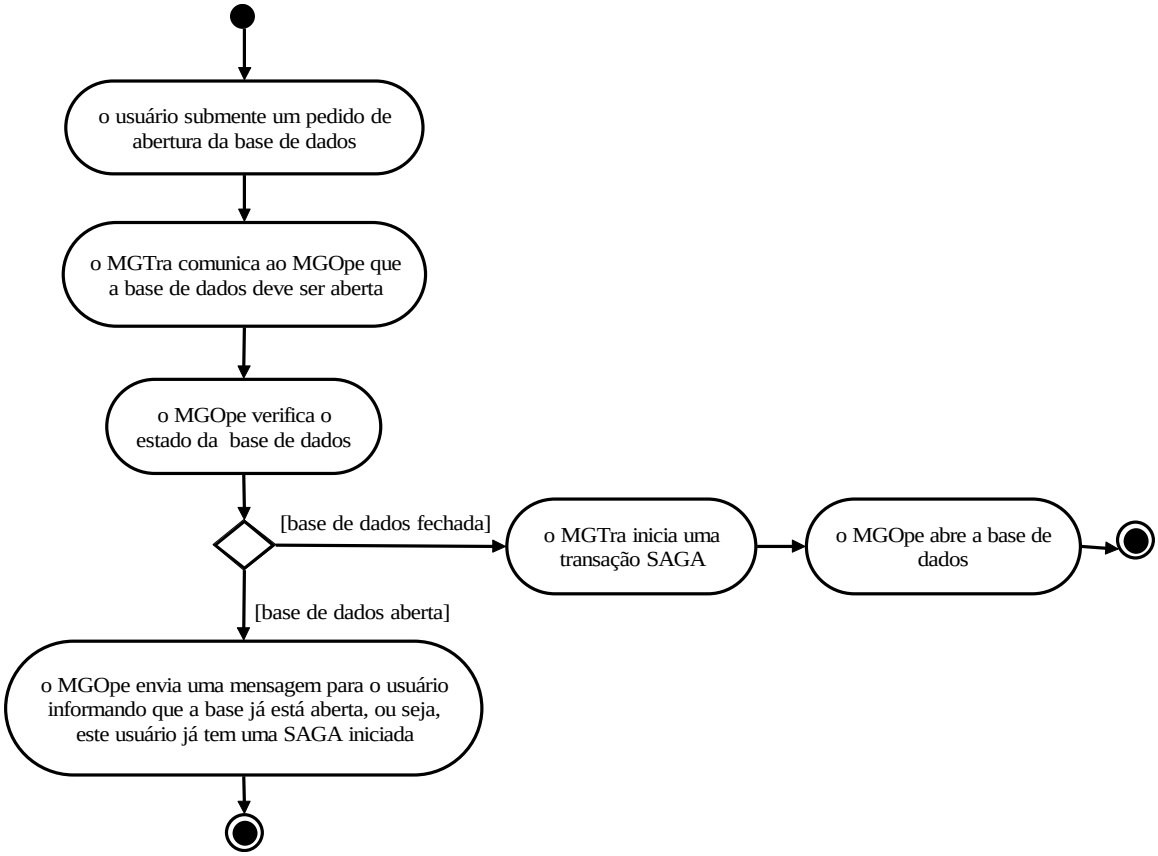



FIGURA 4.14 – Abertura da base de dados

#### 4.2.3.1 Procedimentos para a Inclusão de Feições

Após aberta a base de dados, o usuário submete uma operação de inserção. Ao executar essa ação, o MGTra verifica se existe alguma transação SAGA em execução, conforme ilustra a figura 4.15.

Caso não exista uma SAGA aberta, uma mensagem é enviada ao usuário para que o mesmo abra uma base de dados. Caso uma SAGA esteja aberta, o MGTra verifica se existe uma subtransação em execução. Caso positivo, uma mensagem é enviada ao usuário, comunicando que a subtransação em execução deve ser concluída para que uma nova subtransação para a realização da operação de inserção seja aberta. Não havendo uma subtransação aberta, o MGTra abre uma subtransação e o MGRes inicia o controle das restrições topológicas, invocando o método *beforeInsert()*.

O método *beforeInsert()* realiza todo o controle das restrições espaciais associadas ao tipo da feição passada por parâmetro para o método, e garante a integridade das mesmas. o parâmetro *currentFeature* é a feição que está sendo submetida à inserção no banco de dados. As atividades realizadas pelo método *beforeInsert()* são ilustradas na figura 4.6.

Ao final da verificação da restrição, gerenciada pelo método *beforeInsert()*, a feição é inserida no banco de dados pelo MGOpe e a subtransação é finalizada. Como a geometria da feição inserida no banco de dados pode estar inconsistente, a semântica da subtransação, que é do tipo ACID, é um pouco modificada, de forma que o banco de

dados espacial não precisa estar consistente. No entanto, todos os demais aspectos não referentes a espacialidade do objeto, precisam estar consistentes ao término as subtransação.

O controle dos estados da transação longa e das subtransações, em caso de falhas ou operações canceladas pelo usuário, é realizado pelo MGTra. Este aspecto não é abordado neste contexto, pois desvia-se do escopo do trabalho.

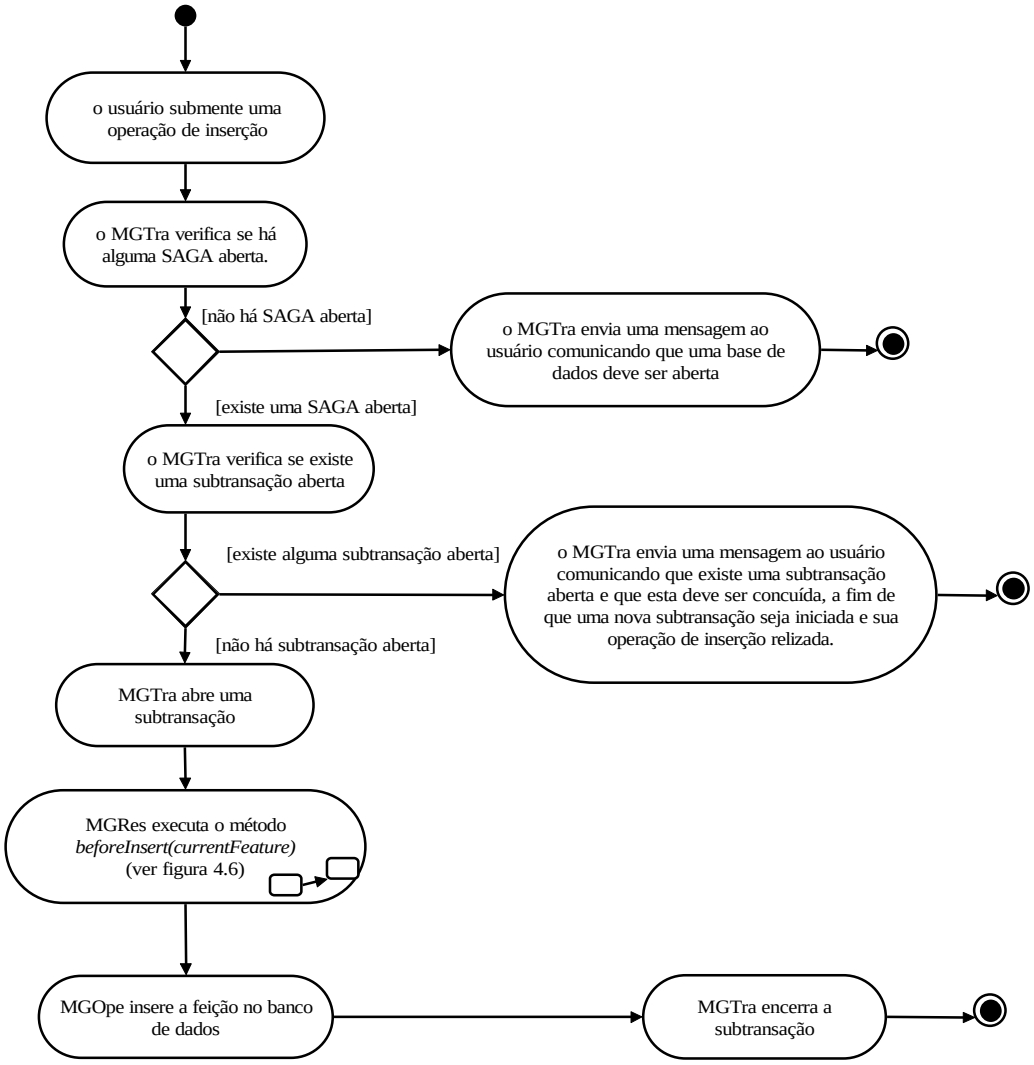



FIGURA 4.15 – Atividades que garantem as restrições na inserção de feições

#### 4.2.3.2 Procedimentos para a Atualização de Feições

Quando a geometria de uma feição é submetida a alteração no banco de dados, o MGRes precisa garantir que as restrições topológicas validadas no processo de inserção da feição sejam mantidas. Convém lembrar que as restrições, das quais trata o presente trabalho, são testadas quando a geometria da feição for modificada. Alterações de valores em atributos descritivos não afetam as restrições de integridade topológica.

Inicialmente, para que uma operação de alteração submetida pelo usuário seja executada, é necessário que o mesmo solicite a abertura de uma base de dados. O processo de abertura da base de dados é ilustrado na figura 4.14.

Uma vez aberta a base de dados, o usuário, então, pode submeter uma operação de atualização. As atividades executadas pelo SIG para gerenciar essa ação do usuário são mostradas na figura 4.16, e seguem o mesmo padrão da operação de inserção. Entretanto, o método *beforeUpdate()* é invocado para garantir as restrições espaciais.

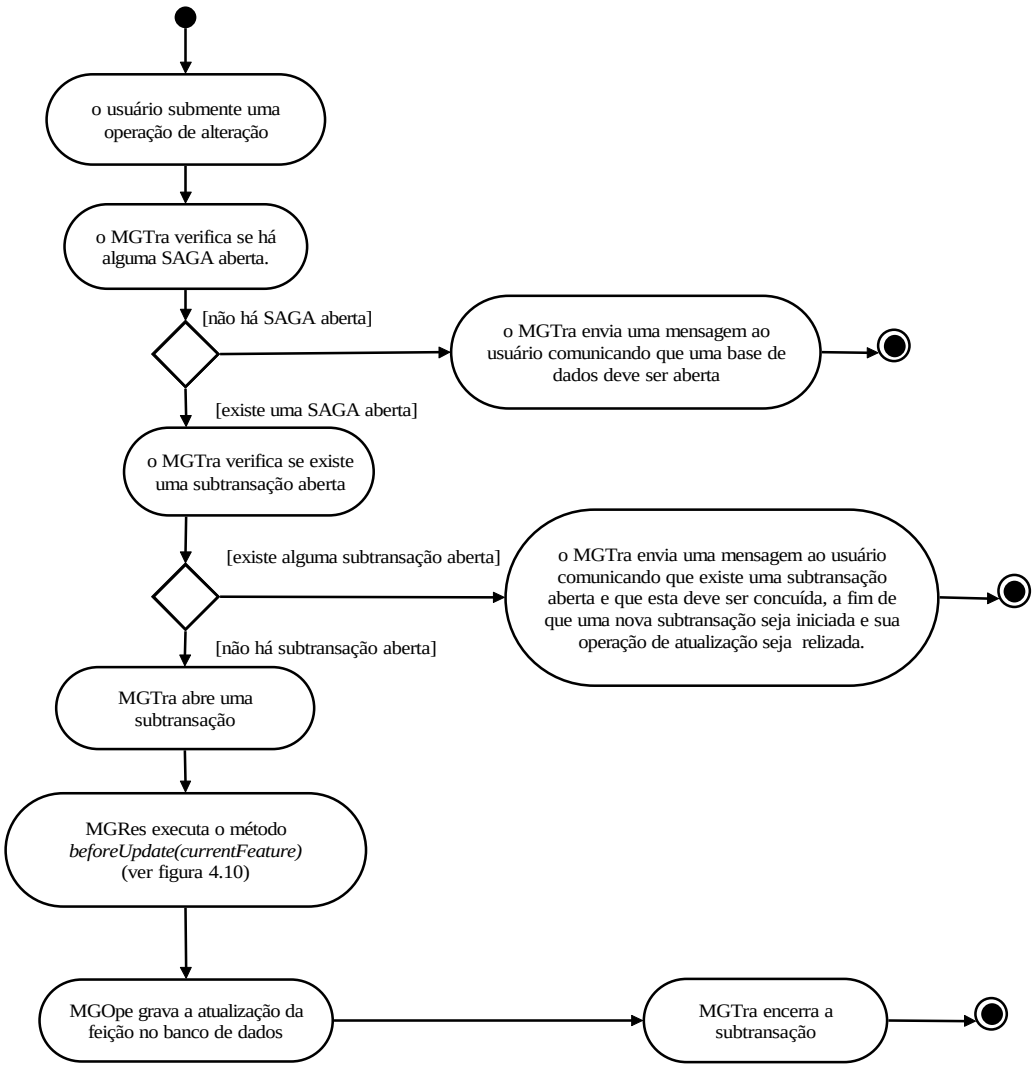



FIGURA 4.16 – Diagrama de atividades para a operação de atualização de feições

A atividade sombreada da figura 4.16 invoca o método *beforeUpdate()*, o qual desempenha uma série de atividades próprias. A ordem de execução dessas atividades é ilustrada pela figura 4.10.

No término das atividades do método *beforeUpdate()*, a feição é alterada no banco de dados pelo MGOpe e o MGTra encerra a subtransação que encapsula a operação. Contudo, a geometria da feição atualizaada no banco de dados pode estar inconsistente. Por isso, a semântica da subtransação, que é do tipo ACID, é um pouco modificada, permitindo que o banco de dados espacial não esteja consistente. Entretanto, todos os demais aspectos que não forem referentes a espacialidade do objeto, precisam estar consistentes no término as subtransação.

#### 4.2.3.3 Procedimentos para a Remoção de Feições

Quando a geometria de uma feição é submetida a exclusão no banco de dados geográficos, o MGRes precisa garantir que as restrições topológicas validadas no processo de inserção ou alteração da feição sejam mantidas.

Inicialmente, para que uma operação de remoção submetida pelo usuário seja executada, é necessário que uma base de dados esteja aberta. O processo de abertura da base de dados é ilustrado na figura 4.14.

Uma vez aberta a base de dados, o usuário pode submeter uma operação de exclusão. As atividades executadas pelo MGRes para gerenciar essa ação do usuário são ilustradas na figura 4.17. As atividades deste processo são semelhantes àquelas realizados em operações de inserção ou alteração. Contudo, o método invocado para garantir as restrições espaciais na exclusão de feições é o *beforeDelete()*.

O método *beforeDelete()* executa uma série de atividades próprias para controlar as restrições espaciais, as quais são ilustradas pela figura 4.13.

Ao final das atividades do método *beforeDelete()*, a feição é removida do banco de dados pelo MGOpe e o MGTra encerra a subtransação que gerenciou o processo de exclusão.

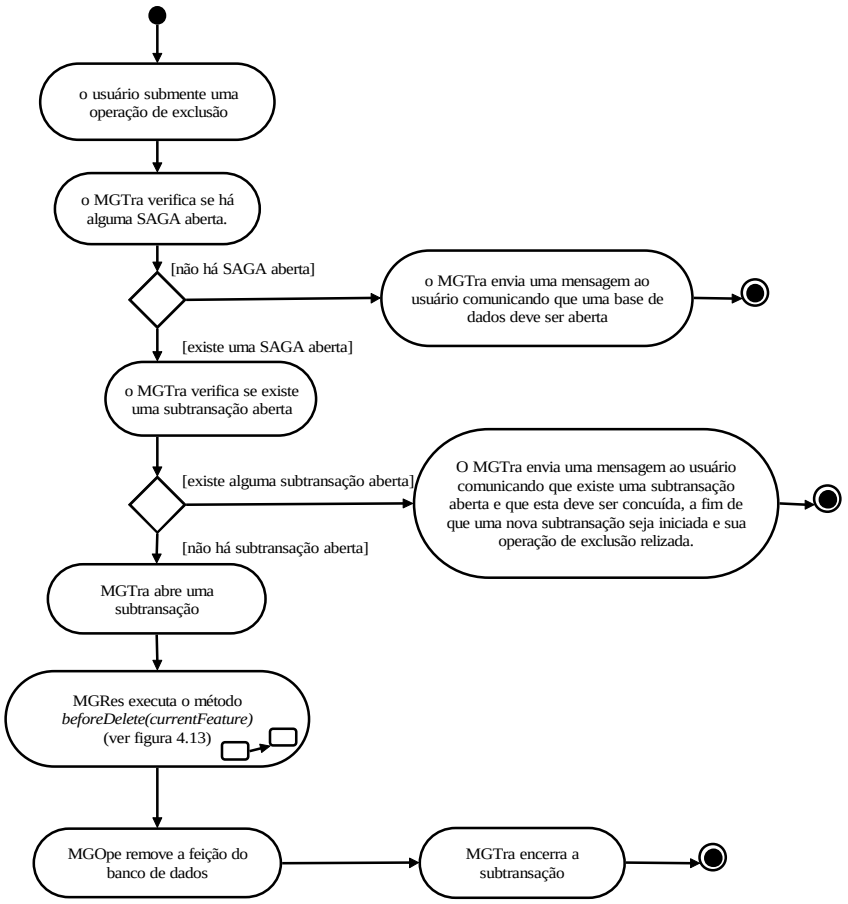



FIGURA 4.17 – Atividades que garantem as restrições na exclusão de feições

#### 4.2.3.4 Tratamento de Feições que Violaram alguma Restrição

Esta fase é responsável pelo tratamento das feições que, por alguma razão, não atenderam uma ou mais restrições espaciais. Ela é iniciada quando o usuário submete uma solicitação de encerramento da base de dados, conforme ilustra a figura 4.18. Neste momento, o MGRes verifica se existe alguma instância na classe *FR_Role,* cujo valor do atributo *status* seja igual a –1. Se encontrar, deve apresentar, através de uma interface, o conjunto de feições que violaram restrições e quais as regras violadas. A partir desta informação, o usuário deve decidir o futuro de cada feição inconsistente.

Ao final da SAGA, quando o usuário encerra a manipulação dos dados da base de dados aberta, as atividades do diagrama 1 da figura 4.18 são executadas:

a) o MGTra aciona o MGRes que verifica se existem feições ainda inconsistentes, ou seja, que apresentam, pelo menos, um papel em *FR_Role* com *status* igual a *–1*. Em caso positivo, o MGRes apresenta a lista dessas feições ao usuário, que deve decidir sobre o futuro das mesmas*;*

b) o usuário pode encerrar a SAGA, sem tratar as feições inconsistentes. Estas deverão, então, ser tratadas em uma nova sessão de trabalho, ou seja, em uma nova SAGA. Caso o usuário decida eliminar *todas* as feições inconsistentes de uma só vez, o MGRes informa ao MGOpe que as feições com algum papel em *FR_Role,* com *status* igual a *–1,* sejam removidas do banco de dados. Então o MGOpe encerra a base de dados e o MGTra encerra a transação SAGA;

c) caso contrário, se o usuário deseja tratar essas feições neste momento, então, para cada feição inconsistente, um nova subtransação é aberta, e o usuário pode realizar uma das seguintes ações:

c.1) *excluir a feição:* o MGRes informa ao MGOpe que a feição deve ser eliminada do banco de dados. Então, o método *beforeDelete()* é invocado para garantir as restrições topológicas das feições envolvidas em relacionamentos com a feição sendo excluída. Em sguida, MGOpe remove a feição, sua geometria, seus papéis, seus atributos e demais aspectos referentes a ela. Feito isso, o MGTra encerra a subtransação;

c.2) *modificar a geometria ou as coordenadas da feição*: nesse caso, o mesmo processo de controle das restrições, durante a atualização de feições, deve ser executado, através da invocação ao método *beforeUpdate()*. Após o término deste, o MGOpe grava as atualizações da feição no banco de dados e o MGTra encerra a subtransação;

c.3) *define a feição como uma exceção em um ou mais papéis violados:* o MGRes altera o valor do atributo *status* dos papéis dessa feição para *1* e em seguida o MGTra encerra a subtransação ;

d) quando todas as feições da lista tiverem sido tratadas, o MGOpe fecha o banco de dados e o MGTra encerra a SAGA.

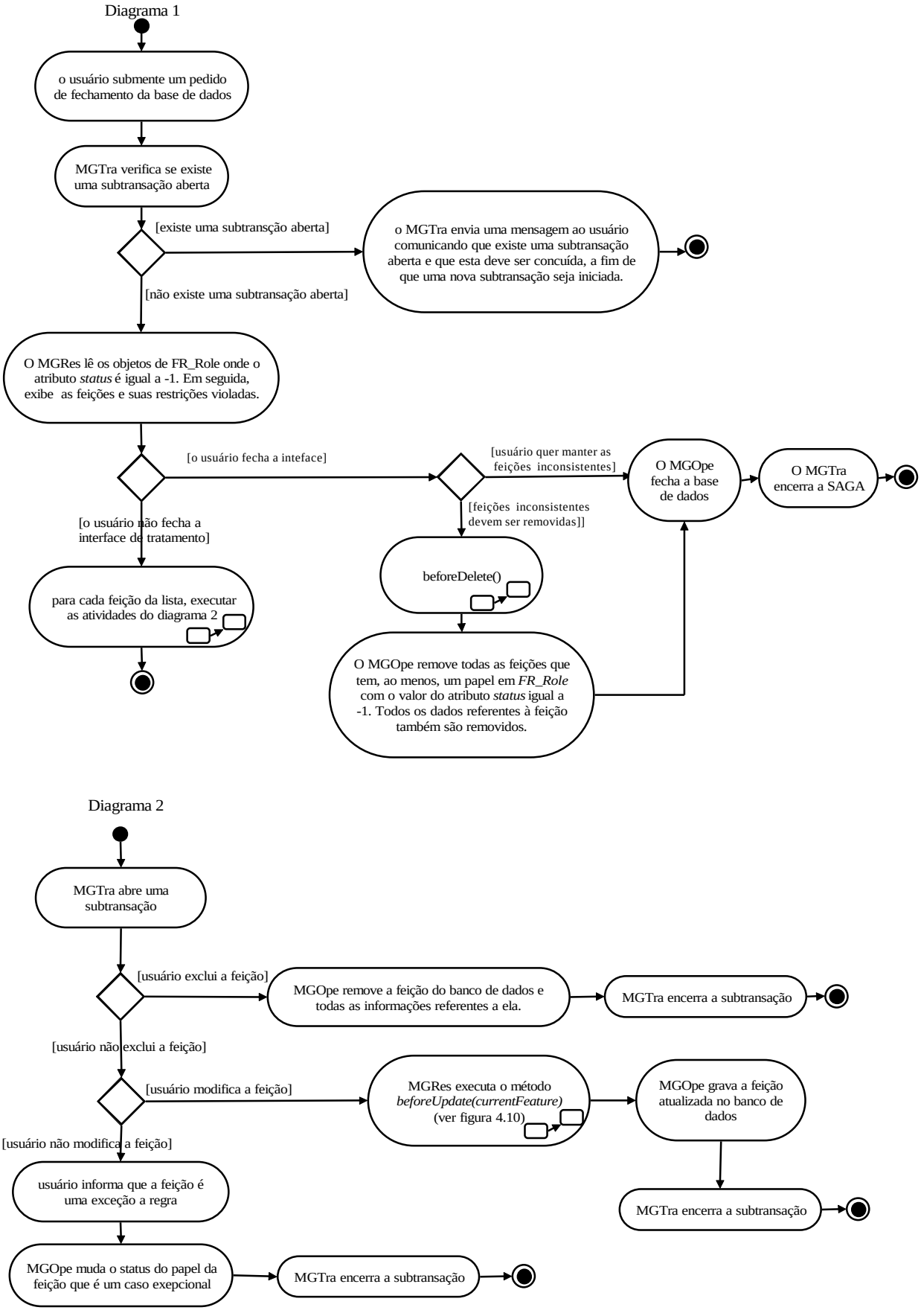



FIGURA 4.18 – Tratamento de feições inconsistentes

## 4.3 Validação da Segunda Proposta de Extensão do Modelo Abstrato *OpenGIS* Estendido

Conforme já mencionado, o *OpenGIS* é uma arquitetura para software de SIG. Para implementar e validar a segunda proposta de extensão ao *Modelo Abstrato*, desenvolvida neste trabalho, existem duas possibilidades: construir um SIG baseado na extensão do *Modelo Abstrato OpenGIS* apresentada no capítulo anterior; ou, simular um SIG baseado na extensão proposta, programando sua funcionalidade como uma aplicação sobre um SIG baseado no *Modelo Abstrato* atual.

A primeira alternativa foi descartada devido a limitações de tempo.

Buscou-se, então, uma outra forma de validar o segundo modelo proposto nesse trabalho e testar algumas das restrições espaciais identificadas no Capítulo 2. A solução encontrada baseou-se no SIG denominado *GOTHIC*, da empresa *Laser Scan* que, através de seu banco de dados geográficos, orientado a objetos, permite programar o comportamento dos objetos espaciais. Assim, foi possível implementar tipos específicos de restrições de integridade topológica. O Anexo 4 apresenta alguns dos comportamentos implementados na linguagem de programação *Lull*, linguagem essa utilizada pelo *GOTHIC*.

Com base nos recursos oferecidos pelo SIG, alguns tipos de restrição espacial foram implementados, testados e validados no *GOTHIC*. Os procedimentos para a definição das restrições são apresentados na Seção 4.3.1. O processo de verificação de restrições é detalhado na Seção 4.3.2. A Seção 4.3.3 menciona o tratamento das feições que violaram alguma restrição topológica.

### 4.3.1 Definição das Restrições Espaciais

Primeiramente, para implementar restrições topológicas foi necessário definir a topologia existente entre os objetos espaciais sobre os quais as restrições foram aplicadas. Entretanto, a topologia define apenas o tipo de relação topológica existente entre duas feições geográficas, não podendo ser definida a cardinalidade do relacionamento entre dois tipos de feição. Isso seria possível somente se o software implementasse fielmente todo *Modelo Abstrato OpenGIS* que prevê esse tipo de definição.

Alguns dos tipos de restrição implementados estão relacionados abaixo:

- rio não cruza edificação;
- rio não cobre rodovia;
- rio não é coberto por rodovia;
- rio não é igual a rodovia;
- rio não é igual a rio; e
- edificação não contém edificação.

Para definir essas restrições espaciais, um método reflexivo[3] foi definido em cada uma das classes nas quais as restrições foram aplicadas, pois não existe uma interface específica para esse fim. Esse método faz a chamada a uma ou mais das seguintes funções que garantem a restrição: *Cobre(), Coberto(), Atravessa(), Contem(), Contido(), Igual() e Toca().*

Por exemplo: para testar a restrição *rio não cruza edificação*, o método reflexivo *Restricao_Hidro_Linha_Drenagem()*, definido na classe *rio*, invoca o método *Atravessa()*, que por sua vez, garante que a relação de cruzamento não aconteça entre os tipos de feição *rio* e *edificação*. Caso a restrição seja violada, uma mensagem de advertência é exibida ao usuário e o identificador dos dois objetos que violaram a restrição, juntamente com o identificador da restrição, são criados em *restriçõesVioladas.*

Os diagramas de classes da figura 4.19 ilustram parte da modelagem de objetos dos temas Hidrografia e Edificações da 1ª DL. O método *Restricao_Hidro_Linha_Drenagem()* foi definido na classe *continental*, classe esta que generaliza os vários tipos de rio. O método garante que, ao ser inserido, atualizado ou removido um objeto do tipo *rio*, a restrição espacial será garantida, ou seja, o rio não cruzará qualquer edificação. Garante também que o *rio* não será *igual* a qualquer *rodovia* (veja o código no Anexo 4).

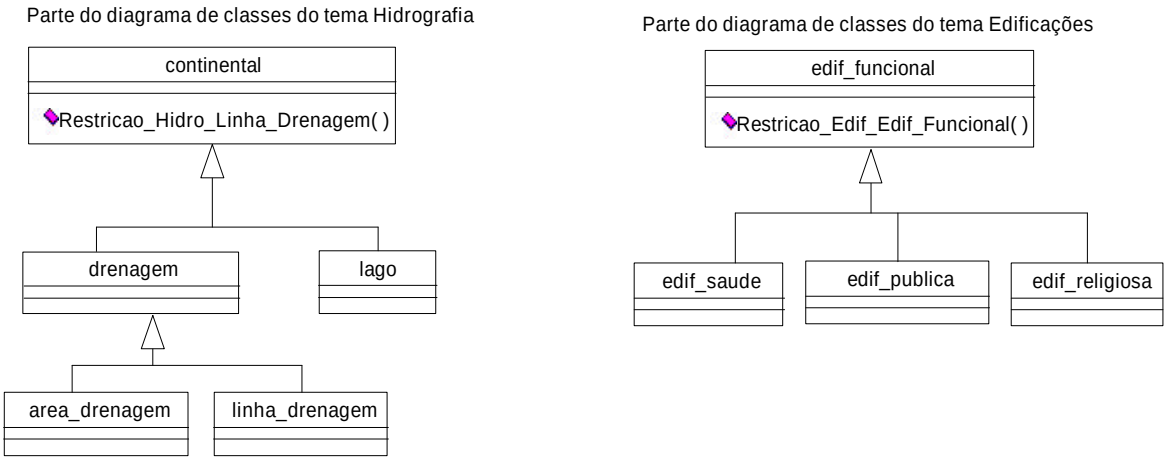



FIGURA 4.19 – Diagrama de classes para Hidrografia e Edificações

O método *Restricao_Edif_Edif_Funcional()*, definido na classe *edif_funcional*, a qual generaliza os diferentes tipos de edificação, garante que uma edificação não será cruzada por qualquer tipo de rio. Ele garante também que uma edificação (*edif_saude)* não conterá outro tipo e edificação *(edif_publica)* (veja o código no Anexo 4).

[3] Método reflexivo, no *GOTHIC*, é aquele invocado toda vez que um objeto da classe, onde o mesmo está definido, é ativado.

### 4.3.2 Verificação das Restrições

No módulo desenvolvido, a verificação das restrições é realizada através das funções genéricas *Cobre(), Coberto(), Atravessa(), Contem(), Contido(), Igual() e Toca().*

Para verificar a existência de um relacionamento topológico entre duas feições, cada uma das funções anteriores, primeiramente, avalia a forma geométrica dos objetos que estão sendo comparados. Se ambos têm a forma geométrica de um polígono, por exemplo, e a restrição é de cruzamento, a mesma não precisa ser testada, pois a relação de cruzamento só existe quando, ao menos, um dos objetos envolvidos tem a forma geométrica de uma linha.

Diferentes testes precisam ser realizados quando um dos objetos envolvidos na restrição tem a forma geométrica de um polígono. Por essa razão, duas funções diferentes foram criadas e podem ser invocadas pelos métodos citados no início da seção. São elas: *Função_Validação* e *Função_Validação_Area* (veja Anexo 4). A primeira compara somente objetos na forma de ponto ou linha. A segunda é usada quando, ao menos, um dos objetos envolvidos no relacionamento tem forma geométrica de um polígono. Dependendo, então, da forma geométrica da feição, uma das duas funções é utilizada para testar a restrição.

O *GOTHIC* gerencia as operações realizadas pelo usuário, através de transações, da mesma forma como o *Modelo Abstrato OpenGIS Estendido.* Quando o usuário abre a base de dados, o sistema abre uma transação longa. Essa transação fica aberta até que o usuário solicite o fechamento da base de dados. Só então ela é concluída.

No contexto da transação longa, o usuário pode realizar operações de inclusão, alteração e exclusão, cada uma das quais é gerenciada por uma subtransação curta. A subtransação é concluída quando o usuário confirma o término da operação.

Se alguma falha ocorrer no sistema durante a execução das subtransações, a transação longa não será desfeita, podendo recomeçar do início da subtransação que falhou.

A subtransação, uma vez concluída, não pode mais ser desfeita. Existe porém, uma função que permite desfazer a transação longa que está aberta. Ela por sua vez pode desfazer todas as suas subtransações.

### 4.3.3 Tratamento das Feições Inconsistentes

As feições inconsistentes estão armazenadas no banco de dados. Contudo, o identificador dos dois objetos que violaram alguma restrição e o tipo de restrição violada estão armazenados na classe *restriçõesVioladas*, classe essa, criada para controlar as feições inconsistentes. A estrutura dessa classe é ilustrada pela figura 4.20.

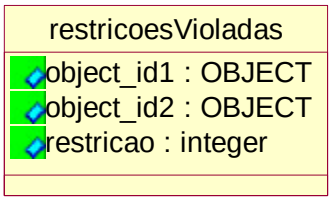



FIGURA 4.20 – Classe de objetos que violaram uma ou mais restrições espaciais

Para tratar as feições inconsistentes é necessário criar uma interface específica no SIG para que o usuário possa interagir com o sistema e tratar essas feições. Essa interface pode ser criada no *GOTHIC*, mas exige conhecimentos mais aprofundados sobre o software e o uso de um módulo específico do SIG que foi adquirido recentemente pela 1ªDL. Por essa razão, o tratamento das feições inconsistentes não foi implementado, podendo ser desenvolvido em trabalhos futuros.

Esse Capítulo apresentou uma extensão ao modelo de objetos do consórcio *OpenGIS*. Novos atributos e comportamentos foram incorporados ao modelo, oferecendo suporte a definição e a garantia de restrições topológicas. Também foi apresentada uma forma de validação da extensão proposta, onde um conjunto de comportamentos foram implementados em um banco de dados geográficos, orientado a objetos, e que permite programar o comportamento dos objetos espaciais.

# 5 Conclusões e Tendências Futuras

Restrições espaciais são um fator de grande importância em SIG porque influem diretamente na integridade e na qualidade dos dados geográficos. Este aspecto está sendo tratado com grande interesse por autores de modelos conceituais de dados geográficos, mas está sendo negligenciada pelos fabricantes de software de SIG. O desenvolvedor da aplicação ainda precisa programar as restrições de integridade espacial, o que muitas vezes é dificultado pelo grande número de regras que precisam ser implementadas.

Esta dissertação oferece uma contribuição para Sistemas de Informação Geográfica através da elaboração de um conjunto genérico de restrições topológicas que foram incorporadas ao *OpenGIS*, uma arquitetura que deve se tornar um padrão mundial para a construção de software de SIG. O SIG baseado nessa arquitetura deverá implementar, automaticamente, as restrições espaciais baseadas em relacionamentos topológicos entre dois objetos geográficos, aumentando assim, a qualidade e confiabilidade das informações espaciais.

Como pôde ser investigado, vários autores propõem diferentes métodos para definir um conjunto significativo de relacionamentos topológicos. A grande maioria dos softwares que manipulam dados geográficos apresentam operações baseadas nesses métodos, as quais verificam o tipo de topologia existente entre duas feições. Contudo, os softwares de SIG, atualmente, não fazem uso dessas operações para garantir restrições espaciais de caráter topológico.

O modelo para software de SIG, criado pelo consórcio *Open GIS*, é uma arquitetura aberta que oferece suporte a um subconjunto dos métodos que definem relacionamentos topológicos, além de permitir a definição das cardinalidades desse tipo de relacionamento. Contudo, esses recursos não são usados para garantir restrições espaciais. Por essa razão, estendeu-se o modelo de objetos *OpenGIS*, descrevendo o processo necessário para que permita a definição, a verificação e a garantia de restrições espaciais de caráter topológico.

Os métodos e atributos criados na proposta de extensão permitiram a construção dos algoritmos que garantem a integridade topológica dos dados geográficos durante a inserção, atualização e remoção de feições no banco de dados.

O uso de diagramas de atividade da UML para especificar os algoritmos que implementam os novos métodos adicionados ao modelo *OpenGIS* facilitou o entendimento do processo de teste das restrições espaciais, bem como, a implementação desses algoritmos, no futuro, por fabricantes de software de SIG que aderirem a arquitetura do OGC e a extensão proposta nesta pesquisa.

A extensão proposta suporta a definição e o processo de garantia das restrições de integridade topológica, sem alterar, demasiadamente, a estrutura original do diagrama de relacionamentos entre feições, aumentando a possibilidade da extensão ser incorporada, formalmente, ao Modelo Abstrato do OGC.

Na extensão preliminar do *Modelo Abstrato*, as feições consistentes e inconsistentes são representadas por duas classes diferentes. Ao tornarem-se consistentes, através da modificação de sua geometria ou pela exclusão/inserção de outros elementos geográficos, as feições são removidas da classe de apoio e inseridas na classe das feições consistentes. Esse processo, porém, pode comprometer o desempenho do sistema se implementado dessa forma, e tornar complexo o controle das restrições.

A avaliação da proposta preliminar, com base em alguns critérios estipulados, originou o desenvolvimento de uma nova extensão ao *Modelo Abstrato OpenGIS*. Ela trata as feições consistentes e inconsistentes como instâncias da mesma classe, porém diferenciadas entre si, através do papel que elas desempenham no relacionamento do qual fazem parte. Para cada feição associada a uma restrição espacial, um papel é armazenado no banco de dados. A situação desse papel indica se a feição desempenha-o corretamente, ou não. Quando, ao menos, um dos papéis da feição não for desempenhado conforme expressa a restrição espacial, a feição é considerada inconsistente no banco de dados.

O controle das restrições, na proposta final, é menos trabalhoso, pois não há necessidade de pesquisar objetos em duas classes diferentes e nem migrá-los de uma classe para outra.

O objetivo dessa pesquisa constitui-se na elaboração de um conjunto de restrições a serem incorporadas na arquitetura para software de SIG do OGC. Esse objetivo foi alcançado mediante o estudo dos tipos de restrições e relacionamentos espaciais, a realização de dois estudos de caso, a definição do conjunto básico de restrições topológicas, a criação de novos atributos e métodos no diagrama de relacionamentos entre feições, a descrição dos algoritmos que implementam as restrições e pela validação desses algoritmos no banco de dados geográficos *GOTHIC*.

A extensão do Modelo Abstrato e os algoritmos apresentados foram testados através de uma aplicação de SIG desenvolvida no Sistema *GOTHIC* da *LaserScan* na 1ª Divisão de Levantamento do Exército do Brasil em Porto Alegre. Através dessa implementação, foi possível constatar que um método genérico para cada tipo de restrição topológica de continência, igualdade, cobertura, cruzamento, etc, é suficiente para representar um grande conjunto de restrições topológicas do mundo real. Para cada nova classe criada no banco de dados, basta que o usuário defina um método reflexivo que faça uma chamada à função genérica desejada e a restrição espacial será testada e garantida para todos os objetos daquela classe.

O trabalho contribuiu para a Primeira Divisão de Levantamento do Exército Brasileiro de forma que o usuário da base cartográfica apenas menciona as restrições espaciais a serem obedecidas e o sistema validará as mesmas automaticamente. A base de dados da 1ªDL, no *GOTHIC*, está em fase de construção, devido a migração dos dados de um banco de dados relacional para o formalismo da orientação a objetos.

Através desta pesquisa, bem como da bibliografia utilizada, este documento poderá ser uma fonte de apoio para trabalhos futuros. Alguns desses trabalhos podem, por exemplo, possibilitar a extensão do conjunto de restrições topológicas para um conjunto maior, envolvendo restrições baseadas em relacionamentos métricos e de orientação. A extensão do modelo do OGC para suportar esses tipos de restrição fica facilitada porque a estratégia a ser usada será a mesma empregada para restrições

topológicas. Além disso, o OGC já oferece alguns comportamentos que são a base para a definição destes tipos de restrição.

Este estudo de relacionamentos e restrições espaciais deverá ser utilizado, também, na extensão do *framework* conceitual de dados geográficos *GeoFrame*. O objetivo é estender o *GeoFrame*, buscando o aperfeiçoamento do mesmo, para que abstraia da melhor forma possível os mais diversos tipos de relacionamentos topológicos e as regras da realidade geográfica que precisam ser identificadas na fase do projeto do banco de dados geográficos.

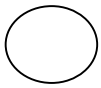
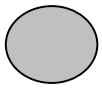
Por último, mas não menos importante, pretende-se traduzir a segunda extensão apresentada neste trabalho para os padrões de documentação do OGC e, em seguida, enviá-la ao consórcio *OpenGIS*, a fim de tornar esse trabalho um submodelo, autêntico, do *Modelo Abstrato OpenGIS.*

# Anexo 1 Representação Gráfica da Interseção entre o Limite e o Interior de Duas Regiões Geográficas

A figura abaixo ilustra a representação gráfica da interseção entre o limite e o interior de dois objetos geográficos representados na forma de polígonos. A mesma representação pode ser aplicada para combinações entre ponto/ponto, ponto/área, ponto/linha, linha/área e linha/linha.

Para que em cada um dos dezesseis casos da figura anterior o conjunto de

combinações forme um relacionamento espacial válido, é necessário que as quatro possíveis interseções sejam analisadas em conjunto.

 Limite  Interior

# Anexo 2 Estudos de Caso de Restrições Espaciais

## 1.1 Estudo de Casos Realizado na Primeira Divisão de Levantamento do Exército Brasileiro

A Primeira Divisão de Levantamento do Exército Brasileiro é uma das organizações responsáveis pela produção da base cartográfica do território brasileiro. As atividades desenvolvidas na 1ª DL são realizadas empregando-se o que há de mais moderno na produção cartográfica digital. Dentre as principais atividades, além daquelas realizadas para o interesse do Exército, destacam-se: o mapeamento sistemático, o levantamento de áreas patrimoniais, o estudo de novas tecnologias e a estruturação de dados para SIG.

Os produtos cartográficos originários das atividades desenvolvidas na instituição são elaborados de acordo com os mais rígidos padrões cartográficos existentes sendo, portanto, de grande confiabilidade para aplicações práticas em geral. Dentre os produtos gerados estão:

- cartas topográficas;
- ortofotocartas;
- cartas de imagem de satélite;
- modelos digitais do terreno;
- bases cartográficas estruturadas para SIG.

As regras e normas descritas, abaixo, foram obtidas através de um estudo de caso realizado na Primeira Divisão de Levantamento do Exército Brasileiro (1ª DL), o qual foi baseado em entrevistas com funcionários da entidade e na documentação de leis criadas para tentar garantir a qualidade e a integridade dos dados geográficos durante o processo de construção da carta. Esse processo é dividido em fases, algumas das quais são apresentadas a seguir.

### 1.1.1 Regras Definidas para as Fases da Restituição e da Validação

As regras relacionadas a seguir, também conhecidas como *leis do modelado,* referem-se às linhas de talvegue[4] e às linhas de festo[5], os principais elementos da modelagem do terreno. São regras que nada têm de absoluto, todas comportam exceções. São variáveis com a superfície terrestre e dizem apenas a forma ideal para a qual tendem os terrenos, normalmente sujeitos à erosão regular das águas.

---

[4] É a ligação de duas encostas que formam um ângulo côncavo. Também conhecida como linha de fundo, de reunião das águas ou fundo de vale.

[5] Ligação de duas encostas que formam um ângulo convexo, localizado na parte mais alta de um monte. Também pode ser chamado de linha de crista, de cumiada ou divisora das águas.

a) De um ponto qualquer do terreno, pode-se descer até o mar sem nunca subir. É a lei da continuidade dos declives. Uma exceção é o caso de uma bacia fechada.

b) A declividade de uma linha de talvegue ou de um curso d'água decresce de montante para jusante. Isso porque tendo, o rio maior volume d'água à jusante, a erosão far-se-á sentir mais à proporção que ele se aproxima do ponto onde desemboca.

c) No ponto de confluência do rio principal e um afluente, o ângulo da montante é menor que a jusante, ou seja, o escoamento do afluente ocorre na mesma direção do principal.

d) Uma linha de festo sempre se liga a outra que, por sua vez, se liga a uma terceira, e assim, sucessivamente.

e) Qualquer curso d'água está compreendido entre duas linhas de festo que, desde a origem até a foz, vão se afastando à medida que descem e seu declive vai diminuindo.

f) Quando diversos cursos d'água, partindo de um ponto central, seguem direções diversas, há ordinariamente na sua origem, um ponto culminante.

Além de regras para a modelagem do terreno, existem algumas outras, necessárias para abstrair e representar corretamente a forma do terreno e os elementos que o constituem. Dentre elas, existem regras que devem ser obedecidas durante a criação das curvas de nível e na representação da planimetria.

a) Eqüidistância entre as curvas: as curvas que compõem as isolinhas devem ser traçadas na mesma variação de altitude, dentro de uma determinada escala. Na escala 1:25.000, por exemplo, a variação da altitude é de 10 metros; na escala 1:50.000, a variação é de 20 metros. A cada cinco isolinhas projetadas é definida uma cota, representando o valor da altitude daquela linha. Ela é denominada *mestra*.

b) Regra do paralelismo: em escalas pequenas, as curvas de nível apresentam uma simetria entre si.

c) Uma isolinha, onde intercepta rios e afluentes, deve estender-se mais sobre o rio principal.

d) A isolinha, onde intercepta um corpo d'água, deve formar um vértice, indicando o volume d'água.

e) Topo da elevação: o topo de uma elevação deve sempre estar associado a uma cota, que corresponde à altura da elevação.

f) Sela: toda sela (depressão entre duas elevações) deve estar associada a uma cota.

g) Anomalias do terreno: toda a anomalia de terreno, como uma cachoeira, por exemplo, também precisa estar associada a uma cota.

h) Todo limite de linha precisa tocar o limite ou o interior de outro objeto geométrico. Exemplos: uma cerca inicia numa estrada e termina em um rio; toda estrada precisa ter o ponto final tocando algum objeto geométrico como uma casa, um bosque, outra estrada, etc. Nenhuma linha pode ter ponta livre.

i) Cada porção de tipo de vegetação é tratado, espacialmente, como um polígono. Toda a área desse polígono precisa ser fechada.

### 1.1.2 Normas Provisórias para a Fase da Generalização

O termo *generalização* é usado em muitos contextos. Na cartografia, a generalização de mapas [WEI 98] é o processo de derivar, a partir de uma detalhada base de dados espacial de origem, um mapa ou uma base de dados com conteúdo e complexidade reduzidos, eliminando os menores objetos espaciais da base, mas conservando as características estruturais e a semântica dos dados.

Considerando, na escala 1:50.000, que a eqüidistância entre curvas de nível varia de 20 em 20 metros (20, 40, 60...), que a cada 50 metros existe uma curva denominada mestra, e que a visão humana permite visualizar objetos com dimensão acima de 2,5 milímetros, algumas das restrições que precisam ser observadas e aplicadas durante o processo de generalização, estão relacionadas a seguir:

a) caminhos menores que 1 cm na escala da carta precisam ser eliminados, desde que não conduzam a nenhum lugar importante e que não necessitem ser representados no mapa;

b) edificações impossíveis de serem representadas pelo símbolo mínimo devem ser eliminadas;

c) cercas menores que 1 cm na escala da carta também precisam ser eliminadas;

d) eliminar curvas-de-nível intermediárias de 10, 30, 70, 90 ... metros;

e) curvas-de-nível mestras de 50, 150, 250 ... metros devem ser eliminadas;

f) limites de matas, macegas, culturas e arrozais menores que 25 milímetros quadrados na escala da carta devem ser excluídos;

g) eliminação de cursos d'água (fundos de vale) menores que 2 cm na escala da carta, desde que não tenham justificado a construção de uma obra de arte que deva ser representada.

h) linhas não representativas na escala da carta também devem ser eliminadas;

i) eliminação de cotas excessivas e respectivos textos, através de uma interação com o operador.

Além da eliminação dos elementos acima relacionados, as seguintes tarefas devem ser executadas:

- reescalonamento de células e toponímia;

- substituição dos fenômenos espaciais não mais passíveis de representação em escala por células;
- no caso da ligação interna de um conjunto de áreas em um objeto maior, deverá ser mantido apenas um centróide para cada nova área. Após isso, deve-se novamente ressimbolizar as feições e gerar um novo arquivo de áreas da categoria modificada.

### 1.1.3 Normas Para a Fase da Edição

Terminada a generalização, realiza-se a edição. A edição vetorial consiste na transformação dos arquivos vetoriais gerados nos processos de aquisição de dados (vetorização de arquivos gerados da digitalização matricial dos originais cartográficos, restituição fotogramétrica, etc.) em produtos que, face as suas características visuais, facilitarão a interpretação humana. Nesta fase são acrescentados os símbolos, que representam as diversas feições topográficas estabelecidas nos manuais técnicos T34-700 1ª Parte – Convenções Cartográficas: Normas para o Emprego de Símbolos e T34-700 2ª Parte – Convenções Cartográficas – Catálogo de Símbolos, além dos cabeçalhos, rodapés, inscrições marginais e textos.

As regras citadas abaixo, são algumas das diversas operações a serem realizadas na fase de edição vetorial, estabelecendo critérios de normalização para a sua execução:

a) revisão do posicionamento da toponímia, o qual já vem preparado da fase de validação, com o conteúdo das fotos preambuladas;

b) montagem do submodelo articulação da folha, colocando os nomes das folhas vizinhas. Os nomes das folhas em trabalho, quando definidos, também devem ser acrescentados;

c) impressão de altimetria e hidrografia;

d) revisão dos nomes de rios e outros elementos hidrográficos, os quais já vêm preparados da fase de validação, com o conteúdo das fotos preambuladas;

e) indicação das cotas a serem representadas;

g) os textos ao longo de linhas (rodovias, ferrovias, rios, etc.) devem ser colocados com um afastamento de

h) 1mm na escala da carta;

g) as passagens elevadas de rodovias, ferrovias e pontes, em rio de margem dupla, quando necessário, podem ser desmembradas, a fim de estender-se as linhas retas, movendo-se, após, as cabeceiras;

## 1.2 Estudo de Casos do Programa Pró-Guaíba

O Estado do Rio Grande do Sul, através de sua Secretaria de Coordenação e Planejamento (SCP), elaborou o Programa para o Desenvolvimento Racional,

Recuperação e Gerenciamento Ambiental da Bacia Hidrográfica do Guaíba, denominado Programa Pró-Guaíba.

A recuperação da Bacia Hidrográfica do Guaíba envolve a regulamentação e o planejamento do uso da água e de outros recursos naturais da região, através da integração de esforços de órgãos públicos e privados e da definição de prioridades no uso e preservação dos recursos naturais [GAR 98].

O principal objetivo do Programa é criar, para a área da Bacia Hidrográfica do Guaíba, condições necessárias para a utilização racional de seus recursos naturais, a recuperação da qualidade ambiental nas áreas urbanas e rurais, bem como o manejo, ambientalmente sustentado da produção agrícola, pecuária, florestal e industrial.

A área geográfica abrangida pelo Programa compreende a Bacia Hidrográfica do Lago Guaíba. Conforme o Conselho de Recursos Hídricos do Estado, a mesma é formada por oito Sub-Bacias, caracterizando-se como a mais importante do Estado do Rio Grande do Sul [GAR 98].

O Programa constitui-se de um conjunto de projetos coordenados e executados pela Secretaria de Coordenação e Planejamento e co-executados por nove instituições de âmbito estadual e municipal.

Entre os projetos que fazem parte do Programa está o Sistema de Informações Geográficas denominado SIGPROGB. Ele trata da criação de uma base de dados geográfica para apoio ao planejamento e gerenciamento ambiental da Bacia Hidrográfica do Guaíba. Essa base de informações é composta pelos principais indicadores do Sistema, com abrangência em todos os projetos do Programa. Durante a modelagem conceitual dessa base de dados, foi identificado um conjunto de regras da realidade geográfica que precisam ser garantidas pelo sistema. Essas regras, de acordo com o seu tipo, são apresentadas nas Seções abaixo.

### 1.2.1 Regras de Continência (*contains/inside*)

Regras de continência são aquelas em que toda a parte interna de um objeto do mundo real, faz parte do interior de outro objeto, com o qual está relacionado. Com base nesse conceito, foram encontradas as seguintes restrições espaciais de continência:

a) uma bacia hidrográfica contém um ou mais recursos hídricos (bacia hidrográfica $_{[1,n]}$ *contains* recurso hídrico $_{[1,n]}$);

b) um recurso hídrico é formado por um ou mais trechos. O conjunto de trechos compõe um recurso hídrico. (recurso hídrico $_{[1,1]}$ *contains* trecho $_{[1,n]}$ );

c) uma bacia hidrográfica é composta ou não de sub-bacias (bacia hidrográfica $_{[1,1]}$ *contains* sub-bacias $_{[0,n]}$ );

d) toda capital precisa estar contida dentro de um e somente um país (capital $_{[1,1]}$ *inside* país $_{[1,1]}$);

e) uma bacia contém vários municípios (bacia hidrográfica $_{[1,n]}$ *contains* municípios $_{[1,n]}$ );

f) cada ponto de captura e lançamento de líquido precisa estar contido em um e somente um trecho de um recurso hídrico
(ponto de captura/lançamento $_{[0,n]}$ *inside* trecho $_{[1,1]}$);

g) uma rodovia contém um ou mais trechos de rodovia
(rodovia $_{[1,1]}$ *contains* trecho de rodovia $_{[1,n]}$);

h) um município contém uma ou muitas zonas de coleta de lixo
(município $_{[1,1]}$ *contains* zona de coleta de lixo $_{[1,n]}$);

i) uma cidade contém um ou muitos bairros
(cidade $_{[1,1]}$ *contains* bairro $_{[1,n]}$);

j) uma ilha precisa estar totalmente contida em um e somente um recurso hídrico, sem interseção de limites (ilha $_{[0,n1]}$ *inside* recurso hídrico $_{[1,1]}$);

k) todo edifício público precisa estar contido em um e somente um município (edifício público$_{[0,n]}$ *inside* município $_{[1,1]}$);

l) toda comporta precisa estar contida em um e somente um recurso hídrico (comporta $_{[0,n]}$ *inside* recurso hídrico $_{[1,1]}$);

### 1.2.2 Regras de Cruzamento (*crosses*)

As regras de cruzamento são aquelas que envolvem dois objetos do mundo real em que um atravessa o outro, de forma que, ambos compartilham alguma parte de seu interior, limite e exterior. Algumas regras desse tipo estão relacionadas abaixo:

a) uma rodovia não cruza uma edificação (rodovia $_{[0,0]}$ *crosses* edificação $_{[0,0]}$).

### 1.2.4 Regras de Sobreposição (*overlaps*)

As regras de sobreposição são fundadas no relacionamento topológico em que um objeto está em cima ou sobre outro objeto espacial. Exemplos desse tipo de regra estão enumerados abaixo:

a) um recurso hídrico e uma rodovia não podem estar sobrepostos (rodovia $_{[0,01]}$ *overlaps* recurso hídrico $_{[0,0]}$).

### 1.2.5 Regras de Adjacência (*touches*)

Regras de adjacência são aquelas em que dois objetos compartilham pontos ou lados sem ter nenhuma parte de seu interior em comum, ou seja, duas feições tocam-se quando a única coisa em comum entre eles for uma parte de seus limites. Algumas restrições de adjacência estão relacionadas a seguir:

a) toda rua toca um ou mais lotes (rua $_{[0,n1]}$ *touches* lote $_{[1,n]}$);

b) toda praia intercepta (toca) um ou mais recursos hídricos
(praia $_{[0,n]}$ *touches* recurso hídrico $_{[1,n]}$);

c) todo brejo ou pântano intercepta (toca) um ou muitos recurso hídricos (brejo/pântano $_{[0,n]}$ *touches* recurso hídrico $_{[1,n]}$);

d) todo porto está associado (toca) um e somente um recurso hídrico (porto $_{[0,n]}$ *touches* recurso hídrico $_{[1,1]}$);

### 1.2.6 Regras de Disjunção

Dois objetos espaciais estão disjuntos quando não existe nenhum tipo de ligação física entre eles, ou seja, não há nenhum ponto de interseção entre ambos. Exemplo: uma rodovia não pode interceptar (atravessar, conter, sobrepor, cobrir) uma edificação. Essa restrição pode ser representada por: (rodovia $_{[1,n]}$ *disjoint* edificação $_{[1,n]}$);

# Anexo 3 Algoritmos para o Controle de Restrições Topológicas no *Modelo Abstrato OpenGIS Estendido*

Os algoritmos de abertura da base de dados e de pedido das operações de inserção ou alteração pelo usuário, por serem um processo gerenciado pelo MGTra, não são especificados. Entretanto, a estrutura das classes e seus métodos são apresentados a seguir.

As funções *load_object()* e *get_number_of_objects()* são funções que o SIG já oferece e por essa razão não estão descritas.

```
#----------DEFINIÇÃO DA CLASSE FT_Feature E SEUS MÉTODOS----------#
Class FT_Feature{
        Integer featureType_id;          # atributo da classe #

#----------DEFINIÇÃO DE UM NOVO MÉTODO DA CLASSE FT_Feature----------#
PUBLIC METHOD verifyConstraint(OBJECT  currentFeature) { # define o método verifyConstraint   #
        Integer i,
        COLLECTION listOfConstraints;
        OBJECT objectRoleType, ;
        FR_RoleType.first;
        i := 1;
        WHILE (not (FR_RoleType.EOF)) {          # Pesquisa todos os objetos de FR_RoleType #
                objectRoleType = FR_RoleType.load_object ;   # carrega o objeto para a memória #
                IF (objectRoleType.featureType_id = currentFeature.featureType_id) THEN {
                        # verifica se o tipo da feição de FR_RoleType é igual ao tipo
                        da feição que está sendo inserida no BDG #
                        listOfConstraints [i]  := objectRoleType; # insere o objeto restrição na coleção#
                        i := i + 1;
                }
                FR_RoleType.next( );
        }
        RETURN (listOfConstraints);
}
#----------DEFINIÇÃO DE UM NOVO MÉTODO DA CLASSE FT_Feature----------#
PUBLIC METHOD beforeInsert (OBJECT currentFeature){
        Integer i, number, v_role;
        Boolean role;
        COLLECTION listOfConstraints;          # definição de variável do tipo coleção de objetos #
        OBJECT objectRoleType2   #  segundo tipo de papel que compõe a restrição, ou seja, a restrição inversa #;
        listOfConstraints := FT_Feature.verifyConstraint (currentFeature);   #  recupera todas as restrições do
                                                                                tipo da feição que está sendo inserida  #
        IF ( listOfConstraints is not empty) THEN {                          # se a lista não for vazia testa as restrições #
                i := 1; v_role := 0;
                number := get_number_of_objects (listOfConstraints); #função que conta o número de restrições da coleção#
                WHILE ( i <= number) do {              # para cada restrição faça: #
                        FR_RoleType.first; role := "FALSE";
                        WHILE (role = "FALSE") do { # procura o segundo tipo de papel do relacionamento #
                                objectRoleType2 = FR_RoleType.load_object();
                                IF (listOfConstraints.object[i].relationshipType_id
                                    =objectRoleType2.relationshipType_id) AND
                                    (listOfConstraints.object[i].ordered < > objectRoleType2.ordered) THEN
                                        # o teste de ordered é necessário para recuperar a segunda parte da restrição#
                                        role := "TRUE"; # sai do laço, pois  encontrou a segunda parte da restrição #
                                }
                                FR_RoleType.next;
                        }
                      # verifica, através do método executeConstraint da classe FR_RoleType, se a restrição foi satisfeita #
                        v_role=FR_RoleType.executeConstraint(currentFeature,listOfConstraints.object[i],objectRoleType2);
                         < o papel da feição é inserido em FR_Role com o  valor do atributo status= v_role>
                          # para cada restrição da coleção que foi testada o papel é inserido em FR_Role com o
                            status adequado #
                         <o papel da feição é atualizado no relacionamento em FR_Relationship >
                         i := i+1;
                 } # conclui o laço que testa cada objeto da coleção #
        }
} # conclui o método beforeInsert () #
 < O MGOpe insere a feicao em FT_Feature>; # no fim dos testes a feição sempre é inserida. O que varia é o status do papel #
```

```
#----------------------------DEFINIÇÃO DE UM NOVO MÉTODO DA CLASSE FT_Feature-----------------------------------------#

PUBLIC METHOD beforeUpdate (OBJECT currentFeature){
        Integer i, number, v_role, v_status;
        Boolean role;
        COLLECTION listOfConstraints, listOfFeatures;
        OBJECT objectRoleType2, currentFeatureRole  #  segundo tipo de papel que compõe a restrição
                                                       e papel da feição modificada #;
        listOfConstraints := FT_Feature.verifyConstraint (currentFeature);   #  recupera todas as restrições do
                                                                                tipo da feição que está sendo inserida  #
        IF ( listOfConstraints is not empty) then {                    # se a lista for vazia não há restrição #
                i := 1; v_role := 0;
                number := get_number_of_objects (listOfConstraints);  #função que conta o número de restrições da coleção#
                WHILE ( i <= number) do {                               # para cada restrição faça: #
                        FR_RoleType.first; role := "FALSE";
                        WHILE (role = "FALSE") do { # procura o segundo tipo de papel do relacionamento #
                                objectRoleType2 = FR_RoleType.load_object();
                                IF  (listOfConstraints.object[i].relationshipType_id
                                   =objectRoleType2.relationshipType_id)  AND
                                     (listOfConstraints.object[i].ordered < > objectRoleType2.ordered) THEN {
                                       # o teste de ordered é necessário para recuperar a  segunda parte da restrição #
                                          role := "TRUE"; # sai do laço, pois  encontrou a segunda parte da restrição #
                                 }
                                FR_RoleType.next;
                        }
                        v_status := 0; flag := 0;
                        FR_Role.first;
                        WHILE (flag=0) {   # recupera o status do papel da feição que está sendo modificada #
                                CurrentFeatureRole = FR_Role.load_object();
                                IF (currentFeatureRole.roleType_id=listOfConstraints.object[i].roleType_id) THEN {
                                        v_status := currentFeatureRole.status;
                                        flag := 1;
                                }
                                FR_Role.next;
                        }
                        If (v_status = 0) THEN {
                                 # recupera as feições cuja restrição sendo testada era satisfeita pela feição
                                 modificada antes de ocorrer   a atualização. O argumento oldFeature é a feição
                                 antes da modificação #
                                 listOfFeatures = FR_RoleType.getOldFeatures (oldFeature, listOfConstraints.object[i],
                                                                                 objectRoleType2);
                                # verifica se a feição modificada continua atendendo a restrição das feições
                                da lista. Caso não atenda, o valor do atributo status do papel de cada feição
                                cuja restrição foi violada é mudado para –1.  Não é preciso testar se a lista está
                                vazia porque o status da feição modificada igual a zero indica que  ao menos
                                uma feição atende a restrição, e esta será recuperada. #
                                 FR_RoleType.testOldFeatures (listOfFeatures,currentFeature,objectRoleType2,
                                                                listOfConstraints.object[i]);
                        }

                        # testa se alguma feição atende esta restrição da a feição modificada  #
                        v_role = executeConstraint (currentFeature, listOfConstraints.object[i], objectRoleType2);

                        < o papel da feição modificada em FR_Role é atualizado com o valor do atributo
                         status= v_role>
                         # para cada tipo de restrição de listOfConstraints, o status do  papel da feição modificada
                          recebe o valor de v_role,  indicando –1 se a restrição foi violada ou 0 se foi atendida #
                         i := i+1;
                } # conclui o laço que testa cada objeto da coleção #
    }
} # conclui o método beforeUpdate () #
 < O MGOpe altera a geometria da feição no banco de dados>; # no fim dos testes a feição sempre é atualizada. O que varia
                                                                 é o status do papel #
```

```
#----------------------DEFINIÇÃO DE UM NOVO MÉTODO DA CLASSE FT_Feature----------------------------------------#

PUBLIC METHOD beforeDelete (OBJECT currentFeature){
        Integer i, cont;
        Boolean role, v_intersection;
        COLLECTION listOfConstraints;
       OBJECT objectRole2, objectRoleType2 ;
        listOfConstraints := FT_Feature.verifyConstraint (currentFeature);   # recupera todas as restrições do
                                                                                tipo da feição que está sendo inserida  #
        IF ( listOfConstraints is not empty) then {                  # se a lista for vazia não há restrição #
                i := 1;
                number := get_number_of_objects (listOfConstraints);  #função que conta o número de restrições da coleção#
                WHILE ( i <= number) do {                              # para cada restrição faça: #
                        FR_RoleType.first;  role := "FALSE";
                        WHILE (role = "FALSE") DO { # procura o segundo tipo de papel do relacionamento #
                                objectRoleType2 = FR_RoleType.load_object();
                                IF  (listOfConstraints.object[i].relationshipType_id
                                   =objectRoleType2.relationshipType_id)  AND
                                     (listOfConstraints.object[i].ordered < > objectRoleType2.ordered) THEN {
                                       # o teste de ordered é necessário para recuperar a  segunda parte da restrição #
                                          role := "TRUE"; # sai do laço, pois  encontrou a segunda parte da restrição #
                                 }
                                FR_RoleType.next;
                        }
                        FR_Role.first; cont := 0;
                        WHILE  NOT (FR_Role.EOF) DO {   #  Lê todos os objetos de FR_Role #
                                objectRole2 = FR_Role.load_object(); # atribui o objeto de FR_Role a variável #
                                IF (objectRole2.roleType_id = objectRoleType2.roleType_id) THEN {
                                        TP_Object1 := getGeometry (objectRole2.feature_id);
                                        TP_Object1 := getGeometry (currentFeature);
                                        v_intersection := FR_RoleType.testIntersection (objectRoleType2,
                                                                                TP_Object1, TP_Object2);
                                        IF (v_intersection = "TRUE") THEN {
                                                IF (objectRole2.status = 0 )  THEN
                                                     IF (objectRoleType2.cardinality = [1,1) THEN {
                                                          < o valor do atributo status (objectRole2.status) da
                                                          feição objectRole2.feature_id é mudado para -1>
                                                          < a instância do relacionamento entre
                                                          objectRole2.feature_id e currentFeature é removida de
                                                          FR_Relationship e os atributos desse relacionamento
                                                          são removidos de FR_RelationshipAttribute>
                                                      }
                                                      ELSE  IF (objectRoleType2.cardinality = [0,1) THEN
                                                          < a instância do relacionamento entre
                                                          objectRole2.feature_id e currentFeature é removida de
                                                          FR_Relationship e os atributos desse relacionamento
                                                          são removidos de FR_RelationshipAttribute>
                                                ELSE
                                                          cont := cont + 1;
                                        }
                                }
                                FR_Role.next;
                        }
                        IF (cont1 = 1) THEN {
                                IF (objectRoleType2.cardinality = [0,0]) THEN
                                        < o valor do atributo status (objectRole2.status) da feição
                                        objectRole2.feature_id é mudado para 0>

                        }
                        ELSE {
                                IF (cont = 2) AND (objectRoleType2.cardinality = [1,1]) THEN {
                                        < o valor do atributo status (objectRole2.status) da feição
                                        objectRole2.feature_id é mudado  para 0>
                                        < a instância do relacionamento entre objectRole2.feature_id e
                                        currentFeature é removida  de FR_Relationship e os atributos desse
                                        relacionamento são removidos de FR_RelationshipAttribute >
                                }
                                IF (cont > 2) AND (objectRoleType2.cardinality = [1,n] ) THEN
                                        < a instância do relacionamento entre objectRole2.feature_id e
                                        currentFeature é removida de FR_Relationship e os atributos desse
                                        relacionamento são removidos de FR_RelationshipAttribute>
                        }
                        i := i + 1;
                } # end WHILE #
        } #  end  IF #
} # end beforeDelete #
```

```
} #  end  FT_Feature #


#----------------------------DEFINIÇÃO DA CLASSE FR_RoleType E SEUS MÉTODOS-----------------------------------------#

Class FR_RoleType {
        Integer featureType_id; relationshipType_id;
        String name, relationType, logicOperator;
        Set of Integer cardinality;
        Boolean ordered;                        # atributos da classe #

#-----------------------------------DEFINIÇÃO DE UM NOVO MÉTODO DA CLASSE FR_RoleType-----------------------------------#
        PUBLIC METHOD testIntersection (OBJECT roleType, GEOMETRY TP_object1, GEOMETRY TP_object2){
        boolean  v_constraint;
        integer v_numberRelation, j, cont;
        v_constraint := "FALSE";
        v_numberRelation = LENGTH (roleType.relationType);
        j := 1;
        WHILE ( j <= v_numberRelation) DO {
                IF (roleType.relationType[j] == "I" ) THEN
                        v_constraint := eRelate (TP_Object1, TP_Object2,'FFNT');
                ELSE {
                        IF (roleType.relationType[j] == "D" ) THEN
                                v_constraint := bRelate (TP_Object1, TP_Object2,'FFFF');
                        IF(roleType.relationType[j] == "T" ) THEN
                                v_constraint := bRelate (TP_Object1, TP_Object2,'TFFF');
                        IF (roleType.relationType[j] == "V" ) THEN
                                v_constraint := bRelate (TP_Object1, TP_Object2,'TFNT');
                        IF (roleType.relationType[j] == "B" ) THEN
                                v_constraint := eRelate (TP_Object1, TP_Object2,'TNFT');
                        IF (roleType.relationType[j] == "E" ) THEN
                                v_constraint := bRelate (TP_Object1, TP_Object2,'TFFT');
                        IF (roleType.relationType[j] == "C" ) THEN
                                v_constraint := bRelate (TP_Object1, TP_Object2,'FNFT');
                        IF (roleType.relationType[j] == "O" ) THEN
                                v_constraint := bRelate (TP_Object1, TP_Object2,'TTTT');
                        IF (roleType.relationType[j] == "S" ) THEN
                                v_constraint := bRelate (TP_Object1, TP_Object2 'TTFT');
                }
                IF (v_numberRelation ==1) THEN
                        RETURN (v_constraint);
                ELSE{
                        IF (v_constraint == "TRUE") THEN
                                IF (roleType.logicOperator == AND) THEN {
                                        v_cont := v_cont + 1;
                                        j := j+1;
                                }
                                ELSE
                                        RETURN (v_constraint);
                        ELSE
                                IF (roleType.logicOperator == AND) THEN
                                        RETURN (v_constraint);
                                ELSE {
                                        v_cont := v_cont + 1;
                                        j := j+1;
                                }
                }
        } # fim do while #
        IF (roleType.logicOperator == AND)  THEN
                RETURN (TRUE);
        ELSE
                RETURN (FALSE);
        } # fim do método #

#-----------------------------------DEFINIÇÃO DE UM NOVO MÉTODO DA CLASSE FR_RoleType-----------------------------------#

PUBLIC METHOD executeConstraint (OBJECT currentFeature, OBJECT listOfConstraints.object[i], OBJECT objectRoleType2)
{
GEOMETRY  TP_Object1, TP_Object2;
OBJECT objectRole2 , v_feature;         #  papel que atende o segundo tipo de papel que compõe a restrição #
OBJECT objectRelationship;
Integer flag, roleStatus, cont ;
String v_intersection;
Boolean v_relationship;
flag := 0; cont := 0; roleStatus := 0;
FR_Role.first;
```

```
WHILE not (FR_Role.EOF) {
        ObjectRole2 := FR_Role.load_object();
        IF (objectRole2.roleType_id=objectRoleType2.roleType_id)  then { # encontrou uma feição que pode ser testada #
                TP_Object1 := getGeometry(currentFeature);
                TP_Object2 := getGeometry(objectRole2.feature_id);
                 v_intersection = FR_RoleType.testIntersection(listOfConstraints.object[i],TP_Object1,TP_Object2);
                 IF (v_intersection = " TRUE"  ) THEN {
                        IF (listOfConstraints.object[i].cardinality = [0,0]) THEN {
                                # o relacionamento topológico existe mas a regra diz que ele não deve existir #
                                WARNING_MESSAGE "Restrição Espacial Violada"  ;
                                flag := 1;
                                IF (objectRole2.status = 0) THEN
                                        <o status do papel da feição objectRole2.feature_id deve ser mudado para –1>
                        }
                        IF (listOfConstraints.object[i].cardinality =[1,n]) THEN {
                                # o relacionamento topológico existe mas  a regra diz que vários podem
                                existir. Então é necessário testar todas as relações porque se o status da
                                feição que atendeu a restrição for igual a –1, este precisa mudar para 0 #
                                flag := 1;
                                IF (objectRole2.status = -1) THEN
                                        < o status do papel de objectRole2  é mudado para 0 >;
                                v_relationship := " FALSE" ; FR_Relationship.first;
                                WHILE (v_relationship = " FALSE" ) DO {
                                        objectRelationship := FR_Relationship.load_object();
                                        IF (listOfConstraints.object[i].relationshipType_id =
                                                objectRelationship.relationshipType_id AND
                                                objectRelationship.role_id1 = objectRole2.role_id  AND
                                                objectRelationship.role_id2 = currentFeature.role_id) THEN
                                                        v_relationship := " TRUE" ;
                                         FR_Relationship.next;
                                }
                                IF (v_relaltionship = " FALSE" ) THEN
                                        <uma instância de relacionamento deve ser criada em FR_Relationship com
                                          os papéis objectRole2.role_id e o papel de currentFeature. Caso
                                          currentFeature ainda não tenha papel, o mesmo fica em branco para ser
                                          preenchido pelo método beforeInsert() >

                        }
                        IF (listOfConstraints.object[i].cardinality = [1,1] OR
                             listOfConstraints.object[i].cardinality =[0,1])) THEN {
                                # o relacionamento topológico  existe mas a regra diz que pode haver no máximo um.
                                Por isso é necessário contar se há mais feições que atendem a restrição. Se houver mais
                                de uma, a restrição foi violada #
                                flag := 1; cont := cont + 1; v_feature := objectRole2;
                        }
                }
         }
         FR_Role.next;
}
IF ( (flag = 0 AND (listOfConstraints.object[i].cardinality = [1,1] OR listOfConstraints.object[i].cardinality = [1,n] ))
        # flag igual a zero indica que nenhuma feição atendeu a restrição #
    OR  (cont > 1)      # mais de uma feição atendeu a restrição. Então ela foi violada porque no máxima uma poderia
                        tê-la satisfeito #
                         WARNING_MESSAGE "Restrição Espacial Violada" ;
                          roleStatus := -1; RETURN (roleStatus);
}
ELSE {
      IF (flag = 1 AND listOfConstraints.object[i].cardinality = [0,0]) THEN
              RETURN ( roleStatus = -1 );
      IF (cont = 1 AND objectRole.status = -1) THEN
         < O status do papel da feição  (v_feature) em FR_Role é mudado para 0 onde: FR_Role.role_id=v_feature.role_id>;
                        IF (objectRole2.status = -1) THEN
                                < o status do papel de objectRole2  também é mudado para 0 >;
                                v_relationship := " FALSE" ; FR_Relationship.first;
                                WHILE (v_relationship = " FALSE" ) DO {
                                        objectRelationship := FR_Relationship.load_object();
                                        IF (listOfConstraints.object[i].relationshipType_id =
                                             objectRelationship.relationshipType_id AND
                                           objectRelationship.role_id1 = objectRole2.role_id  AND
                                             objectRelationship.role_id2 = currentFeature.role_id) THEN
                                                v_relationship := " TRUE" ;
                                        FR_Relationship.next;
                                }
                                IF (v_relaltionship = " FALSE" ) THEN
                                        <uma instância de relacionamento deve ser criada em FR_Relationship com
```

os papéis objectRole2.role_id e o papel de *currentFeature*. Caso *currentFeature* ainda não tenha papel, o mesmo fica em branco para ser preenchido pelo método *beforeInsert()* >

```
}
# se ao final de todos os testes nenhum  item foi satisfeito, o valor de roleStatus continua com o valor inicial (zero). Então a restrição foi satisfeita #
RETURN (roleStatus=0);
}


#-----------------------------------DEFINIÇÃO DE UM NOVO MÉTODO DA CLASSE FR_RoleType-----------------------------------#

PUBLIC METHOD getOldFeatures (OBJECT oldFeature, OBJECT listOfConstraints.object[i], OBJECT objectRoleType2) {
GEOMETRY  TP_Object1, TP_Object2;
OBJECT objectRole;          #  papel que atende o segundo tipo de papel que compõe a restrição #
Integer  j ;
String v_intersection;
FR_Role.first;
 j := 1;
WHILE not (FR_Role.EOF) {
            objectRole := FR_Role.load_object();
            IF (objectRole.roleType_id=objectRoleType2.roleType_id AND objectRole.status=0)  THEN {
                                                            # encontrou feição que pode ser testada #
                        TP_Object1 := getGeometry(oldFeature);
                        TP_Object2 := getGeometry(objectRole.feature_id);
                         v_intersection = testIntersection(listOfConstraints.object[i].relationType, TP_Object1,TP_Object2);
                         IF (v_intersection = "FALSE") THEN {
                                    IF (listOfConstraints.object[i].cardinality = [0,0]) THEN{
                                                # o relacionamento topológico não existe e a cardinalidade indica que o mesmo
                                                 não deve existir. Então a restrição foi satisfeita e a feição e
                                                seu papel são guardados na lista#
                                                ListOfFeatures.object[j] := objectRole;
                                                j := j+1;
                                     }
                          }
                          ELSE {
                                    IF (listOfConstraints.object[i].cardinality =[1,n]) THEN {
                                                # a relação topológica existe e a regra diz que podem ter várias ligações #
                                                ListOfFeatures.object[j] := objectRole;
                                                j := j+1;
                                    }
                                    IF (listOfConstraints.object[i].cardinality = [1,1] OR
                                                listOfConstraints.object[i].cardinality =[0,1]) THEN {
                                                # a relação topológica existe mas a regra diz que pode haver no máximo uma. Não é
                                                necessário contar se há mais feições que atendem a restrição, pois como o status do
                                                papel dessa feição no relacionamento é igual a zero, ela é a única feição que atende a
                                                restrição #
                                                ListOfFeatures.object[j] := objectRole;
                                                j := j+1;
                                    }
                        }
            }
            FR_Role.next;
}
RETURN (listOfFeatures);
}

#-----------------------------------DEFINIÇÃO DE UM NOVO MÉTODO DA CLASSE FR_RoleType-----------------------------------#
PUBLIC METHOD testOldFeatures (COLLECTION listOfFeatures, OBJECT currentFeature, OBJECT objectRoleType2,
                                    OBJECT listOfConstraints.object[i]) {
GEOMETRY  TP_Object1, TP_Object2;
OBJECT objectRole, objectRole2 ;          #  papel que atende o segundo tipo de papel que compõe a restrição #
Integer number, i , flag;
String v_intersection;
j := 1; number := 0;
number := get_number_of_objects(listOfFeatures); # conta o número de objetos da lista #
WHILE  (j <= number) {
            objectRole := listOfFeatures.object[j];
            TP_Object1 := getGeometry(listOfFeatures.object[j].feature_id);
            TP_Object2 := getGeometry(currentFeature);
             v_intersection = FR_RoleType.testIntersection(objectRoleType2.relationType, TP_Object1,TP_Object2);
             IF (v_intersection = "TRUE") THEN {
                        IF (objectRoleType2.cardinality = [0,0]) THEN{
                                    # a relação topológica existe mas a cardinalidade indica que a mesma não deve existir.
```

```
                    Então a restrição foi violada #
                    < o  valor do atributo status do papel da feição da lista é mudado para –1, pois a feição
                     modificada não atende mais a restrição >
                    < a instância do relacionamento entre objectRole2.feature_id e currentFeature é removida de
                    FR_Relationship e os atributos desse relacionamento são removidos de
                    FR_RelationshipAttribute>
                }
        ELSE {
                IF (objectRoleType2.cardinality = [1,1]) THEN {
                    # a relação topológica não existe, mas a cardinalidade indica que ao menos uma feição deveria
                    atender a restrição. Como a feição que atendia a restrição era a feição modificada, porém antes
                    da modificação, então a restrição foi violada, sendo desnecessário contar se mais uma feição
                    atende a restrição. #
                    < o   valor do atributo status do papel da feição da lista é mudado para –1, pois a feição
                     modificada não atende mais a restrição >
                    < a instância do relacionamento entre objectRole2.feature_id e currentFeature é removida de
                    FR_Relationship e os atributos desse relacionamento são removidos de
                    FR_RelationshipAttribute>

                 }
                IF (objectRoleType2.cardinality =[1,n]) THEN {
                    # o relacionamento topológico não existe mas a cardinalidade indica que precisam haver
                    uma ou várias ligações. Como a feição modificada não atende mais esta restrição, é necessário
                    verificar se ela era a única ou se outras feições continuam atendendo esta restrição #
                    FR_Role.first; flag := 0;
                    WHILE (flag = 0) {
                        ObjectRole2 := FR_Role.load_object();
                        IF (objectRole2.roleType_id = listOfConstraints.object[i].roleType_id    AND
                            objectRole2.role_id <> objectRole.role_id) THEN {
                                TP_Object2 = getGeometry(objectRole2.feature_id);
                                 v_intersection=FR_RoleType.testIntersection(objectRoleType2.relationType,
                                                                              TP_Object1,TP_Object2);
                                 IF (v_intersection = "TRUE") THEN {
                                     Flag := 1; #a feição modificada não era a única que atendia  a restrição#
                                   }
                        }
                        FR_Role.next;
                    }
                    IF (flag = 0) THEN { # a feição modificada era a única que atendia  a restrição da feição da lista#
                        < o valor do atributo status do papel da feição da lista é mudado para –1, pois a feição
                        modificada não atende mais a restrição >
                        < a instância do relacionamento entre objectRole2.feature_id e currentFeature é
                        removida de FR_Relationship e os atributos desse relacionamento são removidos de
                        FR_RelationshipAttribute >

                    }
                }
        }
        j := j+1;
}
}
}
#---------- TRATAMENTO DAS FEIÇÕES QUE VIOLARAM ALGUMA RESTRIÇÃO----------------#
IF (close DataBase) THEN
    < usuário encerra o cadastro de feições e inicia automaticamente o tratamento das feições pendentes onde o status do papel é
igual a -1 >
boolean exceção;
FR_Role.first where status = -1;
 WHILE not(FR_Role.EOF) {
        Begin sub-Transaction
        objectRole =  FR_Role.load_object;
        IF (objectRole is exception) then
                < o valor do atributo status do papel é mudado para 1>;
        ELSE {
                <papel permanece até que a feição receba o tratamento adequado > ou
                <a feição, seu papel e todos os dados referentes a ela são removidos do BD pelo MGOpe >
        }
        end sub-Transaction;
        FR_Role.next;
} end SAGA; close DataBase
```

# Anexo 4 Implementação de Restrições Espaciais no GOTHIC

```
function integer Restricao_Edif_Edif_Funcional (VAC vac_id,
                    OBJECT object_id,
                    string return_value)
begin
  string retorno;
  retorno := "OK";

st :=  Contem(vac_id,object_id,"edif_saude","edif_publica",retorno);
st :=  Atravessa(vac_id,object_id,"edif_funcional","continental",retorno);

  return_value := "OK";
  return GOTH__NORMAL;
end;

function integer Restricao_Hidro_Linha_Drenagem (VAC vac_id,
                    OBJECT object_id,
                    string return_value)
begin

  string retorno;
  retorno := "OK";
  #------------ RIO / EDIFICACAO ----------------#
  st :=  Atravessa(vac_id,object_id,"linha_drenagem","edif_funcional",retorno);
  st :=  Igual(vac_id,object_id,"linha_drenagem","rodovia",retorno);
  return_value := "OK";
  return GOTH__NORMAL;

end;

function integer  Funcao_Validacao_Area (VAC vac_id,
                              COLLECTION list_obs,
                              OBJECT object_id,
                              GEOMETRY geom1_id,
                              string nome_classe1,
                              string nome_classe2,
                              string tipo_relac,
                              integer v_contem)
begin
               integer st, number, intersecao, inclusao,class_id, obj_class_id,count, retorno,status;
               OBJECT obj_id, novo_objeto, objeto_retornado;
               boolean is_null,discard;
               string class_name ;
               COLLECTION objetos_violaram;
               GEOMETRY geom2_id;
               v_contem := 10;

               #--Define a classe que contem os objetos que violaram alguma restricao--#
               st := meta_check_class_exists (vac_id,"RestricoesVioladas",is_null);
               if (st != GOTH__NORMAL) then return error_report_status(st);

              if (not (is_null)) then
              begin
                  st := meta_define_class(vac_id,"RestricoesVioladas","");
                  if (st != GOTH__NORMAL) then return error_report_status(st);

                  st := meta_inherit (vac_id, "RestricoesVioladas","simple");
                  if (st != GOTH__NORMAL) then return error_report_status(st);
             end;

              #--Define atributos--#
              st := meta_check_class_has_value (vac_id,"RestricoesVioladas","objeto",discard,discard,is_null);
              if (st != GOTH__NORMAL) then return error_report_status(st);

              if (not (is_null)) then
              begin
                   st := meta_define_value (vac_id,"RestricoesVioladas","objeto","",
                         META_SCOPE_LOCAL,DT_INTEGER,DDT_INVALID,MVT_SINGLE,MST_PROPERTY);
```

```
    if (st != GOTH__NORMAL) then return error_report_status(st);
    st := meta_define_value (vac_id,"RestricoesVioladas","restricao","",
          META_SCOPE_LOCAL,DT_INTEGER,DDT_INVALID,MVT_SINGLE,MST_PROPERTY);
    if (st != GOTH__NORMAL) then return error_report_status(st);
end;

  # recupera o numero de objetos contidos na colecao #
  number := 0;
  st := coln_number_of_elements(list_obs,number);
  if ( st != GOTH__NORMAL) then return error_report_status(st);
  if (number < 1) then return GOTH__NORMAL;
  count := 1;
  while (count <= number) do
  begin
           # recupera o identificador dos objetos da colecao #
          st := coln_get_value_of_element(list_obs,CPT_POSITION,count,obj_id);
          if ( st != GOTH__NORMAL) then return error_report_status(st);

          # recupera o identificador da classe do objeto #
          class_id := goth_get_object_class_id(obj_id);
          if ( st != GOTH__NORMAL) then return error_report_status(st);
          # recupera o nome da classe pelo identificador da classe #
          st := meta_get_class_name_from_id (vac_id,class_id,class_name);
          if ( st != GOTH__NORMAL) then return error_report_status(st);

         # recupera a geometria do objeto a ser testado #
         st := obj_get_value_from_name(vac_id,obj_id,"geometry",geom2_id,is_null);
         if ( st != GOTH__NORMAL) then return error_report_status(st);
         if ( not (is_null)) then
         begin
              st := geom_area_test_geom_inclusion(geom1_id,geom2_id,intersecao,inclusao);
             if ( st != GOTH__NORMAL) then return error_report_status(st);
                        if ((inclusao == _GINCT_CONTAINS ) and (tipo_relac == "CONTER")) then
             begin
                  #---MOSTRA MENSAGEM DE RESTRICAO VIOLADA---#
                 st := goth_printf(frame_stdout,"\n ATENCAO: um(a) %s nao pode CONTER um(a) %s\n
                                  ",nome_classe1,class_name);
                 if (st != GOTH__NORMAL) then return error_report_status(st);
                 st := obj_construct_object_from_values(vac_id,"RestricoesVioladas",novo_objeto,status);
                 if (st != GOTH__NORMAL) then return error_report_status(st);
                 st := obj_set_value_from_name(vac_id,novo_objeto,"objeto",object_id,is_null);
                 if (st != GOTH__NORMAL) then return error_report_status(st);
                 st := obj_set_value_from_name(vac_id,novo_objeto,"restricao",inclusao,is_null);
                  if (st != GOTH__NORMAL) then return error_report_status(st);
                        end;
              if ((inclusao == _GINCT_DISJOINT) and (tipo_relac == "DISJUNTO")) then
              begin
                  #---MOSTRA MENSAGEM DE RESTRICAO VIOLADA---#
                  st := goth_printf(frame_stdout,"\n ATENCAO: um(a) %s nao pode ser DISJUNTA de um(a) %s
                                  ",nome_classe1,class_name);
                  if (st != GOTH__NORMAL) then return error_report_status(st);
                  st := obj_construct_object_from_values(vac_id,"RestricoesVioladas",novo_objeto,status);
                  if (st != GOTH__NORMAL) then return error_report_status(st);
                  st := obj_set_value_from_name(vac_id,novo_objeto,"objeto",object_id,is_null);
                  if (st != GOTH__NORMAL) then return error_report_status(st);
                 st := obj_set_value_from_name(vac_id,novo_objeto,"restricao",inclusao,is_null);
                  if (st != GOTH__NORMAL) then return error_report_status(st);
             end;
              if ((inclusao == _GINCT_SAME) and (tipo_relac == "IGUAL")) then
              begin
                   #---MOSTRA MENSAGEM DE RESTRICAO VIOLADA---#
                  st := goth_printf(frame_stdout,"\n ATENCAO: um(a) %s nao pode ser IGUAL a um(a) %s
                                  ",nome_classe1,class_name);
                  if (st != GOTH__NORMAL) then return error_report_status(st);
                  st := obj_construct_object_from_values(vac_id,"RestricoesVioladas",novo_objeto,status);
                  if (st != GOTH__NORMAL) then return error_report_status(st);
                  st := obj_set_value_from_name(vac_id,novo_objeto,"objeto",object_id,is_null);
                  if (st != GOTH__NORMAL) then return error_report_status(st);
                  st := obj_set_value_from_name(vac_id,novo_objeto,"restricao",inclusao,is_null);
                   if (st != GOTH__NORMAL) then return error_report_status(st);
              end;

              if ((inclusao == _GINCT_CONTAINED) and (tipo_relac == "CONTIDO"))then
              begin
                   #---MOSTRA MENSAGEM DE RESTRICAO VIOLADA---#
                   st := goth_printf(frame_stdout,"\n ATENCAO: um(a) %s nao pode estar CONTIDA em um(a) %s
```

```
                                        ",nome_classe1,class_name);
                                if (st != GOTH__NORMAL) then return error_report_status(st);
                                st := obj_construct_object_from_values(vac_id,"RestricoesVioladas",novo_objeto,status);
                                if (st != GOTH__NORMAL) then return error_report_status(st);
                                st := obj_set_value_from_name(vac_id,novo_objeto,"objeto",object_id,is_null);
                                if (st != GOTH__NORMAL) then return error_report_status(st);
                                st := obj_set_value_from_name(vac_id,novo_objeto,"restricao",inclusao,is_null);
                                if (st != GOTH__NORMAL) then return error_report_status(st);
                            end;
                            if ((inclusao == _GINCT_ON) and (tipo_relac == "COBRIR"))then
                            begin
                                #---MOSTRA MENSAGEM DE RESTRICAO VIOLADA---#
                                st := goth_printf(frame_stdout,"\n ATENCAO: um(a) %s nao pode COBRIR
                                                um(a) %s ",nome_classe1,class_name);
                                if (st != GOTH__NORMAL) then return error_report_status(st);
                                st := obj_construct_object_from_values(vac_id,"RestricoesVioladas",novo_objeto,status);
                                if (st != GOTH__NORMAL) then return error_report_status(st);
                                st := obj_set_value_from_name(vac_id,novo_objeto,"objeto",object_id,is_null);
                                if (st != GOTH__NORMAL) then return error_report_status(st);
                                st := obj_set_value_from_name(vac_id,novo_objeto,"restricao",inclusao,is_null);
                                if (st != GOTH__NORMAL) then return error_report_status(st);
                            end;
                        end;
                        count := count + 1;
                end;
        v_contem := 10;
        return GOTH__NORMAL;
end;


function integer  Funcao_Validacao (VAC vac_id,
                                    COLLECTION list_obs1,
                                    string nome_classe1,
                                    string nome_classe2,
                                    OBJECT object_id,
                                    string tipo_relac,
                                    string str_v_contem)
begin
        string str_classes_testadas, classe_erro;
        OBJECT obj_id,novo_objeto;
        integer class_id, obj_class_id,count,st, number, num_classes1, classes_testadas1[0], cont2, status;
        boolean discard, is_null;

        #--Define a classe que contem os objetos que violaram alguma restricao--#
        st := meta_check_class_exists (vac_id,"RestricoesVioladas",is_null);
        if (st != GOTH__NORMAL) then return error_report_status(st);

        if (not (is_null)) then
        begin
                st := meta_define_class(vac_id,"RestricoesVioladas","");
                        if (st != GOTH__NORMAL) then return error_report_status(st);

                        st := meta_inherit (vac_id, "RestricoesVioladas","simple");
                        if (st != GOTH__NORMAL) then return error_report_status(st);
        end;

        #--Define atributos--#
        st := meta_check_class_has_value (vac_id,"RestricoesVioladas","objeto",discard,discard,is_null);
        if (st != GOTH__NORMAL) then return error_report_status(st);
        if (not (is_null)) then
        begin
                st := meta_define_value (vac_id,"RestricoesVioladas","objeto","",
                        META_SCOPE_LOCAL,DT_INTEGER,DDT_INVALID,MVT_SINGLE,MST_PROPERTY);
                if (st != GOTH__NORMAL) then return error_report_status(st);
                st := meta_define_value (vac_id,"RestricoesVioladas","restricao","",
                                META_SCOPE_LOCAL,DT_INTEGER,DDT_INVALID,MVT_SINGLE,MST_PROPERTY);
                if (st != GOTH__NORMAL) then return error_report_status(st);
         end;

        # recupera o numero de objetos contidos na colecao #
        st := coln_number_of_elements(list_obs1,number);
        if ( st != GOTH__NORMAL) then return error_report_status(st);
        count := 1;
        while (count <= number) do
        begin
                # recupera o identificador dos objetos da colecao #
```

```
st := coln_get_value_of_element(list_obs1,CPT_POSITION,count,obj_id);
if ( st != GOTH__NORMAL) then return error_report_status(st);

# recupera o identificador da classe que contem o objeto #
 obj_class_id :=  goth_get_object_class_id(obj_id);
  # testa ligacao com a outra classe com a qual esta relacionada #
 st := meta_get_class_id_from_name(vac_id, nome_classe2,class_id);
 if ( st != GOTH__NORMAL) then return error_report_status(st);

 if (class_id == obj_class_id) then
 begin
            classe_erro := nome_classe2;
            st := goth_printf(frame_stdout,"\n ATENCAO: um(a) %s nao pode %s um(a) %s\n"
                    ,nome_classe1,tipo_relac,classe_erro);
end;
 #--- RECUPERA TODAS AS SUBCLASSES DA SEGUNDA CLASSE QUE COMPOE A RESTRICAO---#
 st := meta_get_sub_classes (vac_id,nome_classe2,num_classes1,classes_testadas1);
 if (st != GOTH__NORMAL) then return sch_print_message_stack();
 cont2 := 1;
while (cont2 <= num_classes1) do
begin
        if (obj_class_id == classes_testadas1[cont2]) then
        begin
                        st := meta_get_class_name_from_id (vac_id,classes_testadas1[cont2],classe_erro);
                         if ( st != GOTH__NORMAL) then return error_report_status(st);
                           if (tipo_relac == "CONTER") then
                         begin
                                  st := goth_printf(frame_stdout,"\n ATENCAO: um(a) %s nao pode CONTER um(a) %s\n"
                                          ,nome_classe1,classe_erro);
                                 if (st != GOTH__NORMAL) then return error_report_status(st);
                                 st := obj_construct_object_from_values(vac_id,"RestricoesVioladas",novo_objeto,status);
                                 if (st != GOTH__NORMAL) then return error_report_status(st);
                                 st := obj_set_value_from_name(vac_id,novo_objeto,"objeto",object_id,is_null);
                                 if (st != GOTH__NORMAL) then return error_report_status(st);
                                 st := obj_set_value_from_name(vac_id,novo_objeto,"restricao",_GINCT_CONTAINS,is_null);
                                 if (st != GOTH__NORMAL) then return error_report_status(st);
                         end;
                         if  (tipo_relac == "DISJUNTO") then
                         begin
                                  #---MOSTRA MENSAGEM DE RESTRICAO VIOLADA---#
                                st := goth_printf(frame_stdout,"\n ATENCAO: um(a) %s nao pode ser DISJUNTA de um(a) %s"
                                          ,nome_classe1,classe_erro);
                                 if (st != GOTH__NORMAL) then return error_report_status(st);
                                 st := obj_construct_object_from_values(vac_id,"RestricoesVioladas",novo_objeto,status);
                                 if (st != GOTH__NORMAL) then return error_report_status(st);
                                 st := obj_set_value_from_name(vac_id,novo_objeto,"objeto",object_id,is_null);
                                  if (st != GOTH__NORMAL) then return error_report_status(st);
                                 st := obj_set_value_from_name(vac_id,novo_objeto,"restricao",_GINCT_DISJOINT,is_null);
                                 if (st != GOTH__NORMAL) then return error_report_status(st);
                        end;
                        if (tipo_relac == "IGUAL") then
                        begin
                                  #---MOSTRA MENSAGEM DE RESTRICAO VIOLADA---#
                                  st := goth_printf(frame_stdout,"\n ATENCAO: um(a) %s nao pode ser IGUAL a um(a) %s",
                                          nome_classe1,classe_erro);
                                 if (st != GOTH__NORMAL) then return error_report_status(st);
                                 st := obj_construct_object_from_values(vac_id,"RestricoesVioladas",novo_objeto,status);
                                 if (st != GOTH__NORMAL) then return error_report_status(st);
                                 st := obj_set_value_from_name(vac_id,novo_objeto,"objeto",object_id,is_null);
                                 if (st != GOTH__NORMAL) then return error_report_status(st);
                                 st := obj_set_value_from_name(vac_id,novo_objeto,"restricao",_GINCT_SAME,is_null);
                                 if (st != GOTH__NORMAL) then return error_report_status(st);
                        end;
                        if  (tipo_relac == "CONTIDO") then
                        begin
                                  #---MOSTRA MENSAGEM DE RESTRICAO VIOLADA---#
                               st := goth_printf(frame_stdout,"\n ATENCAO: um(a) %s nao pode estar CONTIDA em um(a) %s",
                                          nome_classe1,classe_erro);
                                 if (st != GOTH__NORMAL) then return error_report_status(st);
                                 st := obj_construct_object_from_values(vac_id,"RestricoesVioladas",novo_objeto,status);
                                 if (st != GOTH__NORMAL) then return error_report_status(st);
                                 st := obj_set_value_from_name(vac_id,novo_objeto,"objeto",object_id,is_null);
                                 if (st != GOTH__NORMAL) then return error_report_status(st);
                                st := obj_set_value_from_name(vac_id,novo_objeto,"restricao",_GINCT_CONTAINED,is_null);
                                 if (st != GOTH__NORMAL) then return error_report_status(st);
                        end;
```

```
                        if (tipo_relac == "COBRIR")then
                        begin
                                #---MOSTRA MENSAGEM DE RESTRICAO VIOLADA---#
                                st := goth_printf(frame_stdout,"\n ATENCAO: um(a) %s nao pode COBRIR um(a) %s",
                                        nome_classe1,classe_erro);
                                if (st != GOTH__NORMAL) then return error_report_status(st);
                                st := obj_construct_object_from_values(vac_id,"RestricoesVioladas",novo_objeto,status);
                                if (st != GOTH__NORMAL) then return error_report_status(st);
                                st := obj_set_value_from_name(vac_id,novo_objeto,"objeto",object_id,is_null);
                                if (st != GOTH__NORMAL) then return error_report_status(st);
                                st := obj_set_value_from_name(vac_id,novo_objeto,"restricao",_GINCT_ON,is_null);
                                if (st != GOTH__NORMAL) then return error_report_status(st);
                        end;
                        if (tipo_relac == "ATRAVESSAR")then
                        begin
                                #---MOSTRA MENSAGEM DE RESTRICAO VIOLADA---#
                                st := goth_printf(frame_stdout,"\n ATENCAO: um(a) %s nao pode ATRAVESSAR um(a) %s",
                                        nome_classe1,classe_erro);
                                if (st != GOTH__NORMAL) then return error_report_status(st);
                                st := obj_construct_object_from_values(vac_id,"RestricoesVioladas",novo_objeto,status);
                                if (st != GOTH__NORMAL) then return error_report_status(st);
                                st := obj_set_value_from_name(vac_id,novo_objeto,"objeto",object_id,is_null);
                                if (st != GOTH__NORMAL) then return error_report_status(st);
                                st := obj_set_value_from_name(vac_id,novo_objeto,"restricao",_GINCT_INTERSECT,is_null);
                                if (st != GOTH__NORMAL) then return error_report_status(st);
                        end;
                    end;
                    cont2 := cont2 + 1;
                end;
                count := count + 1;
        end;
        return GOTH__NORMAL;
end;



function integer  Contem (VAC vac_id,
                    OBJECT object_id,
                    string nome_classe1,
                    string nome_classe2,
                    string retorno)
begin
    boolean is_null, existe_classe;
    integer st, num_classes, classes_testadas[0], id_classes [200], tipo_geometria,
    classe_id, cont, v_contem;
    COLLECTION list_obs, list_obs1;
    GEOMETRY geom1_id;
    string str_v_contem,tipo_relac;
    v_contem := 10;
    num_classes := 0;

    #--- VERIFICA SE A SEGUNDA CLASSE QUE COMPOE A RESTRICAO EXISTE-----#
    st := meta_check_class_exists (vac_id,nome_classe2,existe_classe);
    if (st != GOTH__NORMAL) then return sch_print_message_stack();
    if (NOT existe_classe) then return GOTH__NORMAL;

    #--- RECUPERA A GEOMETRIA DO PRIMEIRO OBJETO---#
    st := obj_get_value_from_name(vac_id,object_id,"geometry",geom1_id,is_null);
    if ( st != GOTH__NORMAL) then return error_report_status(st);
    #--- VERIFICA O TIPO DO PRIMEIRO OBJETO, SE EH AREA OU NAO---#
    st := geom_get_type (geom1_id,tipo_geometria);
    if ( st != GOTH__NORMAL) then return error_report_status(st);
    #--- TESTA SE A GEOMETRIA EH UMA AREA SIMPLES OU COMPLEXA #
    tipo_relac := "CONTER";
    if (tipo_geometria == _GT_SIMP_AREA ) then
    begin
                #--- SE A SEGUNDA CLASSE EXISTIR RECUPERA TODAS AS SUAS SUBCLASSES---#
                st := meta_get_sub_classes (vac_id,nome_classe2,num_classes,classes_testadas);
                if (st != GOTH__NORMAL) then return sch_print_message_stack();

                cont := 1;

                while (cont <= num_classes) do
                begin
                    id_classes[cont] := classes_testadas[cont];
```

```
                cont := cont + 1;
            end;

            #--- SE NAO TIVER SUBCLASSES TESTA COM ELA MESMA---#
            st := meta_get_class_id_from_name(vac_id,nome_classe2,classe_id);
            if ( st != GOTH__NORMAL) then return error_report_status(st);

            if (num_classes <= 0) then
            begin
                num_classes := 1;
                id_classes[1] := classe_id;
             end;
             else
                num_classes := num_classes + 1;
                id_classes[num_classes] :=  classe_id;
                #-- RECUPERA OS OBJETOS QUE INTERCEPTAM A AREA DO OBJETO ATIVO---#
                st := quadx_classed_object_in_area(vac_id,vac_id,0,num_classes,id_classes,
                                                   geom1_id,0.0,0.0,list_obs);
                if ( st != GOTH__NORMAL) then return error_report_status(st);
                 #--- EXECUTA A FUNCAO QUE RETORNA O TIPO DE INTERSECAO ENTRE O OBJETO
                   ATIVO E OS DEMAIS QUE O INTERSEPTAM---#
                 st :=  Funcao_Validacao_Area (vac_id, list_obs, object_id,geom1_id,nome_classe1,
                                                                 nome_classe2,tipo_relac, v_contem);
                if ( st != GOTH__NORMAL) then return error_report_status(st);
                        #--- TESTA SE O TIPO DE INTERSECAO EH DE CONTINENCIA---#
            end;

         #--- TESTA SE O OBJETO ATIVO TEM A FORMA DE LINHA SIMPLES OU COMPLEXA.---#
         #--- O PONTO NAO EH TESTADO PORQUE NAO PODE CONTER OUTRO OBJETO ---#
        if ((tipo_geometria == _GT_SIMP_LINE) or (tipo_geometria == _GT_COMP_LINE)) then
        begin
            #--- VERIFICA QUE SE O OBJETO ATIVO CONTEM ALGUM OUTRO OBJETO---#
            st := obj_get_value_from_name(vac_id,object_id,"contains",list_obs1,is_null);
            if ( st != GOTH__NORMAL) then return error_report_status(st);
            if ( not (is_null)) then
            begin
                #--- EXECUTA A FUNCAO QUE CONFIRMA SE O RELACIONAMENTO COM OUTRO
                OBJETO EH DE CONTINENCIA---#
                str_v_contem := "OK";
                st :=  Funcao_Validacao (vac_id, list_obs1, nome_classe1,nome_classe2,
                                                    object_id,tipo_relac,str_v_contem);
                if ( st != GOTH__NORMAL) then return error_report_status(st);
              end;
         end;

  retorno := "OK";
  return GOTH__NORMAL;

end;

function integer  Contido (VAC vac_id,
                OBJECT object_id,
                string nome_classe1,
                string nome_classe2,
                string retorno)
begin
    boolean is_null, existe_classe;
    integer st, num_classes, classes_testadas[0], id_classes [200], tipo_geometria,
    classe_id, cont, v_contem;
    COLLECTION list_obs, list_obs1;
    GEOMETRY geom1_id;
    string str_v_contem,tipo_relac;
    v_contem := 10;
    num_classes := 0;

    #--- VERIFICA SE A SEGUNDA CLASSE QUE COMPOE A RESTRICAO EXISTE-----#
    st := meta_check_class_exists (vac_id,nome_classe2,existe_classe);
    if (st != GOTH__NORMAL) then return sch_print_message_stack();
    if (NOT existe_classe) then return GOTH__NORMAL;

    #--- RECUPERA A GEOMETRIA DO PRIMEIRO OBJETO---#
    st := obj_get_value_from_name(vac_id,object_id,"geometry",geom1_id,is_null);
    if ( st != GOTH__NORMAL) then return error_report_status(st);
    #--- VERIFICA O TIPO DO PRIMEIRO OBJETO, SE EH AREA OU NAO---#
    st := geom_get_type (geom1_id,tipo_geometria);
    if ( st != GOTH__NORMAL) then return error_report_status(st);
```

```
    #--- TESTA SE A GEOMETRIA EH UMA AREA SIMPLES OU COMPLEXA #
    tipo_relac := "CONTIDO";
    if (tipo_geometria == _GT_SIMP_AREA ) then
    begin
                        #--- SE A SEGUNDA CLASSE EXISTIR RECUPERA TODAS AS SUAS SUBCLASSES---#
                st := meta_get_sub_classes (vac_id,nome_classe2,num_classes,classes_testadas);
                if (st != GOTH__NORMAL) then return sch_print_message_stack();

                cont := 1;

                while (cont <= num_classes) do
                begin
                    id_classes[cont] := classes_testadas[cont];
                    cont := cont + 1;
                end;

                #--- SE NAO TIVER SUBCLASSES TESTA COM ELA MESMA---#
                st := meta_get_class_id_from_name(vac_id,nome_classe2,classe_id);
                if ( st != GOTH__NORMAL) then return error_report_status(st);

                if (num_classes <= 0) then
                begin
                   num_classes := 1;
                   id_classes[1] := classe_id;
                end;
                else
                begin
                            num_classes := num_classes + 1;
                            id_classes[num_classes] :=  classe_id;

                            #-- RECUPERA OS OBJETOS QUE INTERCEPTAM A AREA DO OBJETO ATIVO---#
                           st := quadx_classed_object_in_area(vac_id,vac_id,0,num_classes,id_classes,
                              geom1_id,0.0,0.0,list_obs);
                            if ( st != GOTH__NORMAL) then return error_report_status(st);
                             #--- EXECUTA A FUNCAO QUE RETORNA O TIPO DE INTERSECAO ENTRE O OBJETO
                              ATIVO E OS DEMAIS QUE O INTERSEPTAM---#
                          st :=  Funcao_Validacao_Area (vac_id, list_obs, object_id,geom1_id,nome_classe1,
                                nome_classe2,tipo_relac, v_contem);
                           if ( st != GOTH__NORMAL) then return error_report_status(st);
                end;
     end;

    #--- TESTA SE O OBJETO ATIVO TEM A FORMA DE LINHA SIMPLES OU COMPLEXA.---#
    #--- O PONTO NAO EH TESTADO PORQUE NAO PODE CONTER OUTRO OBJETO ---#
    if ((tipo_geometria == _GT_SIMP_LINE) or (tipo_geometria == _GT_COMP_LINE) or
       (tipo_geometria == _GT_SIMP_POINT)) then
    begin
       #--- VERIFICA QUE SE O OBJETO ATIVO CONTEM ALGUM OUTRO OBJETO---#
       st := obj_get_value_from_name(vac_id,object_id,"contained",list_obs1,is_null);

       if ( st != GOTH__NORMAL) then return error_report_status(st);
       if ( not (is_null)) then
       begin
           #--- EXECUTA A FUNCAO QUE CONFIRMA SE O RELACIONAMENTO COM OUTRO
               OBJETO EH DE CONTINENCIA---#
           str_v_contem := "OK";
           st :=  Funcao_Validacao (vac_id, list_obs1, nome_classe1,nome_classe2,
                           object_id,tipo_relac,str_v_contem);
           if ( st != GOTH__NORMAL) then return error_report_status(st);

        end;
      end;

  retorno := "OK";
  return GOTH__NORMAL;

end;



function integer  Igual (VAC vac_id,
                    OBJECT object_id,
                    string nome_classe1,
                    string nome_classe2,
                    string retorno)
begin
```

```
boolean is_null, existe_classe;
integer st, num_classes, classes_testadas[0], id_classes [200], tipo_geometria,
classe_id, cont, v_contem;
COLLECTION list_obs, list_obs1;
GEOMETRY geom1_id;
string str_v_contem,tipo_relac;
v_contem := 10;
num_classes := 0;

#--- VERIFICA SE A SEGUNDA CLASSE QUE COMPOE A RESTRICAO EXISTE-----#
st := meta_check_class_exists (vac_id,nome_classe2,existe_classe);
if (st != GOTH__NORMAL) then return sch_print_message_stack();
if (NOT existe_classe) then return GOTH__NORMAL;

#--- RECUPERA A GEOMETRIA DO PRIMEIRO OBJETO---#
st := obj_get_value_from_name(vac_id,object_id,"geometry",geom1_id,is_null);
if ( st != GOTH__NORMAL) then return error_report_status(st);
#--- VERIFICA O TIPO DO PRIMEIRO OBJETO, SE EH AREA OU NAO---#
st := geom_get_type (geom1_id,tipo_geometria);
if ( st != GOTH__NORMAL) then return error_report_status(st);
#--- TESTA SE A GEOMETRIA EH UMA AREA SIMPLES OU COMPLEXA #
tipo_relac := "IGUAL";
if (tipo_geometria == _GT_SIMP_AREA ) then
begin
                #--- SE A SEGUNDA CLASSE EXISTIR RECUPERA TODAS AS SUAS SUBCLASSES---#
        st := meta_get_sub_classes (vac_id,nome_classe2,num_classes,classes_testadas);
        if (st != GOTH__NORMAL) then return sch_print_message_stack();

        cont := 1;

        while (cont <= num_classes) do
        begin
            id_classes[cont] := classes_testadas[cont];
            cont := cont + 1;
        end;

        #--- SE NAO TIVER SUBCLASSES TESTA COM ELA MESMA---#
        st := meta_get_class_id_from_name(vac_id,nome_classe2,classe_id);
        if ( st != GOTH__NORMAL) then return error_report_status(st);

        if (num_classes <= 0) then
        begin
            num_classes := 1;
            id_classes[1] := classe_id;
        end;
                else
                    num_classes := num_classes + 1;
        id_classes[num_classes] :=  classe_id;
        #-- RECUPERA OS OBJETOS QUE INTERCEPTAM A AREA DO OBJETO ATIVO---#
        st := quadx_classed_object_in_area(vac_id,vac_id,0,num_classes,id_classes,
                    geom1_id,0.0,0.0,list_obs);
        if ( st != GOTH__NORMAL) then return error_report_status(st);
        #--- EXECUTA A FUNCAO QUE RETORNA O TIPO DE INTERSECAO ENTRE O OBJETO
            ATIVO E OS DEMAIS QUE O INTERSEPTAM---#
            st :=  Funcao_Validacao_Area (vac_id, list_obs, object_id,geom1_id,nome_classe1,
                        nome_classe2,tipo_relac, v_contem);
            if ( st != GOTH__NORMAL) then return error_report_status(st);
        #--- TESTA SE O TIPO DE INTERSECAO EH DE CONTINENCIA---#


end;

#--- TESTA SE O OBJETO ATIVO TEM A FORMA DE LINHA SIMPLES OU COMPLEXA.---#
#--- O PONTO NAO EH TESTADO PORQUE NAO PODE CONTER OUTRO OBJETO ---#
if ((tipo_geometria == _GT_SIMP_LINE) or (tipo_geometria == _GT_COMP_LINE) or
  (tipo_geometria == _GT_SIMP_POINT)) then
begin
    #--- VERIFICA QUE SE O OBJETO ATIVO CONTEM ALGUM OUTRO OBJETO---#
    st := obj_get_value_from_name(vac_id,object_id,"equal",list_obs1,is_null);

    if ( st != GOTH__NORMAL) then return error_report_status(st);
    if ( not (is_null)) then
    begin
        #--- EXECUTA A FUNCAO QUE CONFIRMA SE O RELACIONAMENTO COM OUTRO
            OBJETO EH DE CONTINENCIA---#
        str_v_contem := "OK";
```

```
            st :=  Funcao_Validacao (vac_id, list_obs1, nome_classe1,nome_classe2,
                                object_id,tipo_relac,str_v_contem);
            if ( st != GOTH__NORMAL) then return error_report_status(st);


        end;
      end;

  retorno := "OK";
  return GOTH__NORMAL;

end;

function integer  Toca (VAC vac_id,
                        OBJECT object_id,
                        string nome_classe1,
                        string nome_classe2,
                        string retorno)
begin
     boolean is_null, existe_classe;
     integer st, num_classes, classes_testadas[0], id_classes [200], tipo_geometria,
     classe_id, cont, v_contem;
     COLLECTION list_obs, list_obs1;
     GEOMETRY geom1_id;
     string str_v_contem,tipo_relac;
     v_contem := 10;
     num_classes := 0;

     #--- VERIFICA SE A SEGUNDA CLASSE QUE COMPOE A RESTRICAO EXISTE-----#
     st := meta_check_class_exists (vac_id,nome_classe2,existe_classe);
     if (st != GOTH__NORMAL) then return sch_print_message_stack();
     if (NOT existe_classe) then return GOTH__NORMAL;

     #--- RECUPERA A GEOMETRIA DO PRIMEIRO OBJETO---#
     st := obj_get_value_from_name(vac_id,object_id,"geometry",geom1_id,is_null);
     if ( st != GOTH__NORMAL) then return error_report_status(st);
     #--- VERIFICA O TIPO DO PRIMEIRO OBJETO, SE EH AREA OU NAO---#
     st := geom_get_type (geom1_id,tipo_geometria);
     if ( st != GOTH__NORMAL) then return error_report_status(st);
     #--- TESTA SE A GEOMETRIA EH UMA AREA SIMPLES OU COMPLEXA #
     tipo_relac := "TOCAR";
     if (tipo_geometria == _GT_SIMP_AREA ) then
     begin
                  #--- SE A SEGUNDA CLASSE EXISTIR RECUPERA TODAS AS SUAS SUBCLASSES---#
                  st := meta_get_sub_classes (vac_id,nome_classe2,num_classes,classes_testadas);
                  if (st != GOTH__NORMAL) then return sch_print_message_stack();

                  cont := 1;

                  while (cont <= num_classes) do
                  begin
                      id_classes[cont] := classes_testadas[cont];
                      cont := cont + 1;
                  end;

                  #--- SE NAO TIVER SUBCLASSES, TESTA COM ELA MESMA---#
                  st := meta_get_class_id_from_name(vac_id,nome_classe2,classe_id);
                  if ( st != GOTH__NORMAL) then return error_report_status(st);

                  if (num_classes <= 0) then
                  begin
                     num_classes := 1;
                     id_classes[1] := classe_id;
                  end;
                        else
                            num_classes := num_classes + 1;
                            id_classes[num_classes] :=  classe_id;

                           #-- RECUPERA OS OBJETOS QUE INTERCEPTAM A AREA DO OBJETO ATIVO---#
                         st := quadx_classed_object_in_area(vac_id,vac_id,0,num_classes,id_classes,
                               geom1_id,0.0,0.0,list_obs);
                         if ( st != GOTH__NORMAL) then return error_report_status(st);
               #--- EXECUTA A FUNCAO QUE RETORNA O TIPO DE INTERSECAO ENTRE O OBJETO
                    ATIVO E OS DEMAIS QUE O INTERSEPTAM---#
                    st :=  Funcao_Validacao_Area (vac_id, list_obs, object_id,geom1_id,nome_classe1,
                                nome_classe2,tipo_relac, v_contem);
```

```
                    if ( st != GOTH__NORMAL) then return error_report_status(st);
            #--- TESTA SE O TIPO DE INTERSECAO EH DE ADJACENCIA---#


    end;

    #--- TESTA SE O OBJETO ATIVO TEM A FORMA DE LINHA SIMPLES OU COMPLEXA.---#
    #--- O PONTO NAO EH TESTADO PORQUE NAO PODE CONTER OUTRO OBJETO ---#
    if ((tipo_geometria == _GT_SIMP_LINE) or (tipo_geometria == _GT_COMP_LINE) or
      (tipo_geometria == _GT_SIMP_POINT)) then
    begin
       #--- VERIFICA QUE SE O OBJETO ATIVO TOCA ALGUM OUTRO OBJETO---#
       st := obj_get_value_from_name(vac_id,object_id,"touch",list_obs1,is_null);

       if ( st != GOTH__NORMAL) then return error_report_status(st);
       if ( not (is_null)) then
       begin
          #--- EXECUTA A FUNCAO QUE CONFIRMA SE O RELACIONAMENTO COM OUTRO
             OBJETO EH DE ADJACENCIA---#
          str_v_contem := "OK";
          st :=  Funcao_Validacao (vac_id, list_obs1, nome_classe1,nome_classe2,
                          object_id,tipo_relac,str_v_contem);
          if ( st != GOTH__NORMAL) then return error_report_status(st);


       end;
     end;

  retorno := "OK";
  return GOTH__NORMAL;

end;

function integer  Cobre (VAC vac_id,
                    OBJECT object_id,
                    string nome_classe1,
                    string nome_classe2,
                    string retorno)
begin
    boolean is_null, existe_classe;
    integer st, num_classes, classes_testadas[0], id_classes [200], tipo_geometria,
    classe_id, cont, v_contem;
    COLLECTION list_obs, list_obs1;
    GEOMETRY geom1_id;
    string str_v_contem,tipo_relac;
    v_contem := 10;
    num_classes := 0;

    #--- VERIFICA SE A SEGUNDA CLASSE QUE COMPOE A RESTRICAO EXISTE-----#
    st := meta_check_class_exists (vac_id,nome_classe2,existe_classe);
    if (st != GOTH__NORMAL) then return sch_print_message_stack();
    if (NOT existe_classe) then return GOTH__NORMAL;

    #--- RECUPERA A GEOMETRIA DO PRIMEIRO OBJETO---#
    st := obj_get_value_from_name(vac_id,object_id,"geometry",geom1_id,is_null);
    if ( st != GOTH__NORMAL) then return error_report_status(st);
    #--- VERIFICA O TIPO DO PRIMEIRO OBJETO, SE EH AREA OU NAO---#
    st := geom_get_type (geom1_id,tipo_geometria);
    if ( st != GOTH__NORMAL) then return error_report_status(st);
    #--- TESTA SE A GEOMETRIA EH UMA AREA SIMPLES OU COMPLEXA #
    tipo_relac := "COBRIR";
    if (tipo_geometria == _GT_SIMP_AREA ) then
    begin
                #--- SE A SEGUNDA CLASSE EXISTIR RECUPERA TODAS AS SUAS SUBCLASSES---#
                st := meta_get_sub_classes (vac_id,nome_classe2,num_classes,classes_testadas);
                if (st != GOTH__NORMAL) then return sch_print_message_stack();

                cont := 1;

                while (cont <= num_classes) do
                begin
                   id_classes[cont] := classes_testadas[cont];
                   cont := cont + 1;
                end;

                #--- SE NAO TIVER SUBCLASSES TESTA COM ELA MESMA---#
```

```
                st := meta_get_class_id_from_name(vac_id,nome_classe2,classe_id);
                if ( st != GOTH__NORMAL) then return error_report_status(st);

                if (num_classes <= 0) then
                begin
                   num_classes := 1;
                   id_classes[1] := classe_id;
                      end;
                      else
                         num_classes := num_classes + 1;
                         id_classes[num_classes] :=  classe_id;

                        #-- RECUPERA OS OBJETOS QUE INTERCEPTAM A AREA DO OBJETO ATIVO---#
                      st := quadx_classed_object_in_area(vac_id,vac_id,0,num_classes,id_classes,
                           geom1_id,0.0,0.0,list_obs);
                      if ( st != GOTH__NORMAL) then return error_report_status(st);
            #--- EXECUTA A FUNCAO QUE RETORNA O TIPO DE INTERSECAO ENTRE O OBJETO
                 ATIVO E OS DEMAIS QUE O INTERSEPTAM---#
                 st :=  Funcao_Validacao_Area (vac_id, list_obs, object_id,geom1_id,nome_classe1,
                              nome_classe2,tipo_relac, v_contem);
                      if ( st != GOTH__NORMAL) then return error_report_status(st);
            #--- TESTA SE O TIPO DE INTERSECAO EH COBRE---#


    end;

    #--- TESTA SE O OBJETO ATIVO TEM A FORMA DE LINHA SIMPLES OU COMPLEXA.---#
    #--- O PONTO NAO EH TESTADO PORQUE NAO PODE CONTER OUTRO OBJETO ---#
    if ((tipo_geometria == _GT_SIMP_LINE) or (tipo_geometria == _GT_COMP_LINE) or
      (tipo_geometria == _GT_SIMP_POINT)) then
    begin
       #--- VERIFICA QUE SE O OBJETO ATIVO COBRE ALGUM OUTRO OBJETO---#
       st := obj_get_value_from_name(vac_id,object_id,"on",list_obs1,is_null);

       if ( st != GOTH__NORMAL) then return error_report_status(st);
       if ( not (is_null)) then
       begin
          #--- EXECUTA A FUNCAO QUE CONFIRMA SE O RELACIONAMENTO COM OUTRO
              OBJETO EH COBRE---#
          str_v_contem := "OK";
          st :=  Funcao_Validacao (vac_id, list_obs1, nome_classe1,nome_classe2,
                          object_id,tipo_relac,str_v_contem);
          if ( st != GOTH__NORMAL) then return error_report_status(st);


       end;
     end;

  retorno := "OK";
  return GOTH__NORMAL;

end;

function integer  Coberto (VAC vac_id,
                   OBJECT object_id,
                   string nome_classe1,
                   string nome_classe2,
                   string retorno)
begin
    boolean is_null, existe_classe;
    integer st, num_classes, classes_testadas[0], id_classes [200], tipo_geometria,
    classe_id, cont, v_contem;
    COLLECTION list_obs, list_obs1;
    GEOMETRY geom1_id;
    string str_v_contem,tipo_relac;
    v_contem := 10;
    num_classes := 0;

    #--- VERIFICA SE A SEGUNDA CLASSE QUE COMPOE A RESTRICAO EXISTE-----#
    st := meta_check_class_exists (vac_id,nome_classe2,existe_classe);
    if (st != GOTH__NORMAL) then return sch_print_message_stack();
    if (NOT existe_classe) then return GOTH__NORMAL;

    #--- RECUPERA A GEOMETRIA DO PRIMEIRO OBJETO---#
    st := obj_get_value_from_name(vac_id,object_id,"geometry",geom1_id,is_null);
    if ( st != GOTH__NORMAL) then return error_report_status(st);
```

```
    #--- VERIFICA O TIPO DO PRIMEIRO OBJETO, SE EH AREA OU NAO---#
    st := geom_get_type (geom1_id,tipo_geometria);
    if ( st != GOTH__NORMAL) then return error_report_status(st);
    #--- TESTA SE A GEOMETRIA EH UMA AREA SIMPLES OU COMPLEXA #
    tipo_relac := "COBERTO";
    if (tipo_geometria == _GT_SIMP_AREA ) then
    begin
                #--- SE A SEGUNDA CLASSE EXISTIR RECUPERA TODAS AS SUAS SUBCLASSES---#
                 st := meta_get_sub_classes (vac_id,nome_classe2,num_classes,classes_testadas);
                 if (st != GOTH__NORMAL) then return sch_print_message_stack();

                 cont := 1;

                 while (cont <= num_classes) do
                 begin
                     id_classes[cont] := classes_testadas[cont];
                     cont := cont + 1;
                 end;

                 #--- SE NAO TIVER SUBCLASSES TESTA COM ELA MESMA---#
                 st := meta_get_class_id_from_name(vac_id,nome_classe2,classe_id);
                 if ( st != GOTH__NORMAL) then return error_report_status(st);

                 if (num_classes <= 0) then
                 begin
                    num_classes := 1;
                    id_classes[1] := classe_id;
                 end;
                       else
                          num_classes := num_classes + 1;
                          id_classes[num_classes] :=  classe_id;

                        #-- RECUPERA OS OBJETOS QUE INTERCEPTAM A AREA DO OBJETO ATIVO---#
                        st := quadx_classed_object_in_area(vac_id,vac_id,0,num_classes,id_classes,
                              geom1_id,0.0,0.0,list_obs);
                        if ( st != GOTH__NORMAL) then return error_report_status(st);
            #--- EXECUTA A FUNCAO QUE RETORNA O TIPO DE INTERSECAO ENTRE O OBJETO
                  ATIVO E OS DEMAIS QUE O INTERSEPTAM---#
                  st :=  Funcao_Validacao_Area (vac_id, list_obs, object_id,geom1_id,nome_classe1,
                              nome_classe2,tipo_relac, v_contem);
                        if ( st != GOTH__NORMAL) then return error_report_status(st);
            #--- TESTA SE O TIPO DE INTERSECAO EH DE COBERTURA---#


    end;

    #--- TESTA SE O OBJETO ATIVO TEM A FORMA DE LINHA SIMPLES OU COMPLEXA.---#

    if ((tipo_geometria == _GT_SIMP_LINE) or (tipo_geometria == _GT_COMP_LINE) or
      (tipo_geometria == _GT_SIMP_POINT)) then
    begin
       #--- VERIFICA QUE SE O OBJETO ATIVO COBRE ALGUM OUTRO OBJETO---#
       st := obj_get_value_from_name(vac_id,object_id,"off",list_obs1,is_null);

       if ( st != GOTH__NORMAL) then return error_report_status(st);
       if ( not (is_null)) then
       begin
           #--- EXECUTA A FUNCAO QUE CONFIRMA SE O RELACIONAMENTO COM OUTRO
               OBJETO EH DE COBERTURA---#
           str_v_contem := "OK";
           st :=  Funcao_Validacao (vac_id, list_obs1, nome_classe1,nome_classe2,
                          object_id,tipo_relac,str_v_contem);
           if ( st != GOTH__NORMAL) then return error_report_status(st);


       end;
     end;
   retorno := "OK";
  return GOTH__NORMAL;
end;

function integer  Atravessa (VAC vac_id,
                     OBJECT object_id,
                     string nome_classe1,
                     string nome_classe2,
                     string retorno)
```

```
begin
    boolean is_null, existe_classe;
    integer st, num_classes, classes_testadas[0], id_classes [200], tipo_geometria,
    classe_id, cont, v_contem;
    COLLECTION list_obs, list_obs1;
    GEOMETRY geom1_id;
    string str_v_contem,tipo_relac;
    v_contem := 10;
    num_classes := 0;

    #--- VERIFICA SE A SEGUNDA CLASSE QUE COMPOE A RESTRICAO EXISTE-----#
    st := meta_check_class_exists (vac_id,nome_classe2,existe_classe);
    if (st != GOTH__NORMAL) then return sch_print_message_stack();
    if (NOT existe_classe) then return GOTH__NORMAL;

    #--- RECUPERA A GEOMETRIA DO PRIMEIRO OBJETO---#
    st := obj_get_value_from_name(vac_id,object_id,"geometry",geom1_id,is_null);
    if ( st != GOTH__NORMAL) then return error_report_status(st);
    #--- VERIFICA O TIPO DO PRIMEIRO OBJETO, SE EH AREA OU NAO---#
    st := geom_get_type (geom1_id,tipo_geometria);
    if ( st != GOTH__NORMAL) then return error_report_status(st);
    #--- TESTA SE A GEOMETRIA EH UMA AREA SIMPLES OU COMPLEXA #
    tipo_relac := "ATRAVESSAR";


    #--- TESTA SE O OBJETO ATIVO TEM A FORMA DE LINHA SIMPLES OU COMPLEXA.---#
    #--- O PONTO NAO EH TESTADO PORQUE NAO PODE CRUZAR OU SER CRUZADO POR  OUTRO OBJETO ---#
    if (tipo_geometria == _GT_SIMP_LINE) or (tipo_geometria == _GT_COMP_LINE) or (tipo_geometria == _GT_SIMP_AREA
) then
    begin
            #--- SE A SEGUNDA CLASSE EXISTIR RECUPERA TODAS AS SUAS SUBCLASSES---#
            st := meta_get_sub_classes (vac_id,nome_classe2,num_classes,classes_testadas);
            if (st != GOTH__NORMAL) then return sch_print_message_stack();

            cont := 1;

            while (cont <= num_classes) do
             begin
                    id_classes[cont] := classes_testadas[cont];
                    cont := cont + 1;
             end;

            #--- SE NAO TIVER SUBCLASSES TESTA COM ELA MESMA---#
            st := meta_get_class_id_from_name(vac_id,nome_classe2,classe_id);
            if ( st != GOTH__NORMAL) then return error_report_status(st);

            if (num_classes <= 0) then
            begin
                  num_classes := 1;
                  id_classes[1] := classe_id;
            end;
            else
            begin
                  num_classes := num_classes + 1;
                  id_classes[num_classes] :=  classe_id;

                  #-- RECUPERA OS OBJETOS QUE INTERCEPTAM A AREA DO OBJETO ATIVO---#
                  st := quadx_classed_object_in_area(vac_id,vac_id,0,num_classes,id_classes,
                              geom1_id,0.0,0.0,list_obs);
                  if ( st != GOTH__NORMAL) then return error_report_status(st);
           end;
       #--- VERIFICA QUE SE O OBJETO ATIVO ATRAVESSA ALGUM OUTRO OBJETO---#
       st := obj_get_value_from_name(vac_id,object_id,"intersect",list_obs1,is_null);

       if ( st != GOTH__NORMAL) then return error_report_status(st);
       if ( not (is_null)) then
       begin
          #--- EXECUTA A FUNCAO QUE CONFIRMA SE O RELACIONAMENTO COM OUTRO
              OBJETO EH DE CRUZAMENTO---#
          str_v_contem := "OK";
          st :=  Funcao_Validacao (vac_id, list_obs1, nome_classe1,nome_classe2,
                              object_id,tipo_relac,str_v_contem);
          if ( st != GOTH__NORMAL) then return error_report_status(st);


       end;
```

```
      end;

  retorno := "OK";
  return GOTH__NORMAL;

end;
```

# Referências Bibliográficas


[BOG 99] BOGORNY, V. **Um estudo sobre o *OpenGIS***: a proposta da OGC para interoperabilidade e distribuição em Sistemas de Informação Geográfica. Porto Alegre: PPGC da UFRGS, 1999. (TI-865).

[BOG 99a] BOGORNY, V. Exercício de representação do *framework* conceitual GeoFrame no modelo proposto pelo consórcio OpenGIS. Porto Alegre: PPGC da UFRGS, 1999. (TI-898).

[BOO 98] BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **The Unified Modeling Language user guide**. Reading: Addison-Wesley, 1998.

[BOO 2000] BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. **UML:** guia do usuário. Rio de Janeiro: Campus, 2000.

[BOR 97] 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.

[BOR 99] BORGES, K. A. V.; LAENDER, A. H. F.; DAVIS Jr., C. A. Spatial data integrity constraints in object oriented geographic data modeling. In: ACM GIS SYMPOSIUM ON ADVANCES IN GEOGRAPHIC INFORMATION SYSTEMS, 7., 1999, Kansas City, USA. **Proceedings...** Kansas City: ACM, 1999.

[BUE 98] BUEHLER, K.; MCKEE, L. **The OpenGIS guide.** Massachusetts, USA. Disponível em: <http://www.*OpenGIS*.org/techno/guide.html>. Acesso em: jun. 1998.

[CAM 96] CAMARA, G. et al. **Anatomia de Sistemas de Informação Geográfica.** Campinas: Instituto de Computação, UNICAMP, 1996.

[CAM 96a] CAMARA, G.; MEDEIROS, J. S. de. **Geoprocessamento para projetos ambientais.** São José dos Campos, SP: Instituto Nacional de Pesquisas Espaciais, 1996. Relatório Técnico.

[CIL 96] CILIA, M.; MEDEIROS, C. B. Combinando bancos de dados ativos e Sistemas de Informação Geográfica para manutenção de restrições topológicas. In: SIMPÓSIO BRASILEIRO DE BANCO DE DADOS, 11., 1996, São Carlos, SP. **Anais...** São Carlos: Departamento de Computação / UFSCAR, 1996.

[CLE 93] CLEMENTINI, E.; DI FELICE, P.; VAN OSTERN, P. A small set of formal topological relationships for end-user interaction. In: ABEL, D; OOI, B.C. (Eds.). **Advances in Spatial Databases**. [S.l.]: Springer-Verlag, 1993. p. 277-295. (Lecture Notes in Computer Science, v. 692)

[CLE94] CLEMENTINI, E.; SHARMA, J.; EGENHOFER, M. Modeling topological spatial relations: strategies for query processing. **Computers & Graphics,** Oxford, v.18, n.6, p. 815-822, Nov./Dec. 1994.

[COC 96] COCKCROFT, S. Towards the automatic enforcement of integrity rules in spatial database systems. In: ANNUAL COLLOQUIUM OF THE SPATIAL INFORMATION RESEARCH CENTRE, 8., 1995. **Proceedings...** Otago: University of Otago, 1996. p. 33-42.

[COC 97] COCKCROFT, S. A taxonomy of spatial data integrity constraints. **Geoinformatica**, [S.l.], v.1, n.4, p.327-343, 1997.

[EGE 91] EGENHOFER, M. et al. A framework for the definition of topological relationships and na Algebraic Approach to Spatial Reasoning Within this Framework. [S.l.]: NCGIA, 1991. (Technical Report 91-7).

[EGE 93] EGENHOFER, M. A model for detailed binary topological relationships. **Geomatica,** [S.l.], v.47, n.3-4, p. 261-273, 1993.

[EGE 93a] EGENHOFER, M.; HERRING, J. Categorizing binary topological relationships between regions, lines and points in geographic databases. Orono: University of Maine, 1993.

[EGE 94] EGENHOFER, M.; CLEMENTINI, E.; FELICE, P. di. Topological relations between regions with holes. **International Journal of Geographical Information Systems,** [S.l.], v.8, n.2, p. 129-144, 1994.

[EGE 95] EGENHOFER, M.; FRANZOSA, R. On the equivalence of topological relations. **International Journal of Geographical Information Systems,** [S.l], v.9, n.2, p. 133-152, 1995.

[EGE 95a] EGENHOFER, M.; MARK, D. M. Modeling conceptual neighborhoods of topological line-regions relaions. **International Journal of Geographical Information Systems,** [S.l], v.9, n.2, p. 555-565, 1995.

[ELM 90] ELMAGARMID, A. K. **Database transaction models.** San Mateo, California: Morgan Kaufmann, 1990.

[FAR 98] FARIA, G. **Um Banco de dados espaço-temporal para desenvolvimento de aplicações em Sistemas de Informação Geográfica**. Campinas: Instituto de Computação – Universidade Estadual de Campinas, 1998. Dissertação de Mestrado.

[FOW 96] FOWLER, M.; SCOTT, K. **UML Distilled.** Reading: Addison-Wesley, 1997. 179 p.

[FUR 98] FURLAN, J. D. **Modelagem de objetos através da UML**. São Paulo: Makron Books, 1998.

[GAR 98] GARAFFA, I. M. Análise da adequação de uma hierarquia de classes básicas para modelagem conceitual de SIG através de um Estudo de Caso. Porto Alegre: CPGCC da UFRGS, 1998. Dissertação de Mestrado.

[GUT 94] Güting, H. R. An Introduction to Spatial Database Systems. **VLDB Journal**, [S.l.], v.10, n.4, p. 357-399, 1994.

[HAD 92] HADZILACOS, T.; TRYFONA, T. A model for expressing topological integrity constraints in geographic databases**.** In: INTERNATIONAL CONFERENCE GIS, 1995. **Proceedings...** Pisa, Italy: [s.n.], 1992. (Lecture Notes in Computer Science, v.639).

[HAD 96] HADZILACOS, T.; TRYFONA, N. Logical data modelling for geographical applications. **International Journal of Geographical Infomation Systems,** [S.1.], v.10, n.2, p. 179-203, 1996.

[HAE 87] HAERDER, T.; ROTHERMEL, K. Concepts for transaction recovery in nested transactions. In: ACM SIGMOD, 1987. **Proceedings...** [S.l..:s.n.], 1987.

[HEB 99] HEBERT, J.; MURRAY, C. **Oracle spatial user´s guide and reference.** [S.l.]: Oracle Corporation, 1999.

[KOS 95] KOSTERS, G.; PAGEL, B.; SIX, H. **GeoOOA:** object-oriented analisys for GIS-aplications**.** Germany: University of Hagen, 1995.

[LIS 96] LISBOA FILHO, J.; IOCHPE, C. Introdução a Sistemas de Informações Geográficas com ênfase em banco de dados. In: SBC, 1996. **Anais...** [S.l.:s.n], 1996.

[LIS 97] LISBOA FILHO, J. **Modelos Conceituais de Dados Para Sistemas de Informações Geográficas:** exame de qualificação. Porto Alegre: CPGCC da UFRGS, 1997.

[LIS 99] LISBOA FILHO, J.; IOCHPE, C. GeoFrame: um *Framework* Conceitual para Especificação de Padrões de Análise em Banco de Dados Geográficos. In: ACMGIS, 1999. **Anais...** [S.l.:s.n], 1999.

[MAN 95] MANGAN, M. A. **Um gerente de execução para o modelo ConTracts de transações longas.** Porto Alegre: Instituto de Informática da UFRGS, 1995. Trabalho de Conclusão.

[MOL 87] MOLINA, H. G.; SALEN, K. SAGAS. In: ACM SIGMOD ANNUAL CONFERENCE ON MANAGMENTO DATA, 1987. **Proceedings...** San Francisco, CA, [s.n.], 1987.

[OGC 99] OPEN GIS CONSORTIUM. **Topic 0, the *OpenGIS* abstract specification overview. Version 4**. Disponível em: <http://www.*OpenGIS*.org/techno/specs.htm>. Acesso em: jul. 1999.

[OGC 99a] OPEN GIS CONSORTIUM. **Topic 1, the *OpenGIS* abstract specification – feature geometry – Version 4**. Disponível em: <http://www.*OpenGIS*.org/techno/specs.htm>. Acesso em: jul. 1999.

[OGC 99b] OPEN GIS CONSORTIUM. **Topic 5, the *OpenGIS* abstract specification – *OpenGIS* features – Version 4**. Disponível em: <http://www.*OpenGIS*.org/techno/specs.htm>. Acesso em: jul. 1999.

[OGC 99c] OPEN GIS CONSORTIUM. **Topic 6, the *OpenGIS* abstract specification – The coverage type and its subtypes – Version 4**. Disponível em: <http://www.*OpenGIS*.org/techno/specs.htm>. Acesso em: jul. 1999.

[OGC 99d] OPEN GIS CONSORTIUM. **Topic 8, the *OpenGIS* abstract specification relationships between features – Version 4**. Disponível em: <http://www.*OpenGIS*.org/techno/specs.htm>. Acesso em: jul. 1999.

[OGC 99e] OPEN GIS CONSORTIUM. **Topic 11, the *OpenGIS* abstract specification – metadata – Version 4**. Disponível em: <http://www.*OpenGIS*.org/techno/specs.htm>. Acesso em: jul. 1999.

[OGC 2000] OPEN GIS CONSORTIUM - HERRING, J. R. **Geometry Model - OpenGIS Project Document 2000-32,** 2000. Disponível em: <http://www.*OpenGIS*.org>. Acesso em ago. 2000.

[OMG 99] OBJECT MANAGEMENT GROUP. **Unified Modeling Language Specification.** [S.l.], 1999.

[ROA 99] ROAD, M. **Gothic database concepts.** United Kingdom: Laser-Scan, 1999.

[PAR 98] PARENT, C. et al. Modeling spatial data in the MADS conceptual model. In: INTERNATIONAL SYMPOSIUM ON SPATIAL DATA HANDLING, 1998. **Proceedings...** Canada: [s.n.], 1998.

[PIR 97] PIRES, Fátima. **Um ambiente computacional para modelagem de aplicações ambientais**. Campinas: Universidade Estadual de Campinas, 1997. Tese de doutorado.

[SER 2000] SERVIGNE, S. et al. A methodology for spatial consistency improvement of geographic databases. **Geoinformatica**, [S.l.], v.4, n.1, p.7-34, 2000.

[SIL 95] SILBERCHATZ, A.; KORTH, H. F. **Sistemas de bancos de dados.** 2. ed. São Paulo: Makron Books, 1995.

[THO 98] THOMÉ, R. **Interoperabilidade em geoprocessamento**: conversão entre modelos conceituais de Sistemas de Informação Geográfica e comparação com o padrão OpenGIS. São José dos Campos, SP: [s.n.], 1998. Dissertação de Mestrado.

# Bibliografia Complementar


[ABR 94] ABRANTES, G.; CARAPUCA, R. Explicit representation of data that depend on topological relatinships and control over data consistency. In: EUROPEAN CONFERENCE AND EXHIBITION ON GEOGRAPHICAL INFORMATION SYSTEMS – EGIS/MARI, 1994. **Proceedings...** [S.l:s.n], 1994.

[BOG 2001] BOGORNY, V.; IOCHPE, C. Incorporando suporte a restrições espaciais no Padrão *OpenGIS.* In: GISBRASIL, 2001, Curitiba. **Anais...** Curitiba, PR: [s.n], 2001.

[CAM 94] CAMARA, G. **Análise de arquiteturas para Banco de Dados Geográficos Orientados a Objetos**. São José dos Campos, SP: Instituto Nacional de Pesquisas Espaciais, 1994. Relatório Técnico.

[CAM 95] CAMARA, G. **Modelos, linguagens e arquiteturas para Banco de Dados Geográficos**. São José dos Campos, SP: Instituto Nacional de Pesquisas Espaciais, 1995. Tese de Doutorado.

[CHA 92] CHAMPOUX, P. Notions fondamentales d'analyse spatiale et d'opérateyrs spatiaux. **Revue des sciences de 1'Information Géographique et de 1'Analyse Spatiale**, [S.l.], v.2, n.2, 1992.

[CLE 94a] CLEMENTINI, E.; DI FELICE, P. An algebraic model for spatial objects with undetermined boundaries. In: GISDATA SPECIALIST MEETING, 1995. **Proceedings...** Italy: [s.n.], 1995.

[CLE 94b] CLEMENTINI, E.; DI FELICE, P. A model for representing topological relationships between complex geometric features in spatial databases. **Information Sciences**, [S.l.], v.90, n.1-4, p. 121-136, 1994.

[CLE 94c] CLEMENTINI, E.; DI FELICE, P. **A comparision of methods for representing topological relationships**. Italy: University of L'Aquila, 1994.

[CLE 95] CLEMENTINI, E.; DI FELICE, P**.** A comparison of methods for representing topological relationships. **Information Sciences**, [S.l.], v.3, p. 149-178, 1995.

[EGE 92] EGENHOFER, M.; SHARMA, J. Topological consistency. In: INTERNATIONAL SYMPOSIUM ON SPATIAL DATA HANDLING 5., 1992. **Proceedings...** Charleston:[s.n], 1992.

[EGE 93b] EGENHOFER, M. **Deriving the composition of binary topological relations**. Orono: University of Maine,1993.

[EGE 94a] EGENHOFER, M. Pre-Processing queries with spatial constraints. **Photogrammetric Engineering & Remote Sensing,** [S.l.], v.60, n.6, p.783-790, 1994.

[EGE 94b] EGENHOFER, M. Spatial SQL: a query and presentation language**. IEEE Transactions on Knowledge and Data Engineering**, [S.l], v.6, n.1, p. 86-95, 1994.

[EGE 94c] EGENHOFER, M.; SHARMA, J. Assessing the consistency of complete and incomplete topological information. Orono: University of Maine, 1994.

[GRO 97] GROGER, G.; PLUMER, L. **Provably correct and complete transaction rules for GIS.** Germany: Universitat Bonn, 1997.

[LAT 92] LAURINI, R.; THOMPSON, D. **Fundamentals of spatial information systems**. London: Academic Press, 1992.

[OGC 97] OPEN GIS CONSORTIUM. **The OGC technical committee technology development process,** 1997. Disponível em: <http://www.*OpenGIS*.org> Acesso em: set. 1997.

[PAP 97] PAPADIAS, D.; THEODORIDIS, Y. Spatial relations, minimum bounding rectangles, and spatial data structures. Orono: University of Maine, 1997.

[RUM 94] RUMBAUGH, J. et al. **Modelagem e projeto baseados em objetos**. Rio de Janeiro: Campus, 1994.

[SAU 94] SAUNDERS, C.A.B. **Notas de cartografia.** 2.ed. Rio de Janeiro: IME, 1994. v.1.

[SIL 97] SILBERCHATZ, A.; KORTH, H. F. ; SUDARSHAN, S. **Sistema de bancos de dados. 3**. ed. São Paulo: Makron Books, 1997.

[SU 98] SU, B.; LODWICK, G.; LI, Z. Morphological models for the collapse of area features in digital map generalization. **GeoInformatica**, [S.l.], v.2, n.4, p. 359-382, 1998.

[TRY 97] TRYFONA, N.; EGENHOFER, M. Consistency among parts and aggregates: a computational model transactions. **International Conference GIS**, [S.l.], v.1, n.3, p. 189-206, 1997.

[TRY 2000] TRYFONA, N. A framework for constraint-based spatial data mining. In: INTERNATIONAL WORKSHOP ON EMERGING TECHNOLOGIES FOR GEO-BASED APPLICATIONS, 2000. **Proceedings...** Switzerland: [s.n.], 2000.

[WEI 98] WEIBEL, R.; JONES, C.B. Computational perspectives on map generalization**. GeoInformatica**, [S.l.], v.2, n.4, p. 307-314, 1998.

[WIN 2000] WINTER, S.; FRANK, A. U. Topology in Raster and Vector Representation. **GeoInformatica**, [S.l.], v.4, n.1, p. 35-65, 2000.