

**Banco de Dados Ativos como
Suporte a Restrições Topológicas
em Sistemas de Informação
Geográfica**

Mariano A. Cilia

Banco de Dados Ativos como Suporte a Restrições Topológicas em Sistemas de Informação Geográfica

Este exemplar corresponde à redação final da tese devidamente corrigida e defendida pelo Sr. Mariano A. Cilia e aprovada pela Comissão Julgadora.

Campinas, 11 de Março de 1996.


Cláudia Bauzer Medeiros
Orientadora

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

UNIDADE	BC
N.º CHEQUE	
	UNICAMP
	C489b
	27439
	667/96
	x
	R\$ 11,00
	25/04/96
N.º C.F.O.	C.M.000.37400-9

FICHA CATALOGRAFICA ELABORADA PELA
BIBLIOTECA DO IMECC DA UNICAMP

Cilia, Mariano Ariel

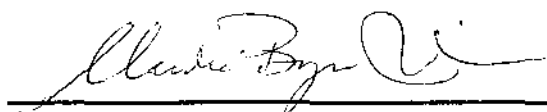
C489b Banco de dados ativos como suporte a restrições topológicas em sistemas de informação geográfica / Mariano Ariel Cilia -- Campinas, [S.P. :s.n.], 1996.

Orientadora: Claudia Bauzer Medeiros

Dissertação (mestrado) - Universidade Estadual de Campinas, Instituto de Matemática, Estatística e Ciência da Computação.

1. Banco de dados orientado a objetos. 2. Sistemas de informação geográfica. I. Medeiros, Claudia Bauzer. II. Universidade Estadual de Campinas. Instituto de Matemática, Estatística e Ciência da Computação. III. Título.

Tese de Mestrado defendida e aprovada em 11 de março de 1996
pela Banca Examinadora composta pelos Profs. Drs.



Prof (a). Dr (a). CLAUDIA MARIA BAUER MEDEIROS



Prof (a). Dr (a). GIOVANE CAYRES MAGALHÃES



Prof (a). Dr (a). GILBERTO CAMARA

Banco de Dados Ativos como Suporte a Restrições Topológicas em Sistemas de Informação Geográfica

Mariano A. Cilia

Departamento de Ciência da Computação
IMECC - UNICAMP

Banca Examinadora:

- Claudia Bauzer Medeiros¹ (Orientadora)
- Geovane Cayres Magalhães²
- Gilberto Câmara Neto³
- Luiz Eduardo Buzato⁴ (Suplente)

¹ Professora do Departamento de Ciência da Computação - IMECC - UNICAMP.

² Professor do Departamento de Ciência da Computação - IMECC - UNICAMP e Pesquisador do Centro de Pesquisas e Desenvolvimento (CPqD) da Telebrás.

³ Pesquisador do Instituto de Pesquisas Espaciais (INPE).

⁴ Professor do Departamento de Ciência da Computação - IMECC - UNICAMP.

DEDICATÓRIA

a Maria

AGRADECIMENTOS

- Em primeiro lugar, a Cláudia Bauzer Medeiros, pela assistência que recebi no decorrer da tese.
- À Faculdade de Ciências Exatas, UNICEN pelo apoio financeiro recebido.
- Ao projeto CNPq-PROTEM GEOTEC e o projeto ICDT-116 da Comunidade Européia, que financiaram parcialmente esta dissertação.
- A Maria pela enorme paciência e constante apoio.
- Ao grupo trabalho de Tandil: Jony, Pitty, Alvaro, e especialmente a Marcelo.
- Especialmente a Ana Maria, Sandra e Horacio, com quem certamente comecei uma forte amizade.
- Ao Juliano, pela amizade e por todas as discussões técnicas.
- A Cristina e Ricardo, entre outras coisas, pelas inúmeras revisões do texto.
- Ao Ricardo pelo apoio, amizade e os incontáveis favores.
- Aos amigos hispano-falantes Diego, Sérgio e Roly, pelos momentos de descontração.
- Aos meus amigos do grupo de banco de dados (Pedro Rafael, Márcio, Cereja, Luis Mariano, Fátima, Raimundo, Walter, Henrique).
- A todos meus companheiros e amigos de turma, especialmente a Sueli, Rosiane, Raul e Maurício.
- Aos meus amigos da minha terra, que não me deixaram esquecer o muito que eu tinha.
- A minha família, a Martha e José pelo constante apoio.
- Aos professores Gilberto Câmara e Geovane Magalhães, pelas contribuições à redação final deste documento.

RESUMO

Esta dissertação trata da utilização de sistemas ativos no contexto de aplicações geográficas. Os resultados aqui apresentados estendem o paradigma de SGBD ativos com o objetivo de solucionar o problema de manutenção de relacionamentos espaciais (topológicos) na presença de atualizações.

A solução apresentada para este problema está dividida em três etapas: i) especificação da restrição topológica; ii) transformação da restrição em regras; e iii) manutenção automática da restrição, com base nas regras geradas. Esta abordagem foi utilizada no desenvolvimento de um protótipo de sistema ativo que incorpora um modelo geográfico OO, eliminando o problema da impedância existente entre Sistemas de Informação Geográfica (SIG) e sistemas de regras.

As principais contribuições deste trabalho são um estudo detalhado sobre relacionamentos topológicos binários; uma proposta integrada para o problema de manutenção desses relacionamentos; e a definição de algoritmos para a verificação de integridade topológica, implementados no protótipo.

ABSTRACT

This dissertation concerns the use of active databases in geographic applications. The results presented here extend the active database systems paradigm to solve the problem of maintaining spatial (topological) constraints.

The solution for this problem is divided in three steps: i) topological constraint specification; ii) translation of the constraint into rules; and iii) automatic constraint maintenance, using the generated rules. This approach was used in the development of an active system prototype that incorporates an object-oriented geographic model, thus removing the gap between Geographic Information Systems (GIS) and rule systems.

The main contributions presented are a detailed study about binary topological relationships; an integrated proposal for the problem of maintaining these relationships; and the definition of algorithms to verify the topological integrity (these algorithms are incorporated in the prototype).

CONTEÚDO

DEDICATÓRIA.....	iv
AGRADECIMENTOS.....	v
RESUMO	vi
ABSTRACT	vii
LISTA DE TABELAS.....	xi
LISTA DE FIGURAS.....	xii
I. Introdução e Motivação	
1. INTRODUÇÃO.....	1
2. OBJETIVOS E ORGANIZAÇÃO DA DISSERTAÇÃO.....	3
II. Bancos de Dados Ativos	
1. INTRODUÇÃO.....	5
2. USO DOS SISTEMAS ATIVOS.....	8
3. REGRAS E GATILHOS.....	9
3.1 REGRAS E-C-A	9
3.2 LINGUAGENS PARA ESPECIFICAR EVENTOS	10
3.3 LINGUAGENS PARA ESPECIFICAR CONDIÇÕES.....	11
3.4 LINGUAGENS PARA ESPECIFICAR AÇÕES	11
3.5 COMPONENTES.....	12
4. MODELOS DE EXECUÇÃO.....	13
4.1 MODOS DE ACOPLAMENTO	13
4.2 GRANULARIDADE DE ATIVAÇÃO.....	15
4.3 RESOLUÇÃO DE CONFLITOS.....	16
4.4 ATIVAÇÃO EM CASCATA	16
5. OTIMIZAÇÃO	16
6. ALGUNS PROTÓTIPOS DE SISTEMAS ATIVOS.....	18
6.1 ARIEL [HAN89].....	18
6.1.1 Regras.....	18
6.1.2 Modelo de execução.....	19
6.1.3 Otimização.....	19
6.2 HiPAC [CBB+89]	19
6.2.1 Regras.....	19
6.2.2 Modelo de execução.....	20
6.2.3 Otimização.....	20
6.3 ODE [GJ91]	21

6.3.1 Restrições e Gatilhos.....	21
6.3.2 Modelo de Execução	22
6.4 POSTGRES [SJGP90]	22
6.4.1 Regras.....	22
6.4.2 Modelo de Execução	22
6.4.3 Otimização.....	23
6.5 SAMOS [GD92].....	23
6.5.1 Regras.....	23
6.5.2 Modelo de Execução	24
6.5.3 Otimização.....	24
6.6 STARBURST [WF90].....	24
6.6.1 Regras.....	24
6.6.2 Mecanismo de Execução	25
6.7 SENTINEL [CAM93].....	25
6.7.1 Regras.....	25
6.7.2 Modelo de Execução	26
6.7.3 Otimização.....	26
7. ALGUMAS FERRAMENTAS ÚTEIS	26
8. VISÃO GERAL.....	27

III. Manutenção de Restrições num Modelo Ativo Orientado a Objetos

1. RESTRIÇÕES DE INTEGRIDADE	29
1.1 REPRESENTAÇÃO DE RESTRIÇÕES DE INTEGRIDADE EM SGBDOO	30
1.2 REQUISITOS DA LINGUAGEM DE ESPECIFICAÇÃO	31
1.3 TIPOS DE MONITORAMENTO	31
2. BANCOS DE DADOS ATIVOS COMO SUPORTE A RESTRIÇÕES.....	32
3. MANUTENÇÃO DE RESTRIÇÕES USANDO UM SISTEMA ATIVO (OO)	33
3.1 CLASSIFICAÇÃO DOS OBJETOS.....	34
3.2 EVENTOS	35
3.3 REGRAS	35
3.4 ASSOCIAÇÃO DAS REGRAS A OBJETOS	36
4. QUESTÕES DE IMPLEMENTAÇÃO	36
5. DESCRIÇÃO DO MECANISMO ATIVO IMPLEMENTADO	39
5.1 HIERARQUIA DE CLASSES.....	40
5.1.1 Classe Reativa.....	40
5.1.2 Classe Notificável	42
5.1.3 Hierarquia de Eventos.....	42
5.1.4 A Classe Regra.....	43
5.2 DETECÇÃO DE EVENTOS (PRIMITIVOS E COMPOSTOS).....	44
5.3 MODELO DE EXECUÇÃO.....	45

IV. Sistemas de Informação Geográfica

1. CONCEITOS BÁSICOS.....	47
1.1 MODELOS DE DADOS	48
1.2 TIPOS DE DADOS	48
1.3 FUNCIONALIDADE	49
2. MODELO ADOTADO	51
3. OPERAÇÕES SOBRE OBJETOS GEOGRÁFICOS	52
4. SIG E BANCO DE DADOS ATIVOS	56

V. Restrições de Integridade Topológica

1. RELACIONAMENTOS ESPACIAIS	58
1.1 RELACIONAMENTOS DE ORIENTAÇÃO	58
1.2 RELACIONAMENTOS MÉTRICOS (ESCALARES).....	59
1.3 RELACIONAMENTOS TOPOLÓGICOS	59
2. ESPECIFICAÇÃO DE RESTRIÇÕES TOPOLÓGICAS.....	63
2.1 PREDICADO TOPOLÓGICO	63
2.2 SENTENÇA TOPOLÓGICA.....	64
2.3 LINGUAGEM PARA ESPECIFICAR RESTRIÇÕES TOPOLÓGICAS BINÁRIAS	65
3. TRANSFORMAÇÃO DE RESTRIÇÕES TOPOLÓGICAS EM REGRAS ECA.....	66
3.1 IDÉIAS CENTRAIS DA TRANSFORMAÇÃO.....	66
3.2 EVENTOS QUE PODEM VIOLAR UM PREDICADO TOPOLÓGICO	66
3.3 CONDIÇÕES PARA VERIFICAR UM PREDICADO TOPOLÓGICO	67
3.4 PREDICADOS TOPOLÓGICOS INTRA-CLASSE.....	69
3.5 TRATAMENTO DO ESTADO INICIAL.....	70
3.6 RESTRIÇÕES TOPOLÓGICAS COMPLEXAS.....	73
3.7 GERAÇÃO DE REGRAS	76

VI. Conclusões e Futuras Extensões

1. CONCLUSÕES.....	78
2. FUTURAS EXTENSÕES.....	79

Apêndice A. Implementação do Modelo Geográfico.....	81
--	-----------

Apêndice B. Análise Sintática	85
--	-----------

Apêndice C. Exemplos de Restrições Topológicas.....	88
--	-----------

Bibliografia	100
---------------------------	------------

LISTA DE TABELAS

TABELA 2.1. RESUMO DAS CARACTERÍSTICAS DOS PRINCIPAIS PROTÓTIPOS.....	28
TABELA 5.1. CASOS POSSÍVEIS DE RELACIONAMENTOS TOPOLÓGICOS BINÁRIOS UTILIZANDO O MÉTODO DAS 4-INTERSEÇÕES.....	60
TABELA 5.2. EVENTOS QUE PODEM VIOLAR UM PREDICADO TOPOLÓGICO.....	67
TABELA 5.3. CONSULTAS QUE REPRESENTAM O PREDICADO TOPOLÓGICO, SEGUNDO A LIGAÇÃO DE SUAS VARIÁVEIS. TAMBÉM SÃO DESCRITOS OS EVENTOS QUE ATIVAM A AVALIAÇÃO DO PREDICADO.....	68

LISTA DE FIGURAS

FIGURA 1.1. ORAGANIZAÇÃO DA DISSERTAÇÃO.....	3
FIGURA 2.1. EVOLUÇÃO DA GERÊNCIA DOS DADOS [TAN95].	5
FIGURA 2.2. POLLING [BUCH94].	6
FIGURA 2.3. INCORPORAÇÃO DE REGRAS AO BANCO DE DADOS.....	7
FIGURA 2.4. CLASSIFICAÇÃO DE EVENTOS EM SISTEMAS ATIVOS.....	10
FIGURA 2.5. COMPONENTES DE UM BANCO DE DADOS ATIVO [BUCH94].	12
FIGURA 2.6. ALGORITMO SIMPLES DE PROCESSAMENTO DE REGRAS [CFPW94].	13
FIGURA 2.7. MODOS DE ACOPLAMENTO.	14
FIGURA 2.8. MODOS DE ACOPLAMENTO DESACOPLADOS.....	15
FIGURA 2.9. MODO DE ACOPLAMENTO SEQUENCIAL INDEPENDENTE.....	15
FIGURA 3.1. ARQUITETURAS PARA MONITORAMENTO DE EVENTOS [ZJ93].	32
FIGURA 3.2. ASSOCIAÇÃO ENTRE EVENTOS, REGRAS E OBJETOS REATIVOS NO SENTINEL [CAM93].	37
FIGURA 3.3. CÓDIGO CORRESPONDENTE A UM MÉTODO GERADOR DE EVENTOS ANTES E APÓS O PRE-PROCESSAMENTO.....	37
FIGURA 3.4. HIERARQUIA DE CLASSES DO SISTEMA SENTINEL.	38
FIGURA 3.5. ASSOCIAÇÃO ENTRE OBJETOS NO MECANISMO PROPOSTO.....	40
FIGURA 3.6. DEFINIÇÃO DA CLASSE REATIVA.	41
FIGURA 3.7. DEFINIÇÃO DA CLASSE EVENTO.....	42
FIGURA 3.8. DEFINIÇÃO DA CLASSE PRIMITIVA.....	42
FIGURA 3.9. DEFINIÇÃO DA CLASSE CONJUNÇÃO.....	43
FIGURA 3.10. DEFINIÇÃO DA CLASSE REGRA.....	44
FIGURA 3.11. GRAFO DE EVENTOS.	45
FIGURA 3.12. VISÃO GLOBAL DO MODELO DE EXECUÇÃO.....	46
FIGURA 4.1. DIFERENTES RESPOSTAS PARA A CONSULTA <i>ONDE ESTÁ LOCALIZADA A UNICAMP?</i>	48
FIGURA 4.2. FOTOGRAFIA AÉREA.....	49
FIGURA 4.3. MODELO GEOGRÁFICO IMPLEMENTADO.	52
FIGURA 4.4. CLASSIFICAÇÃO DAS OPERAÇÕES SOBRE OBJETOS GEOGRÁFICOS.	52
FIGURA 4.5. ANÁLISE DE PROXIMIDADE AO REDOR DE UMA REGIÃO.....	54
FIGURA 4.6. ANÁLISE DE PROXIMIDADE AO REDOR DE UM PONTO.	54
FIGURA 4.7. EXEMPLOS DE RESTRIÇÕES DE INTEGRIDADE ESPACIAIS.	56
FIGURA 4.8. VISÃO CONCEITUAL DO ESQUEMA DE CAMADAS.....	57
FIGURA 5.1. PARTIÇÕES DO PLANO CONSIDERANDO UM ÚNICO PONTO DE REPRESENTAÇÃO.....	59
FIGURA 5.2. REPRESENTAÇÃO DO MÉTODO DAS 9-INTERSEÇÕES.....	60
FIGURA 5.3. EXEMPLOS DO RELACIONAMENTO <i>DISJOINT</i>	61
FIGURA 5.4. EXEMPLOS DO RELACIONAMENTO <i>OVERLAP</i>	61
FIGURA 5.5. EXEMPLOS DO RELACIONAMENTO <i>TOUCH</i>	62
FIGURA 5.6. EXEMPLOS DO RELACIONAMENTO <i>IN</i>	62
FIGURA 5.7. EXEMPLOS DO RELACIONAMENTO <i>CROSS</i>	63
FIGURA 5.8. EXEMPLOS DE PREDICADOS TOPOLÓGICOS.	64
FIGURA 5.9. EXEMPLO DE SENTENÇA TOPOLÓGICA.	65
FIGURA 5.10. MAQUINA DE ESTADOS QUE REPRESENTA EVOLUÇÃO CONSISTENTE.	71
FIGURA 5.11. ÁRVORE CORRESPONDENTE AO PASSO 1.....	74
FIGURA 5.12. ÁRVORES CORRESPONDENTES AO PASSO 3.....	75

FIGURA 5.13. REGRAS E-C-A CORRESPONDENTES À MANUTENÇÃO DA RESTRIÇÃO R (FIGURA 5.9).....	77
FIGURA 6.1. ETAPAS PARA A RESOLUÇÃO DO PROBLEMA DAS RESTRIÇÕES TOPOLÓGICAS.....	79
FIGURA A.1. CLASSES E RELACIONAMENTOS CORRESPONDENTES AO MODELO ADOTADO.....	82
FIGURA A.2. ALGORITMO QUE VERIFICA A EXISTÊNCIA DE UM RELACIONAMENTO TOPOLÓGICO <i>DISJOINT</i> ENTRE UM OBJETO GEOGRÁFICO FONTE E UM CONJUNTO DE OBJETOS GEOGRÁFICOS	83
FIGURA A.3. ALGORITMO QUE VERIFICA A INEXISTÊNCIA DE UM RELACIONAMENTO TOPOLÓGICO <i>DISJOINT</i> ENTRE UM DETERMINADO OBJETO E UM CONJUNTO DE OBJETOS.....	83
FIGURA A.4. ALGORITMO QUE VERIFICA A EXISTÊNCIA DE UM RELACIONAMENTO $TopREL_2$ ENTRE UM DETERMINADO OBJETO GEOGRÁFICO E UM CONJUNTO DE OBJETOS GEOGRÁFICOS.....	83
FIGURA A.5. ALGORITMO QUE VERIFICA A INEXISTÊNCIA DE UM RELACIONAMENTO $TopREL_2$ ENTRE UM DETERMINADO OBJETO GEOGRÁFICO E UM CONJUNTO DE OBJETOS GEOGRÁFICOS	84
FIGURA A.6. ALGORITMO QUE VERIFICA A EXISTÊNCIA DE UM RELACIONAMENTO $TopREL$ ENTRE UM PAR DE CONJUNTOS DE OBJETOS GEOGRÁFICOS.....	84
FIGURA C.1. HÍERARQUIA DE CLASSES DE ALGUMAS ENTIDADES DA TELEBRÁS.....	89
FIGURA C.2. EM TODOS OS DUTOS HÁ PELO MENOS UM CABO.....	89
FIGURA C.3. EXISTE PELO MENOS UM POSTE ASSOCIADO A UM CABO AÉREO.....	92
FIGURA C.4. ESTRUTURAS CORRESPONDENTES AO EXEMPLO 2.1.....	96
FIGURA C.5. EXEMPLO DE RESTRIÇÃO INTRA-CLASSE.....	97
FIGURA C.6. EXEMPLO DE RESTRIÇÃO INTER-CLASSE.....	98

I

INTRODUÇÃO E MOTIVAÇÃO

1. INTRODUÇÃO

Esta dissertação mostra a utilização de sistemas de gerenciamento de banco de dados (SGBD) ativos no contexto de aplicações Sistema de Informação Geográfico (SIG). Os resultados apresentados neste trabalho estendem o paradigma dos sistemas de banco de dados ativos para incorporar a manutenção de relacionamentos espaciais na presença de atualizações.

Sistemas de bancos de dados ativos são SGBD que respondem a eventos (gerados interna ou externamente ao sistema) sem a intervenção do usuário. A dimensão ativa é suportada por um mecanismo de *regras E-C-A*, fornecido pelo próprio gerenciador de banco de dados. Regras E-C-A são definidas como: *when E if C then A*, onde **E** é o acontecimento de um evento, **C** é uma condição a ser verificada, e **A** é uma ação a ser executada se a condição é verdadeira. Os bancos de dados ativos são uma solução apropriada para a manutenção de restrições em aplicações convencionais. Estas idéias ainda não foram aplicadas a dados georeferenciados.

Um SIG é um sistema de computação capaz de captar, armazenar, processar, analisar, manipular e mostrar grandes volumes de dados georeferenciados (aqueles identificados de acordo com a sua localização) [Mag91].

A utilização de regras de produção em SIG não é nova [KWB+93, SR93, SRD+91, YS91, AYA+92]. Regras de produção são definidas como cláusulas *if X then Y*, onde **X** é um predicado a ser verificado e **Y** é uma ação a ser executada se o predicado é verdadeiro. No contexto dos SIGs, regras são utilizadas em um enfoque dedutivo: ajudando no processamento de consultas, analisando dados ou derivando relacionamentos entre dados espaciais. Regras e dados são gerenciados por componentes isolados (sistema de gerenciamento de regras e SGBD espaciais), necessitando de módulos intermediários para permitir a sua comunicação. Geralmente, um sistema de gerenciamento de regras é externo, tipicamente um *shell* de sistema especialistas, que também suporta a tomada de decisões espaciais [AD90].

O acoplamento de regras com o SGBD espacial requer vários níveis de modelagem e tradução entre módulos, além do aprendizado, por parte do usuário, de diferentes linguagens (para manipular dados e para utilizar regras). Um exemplo típico deste enfoque é apresentado em [KWB+93], onde ARC/INFO gerencia os dados espaciais e um sistema de regras acoplado é usado para a classificação, refinamento e generalização destes dados.

Outros exemplos de pesquisas que combinam SIG e regras utilizam as seguintes abordagens:

- *shells* de sistemas especialistas (por exemplo, [SR93, SRD+91, LL93]),
- sistemas de suporte a tomada de decisões espaciais (construídos acoplando SGBDs espaciais e sistemas de regras (por exemplo, [YS91, AYA+92])),
- sistemas especialistas para colocar rótulos em mapas [DF92],
- sistemas de consultas baseados em regras [WCY89],
- sistemas dedutivos baseados em Prolog [Web90].

Nenhuma destas propostas é satisfatória do ponto de vista de gerenciamento de dados: o programador da aplicação precisa desenvolver módulos de manutenção de regras e dados e seu acoplamento. Este tipo de solução é conhecida como *baseada em linguagens*, e requer um esforço adicional para integrar a linguagem de programação das regras com o respectivo SGBD.

[SA93, PMP93, AWP93] são trabalhos que tentam diminuir a impedância entre sistemas de regras e SGBDs espaciais. [PMP93] discute a importância da integração do gerenciamento de regras em um SGBD espacial. Em [SA93], as regras são modeladas como uma classe específica (utilizando uma metodologia de projeto orientada a objetos para sistemas geográficos) e implementadas em uma linguagem orientada a objetos. Outra proposta é o uso de SGBDs dedutivos: [AWP93] analisa os relacionamentos topológicos expressos dentro de um banco de dados dedutivo com uma linguagem lógica, mas a ênfase é na especificação de restrições e não na sua manutenção.

A integração e gerenciamento de relacionamentos espaciais em um banco de dados ativos, tem a seguintes vantagens sobre os enfoques existentes na literatura:

- a especificação e preservação de relacionamentos espaciais são gerenciados pelo próprio banco de dados, permitindo assim, a evolução independente dos dados, das regras e da aplicação;
 - o mesmo banco de dados (geográfico) subjacente pode ser utilizado por todas as aplicações que precisam ter a mesma visão do mundo, sem necessidade de re-escrever a aplicação para verificar as restrições;
 - diferentes aplicações podem ter distintas visões do mesmo banco de dados, habilitando e desativando diferentes conjuntos de regras;
 - os relacionamentos espaciais podem ser tratados no mesmo nível (da arquitetura) que os dados que estes referenciam. Isto ajuda nas questões de
-

implementação de baixo nível, como exemplo, deixar que o gerenciador de banco de dados otimize as operações espaciais.

2. OBJETIVOS E ORGANIZAÇÃO DA DISSERTAÇÃO

Esta dissertação mostra como utilizar SGBD ativos na manutenção de restrições topológicas entre dados georeferenciados.

Este trabalho está restrito ao problema das restrições topológicas binárias entre objetos geográficos armazenados em formato vetorial. A solução deste problema pode ser dividido em três etapas: i) especificação da restrição topológica, ii) tradução da restrição em regras e iii) manutenção automática da restrição (baseada nas regras). A figura 1.1 mostra como estas etapas são abordadas neste texto.

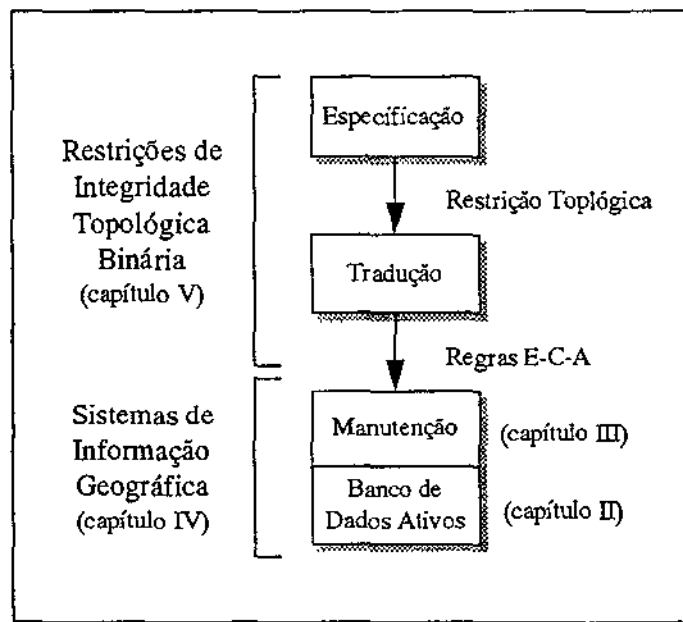


Figura 1.1. Organização da dissertação.

As principais contribuições da dissertação são:

- estudo exaustivo dos relacionamentos topológicos binários;
- desenvolvimento de um protótipo de sistema ativo que incorpora um modelo orientado a objetos espacial;
- definição de algoritmos para verificação de restrições de integridade topológica, com sua implementação no protótipo.

Além deste capítulo introdutório, a dissertação é composta por mais cinco capítulos. O capítulo 2 apresenta uma revisão bibliográfica das principais propostas em bancos de dados ativos, descrevendo também os conceitos utilizados na literatura e a nomenclatura a ser adotada.

O capítulo 3 descreve os principais mecanismos para a manutenção de restrições de integridade em SGBD. São descritos também alguns dos trabalhos na especificação e manutenção de integridade utilizando banco de dados ativos. Ao final o capítulo descreve o mecanismo ativo implementado pela dissertação, para dar suporte às regras E-C-A que manterão a restrições.

O capítulo 4 descreve uma visão geral dos SIG, destacando as suas funcionalidades sob o ponto de vista de sistemas de informação e apresenta o modelo geográfico orientado a objetos adotado nesta dissertação. Este capítulo também relaciona as funcionalidades dos SIG com o suporte de características ativas.

O capítulo 5 apresenta uma descrição dos relacionamentos espaciais (métricos, de direção e topológicos), dando maior ênfase aos relacionamentos topológicos binários. Neste capítulo também é descrita a especificação das restrições de integridade topológicas e a transformação destas em regras E-C-A.

Finalmente, o capítulo 6 contém conclusões e extensões futuras deste trabalho. Informações adicionais sobre a implementação do modelo geográfico, e a descrição da linguagem para expressar restrições são encontradas, respectivamente, nos apêndices A e B. O apêndice C contém exemplos da transformação de restrições topológicas em regras, segundo os algoritmos propostos. Algumas destas restrições foram fornecidas pelo Centro de Pesquisas e Desenvolvimento (CPqD) da Telebrás.

II

BANCOS DE DADOS ATIVOS

1. INTRODUÇÃO

A Figura 2.1 apresenta uma visão da evolução histórica da gerência de dados. Devido à inexistência de tecnologia de armazenamento de dados com acesso “on line”, a memória principal era compartilhada por programas e dados (Figura 2.1.a). Os dados passaram a ser armazenados em arquivos externos devido à evolução dos dispositivos magnéticos de acesso direto. No entanto, os programas ainda continuavam com a função de gerência dos dados (Figura 2.1.b). Os sistemas de gerenciamento de banco de dados (SGBD) surgiram como um módulo independente para gerência de dados, que provê independência de dados e programas: toda a responsabilidade de gerência do banco de dados passa a ser do SGBD, enquanto os programas só tratam do processamento e controle da aplicação (Figura 2.1.c).

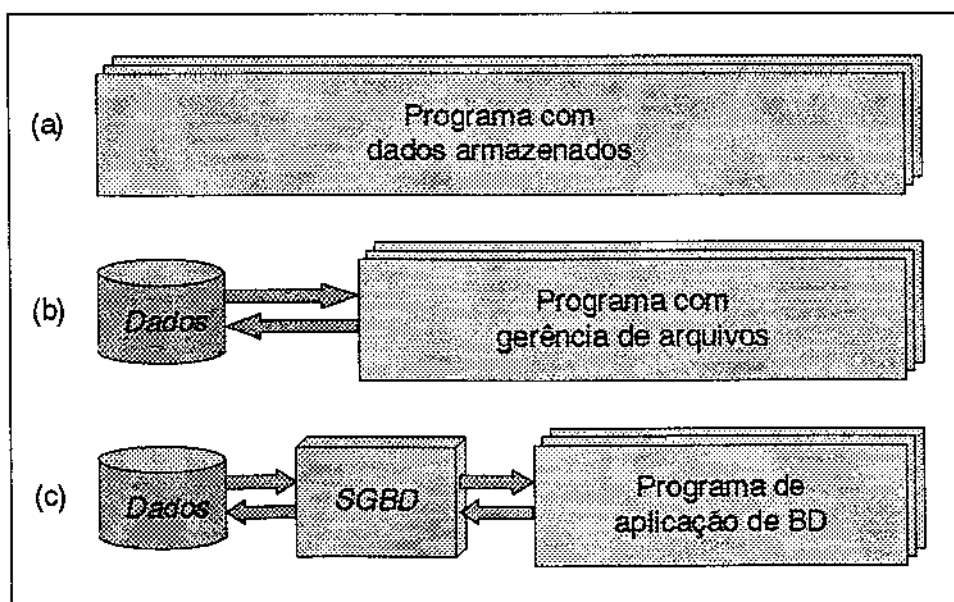


Figura 2.1. Evolução da gerência dos dados [Tan95].

Um banco de dados é *passivo* quando não oferece suporte para o gerenciamento automático de condições definidas sobre o estado do banco de dados em resposta a estímulos externos. Os SGBDs convencionais são passivos, só executando transações quando são explicitamente requisitadas pelo usuário ou aplicação.

Novas aplicações, como por exemplo, controle de processos, redes de distribuição de energia, controle automatizado de fluxo de documentos (workflow), requerem repostas oportunas em situações críticas. Para este tipo de aplicações, determinadas condições são definidas sobre estados do banco de dados os quais devem ser monitorados. Quando estas condições forem satisfeitas ações devem ser executadas.

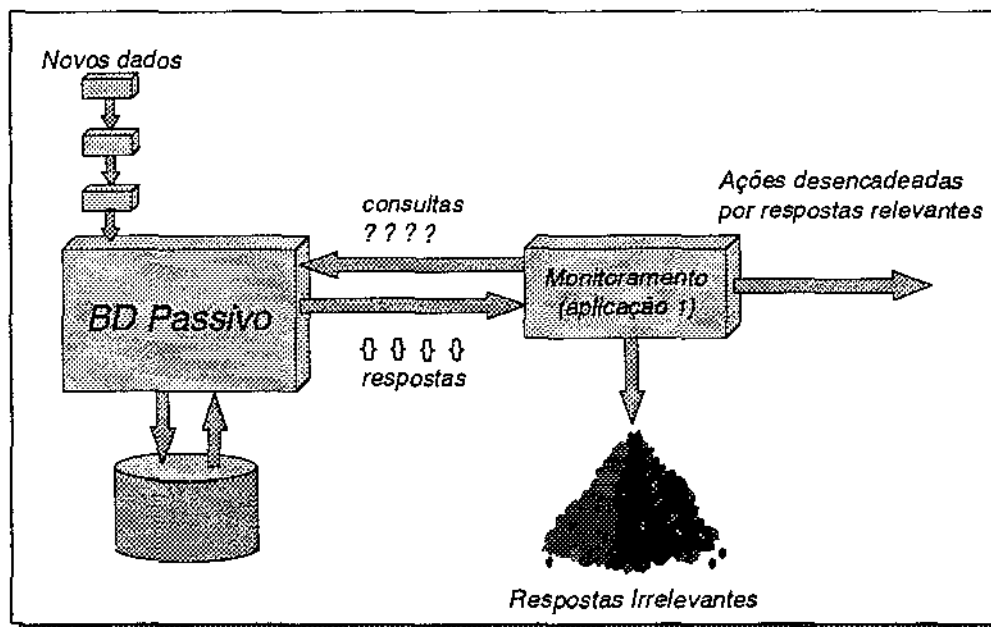


Figura 2.2. Polling [Buch94].

Há diferentes formas de satisfazer as necessidades destas novas aplicações. Segundo [CN90], as mais tradicionais são: i) escrever no próprio programa de aplicação a condição que se deseja testar; ii) avaliar continuamente a condição, ou seja, *polling* (Figura 2.2). Uma desvantagem da primeira alternativa é que a avaliação da condição é responsabilidade do programador. No segundo caso, o problema pode ser a baixa utilização dos recursos se houver uma frequência excessiva dos testes de condição.

Um banco de dados é *ativo* quando *eventos* gerados interna ou externamente ao sistema provocam uma resposta do próprio banco de dados (BD), independente da solicitação do usuário. Neste caso, alguma *ação* é tomada automaticamente, dependendo das *condições* que foram especificadas sobre o estado do banco de dados. Estas situações podem ser especificadas utilizando regras.

A incorporação ao banco de dados, não de apenas dados, mas também regras aplicáveis a estes, tem várias conseqüências. As principais vantagens são:

- não há necessidade de manter as regras dentro dos programas de aplicação,

- existe independência de conhecimento em relação aos programas,
- o banco de dados executa automaticamente as regras quando necessário,
- várias aplicações, com a mesma visão do mundo, podem compartilhar um conjunto de regras.

Esta nova funcionalidade requer uma capacidade adicional do SGBD, a gerência de regras (Figura 2.3).

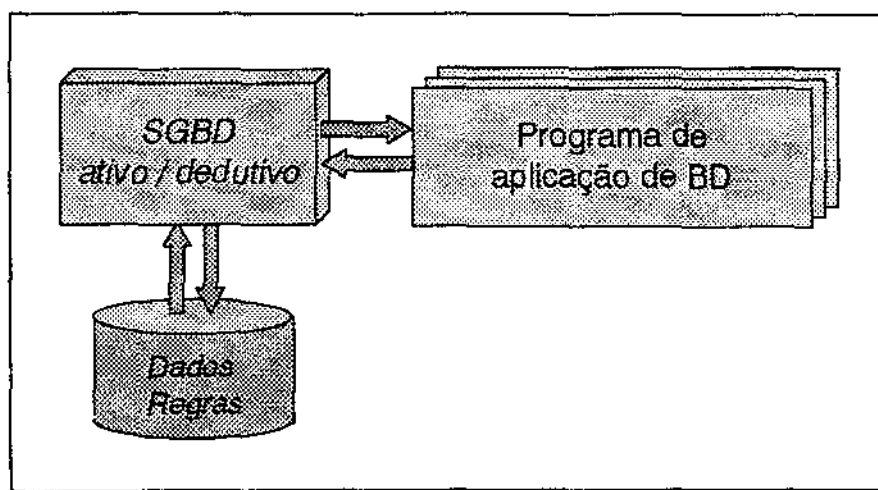


Figura 2.3. Incorporação de regras ao banco de dados.

Bancos de dados *dedutivos* fornecem um mecanismo para derivar dados que não estão explicitamente armazenados no banco de dados (conhecidos como dados virtuais ou dados derivados). São mais poderosos e expressivos que as visões, embora mais problemáticos para serem suportados [LT92].

Não existe uma divisão clara entre os bancos de dados dedutivos e os ativos. A principal diferença está baseada no modelo de execução. No primeiro tipo, geralmente a preocupação é a derivação de informação, e as regras são executadas explicitamente pela aplicação. No segundo, as regras são disparadas como efeito colateral das ações normais do banco de dados [Wid93].

As pesquisas em banco de dados ativos (BDA) podem ser divididas em três categorias: i) definição das regras ou gatilhos, ii) seu correspondente modelo de execução, e iii) sua otimização. Na primeira categoria definem-se quais são os tipos de eventos, condições e ações. Uma questão muito importante é a expressividade da linguagem de especificação. Na segunda categoria discute-se quando devem ser avaliadas as condições, ou como devem ser resolvidos os conflitos (quando mais de uma regra é habilitada ao mesmo tempo). A maioria dos artigos existentes nesta área especifica com detalhe os modelos de execução [HLM88,CBB+89,LT92,VK93,Wid93,Buch94,CFPW94], enquanto alguns outros só esquematizam o sistema implementado [SHH88,Han89,CN90,GJ91,GD92,Wid92,CAM93]. Por último, existe o problema da otimização da execução, levando em consideração estratégias eficientes para avaliação da condição.

Na seção seguinte são descritas algumas das aplicações dos sistemas de gerenciamento de BDA. A seguir são apresentadas as características dos BDA segundo as três categorias descritas anteriormente (definição de regras, modelo de execução e otimização da execução). Logo após, são descritas as características "ativas" dos principais protótipos desenvolvidos na área. Finalmente, são apresentadas algumas ferramentas úteis e as conclusões.

2. USO DOS SISTEMAS ATIVOS

Estes sistemas podem ser usados para aplicações financeiras [CS94] (*commodity trading, portfolio management, currency trading*), aplicações multimídia, controle da produção industrial (*CIM, controle de inventario*), monitoramento (controle de tráfego aéreo, controle de plantas nucleares, controle ambiental), entre outras.

Os sistemas de bancos de dados ativos podem ser classificados em três grupos, segundo seu uso [VK93]:

- ***Suporte automático ao usuário***

- *Notificação.* Em algumas situações o banco de dados precisa da intervenção do usuário. As regras podem ser usadas para detectar automaticamente estas situações e informar ao usuário. Também conhecido como *alerter*.

- *Execução automática de procedimentos.* Ante a ocorrência de um evento, um procedimento pode ser executado mediante o uso do sistema de regras.

- *Provisionamento de valores default.* Uma operação comum nas aplicações é a atribuição de valores *default*, que pode ser feita diretamente com regras.

- ***Funcionalidade do modelo de dados***

- *Manutenção da integridade.* Uma *restrição de integridade*, num ambiente de BD, é um predicado sobre estados do BD que deve ser mantida, para assegurar a consistência dos dados. Num sistema ativo, as condições que violam a integridade são especificadas na parte da condição da regra e a ação corretora especificada na parte correspondente à ação. Este esquema traz duas vantagens importantes: i) o BD suporta especificação e manutenção de restrições, sem depender do código da aplicação, ii) o BD serve para todas as aplicações que têm a mesma visão do mundo. Detalhes sobre manutenção de integridade podem ser encontrados no capítulo 4.

- *Proteção.* O acesso ao banco de dados pode ser controlado usando regras. Com este esquema não só pode ser aceito ou rejeitado o acesso aos dados, como também o sistema pode fornecer dados fictícios nos campos protegidos.

- **Gerenciamento dos recursos.** As regras são usadas dentro do próprio núcleo do banco de dados.
 - **Otimização do armazenamento físico.** Podem ser usadas para ações de *checkpointing*, *clustering*, caminhos de acesso, ou também para adaptar as estruturas de armazenamento segundo estatísticas das consultas.
 - **Gerenciamento de visões.** Visões materializadas complexas podem ser mantidas com regras [SJGP90].

3. REGRAS E GATILHOS

Bancos de dados ativos são via de regra implementados a partir de mecanismos de regras de produção. Regras são descrições de comportamento a serem adotadas por um sistema. As regras estão geralmente baseadas em três componentes: *evento*, *condição* e *ação* (E-C-A) [DBM88]. O evento é um indicador da ocorrência de uma determinada situação. Uma condição é um predicado sobre o estado do banco de dados. Uma ação é um conjunto de operações a ser executado quando um determinado evento ocorre e a sua condição é avaliada como verdadeira. Um evento pode disparar uma ou mais regras.

O primeiro na formalização de sistemas ativos foi Morgenstern. No trabalho descrito em [Mor84], os sistemas ativos são utilizados para manter restrições através das equações de restrições (Constraint Equation, CE). As CEs permitem expressar restrições semânticas que requerem consistência entre muitas relações, de forma similar à manutenção de relações. As CEs constituem uma forma mais concisa que a escrita de procedimentos para expressar e garantir restrições, por possuírem representação declarativa, e uma interpretação executável, sendo compiladas em rotinas para garantir automaticamente as restrições.

De acordo com [SR88], os gatilhos são associações de condições e ações. A execução da ação ocorre quando o banco de dados evolui para um estado que leva o gatilho à condição verdadeira.

3.1 REGRAS E-C-A

A forma atualmente aceita para considerar um BDA é a adoção de regras de produção E-C-A, que basicamente respondem a três perguntas.

- **Evento.** O evento é um indicador da ocorrência de uma determinada situação respondendo a *quando* avaliar. Na Figura 2.4 é apresentada a classificação de eventos em sistemas ativos [Sam94]. Existem basicamente três tipos de eventos primitivos: temporais (às 8:30, 5 minutos após o *user-logout*, toda sexta às 10:00), operações próprias do BD (*insert*, *delete*, *update*, *select*), e explícitos (alta temperatura, *user-login*). Os eventos compostos são formados a partir da aplicação de operadores de composição sobre eventos primitivos e/ou outros eventos compostos.

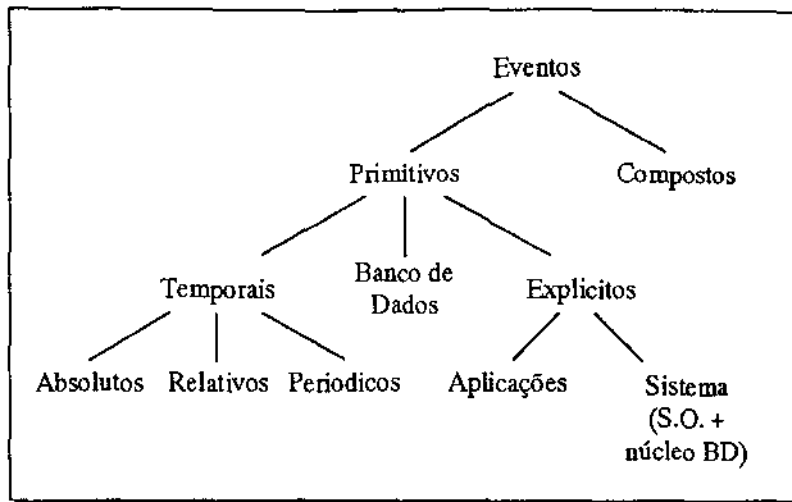


Figura 2.4. Classificação de eventos em sistemas ativos.

- **Condição.** Uma condição é um predicado sobre o estado do banco de dados, respondendo à pergunta: *o que* avaliar. Condições são implementadas por consultas, predicados ($\text{produto.quantidade} < 20$) ou por procedimentos da aplicação (máximo-atingido).

- **Ação.** Uma ação é um conjunto de operações a ser executado quando um determinado evento ocorre e a condição associada é avaliada como verdadeira. A ação representa o *como* responder. Um evento pode disparar uma ou mais regras. As ações típicas são: operações de modificação (*insert*, *delete*, *update*) ou consultas (*select*), comandos do BD (*commit*, *rollback*), ou procedimentos da aplicação (podendo ou não acessar o BD).

Existem dois aspectos importantes no projeto da linguagem de regras de produção: a sintaxe para a criação, modificação, e eliminação de regras; e a semântica do processamento das regras na execução. A sintaxe da maioria das linguagens é similar (baseada na extensão da linguagem de consulta). No entanto, a semântica do processamento varia consideravelmente.

3.2 LINGUAGENS PARA ESPECIFICAR EVENTOS

Um BDA precisa detectar a ocorrência de qualquer evento definido para poder iniciar a ativação das regras. Para que as situações reais possam ser monitoradas, uma linguagem de definição de eventos deve ser empregada permitindo a modelagem de situações complexas. Alguns exemplos deste tipo de linguagens são Ode [GJS92b], Samos [GD93], Compose [GJS92a] e Snoop [CM93], entre outros.

Álgebras de eventos são incorporadas às linguagens de especificação de eventos para permitir a composição de eventos. Os principais operadores encontrados em álgebras de composição como as de Ode, Samos, Compose e Snoop são:

- **Seqüência.** Indica que o evento composto acontece quando os eventos que constituem a seqüência tiverem ocorrido na ordem determinada.

- **Conjunção.** Indica que o evento composto acontece se todos eventos que formam a conjunção ocorrerem, independente da ordem relativa entre eles.

- **Disjunção.** Indica que o evento composto ocorre se pelo menos um dos eventos que formam a disjunção tiver acontecido.

- **Negação.** Indica que o evento composto acontece se os eventos descritos na expressão de negação não tiverem acontecido num determinado intervalo de tempo.

A especificação dos eventos pode ter parâmetros, e estes podem ser referenciados na condição e ação da regra.

3.3 LINGUAGENS PARA ESPECIFICAR CONDIÇÕES

Condições são expressas por fórmulas, às quais é atribuído um valor booleano quando executadas. Uma classificação das possíveis teorias da lógica para linguagens de especificação de condições é descrita em [VK93]. Entre as teorias estão a lógica proposicional, lógica de primeira ordem, e lógica temporal.

As linguagens de consulta são geralmente usadas para a especificação de condições. Neste caso, uma convenção deve ser adotada para a condição ser verdadeira (se a consulta produz uma resposta vazia ou não). Os resultados destas consultas podem ser referenciadas na ação correspondente à regra em questão.

3.4 LINGUAGENS PARA ESPECIFICAR AÇÕES

As linguagens para especificação de ações podem ser divididas em três grupos: linguagens de consulta, linguagens de consulta estendidas, e acesso direto ao banco de dados.

- **Linguagens de consulta.** As ações são especificadas com a mesma linguagem de consulta dos sistemas de bancos de dados, como exemplo SQL. A vantagem é que o acesso ao banco de dados fica sob controle do próprio sistema, e a desvantagem é que limita a funcionalidade. Por exemplo, não podem ser executadas ações externas ao ambiente do banco de dados.

- **Linguagens de consulta estendidas.** Para que as regras possam interagir com o ambiente externo, a linguagem de consulta precisa ser estendida. Assim a linguagem de especificação de ações pode basear-se em duas partes, a linguagem de consulta e a linguagem de comunicação. Esta última é baseada em chamadas a procedimentos convencionais ou em primitivas *send* e *receive*.

- **Acesso direto ao banco de dados.** Uma linguagem algorítmica aumenta a expressividade das ações, quando não podem ser expressas com uma linguagem de consulta estendida. A desvantagem deste tipo de linguagem é a redução das possibilidades de otimização, que é de vital importância para obter uma performance aceitável.

3.5 COMPONENTES

Na Figura 2.5 são mostrados os componentes de um BDA, que adicionam características ativas às funções características dos BD convencionais, como exemplo, controle de concorrência, recuperação de falhas, persistência, otimização de consultas [ABD+92].

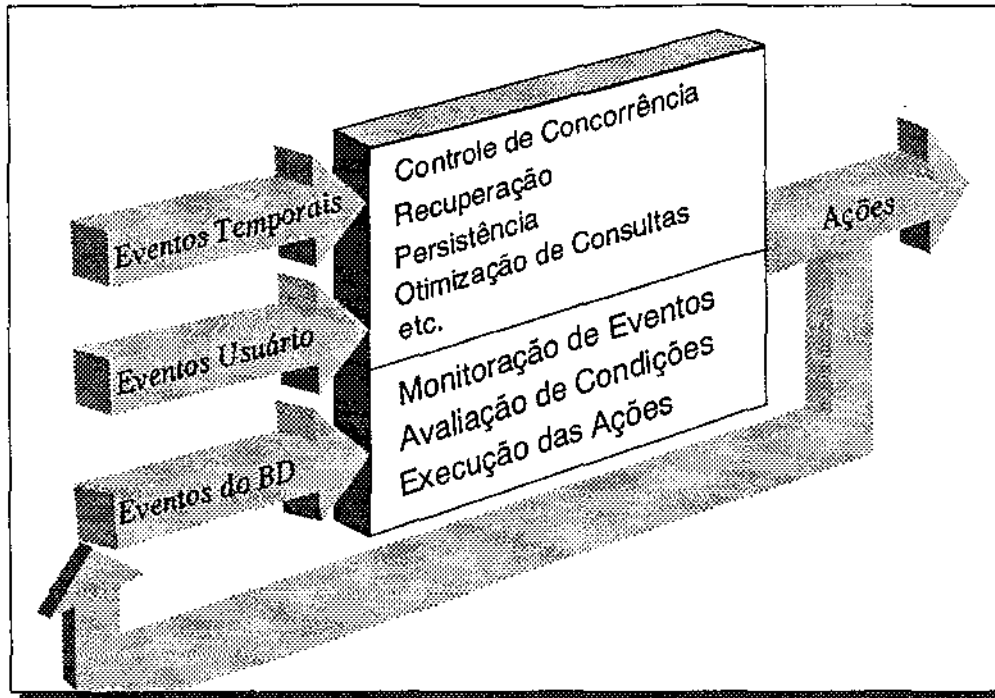


Figura 2.5. Componentes de um Banco de Dados Ativo [Buch94].

São três as componentes básicas para a obtenção de um sistema ativo:

- **Monitoramento de Eventos.** É o módulo encarregado de detectar eventos e ativar as regras que dependam desses eventos.
- **Avaliação da Condição.** Depois que o evento foi detectado o avaliador da condição é responsável pela avaliação eficiente das condições. Aquelas regras cujas condições sejam verdadeiras são passadas para o *executor de ações*.
- **Execução de Ações.** Este componente coordena o sincronismo entre detecção de eventos e execução de ações. As ações podem ser executadas de imediato (antes do fim da transação que disparou a regra), ou depois (numa transação independente). A ligação entre a execução de ações e regra é denominada *modo de acoplamento*.

Dependendo dos *modos de acoplamento* a serem adotados pelo sistema de regras, o gerenciador de transações subjacente deve dar suporte transações aninhadas.

4. MODELOS DE EXECUÇÃO

A inclusão de regras num banco de dados afeta significativamente o modelo de execução. Quando uma regra é selecionada para execução, sua condição é avaliada e sua ação é executada dependendo do resultado da avaliação da condição. A semântica de execução de regras pode ser expressa pelo algoritmo apresentado em [CFPW94], onde é assumida a existência de um conjunto de regras disparadas por eventos que foram detectados até um dado momento do processamento. Este algoritmo é apresentado na Figura 2.6.

Enquanto houver regras disparadas:

1. Selecionar uma regra disparada R
2. Avaliar a condição de R
3. IF a condição de R for verdadeira THEN executar a ação de R

Figura 2.6. Algoritmo simples de processamento de regras [CFPW94].

Este algoritmo simples de execução iterativo não considera: 1) o processamento recursivo de regras, em que a ação de uma regra pode causar um evento que dispara outra regra; 2) a interação entre as transações “ordinárias” do BD e as transações geradas pela ação das regras (modos de acoplamento).

O modelo de transações [HLM88] que é utilizado no HiPAC é usado como referência na literatura. Neste modelo as transações podem ser aninhadas a níveis arbitrários, formando uma árvore de transações com uma transação topo na raiz.

Outras questões devem ser consideradas no modelo de execução, como exemplo, os modos de acoplamento, a frequência com que são processadas as regras, como são resolvidos os conflitos quando mais de uma regra está pronta para executar, e a ativação de outras regras por parte de regras já disparadas. Todas estas questões são tratadas nas próximas seções.

4.1 MODOS DE ACOPLAMENTO

O modo de acoplamento especifica o relacionamento entre o evento disparador e a avaliação da condição (E-C), ou entre a condição e a execução da ação (C-A), definindo quando a execução da regra deve começar ou terminar, e também em que transação deve ser executada. Por uma questão de legibilidade, e sem perda de generalidade, é considerado o relacionamento E-C, executando a ação logo após a avaliação da condição. A notação utilizada para a descrição dos modos de acoplamentos é a seguinte: BOT - begin of transaction, EOT - end of transaction, E - evento, C - condição, e A - ação. A seguir são descritos os modos de acoplamento, apresentados em HiPAC [CBB+89], que são utilizados como referência nas pesquisas da área.

- **Imediato.** A avaliação da condição ocorre imediatamente quando o evento é sinalizado (ou a condição é avaliada como verdadeira), dentro da mesma transação (Figura 2.7.a).

- **Retardado.** A avaliação da condição é adiada até que a transação-"raiz" termine, mas antes do ponto de término (EOT). Este modo é descrito na Figura 2.7.b.

Por exemplo, para a manutenção de consistência sobre um atributo de um objeto, é suficiente o modo imediato; mas sobre vários atributos, de um objeto é preciso o modo retardado.

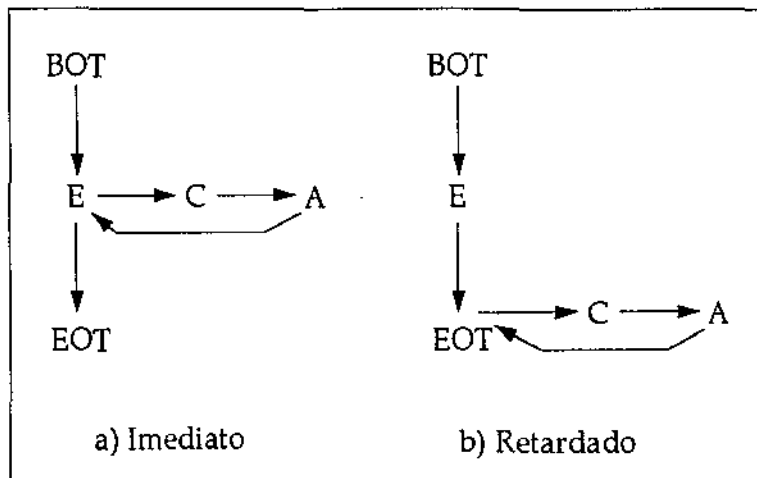


Figura 2.7. Modos de acoplamento.

Por questões de eficiência, algumas vezes é preciso executar uma regra em uma transação separada. Enquanto a regra é executada, a transação que a disparou pode estar sendo atualizada em paralelo. Este tipo de acoplamento é conhecido como modo de acoplamento desacoplado, e pode ser:

- **Desacoplado-independente.** A avaliação da condição é feita em uma transação separada e é realizada independentemente do término ou não da transação-"raiz" (Figura 2.8.a).

- **Desacoplado-dependente.** A avaliação da condição ocorre depois que a transação-"raiz" termina completamente. Se o término (EOT) da transação não ocorre (por causa de um *rollback*), a condição não é avaliada (Figura 2.8.b)

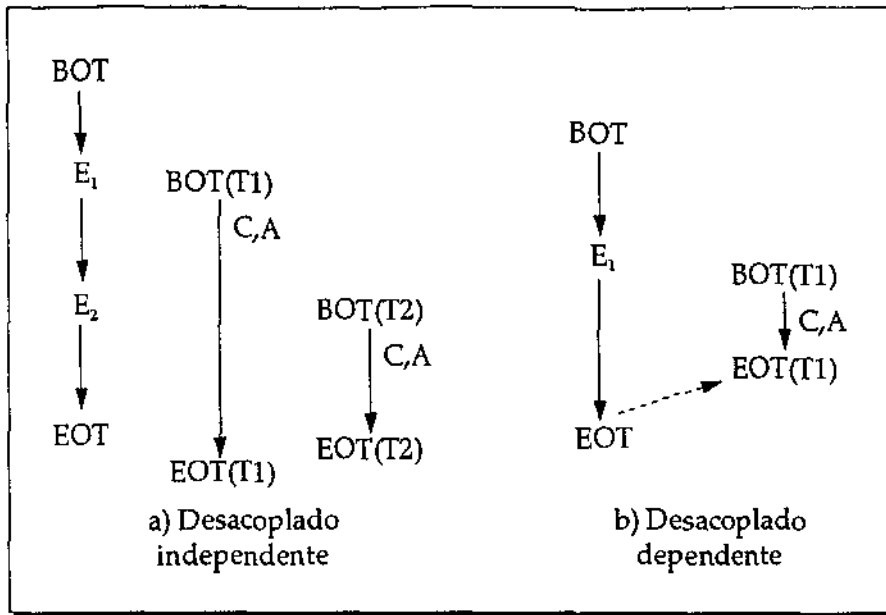


Figura 2.8. Modos de acoplamento desacoplados.

- *Seqüencial-independente.* A avaliação da condição é feita em uma transação separada e é realizada independentemente após o término da transação-"raiz" (Figura 2.9). É útil no caso dos efeitos colaterais que não podem ser desfeitos (*rollback*). Como exemplo, a ação é uma ordem para um robô para cortar um pedaço de metal, a qual não deveria ser executada até que a transação seja completada (figura 2.9).

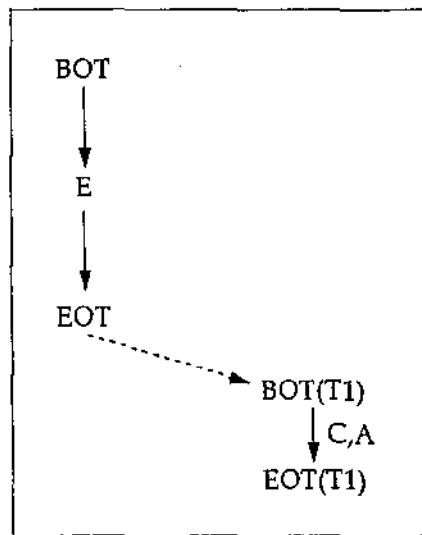


Figura 2.9. Modo de acoplamento seqüencial independente.

4.2 GRANULARIDADE DE ATIVAÇÃO

A granularidade do processamento de regras especifica o intervalo entre verificação das regras a serem disparadas, ou seja, com que frequência é executado o algoritmo

apresentado na figura 2.6. A granularidade pode ser classificada em dois grupos: *set-oriented* e *instance-oriented*. No primeiro grupo existem três opções: 1) sessão do próprio banco de dados (*user-login*), 2) transação (*commit*, *rollback*), 3) operação coletiva ou comando (sentenças SQL). O segundo grupo é em nível de operação individual, como exemplo, *insert*, *update* ou *delete*.

Uma granularidade mais fina leva a maiores possibilidades no uso de regras. Por exemplo, se é em nível de comando, então podem-se definir regras para a interação com o usuário, o que seria impossível com a granularidade de ativação em nível de primitivas.

No caso da manutenção de integridade, é necessário que a integridade seja verificada (ou até corrigida) antes do *commit* da transação. Para isto é necessário ter o controle em nível de comando, para que as regras possam ser executadas imediatamente antes do *commit*.

4.3 RESOLUÇÃO DE CONFLITOS

Múltiplas regras podem estar habilitadas ao mesmo tempo. Estas são conhecidas como o conjunto *prontas para executar*. A resolução de conflitos consiste em um processo de seleção de uma regra a partir deste conjunto. Este processo pode introduzir um grau de não determinismo. Há várias alternativas para resolver este problema:

- *Execução em paralelo*. Todos os gatilhos ou regras são executados em paralelo. Isto só pode ser feito se as regras não interferem entre si.
- *Ordem seqüencial de execução*. Todos os gatilhos ou regras são executados seqüencialmente, a ordem podendo ser aleatória, por prioridade, ou outras.
- *Execução simples de uma regra*. Somente uma regra é escolhida, com as mesmas opções que no caso anterior.

4.4 ATIVAÇÃO EM CASCATA

A execução de uma regra pode também ativar outras regras, o que é conhecido como ativação em cascata. É necessário definir: i) se a execução é suspensa para executar a nova regra; ii) se a execução de uma determinada regra começa quando termina a execução de outra (isto depende dos modos de acoplamento E-C e C-A); iii) se é permitido a recursão, que pode levar a situações de *livelock*.

5. OTIMIZAÇÃO

Existem várias técnicas de otimização para execução de regras, que variam segundo os diferentes objetivos dos sistemas ativos e das arquiteturas subjacentes. As características principais destas estratégias são classificadas em quatro grupos: otimização na execução das consultas, redução na execução de consultas, otimização no armazenamento, e mecanismos de detecção de eventos.

As estratégias do primeiro grupo são as mesmas que as otimizações de ordenação de operações usadas nos sistemas de bancos de dados convencionais e são descritas com detalhe em [Ull82].

No segundo grupo estão os algoritmos para a redução da quantidade de cálculo. Neste caso o objetivo é tentar excluir código redundante de subexpressões comuns e evitar execução de regras ou gatilhos que não são necessários. Há duas estratégias relevantes dentro deste grupo. A primeira é a otimização da avaliação da condição, por exemplo, avaliação incremental das condições. A segunda é a redução do tempo de computação das ações. Isto pode ser feito executando as regras o mais cedo possível, ou deixando-as para depois, quando talvez seja possível otimizar a execução através de abandono de regras supérfluas ou avaliação em conjunto [VK93].

O terceiro grupo consiste em estratégias para a otimização no armazenamento de dados e regras. A indexação dos dados neste caso é feita levando em consideração os predicados das regras, reduzindo assim a quantidade de objetos considerados no momento da avaliação da condição.

É óbvio que a implementação de um detector de eventos eficiente é crucial para ter um bom desempenho num BDA. Existem várias formas de detectar eventos, alguma das quais são descritas a seguir:

- **Centralizada.** Quando um evento é gerado, todas as regras são verificadas. É o mais fácil de implementar, mas também o de pior desempenho.

- **Índices.** Regras podem ser indexadas segundo os eventos que as ativam. Assim, quando o evento ocorre, as regras que podem ser disparadas são encontradas rapidamente.

- **Subscrição.** Associação de regras a objetos geradores de eventos. Quando um evento é gerado, este notifica a todas as regras que subscreveram aquele evento. No Sentinel [CHS92] é usado este mecanismo, mas existe um detector de eventos local a cada regra.

- **Rede de Eventos.** Para cada evento composto é construída uma rede. O sistema trabalha com uma combinação de redes que inclui todas as redes de todos os eventos compostos e primitivos. Um evento pode participar em mais de uma composição, mas na combinação das redes este evento tem uma única representação, e é compartilhado. Uma vantagem do uso destes mecanismo é a diminuição do espaço para representar os eventos associados às regras. Este mecanismo de detecção é implementado em alguns dos protótipos (que serão detalhados a seguir) usando autômatos finitos [GJS92b], redes de Petri [GD94] e grafos [CBB+89, Ber94].

6. ALGUNS PROTÓTIPOS DE SISTEMAS ATIVOS

O gerenciamento de regras e/ou gatilhos pode ser implementado separadamente do sistema de banco de dados, constituindo uma camada adicional que não está integrada ao núcleo do SGBD [BL92, SKdM92]. Outros sistemas de bancos de dados ativos, tanto relacionais como orientado a objetos, foram projetados ou estendidos para tal fim. Ariel [Han89], HiPAC [CBB⁺89], Ode [GJ92], Postgres [SJGP90], Samos [GD92], Starburst [Wid92] e Sentinel [CHS92] são os projetos mais destacados na área. A seguir são descritas as características ativas destes projetos.

6.1 ARIEL [HAN89]

Ariel é um sistema de gerenciamento de banco de dados implementado sobre Exodus [CFM⁺86] com um sistema de regras integrado. As principais questões de projeto são a integração do sistema de regras com o processamento de transações e a eficiência do sistema todo. O resultado é um sistema convencional de gerenciamento de banco de dados relacional estendido com um sistema de regras [Han89].

6.1.1 Regras

A linguagem de especificação de regras de Ariel é uma linguagem projetada para ambientes de banco de dados. A condição de uma regra pode basear-se numa combinação de eventos do banco de dados e/ou em um casamento sobre seqüências de eventos.

A sintaxe das regras é a seguinte:

```
[ priority prioridade ]  
[ on evento ]  
[ if condição ]  
[ then ação ]
```

- *prioridade*: fornece um controle sobre a ordem de execução das regras.
 - *evento*: permite a especificação do evento que vai ativar a regra. Existem dois tipos de eventos: eventos temporais e eventos próprios dos bancos de dados, tais como *append*, *delete*, *replac*e *retrieve*.
 - *condição*: baseia-se em uma combinação de eventos de bancos de dados ou casamento de padrões sobre uma seqüência de eventos. As condições também podem testar condições de transição (baseado no estado prévio e atual).
 - *ação*: especifica as ações a serem executadas quando a regra é disparada. A ação é uma seqüência de comandos do banco de dados.
-

6.1.2 Modelo de execução

As regras podem ser definidas para reagir a simples comandos do banco de dados, tal como *delete* ou *replace*, ou seja, os eventos são detectados a nível de comando. A ação é executada no fim do comando de mais alto nível onde aconteceu o evento. Ariel define um comando composto como um bloco ou lista de comandos. Um comando é chamado de *top-level* se não está aninhado dentro de um comando composto.

A ação é executada uma vez para cada um dos objetos no conjunto *prontos para executar*, e a execução da regra é parte da transação onde a própria regra foi ativada. Se a ação da regra falhar, a transação é abortada.

A seleção da regra a ser executada está baseada em prioridades e tempo de ativação. As regras podem acessar tanto o estado do banco de dados, quanto o conjunto de variáveis do evento que causou sua execução.

6.1.3 Otimização

A avaliação do evento e da condição estão baseadas no algoritmo Treat [HC+90], chamado A-Treat (Ariel-TREAT). Este algoritmo mantém incrementalmente conjuntos de elementos que satisfazem determinadas condições. Quando um evento ativa uma regra, todos os elementos que satisfazem a condição da regra podem ser encontrados no conjunto.

6.2 HiPAC [CBB+89]

O nome HiPAC vem de *High Performance ACtive database system*. Este projeto considera dois problemas no gerenciamento de restrições de dados: a manipulação de restrições temporais em banco de dados, e a redução do *polling* das aplicações que usam regras.

Este projeto trabalha sobre um modelo de dados estendido próprio [DBB+88], e inclui construtores para representar suas regras. Estes construtores utilizam o mecanismo de regras ECA, conseguindo transformar o HiPAC em um sistema de banco de dados orientado a objetos ativo.

6.2.1 Regras

No HiPAC as regras são tratadas como objetos. Existe uma classe de objetos do tipo regra e toda regra é uma instância desta classe. O tratamento de regras como objetos permite que estas se relacionem com outros objetos e possuam atributos. Além disso, podem ser criadas, modificadas ou excluídas da mesma forma que outros objetos [DBM88].

A sintaxe das regras é a seguinte:

```
identificador-da_regra
event evento
condition:
    coupling: modo
    query: consulta
```

action:

coupling: *modo*

operation: *operação*

[**timing-constraints** *lista-de-restrições-temporais*]

[**contingency-plans** *exceções*]

- *evento*: O evento é uma instância que possui um identificador e uma lista de argumentos formais tipados. Os eventos podem ser: operações sobre o banco de dados; temporais (por exemplo, o evento ocorre sempre às cinco da tarde); abstratos (eventos gerados pelas aplicações); compostos (por exemplo, é preciso que ocorram dois eventos em seqüência).

- *condição*: A condição de uma regra é um objeto. A condição está baseada na lógica de Horn. Sua estrutura é descrita por duas funções: modo de acoplamento e uma coleção de consultas. As consultas que formam parte da condição de uma regra devem ser todas satisfeitas para que a condição seja verdadeira.

- *ação*: É um objeto complexo. Sua estrutura é definida por duas funções: um modo de acoplamento (similar à definição da condição) e uma operação (que pode ser por exemplo uma mensagem para algum processo ou algum programa escrito na linguagem de manipulação de dados do sistema).

- *lista-de-restrições-temporais*: Corresponde a *deadlines*, prioridades, urgências ou funções de valores. Não são propriedades exclusivas de regras, mas podem ser acopladas a qualquer tarefa no sistema HiPAC.

- *exceções*: São ações alternativas que podem ser invocadas sempre que a ação especificada numa regra não possa ser executada.

As operações sobre os objetos regras são: *create* (cria uma regra), *delete* (elimina uma regra), *enable* (ativa uma regra para que possa ser usada), *disable* (desativa uma regra) e *fire* (faz com que a regra seja executada).

6.2.2 Modelo de execução

Um dos objetivos no projeto HiPAC é fornecer um bom tempo de resposta aos eventos críticos. Desta forma, é importante avaliar a condição logo após que aconteça o evento, e executar a seguir a ação. Para cumprir estes objetivos, as regras foram estendidas com os modos de acoplamento já descritos.

Se um evento ativa mais de uma regra, então para cada par condição-ação é criada uma sub-transação. Uma ação é executada uma vez para cada um dos objetos no *conjunto prontos para executar*. Estes objetos são passados para a ação por meio dos argumentos. As variáveis livres são ligadas ao conjunto de objetos que satisfaçam ao evento e à condição, sendo a ação executada uma vez para o conjunto.

6.2.3 Otimização

Como um dos objetivos de HiPAC é o desempenho, foram identificadas várias técnicas para a avaliação de condições complexas. Algumas são: otimização de condições múltiplas, gerenciamento de dados derivados, e avaliação incremental da condição. É usada uma estrutura de rede tipo Rete [For82] para o processamento das condições (indexação de dados).

Para otimizar a detecção de eventos é utilizado um mecanismo de rede que utiliza um grafo orientado acíclico, chamado *signal graph*.

6.3 ODE [GJ91]

O banco de dados orientado a objetos Ode suporta o paradigma de objetos de C++. A interface com o usuário é a linguagem O++, que estende C++ fornecendo facilidades para a criação de objetos persistentes.

6.3.1 Restrições e Gatilhos

Ode fornece dois tipos de facilidades de regras: *restrições* para a manutenção da integridade do banco de dados, e *gatilhos* para a execução automática de ações dependendo do estado do banco de dados.

As restrições são usadas para manter a noção de consistência, o que é geralmente expresso com o sistema de tipos. Estas restrições, que são condições, são associadas às definições das classes, como é mostrado no exemplo seguinte. As restrições podem ser herdadas como qualquer outra propriedade dos objetos. Se uma restrição é violada e não é corrigida, a transação é abortada.

```
class exemplo{
...
public:
...
[soft] constraints:
    condição : ação
}
```

O teste da condição pode ser feito imediatamente após o objeto ser acessado (conhecido como *hard*), ou retardado (chamado *soft*). Este último tipo permite violações temporais de restrições que são corrigidas com ações antes que o commit da transação seja executado.

Os gatilhos, como as restrições, monitoram o banco de dados verificando algumas condições, exceto que estas últimas não representam violações de consistência. Um gatilho é especificado na definição da classe e consiste de duas partes: o predicado do evento e a ação. Os gatilhos só podem ser aplicáveis àqueles objetos especificados.

```
trigger:
[perpetual] nome-do-trigger (params) :
    [within expressão ?] condição => ação [: ação-do-time-out]
```

Os gatilhos podem ser *once-only* (desativados logo após a primeira execução, usados por *default*) ou *perpetual* (reativados automaticamente depois de cada execução). Gatilhos podem ter parâmetros, e podem também ser ativados várias vezes com distintos valores como parâmetros. A *ação* associada a um gatilho é executada quando a *condição* (e *expressão*) é verdadeira, sempre e quando o gatilho esteja ativado. Após o *time-out* especificado, a *ação-do-time-out* é executada.

6.3.2 Modelo de Execução

Os gatilhos são associados a objetos, e são ativados explicitamente depois que o objeto foi criado. A ação é executada em uma sub-transação separada, dependendo do *commit* da transação original (desacoplado-dependente). A palavra chave *independent* faz com que a execução da sub-transação seja executada de forma independente (desacoplado-independente). Outros modos de acoplamento podem ser imediato ou retardado.

6.4 POSTGRES [SJGP90]

É uma extensão de um sistema de gerenciamento de bancos de dados relacional que inclui um sistema de gatilhos de propósito geral. O sistema de gatilhos pretende ser usado como uma linguagem para implementação de visões, visões materializadas, visões parciais, e procedimentos. O projeto fundamental do sistema de regras do Postgres é descrito em [SR86, SHH88, SJGP90, SHP89].

6.4.1 Regras

A primeira versão do Postgres usa um paradigma de ativação de regras, onde cada comando Postquel pode ter associado os modificadores: *always*, *refuse* ou *one-time*. A segunda versão do sistema de regras usa um enfoque de sistemas de produção mais tradicional.

```
define rule nome-da-regra [ as exception to nome-da-regra ]
on evento to objeto [[ from clausula ] where clausula ]
then do [ instead ] ação
```

O *evento* pode ser: *retrieve*, *replace*, *delete*, *append*, *new* e *old*. *New* é usado com *replace* ou *append*, e *old* quando usa-se *delete* ou *replace*. A *clausula where* é igual às condições das consultas padrão dos sistemas de banco de dados (fórmulas em lógica de Horn). A *ação* é uma coleção de comandos Postquel mais as variáveis predefinidas *new* e *current*. Estas variáveis contêm os valores do objeto a modificar.

6.4.2 Modelo de Execução

O modelo de execução do sistema de regras de Postgres é definido em nível de tupla. Quando uma tupla é acessada, atualizada, inserida ou eliminada, existe uma tupla *current* (para os casos de *retrieve*, *replace* e *delete*), e uma tupla *new* (no caso de *replace* e *append*). Se o evento e a condição especificada são verdadeiros para a tupla atual, então a ação é executada. A granularidade é a nível de operação.

Postgres não permite recursão, oferecendo um mecanismo de prevenção de ciclos (usando uma estrutura de grafos), verificando que uma regra não dispare a ela mesma diretamente ou por transição.

6.4.3 Otimização

São usadas duas diferentes estratégias de implementação para a otimização da execução das regras: execução em nível de tupla e re-escrita da consulta. A implementação em nível de tupla mantém trancas de eventos (*event locks*) nas tuplas. Estas trancas são colocadas em campos apropriados em todas as tuplas envolvidas na condição e segundo a especificação do evento da regra. Quando um evento acontece em um campo marcado com um *event lock*, a condição é testada. Se for verdadeira, a ação é executada. No processamento da consulta, pode-se saber logo quais são as regras a serem executadas.

A re-escrita é feita entre as etapas de *parsing* e otimização. A componente que re-escreve a consulta compila os comandos junto com as regras que estão ligadas aos comandos, evitando assim buscas dinâmicas de regras.

Dependendo da regra, uma das implementações de otimização é escolhida. Maiores detalhes podem ser encontrados em [SJGP90].

6.5 SAMOS [GD92]

Fornecer as mesmas propriedades que todos os gerenciadores de banco de dados orientados a objetos (SGBDOO) como herança, tipos e operações definíveis pelo usuário, encapsulamento, dentre outros. O protótipo foi implementado baseado no SGBDOO comercial ObjectStore. Samos fornece, como Snoop [CM91] e Compose [GJ92], uma linguagem de eventos que inclui vários construtores para a especificação de eventos. Os eventos podem ser divididos em duas categorias: eventos primitivos (eventos de métodos, de valor, temporais ou abstratos) ou compostos (usando disjunção, conjunção, seqüência, vezes ou negação).

6.5.1 Regras

As regras são representadas como objetos, o que permite maior flexibilidade, podendo ser classificadas segundo seus relacionamentos com as classes (ou objetos). Existem dois tipos de relações entre as regras e as classes. Regras *internas* à classe, que permitem operar ou acessar os valores dos objetos, ou *externas*. As internas formam parte da definição da classe, e estão encapsuladas dentro das instâncias. Condições e ações das regras internas podem operar diretamente com os valores. As regras externas à classe podem ser definidas por qualquer usuário ou aplicação independente da definição da classe. Na definição da classe especifica-se quando precisa ser avaliada a condição, ou quando a ação deve ser executada, segundo os modos de acoplamento imediato, retardado, ou desacoplado.

6.5.2 Modelo de Execução

A avaliação da condição e a execução da ação são implementadas como sub-transações. A ordem de execução das regras disparadas ao mesmo tempo usa um mecanismo de prioridades.

O projeto Samos estende a arquitetura de um sistema de banco de dados (passivo) orientado a objetos (ObjectStore) com novas componentes, tais como: *analizador*, *gerenciador de regras e eventos*, *detector de eventos*, e *executor das ações*. O *analizador* é responsável pela passagem das regras e eventos ao gerenciador correspondente (estes manipulam a base de regras e a história dos eventos, respectivamente). O *detector de eventos* precisa manter a estrutura de dados necessária para a detecção dos eventos. Logo após o evento ser detectado, é inserido no registro de eventos. Baseado nesse registro, o *gerenciador de regras* determina quais regras precisam ser executadas, e o *executor* é encarregado de avaliar as condições e executar as ações.

6.5.3 Otimização

A implementação de Samos está construída sobre ObjectStore como uma caixa preta, havendo portanto uma perda de desempenho.

É obvio que a implementação de um detector de eventos eficiente é crucial para ter uma boa performance num sistema de bancos de dados ativos. As Redes de Petri foram usadas para modelagem e detecção de eventos. Para cada construtor há um padrão de Rede de Petri. O sistema trabalha com uma combinação de redes de Petri que inclui todas as redes de todos os eventos compostos. Um evento pode participar em mais de uma composição, enquanto na combinação das Redes só existe um estado para cada evento. A vantagem do uso destas regras é que os eventos compostos podem ser detectados passo a passo, dada a ocorrência de um evento primitivo, e não é preciso inspecionar um grande número de eventos primários armazenados no registro de eventos [GD94].

6.6 STARBURST [WF90]

O Starburst é um sistema de gerenciamento de banco de dados relacional estendido. Há duas formas de extensão, conhecidos como métodos de armazenamento e *attachments*. Os métodos de armazenamento provêm métodos alternativos para implementação de tabelas. Os *attachments* provêm caminhos de acesso, restrições de integridade e extensões de gatilhos.

6.6.1 Regras

A sintaxe da linguagem para expressar regras em Starburst é descrita a seguir:

```
create rule nome on tabela  
when operações  
[if condição  
then ação  
[precedes lista-de-regras ] [follows lista-de-regras ]
```

As operações são uma ou mais dentre *inserted*, *deleted*, ou *updated(c₁,...,c_n)*, onde *c₁,...,c_n* são atributos de relações. A *condição* é um predicado SQL sobre o banco de dados. A *ação* é uma seqüência de operações do banco de dados, incluindo comandos de manipulação SQL, comandos de definição, e o comando *rollback*. Os comandos podem ser também *alter*, *drop*, *deactivate*, e *activate* sobre as regras. As seções *precedes* e *follows* são usadas para ordenar parcialmente o conjunto de regras [Wid92].

6.6.2 Mecanismo de Execução

As regras são processadas logo após a finalização das transações, sendo também possível adicionar outros pontos de processamento dentro das transações.

A semântica está baseada em transições, que são mudanças de estado do banco de dados que resultam da execução de comandos SQL. A execução da transação é a primeira transição relevante, e algumas regras podem ser disparadas por essa transação. Quando as ações de uma regra são executadas, outras transições são criadas. Estas podem disparar regras adicionais, ou as mesmas regras várias vezes.

As condições e ações fazem referência ao estado atual do banco de dados, através de operações de seleção de SQL. Além disso, as condições e ações das regras fazem referência a tabelas de transição, que são tabelas lógicas que refletem as mudanças que aconteceram durante a transição do disparo das regras.

O sistema de regras inclui mecanismos de controle de concorrência que asseguram a consistência com respeito aos dados e regras, e na ordenação das regras. Também inclui mecanismos para a recuperação de falhas (quando *rollback* é executado) das estruturas de dados do próprio sistema de regras.

6.7 SENTINEL [CAM93]

Este projeto integra as regras de tipo E-C-A, introduzidas em HiPAC [CBB+89], a um sistema de banco de dados orientado a objeto, estendendo também a linguagem de especificação de eventos. Os objetivos são: usar o sistema resultante para monitoramento do próprio banco de dados, fornecer suporte cooperativo na resolução de problemas, e suportar SGBDs ativos multimídia para aplicações científicas [CHS92].

6.7.1 Regras

Os objetos C++ convencionais foram estendidos com uma *interface de eventos*. Com isto, é possível gerar eventos quando os métodos são invocados, e propagá-los a outros objetos. Os eventos podem ser gerados antes ou depois da execução do método. As classes *reativas* são aquelas que, além da definição tradicional, têm a especificação da interface de eventos. Os eventos são definidos pelo usuário na própria definição da classe.

A especificação de eventos está baseada na linguagem Snoop [CM91]. Esta linguagem propõe uma hierarquia de eventos constituída por eventos primitivos e compostos. São definidos operadores para eventos, como seqüência, disjunção, etc. A noção de parâmetros

é usada para computar os parâmetros dos eventos complexos. A especificação de eventos e regras pode ser feita em qualquer uma das classes. São suportados os modos de acoplamento imediato ou retardado.

As regras, que são do tipo ECA, são criadas, modificadas e eliminadas da mesma forma que os outros objetos, tendo assim um tratamento uniforme. As regras são também entidades separadas que existem independentemente de outros objetos no sistema. Cada regra tem: uma identificação (para poder associá-la a outros objetos), uma associação a um objeto de tipo evento, e dois métodos públicos: condição e ação. As regras são associadas a outros objetos utilizando o mecanismo de subscrição [CAM93], que é descrito a seguir.

6.7.2 Modelo de Execução

O sistema Sentinel é desenvolvido usando o gerenciador de banco de dados orientado a objetos Zeitgest. Para incorporar as regras no sistema foram incluídas as classes: Reativa, Notificável, Evento, Regra. A hierarquia tem a classe persistente *zg-pos* (predefinida no sistema), como superclasse de Reativa e Notificável, e as classes Evento e Regra são subclasses desta última. Assim, Eventos e Regras podem receber eventos propagados pelos objetos reativos. Para associar Regras a objetos é introduzido o mecanismo de *subscrição*. Isto permite aos objetos notificáveis (neste caso as Regras) subscrever dinamicamente aos eventos gerados por objetos reativos.

6.7.3 Otimização

O esquema da subscrição tem a vantagem que uma mesma regra pode ser aplicada a objetos de tipos diferentes, o que é mais eficiente que definir a mesma regra múltiplas vezes para cada tipo de objeto. Outra vantagem deste esquema é que as regras que estão subscritas a um objeto reativo são testadas só quando este objeto gera eventos. Isto tem um ganho em desempenho em relação ao esquema centralizado, onde todas as regras definidas no sistema são testadas quando um evento é gerado.

7. ALGUMAS FERRAMENTAS ÚTEIS

As implementações de bancos de dados ativos devem incluir algumas ferramentas para dar suporte à implementação de aplicações. Algumas destas são:

- **Ferramentas para planeamento.** Podem ser utilizadas para tratar o problema de disparo infinito de regras devido à ocorrência de ciclos. Para a detecção de ciclos podem ser usados: i) grafo de execução, no qual os nós representam os estados e os arcos as regras; ii) grafo de disparos, no qual os nós representam as regras, e os arcos representam quais regras as disparam. Caminhos dentro deste grafo representam possíveis seqüências de execução.

- **Detecção de inconsistências.** Útil para ajudar na especificação de eventos e dos modos de acoplamento. É difícil para o usuário especificar corretamente, por causa da complexidade dos eventos.

- *Browser de Regras.* Necessário para agrupar e consultar regras baseadas em vários critérios. Por exemplo, quais eventos disparam que regras, todas as regras de uma classe, todas as regras que podem estar envolvidas em um ciclo, etc.
- *Simulador de Regras.* Útil para visualizar os efeitos de mudanças no BD usando as regras definidas. Serve também para identificar possíveis problemas com algumas regras.
- *Event Log e Rule Tracer.* São ferramentas utilizadas no controle da invocação e execução das regras.

8. VISÃO GERAL

Este capítulo discutiu as vantagens dos bancos de dados ativos, não apenas para execução automática de tarefas internas do sistema, mas também para atender à demanda de novas aplicações de banco de dados.

Também foram discutidas as principais características dos bancos de dados ativos utilizando como marco para sua descrição as seguintes categorias: a definição das regras, o modelo de execução e a otimização da execução. A partir destes itens, foram descritos os principais protótipos desenvolvidos na área. A tabela 2.1 apresenta um resumo das características destes protótipos.

O uso deste tipo de tecnologia, além de ser útil para implementar ou estender funções do próprio banco de dados, apresenta várias outras vantagens, tais como:

- a independência do conhecimento em relação aos programas de aplicação
- a manutenção das regras não precisa ser realizada dentro dos programas
- a preservação da modularidade
- a execução automática das regras pelo banco de dados quando necessário
- o compartilhamento de um conjunto de regras por diferentes aplicações que possuem a mesma visão do mundo.

A tecnologia de banco de dados ativos é ainda recente existindo algumas áreas não pesquisadas até agora, tais como: i) questões de eficiência tanto no processamento de regras quanto na detecção de eventos [SKD95]; ii) ferramentas de análise de conflitos, de especificação, e para desenvolvimento [Buch94, SKD95]; iii) problemas relacionados ao processamento paralelo e distribuído de regras [DHW95].

	Ariel	HiPAC	Ode	Postgres	Samos	Starburst	Sentinel
Modelo	Relacional estendido (Exodus)	Orientado a Objeto	O++ (C++ persist.)	Relacional estendido	Orientado a Objeto (Object-Store)	Relacional estendido	Orientado a Objeto (Zeitgest)
Especificação das Regras E-C-A							
Condição	Subconj. de Postquel	Subconj. das consultas	Expressão C++	Postquel	Linguagem de consulta	SQL	C++ / Zeitgest
Evento	append/ delete/ replace/ temporais	BD / relógio / notificações externas	update	retrieve/ replace/ delete/ append/ new / old	BD / temporais/ métodos / abstratos/ transações	insert / update / delete	BD / transações / temporais / métodos
Ação	Subconj. de Postquel	DML/ Procedim. Externos	O++	Postquel	DML	DML	C++ / Zeitgest
Regras							
Herança	Não	-	Sim	Não	Sim	Não	Sim
Eventos							
Mecanismo de Detecção	baseado em TREAT [HC+ 90]	<i>signal graphs</i>	autômato de estado finitos	Índices	Redes de Petri	Índices (<i>Rule catalogs</i>)	Subscrição (Detecção local à regra)
Composição	Não	Sim	Sim	Não	Sim	Não	Sim
Condição							
Otimização	Re-escrita da consulta	Avaliação incremental / Sub-expressões comuns / Rete [For82]	duas avaliações	re-escrita consulta / a nível de tupla	Depende de Object-Store	-	Depende de Zeitgest
Modelo de Execução							
Resolução de conflitos	prioridade	concorrência	- (concorr.)	-	prioridade	prioridade	concorrência
Modos Acoplamento	Imediato	Imediato/Retardado / Separado (dependen/indepen.)	Imediato / Retardado	Imediato	Imediato / Retardado-independ.	Retardado	Imediato / Retardado

Tabela 2.1. Resumo das características dos principais protótipos.

III MANUTENÇÃO DE RESTRIÇÕES EM UM MODELO ATIVO ORIENTADO A OBJETOS

1. RESTRIÇÕES DE INTEGRIDADE

Restrições de integridade expressam estados admissíveis de um banco de dados. Estas expressões podem ser classificadas em: dependente de estado, dependente da transição de estado e dependente da história do banco de dados [ZJ93]. Aquelas dependentes de estado são expressões booleanas sobre as propriedades dos objetos armazenados. Restrições de transição de estados consideram os estados antigo e novo, geralmente usando as palavras chave *old* e *new*. Expressões dependentes da história do banco de dados usam lógica temporal para expressar condições sobre seqüência de estados do banco de dados.

Além desta classificação, as restrições podem ser classificadas segundo outras características [ZJ93]:

- *Referente a quando deve ser satisfeita.* Se a restrição deve ser satisfeita após cada operação atômica ou se necessita de uma seqüência de operações atômicas (transação) para sua verificação. Restrições *diferidas* devem ser satisfeitas só quando o objeto alterado for consultado após uma atualização. Restrições *dependentes da operação* são aquelas que devem ser satisfeitas no momento da execução de uma operação específica do banco de dados.

- *Referente às condições de manutenção.* Restrições podem ser auto corretoras (quando estabelecem uma ação definida pelo usuário, que re-estabelece o estado consistente após a violação) ou preventivas. Neste segundo caso, podem ser fortes ou fracas. Restrições fortes devem ser satisfeitas por todas as transações. Se são violadas, a integridade do banco de dados é violada e a transação é abortada. Restrições fracas podem ser violadas em situações especiais, sem abortar (às vezes basta notificar o usuário).

- *Referente ao conjunto de objetos envolvidos.* As restrições podem fazer referência a um objeto único, a alguns objetos de uma classe (intra-objeto), a todos os objetos de uma classe (intra-classe), a objetos de diferentes classes (inter-classe) ou a todo o banco de dados (global). Maiores detalhes sobre esta classificação são descritos em [And92].

- *Referente à sua dependência com respeito ao tempo.* As restrições podem ser sempre válidas ou válidas temporariamente.

1.1 REPRESENTAÇÃO DE RESTRIÇÕES DE INTEGRIDADE EM SGBDOO

Esta dissertação considera o uso de SGBD orientados a objetos ativos. Como não existe modelo padrão para orientação a objetos, a dissertação adota o modelo baseado em classes de [Bee89]. Um objeto é uma instância de uma classe e é caracterizado por seu estado (conjunto de valores de atributos), e por seu comportamento (conjunto de métodos que podem ser aplicados ao objeto). Um objeto o pode ser construído utilizando outros objetos o_1, \dots, o_n , onde o é chamado complexo e o_1, \dots, o_n são chamados de componentes de o . Um objeto não complexo é chamado simples. As classes podem estruturar-se em hierarquias de herança. Os objetos comunicam-se com outros objetos mediante métodos.

Nos Sistemas Gerenciadores de Banco de Dados Orientados a Objeto (SGBDOO) comerciais não existem formas explícitas para suportar restrições. O usuário precisa codificar as restrições dentro dos métodos e das transações que alteram o banco de dados. Isto não é uma boa solução porque:

- O programador dos métodos deve conhecer e compreender as restrições definidas.

- As restrições e o conhecimento representado por elas ficam escondidos na implementação. Conseqüentemente, as restrições não podem ser consultadas com uma linguagem de consulta padrão e também não podem ser utilizadas por outros componentes do SGBD, como o processador de consultas ou o gerenciador de transações.

- A codificação manual das restrições é ineficiente e propensa a erros.

Há diferentes possibilidades de representar restrições de integridade no modelo orientado a objetos:

- Codificá-las manualmente em todos os métodos e transações que alteram os objetos. Já foram discutidas anteriormente as desvantagens desta alternativa.

- Modelá-las como objetos. É apropriado para restrições inter-classe, mas não é natural para restrições intra-classe (dado que é importante manter a restrição junto com a classe para a qual foi definida).

- Codificá-las em uma extensão da definição de uma classe. É apropriado para restrições do tipo intra-classe.
- Especificá-las utilizando um sistema geral de regras. A desvantagem deste esquema está relacionada ao fato de que o usuário deve aprender uma nova linguagem de regras para expressar as restrições de integridade de seu interesse.

1.2 REQUISITOS DA LINGUAGEM DE ESPECIFICAÇÃO

O usuário de um sistema de banco de dados deve conhecer a linguagem de definição de dados (DDL) e de manipulação de dados (DML) que inclui consultas e atualizações. Estas linguagens contêm todos os componentes necessários para a especificação de restrições. Na especificação de restrições há a necessidade de utilizar uma linguagem declarativa para expressar a condição da restrição; por conseguinte, a linguagem de consulta é um boa candidata. Outro componente na definição de restrições é a ação, que é executada quando a condição for verdadeira e que podem muitas vezes ser descritas através da DML. Ao contrário dos bancos de dados relacionais, os SGBDOO geralmente incluem uma DML suficiente para especificar ações. Com isto, é possível estabelecer a condição e a ação de uma restrição sem ter que forçar o usuário a aprender uma nova linguagem.

Este enfoque também facilita o trabalho de quem implementa o SGBD, porque pode reutilizar a tecnologia já existente no gerenciador, como por exemplo, o otimizador de consultas, as facilidades de processamento da linguagem, entre outras.

1.3 TIPOS DE MONITORAMENTO

Qualquer componente de um SGBD responsável pela preservação da consistência precisa monitorar os eventos que acontecem nele. Segundo [ZJ93] há três diferentes arquiteturas que podem ser usadas (figura 3.1):

a. *Monitoramento de integridade dependente da aplicação.* Cabe ao programador da aplicação ativar os componentes encarregados da consistência, inclusive o componente responsável pela integridade. Este esquema não requer nenhuma alteração no gerenciador de banco de dados. Uma desvantagem desta alternativa é que a total responsabilidade da integridade fica por parte da codificação do programador.

b. *Garantia de consistência através de ações/transações.* Neste caso, há dois tipos de tratamento. No primeiro, *top down*, os comandos de manipulação de dados das aplicações são modificados pelo componente responsável pela integridade de forma a transformar-se em ações que preservem a consistência. Isto é conhecido como *query modification* e é usado em [SJGP90]. No segundo, *bottom up*, o componente de integridade oferece ao programador da aplicação um conjunto de transações que asseguram a consistência (ao contrário das ações atômicas fornecidas habitualmente por vários SGBD). Esta idéia está baseada na proposta de [BR84] sobre o modelo relacional, mas se aplica também ao modelo orientado a objetos.

c. *Monitoramento da integridade dependente do SGBD.* Neste caso, o componente de integridade está dentro do próprio SGBD e utiliza informação interna ao gerenciador. Esta arquitetura requer alterações no gerenciador. Esta idéia também permite o uso de conhecimento semântico das restrições em outros componentes do sistema, e conseqüentemente reduz a penalidade de desempenho no monitoramento das restrições.

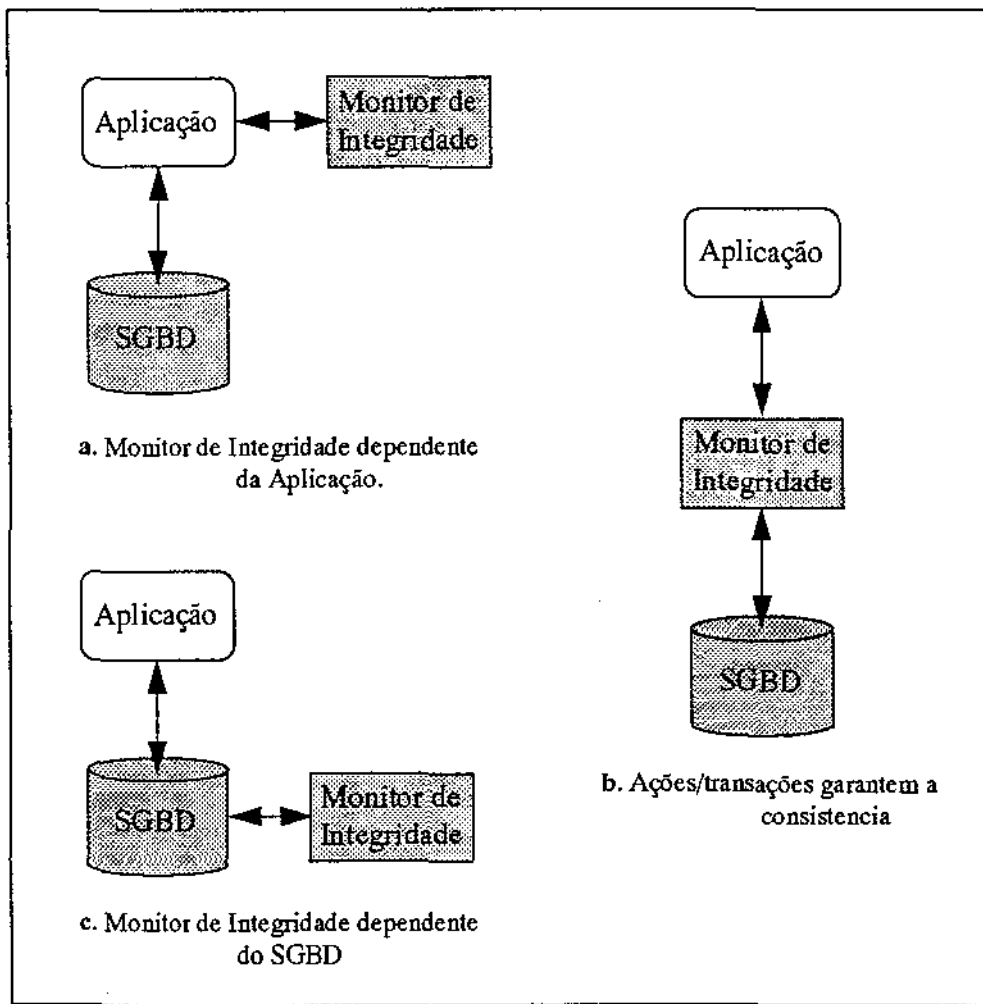


Figura 3.1. Arquiteturas para monitoramento de eventos [ZJ93].

2. BANCOS DE DADOS ATIVOS COMO SUPORTE A RESTRIÇÕES

Existem vários trabalhos de pesquisa que discutem o suporte a restrições utilizando bancos de dados ativos. Neste contexto, as regras E-C-A têm o seguinte significado: o Evento é uma notificação de atualização; a Condição é o predicado da restrição; e a Ação pode ter um comportamento corretivo ou preventivo, como descrito anteriormente.

Os trabalhos nesta área podem dividir-se em: restrições em banco de dados relacionais (e relacionais estendidos), e restrições em sistemas orientados a objetos. No primeiro grupo, [Mor84] descreve algumas características dos bancos de dados ativos e propõe uma

linguagem de restrições. [SHP88, SJGP90] apresentam um mecanismo para o gerenciamento de regras no sistema Postgres, e como as regras podem ser usadas para o suporte a visões, procedimentos, e restrições de integridade. [WF90] estende SQL para incorporar a especificação de restrições, uma restrição gera regras de produção que verificam as alterações sobre as tuplas ou conjuntos de tuplas (no sistema Starburst). [SPAM91] descreve como estender um sistema relacional com uma camada que gerencia relações especiais (ativas) que controlam os eventos e permitem a execução de gatilhos. Em [SZ91] é considerado o problema de otimizar a execução de grandes volumes de regras.

No segundo grupo, [DBM88, CBB+89] introduzem o paradigma E-C-A no sistema HiPAC; [DHL90, DHL91] apresentam um estudo das regras em um contexto de atividades de longa-duração, retardando a execução da ação. Em [UD89, UD90, UKN92] as regras são utilizadas para a manutenção de restrições, propondo também um algoritmo para a detecção de ciclos. [MP91] implementa um mecanismo de regras no sistema O2, o qual pode ser utilizado para a manutenção da consistência. No trabalho [DGP91] é apresentado como especificar regras para a manutenção de restrições no sistema ADAM. [And92] propõe uma solução geral para a geração automática de regras que manterão uma determinada restrição em um sistema ativo orientado a objetos.

3. MANUTENÇÃO DE RESTRICÕES USANDO UM SISTEMA ATIVO (OO)

Esta dissertação utiliza o mecanismo de regras E-C-A baseado no sistema ativo orientado a objetos do Sentinel [CAM93], que ataca as seguintes limitações de outros sistemas de regras:

- Alguns sistemas só permitem a especificação de regras dentro da definição das classes. Isto traz dificuldades quando regras são adicionadas, eliminadas, ou alteradas, dado que todas as instâncias envolvidas nestas mudanças também devem ser alteradas.
- Em alguns sistemas, regras e eventos não são manipulados como objetos, resultando em um tratamento diferenciado entre estes e os outros objetos.

Os critérios de projeto do Sentinel são os seguintes:

- Enriquecer a interface de objetos convencionais com uma interface de eventos, que tem a habilidade de propagar eventos, mantendo assim o encapsulamento.
 - Suportar regras e eventos (primitivos e compostos) como objetos.
 - Permitir que as regras possam ser disparadas por eventos compostos gerados por vários objetos.
 - Permitir que um objeto especifique dinamicamente a que mudanças de estado de outros objetos deve reagir.
-

- Fornecer um mecanismo uniforme para associar regras a todas as instâncias de uma classe, ou um subconjunto delas, possivelmente de diferentes classes.

Para garantir independência entre uma regra e os objetos que ela monitora, estes últimos devem ser capazes de: i) gerar eventos quando seus métodos são ativados e ii) propagar estes eventos a outros objetos. Para conseguir estes objetivos, Sentinel estende os objetos com uma interface de eventos. Isto permite ao objeto designar alguns de seus métodos como *geradores de eventos* primitivos. Tradicionalmente, objetos recebem mensagens (definidas usando a interface convencional), executam certas operações e retornam os resultados. No Sentinel, além disso, os métodos invocados geram eventos (definidos na interface de eventos) e propagam-nos, mediante envio de mensagens, a outros objetos. Os eventos podem ser gerados antes ou após a execução do próprio método. Uma classe com estas características é conhecida como *Reativa*, sendo definida como:

Classe Reativa = Definição de Classe Tradicional + Especificação da Interface de Eventos

A interface de eventos só especifica os eventos que são produzidos pela classe Reativa quando recebe determinadas mensagens. A semântica desta interface é que toda instância de uma classe Reativa deve gerar eventos para os métodos nela especificados. O evento gerado, através do envio de mensagens, tem os seguintes parâmetros:

Evento Primitivo = OID + Classe + Método + Parâmetros + <i>TimeStamp</i>
--

Instâncias de classes Reativas são geradoras (ou produtoras) de eventos, e devem conhecer os consumidores destes. Isto introduz a noção de classes *Notificáveis*. Uma instância de uma classe Notificável é um possível consumidor de um evento que é gerado por um objeto da classe Reativa. A associação entre um evento e um objeto notificável é estabelecida utilizando o mecanismo de *subscrição*, que permite associar dinamicamente objetos reativos a objetos notificáveis.

3.1 CLASSIFICAÇÃO DOS OBJETOS

Em função das considerações anteriores, os objetos em Sentinel são classificados em três categorias: passivos, reativos e notificáveis.

- *Passivos*. Aceitam execução de operações (métodos) mas não geram eventos. Não há nenhum *overhead* sobre a definição e uso destes objetos.

- *Reativos*. Objetos que precisam ser monitorados. Uma vez que um método é declarado como gerador de eventos quando aplicado a certos objetos (usando a interface de eventos), a execução deste método provoca um evento e a conseqüente notificação de outros objetos (previamente definidos através do mecanismo de subscrição).

- *Notificáveis*. Objetos com a capacidade de serem notificados por eventos gerados em objetos reativos.

A notificação é baseada no conceito de *subscrição*. Objetos notificáveis se inscrevem a eventos primitivos gerados por objetos reativos. Em outras palavras, um método gerador de eventos propaga estes eventos aos objetos notificáveis que estejam ligados, por subscrição, a tais eventos. Os objetos notificáveis, ao serem acionados desta forma, executam alguma operação que pode envolver objetos reativos, notificáveis e passivos.

3.2 EVENTOS

Há várias alternativas possíveis para representar eventos em um sistema orientado a objetos. A seguir são discutidas as vantagens e desvantagens de algumas delas.

- *Eventos como expressões*. O processamento da definição de eventos é feito principalmente em tempo de compilação, obtendo assim ganho no momento de execução. A principal desvantagem é que eventos não podem ser adicionados, eliminados, ou alterados durante a execução. Os eventos não podem ter atributos ou métodos. Novos tipos de eventos ou atributos de eventos não podem ser incorporados facilmente, comprometendo a extensibilidade. Exemplo: Ode [GJ91,GJ92a].

- *Eventos como atributos das regras*. Uma vantagem deste enfoque é que a associação entre evento e regra é armazenada como parte da estrutura da regra. Ao contrário do caso anterior, este esquema permite adicionar, eliminar, e alterar eventos durante a execução, mas tem as demais desvantagens do anterior.

- *Eventos como objetos*. Nesta alternativa, os eventos têm estado, estrutura, e comportamento. O estado pode armazenar a ocorrência e os parâmetros dos eventos ocorridos. A estrutura é utilizada para descrever a relação com outros eventos. O comportamento especifica quando informar outros eventos ou regras. Representar eventos como objetos permite adicionar, eliminar, e alterar eventos de forma uniforme, como qualquer outro objeto. Novos tipos de eventos podem ser incorporados alterando a definição de classes, sem comprometer a extensibilidade e modularidade do sistema. Exemplos: Adam [DPG91] e Sentinel [CAM93,CKTB94].

Nesta dissertação é adotada a terceira alternativa, tratando os eventos como objetos (especificamente notificáveis), que podem ser compostos utilizando operadores sobre eventos.

Eventos primitivos são aqueles gerados pela invocação de um método, pelo relógio do sistema ou por uma aplicação. Um *evento composto* é um conjunto de eventos primitivos ou compostos relacionados por meio de operações (disjunção, conjunção, seqüência, etc.).

3.3 REGRAS

Há vários esquemas para incorporar regras dentro de um ambiente orientado a objetos:

- *Regras declaradas só dentro de classes.* As regras são declaradas pelo usuário (na definição da classe) e inseridas no código pelo sistema nos lugares onde possam ser disparadas. As regras são associadas a objetos, fazendo parte de seu comportamento. A principal vantagem é o desempenho, dado que a maior parte do processamento das regras é feito em tempo de compilação. Uma desvantagem é a ineficácia na criação, exclusão e modificação de regras.

- *Regras como tipo de dados.* A vantagem é a reutilização e a extensibilidade. Regras podem ser dinamicamente incorporadas, excluídas e modificadas. A principal desvantagem é que não suporta herança. Além disso, a existência de uma regra depende da existência de outros objetos.

- *Regras como objetos.* Apresentam várias vantagens, uma das quais é que as regras podem ser criadas, modificadas e excluídas da mesma forma que outros objetos, provendo assim um esquema uniforme. Outra vantagem é que as regras são entidades independentes da existência de outros objetos. É fácil introduzir novos atributos ou operações sobre regras, simplesmente modificando a definição da classe Regra [DBM88].

No mecanismo descrito nesta dissertação, as regras são tratadas como objetos, do tipo notificável.

3.4 ASSOCIAÇÃO DAS REGRAS A OBJETOS

Regras em Sentinel são classificadas em duas categorias: regras de classe e regras de instância. As regras de classe são aplicáveis a todas as instâncias de uma classe, enquanto que as regras de instância são aplicáveis a instâncias específicas, possivelmente de diferentes classes. Todas as regras são tratadas como objetos notificáveis.

Regras são associadas a objetos através do mecanismo de subscrição. Como regras são objetos notificáveis, inscrevem-se dinamicamente a eventos gerados por objetos reativos. Desta forma, são notificados, por estes últimos, no acontecimento de um evento, podendo então executar as operações que correspondam.

A figura 3.2 descreve a associação entre eventos, regras e objetos reativos, como definido no sistema Sentinel. O objeto reativo *Objeto1* notifica a regra *RI* quando o evento *e1* acontece. De forma similar o *Objeto2* notifica a mesma regra quando o evento *e2* acontece. Esta notificação corresponde à ativação da regra, que verifica se pode ser disparada (executada), analisando os eventos a ela associados.

4. QUESTÕES DE IMPLEMENTAÇÃO

É necessário distinguir duas etapas no uso de um sistema que adota a arquitetura do Sentinel. A primeira delas é a definição da interface de eventos, e a correspondente criação da estrutura de dados necessária para a notificação destes eventos. A segunda etapa é a manutenção da informação relativa às subscrições.

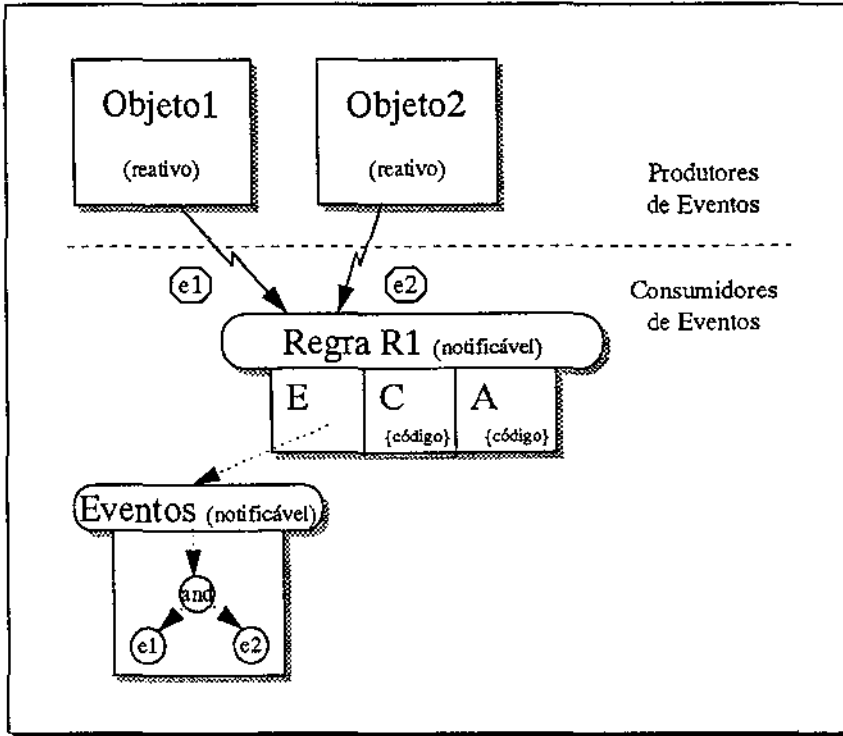


Figura 3.2. Associação entre eventos, regras e objetos reativos no Sentinel [CAM93].

Na interface de eventos, o programador identifica que função gera eventos e em que momento, podendo este último ser de dois tipos: *BeginOfMethod* (BOM) e *EndOfMethod* (EOM). BOM e EOM geram o evento antes e após a execução do código do método correspondente.

Os métodos são pré-processados e tem seu código modificado a partir da análise da interface de eventos. Esta é analisada alterando o comportamento dos métodos correspondentes, colocando como primeira (BOM) e última (EOM) instrução a invocação ao método (*SearchAndNotify*) que notificará os subscribers. Na figura 3.3 é apresentado o código do método gerador (de eventos) antes (a) e após (b) as alterações, respectivamente.

<pre>methodname: params method body</pre> <p style="text-align: center;">(a)</p>	<pre>methodname: params self SearchAndNotify(BOM,methodname,params) method body self SearchAndNotify(EOM,methodname,params)</pre> <p style="text-align: center;">(b)</p>
--	--

Figura 3.3. Código correspondente a um método gerador de eventos antes e após o pré-processamento.

É necessário também criar, se inexistente, a instância de Evento correspondente, para que seja notificado quando necessário pelo método *SearchAndNotify*. A figura 3.4 descreve parte da hierarquia de classes do Sentinel que é de interesse para este trabalho. A classe

objeto tem duas subclasses: *Reativa* e *Notificável*. As classes *Regra* e *Evento* são derivadas a partir da classe *Notificável*.

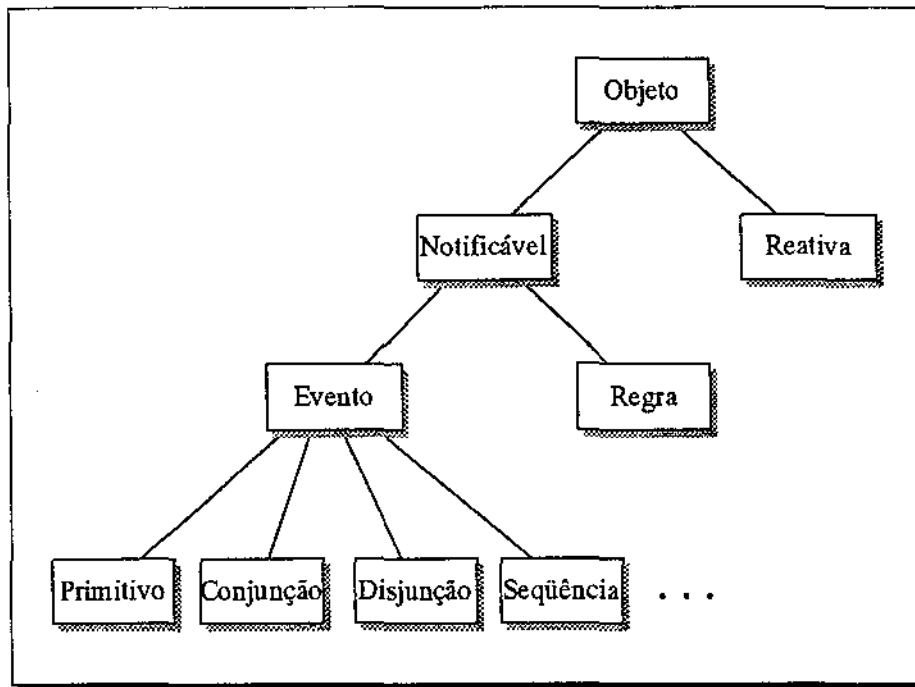


Figura 3.4. Hierarquia de classes do sistema Sentinel.

Em Sentinel os eventos (compostos ou primitivos) subscrevem-se às regras, e estas últimas associam-se aos objetos reativos através do mecanismo de subscrição. Em cada classe *Reativa* há uma referência explícita às regras dela dependentes por subscrição. Cada regra tem seu detector de eventos. Quando um método que gera eventos é executado na instância ou instâncias de interesse, os eventos são enviados às regras subscreitas e em seguida passados para o detector de eventos que avalia se as regras subscreitas devem ser disparadas [CAM93]. Esta implementação tem várias desvantagens:

- A execução de um método em uma instância que não esteja subscreita envolve muitas verificações para concluir que aquele evento não é de interesse.
- As regras participam como intermediárias na detecção de eventos (figura 3.2). Os eventos gerados sempre são passados às regras, para que estas entreguem o controle a seus detectores locais de eventos. Isto implica perda de desempenho.
- Como cada regra tem seu próprio detector de eventos, um evento precisa ser enviado a todas as regras a ele subscreitas. Desta forma deve existir uma cópia de cada evento para cada regra subscreitora. Isto implica em um gasto maior de espaço para manter esta informação.

Estas desvantagens podem ser eliminadas através de algumas modificações propostas nesta dissertação as quais são introduzidas a seguir.

5. DESCRIÇÃO DO MECANISMO ATIVO IMPLEMENTADO

Para validar as propostas desta dissertação foi implementado um protótipo de sistema ativo, baseado no Sentinel. No mecanismo implementado, as questões de desempenho foram resolvidas modificando a arquitetura do Sentinel como se segue:

- Os eventos recebem diretamente mensagens geradas pelos objetos reativos (ao invés de ter as regras como intermediários). Isto é feito utilizando o mesmo mecanismo de subscrição do modelo Sentinel. Desta mesma forma os eventos são associados às regras. Assim que um método gerador é invocado, o evento correspondente é informado e a regra é notificada de imediato.

- Cada método que é gerador de eventos tem uma única instância da classe Evento que lhe corresponde (ao invés de ter várias instâncias de um mesmo evento para cada uma das regras onde está envolvido).

- A estrutura para manter a informação sobre a subscrição está baseada em grafos. Especificamente, esta informação é centrada nos eventos, sendo estes os intermediários entre os produtores de eventos e as regras (figura 3.5).

- Nos objetos reativos, a informação sobre as subscrições passa a ser mais detalhada. A proposta é que cada objeto reativo mantenha uma lista de tuplas, cada uma das quais com a seguinte informação: método, contador, e identificador de uma instância de Evento. Cada tupla associa, assim, um par <objeto reativo, método> à instância de Evento correspondente. Esta instância é notificada sempre que o método correspondente for executado. Com esta informação, os eventos são notificados só quando necessário, evitando assim um *overhead* adicional. O contador controla quantas regras estão associadas ao evento.

O esquema conceitual de associação entre objetos que é implementado no Sentinel [CAM93] é descrito na figura 3.2. A figura 3.5 apresenta uma visão geral dos aperfeiçoamentos do mecanismo descrito nesta dissertação, que tentam otimizar tempo de execução e espaço.

De acordo com esta figura, os objetos reativos *Objeto1* e *Objeto2* geram os eventos *e1*, *e2* respectivamente. É definido o evento (*e1 and e2*) que depende (ou subscreve) dos eventos *e1* e *e2*. A regra *R2* subscreve o evento *e1*, enquanto a regra *R1* subscreve o evento (composto) (*e1 and e2*).

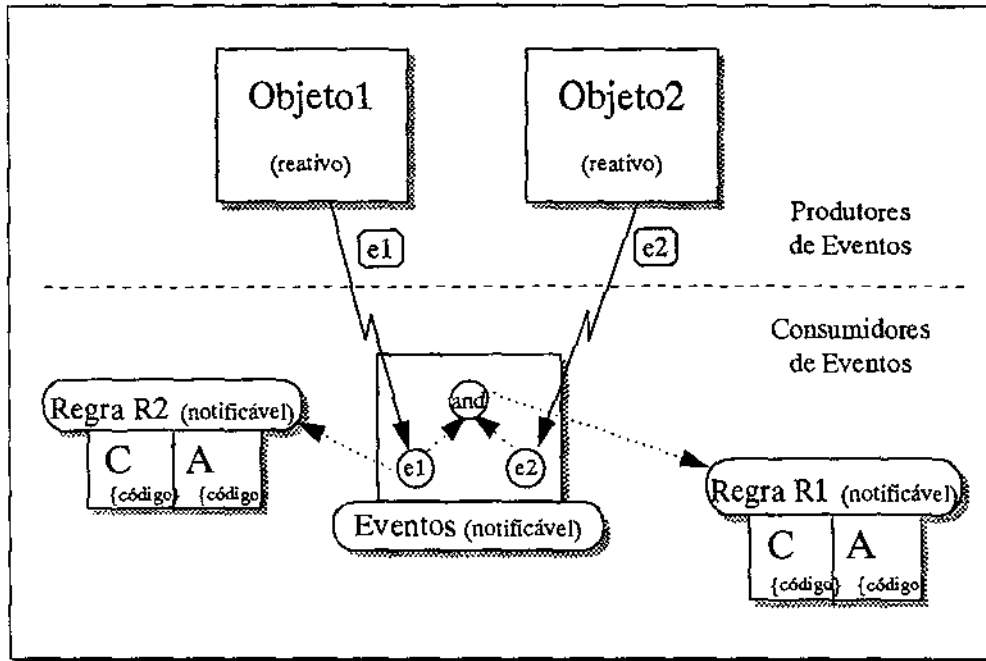


Figura 3.5. Associação entre objetos no mecanismo proposto.

Este esquema pode parecer, em princípio, mais complicado do que o apresentado na figura 3.2, uma vez que possui maior quantidade de ligações entre os objetos. No entanto, este esquema é transparente ao usuário, e utiliza as mesmas primitivas do modelo implementado no Sentinel. Em adição, o comportamento dos objetos é otimizado, como descrito nas próximas seções.

As seções seguintes descrevem a hierarquia de classes apresentadas na figura 3.4, a forma de detecção de eventos, e o modelo de execução. A seguinte notação é utilizada:

- métodos: representados por nomes em **negrito**, por exemplo, **AddEventSubscribe**
- variáveis de instância: representados por nomes em *itálico* com a primeira letra minúscula, por exemplo, *methodName*
- variáveis de classe: representados por nomes em *itálico* com a primeira letra maiúscula, por exemplo, *ClassSubscribers*

5.1 HIERARQUIA DE CLASSES

Nas seções seguintes são descritas as características das classes Reativa, Notificável, Evento e Regra implementadas no protótipo.

5.1.1 Classe Reativa

A figura 3.6 apresenta a definição da classe Reativa. A variável de classe *ClassSubscribers* de uma classe reativa mantém uma lista dos subscritores associados à classe. Esta variável é usada só para subscrições de classe (todas as instâncias de uma classe) e mantém uma lista de tuplas *<methodName, counter, eventId>*, onde

- *methodname*: nome de um método que gera eventos ao ser enviado a qualquer instância da classe;
- *eventid*: é o identificador da instância do evento (primitivo) que deve ser informado quando o método é executado;
- *counter*: mantém o número de regras que dependem deste evento.

Subscrições de instância são realizadas de forma semelhante, usando a variável de instância *instanceSubscribers* com o mesmo tipo de informação.

O método **Subscribe**, utilizado para subscrição, tem como parâmetro a regra que deve ser associada à classe ou instância. Este método coloca automaticamente na lista de subscrições cada um dos eventos primitivos que pertencem à regra correspondente.

Este esquema de implementação, que automaticamente coloca na lista de subscritores todos os eventos envolvidos, permite automatizar tarefas que em [CAM93] são deixadas a cargo do programador. Por exemplo, quando em [CAM93] é definido um evento composto (com eventos de várias classes), a subscrição da regra deve ser feita para cada uma das classes componentes. Já no mecanismo aqui proposto isto é feito automaticamente.

```
class Reactive : Object {
  ClassVar:
    ClassSubscribers;      /* lista de <methodname,counter,eventid> */
  ClassMethods:
    Subscribe(rule)
    UnSubscribe(rule)
  InstanceVar:
    instanceSubscribers;  /* lista de <methodname,counter,eventid> */
  Methods:
    Subscribe(rule)
    UnSubscribe(rule)
    SearchAndNotify(methodname, params)
}
```

Figura 3.6. Definição da classe Reativa.

Quando invocado o método *methodname*, o método **SearchAndNotify** verifica a sua presença na lista de subscritores. Inicialmente, *methodname* é procurado na lista de subscritores, primeiro na lista de instâncias e, se não encontrado, na lista de classes. Se é achado e o contador é maior que zero, então é enviada a mensagem de notificação (**Notify**) para o evento correspondente (*eventid*).

5.1.2 Classe Notificável

A classe Notificável representa objetos consumidores de eventos que são produzidos por objetos da classe Reativa. Como descrito anteriormente, os objetos reativos informam a ocorrência de um evento enviando a mensagem `Notify` aos objetos consumidores (subscritores). É necessário então descrever no método `Notify` (das classes notificáveis) o comportamento correspondente à reação ante um determinado evento.

5.1.3 Hierarquia de Eventos

São suportados os eventos primitivos e os compostos. Os eventos primitivos são aqueles gerados quando um método de uma classe Reativa é executado (podendo ser BOM ou EOM). Eventos compostos são construídos aplicando operadores de composição de eventos (disjunção, conjunção, seqüência, etc.) a eventos primitivos ou compostos.

A definição da classe abstrata `Evento` é apresentada na figura 3.7. Cada instância desta classe tem uma lista de subscritores aos quais deve notificar quando o evento que representa for detectado. A lista de subscritores está dividida em duas variáveis, `eventsToNotify` e `rulesToNotify`, a primeira utilizada para eventos compostos e a segunda para as regras dependentes deste evento.

```
class Event : Notifiable {      /* classe abstrata */
  InstanceVar:
    rulesToNotify;
    eventsToNotify;
  Methods:
    Notify(params)
}
```

Figura 3.7. Definição da classe `Evento`.

O método `Notify`, quando invocado, informa às regras e eventos dele dependentes que o método que ele representa foi executado.

A classe `Primitiva` (figura 3.8), é introduzida para representar os eventos primitivos. Existe uma única instância desta classe para cada par `<methodname, classname>`, criada através do método `Create`. O método `Notify` é herdado da classe abstrata `Evento`.

```
class Primitiva : Event{
  ClassMethods:
    Create(when, class, methodname)      /* when:[BOM|EOM] */
}
```

Figura 3.8. Definição da classe `Primitiva`.

Os operadores sobre eventos são modelados como subclasses de `Evento`, e o comportamento correspondente é descrito no método `Notify`. Este tipo de esquema permite

que estes operadores possam ser estendidos simplesmente através da incorporação de uma nova subclasse de Evento.

Um dos operadores, a classe Conjunção, é descrito na figura 3.9. A variável de instância *listOfEvents* mantém uma lista dos eventos que são passados como parâmetro na criação de uma instância, usando o método **Create**. O método **Notify** é re-escrito para implementar a operação de conjunção. Este método ao verificar que todos os eventos em *listOfEvents* aconteceram, invoca o método **Notify** da super-classe Evento (descrito anteriormente).

```
class Conjunction : Event{
  InstanceVar:
    listOfEvents;
  ClassMethods:
    Create(evs)
  Methods:
    Notify(params)
}
```

Figura 3.9. Definição da classe Conjunção.

5.1.4 A Classe Regra

Uma regra é definida basicamente por: i) evento, que a dispara, ii) condição, avaliada quando a regra é disparada, iii) ação, executada quando a condição é satisfeita. Há duas alternativas básicas para implementar regras em um modelo orientado a objetos. Na primeira, uma classe é criada para cada regra, e a regra é a única instância dessa classe. A principal desvantagem deste enfoque é a quantidade de classes criadas para regras. A segunda alternativa é criar uma única classe Regra, com uma instância para cada regra. A vantagem é definir uma classe para todas as regras, não aumentando exageradamente a quantidade de classes e podendo também especializar esta classe segundo as necessidades.

No mecanismo implementado nesta dissertação é adotada a segunda alternativa, onde as regras são objetos notificáveis.

A definição da classe Regra é ilustrada na figura 3.10. A variável de instância *state* reflete o estado da regra (ativo ou inativo), modificado mediante os métodos **Activate** e **DeActivate**. O comportamento das regras depende do modificador *mode*, que segundo seu valor faz com que a regra seja desativada automaticamente logo após sua execução (*once*), ou que tenha que ser desativada mediante o método **DeActivate** (*always*). A variável *granularity* se corresponde com a frequência (a nível de sessão, transação ou operação) com que é avaliado o conjunto de regras que estão prontas para executar.

A variável *event* é utilizada para associar a regra ao objeto evento correspondente. O código correspondente à condição e à ação é armazenado nas variáveis *condition* e *action* respectivamente. A variável *nameOfRule* representa o nome da regra.

```

class Rule : Notifiable{
  InstanceVar:
    state;          /* [active|inactive] */
    mode;           /* [once|always] */
    granularity;    /* [operation|transaction|session] */
    event;          /* identificador do objeto evento */
    condition;      /* codigo que representa a condição */
    action;         /* codigo que representa a ação*/
    nameOfRule;     /* nome da regra */
  ClassMethods:
    Create(name, event, condCode, actionCode, granularity, mode)
  Methods:
    Activate()
    DeActivate()
    Notify(params)
    Fire(params)
}

```

Figura 3.10. Definição da classe Regra.

O método **Notify** verifica que a regra esteja ativa. Segundo a variável *granularity*, a regra é colocada na fila (*queue*) correspondente para sua posterior execução. O método **Fire** avalia a condição e, se esta for satisfeita, executa a ação correspondente.

Regras podem ser classificadas em regras de classe ou de instância, segundo sua aplicação. As primeiras são aplicáveis a todas as instâncias de uma classe, enquanto que as outras são aplicáveis a instâncias específicas, possivelmente de diferentes classes. Regras de instância são declaradas no código da aplicação.

5.2 DETECÇÃO DE EVENTOS (PRIMITIVOS E COMPOSTOS)

Existem várias formas de detectar eventos em sistemas ativos, como já foi descrito no capítulo 2. Esta dissertação adotou uma mistura dos mecanismos de grafos e de subscrição. A detecção é utilizada a partir de um grafo dirigido acíclico, onde cada nó representa um evento (primitivo ou composto), e os arcos representam a seqüência entre eles. Os nós folha correspondem a eventos primitivos, enquanto que os nós internos correspondem a sub-expressões de eventos (compostos).

Eventos primitivos subscrevem a eventos compostos, enquanto que as regras subscrevem a um evento. Para cada subscrição é criado um arco dirigido, que descreve a comunicação entre os nós.

Os eventos primitivos são notificados diretamente por objetos reativos, e os eventos compostos são notificados por eventos primitivos ou compostos. Toda vez que um evento é notificado, este informa a todos seus subscritores (regras e eventos compostos) que o evento aconteceu.

A figura 3.11 mostra um grafo de eventos. Os métodos $method_1$, $method_2$, $method_n$ (sobre objetos reativos) geram os eventos $event_1$, $event_2$ e $event_n$ respectivamente. São definidos também eventos compostos usando os conectores *AND* e *OR*. A regra *Rule2* subscreve o evento composto por *OR*, enquanto a regra *Rule1* subscreve o evento composto por *AND*.

Os conectores foram implementados como objetos do tipo evento. Os eventos $event_1$ e $event_2$ notificam o evento *AND*. Este evento composto somente irá notificar seus subscretores (regra *Rule1* e evento *OR*) quando ambos eventos $event_1$ e $event_2$ tenham acontecido (em qualquer ordem). O evento *OR*, por sua vez, dispara a regra *Rule2* somente quando algum dos eventos que lhe subscrevem ocorrer.

Esta estrutura de grafo tem a vantagem de permitir o compartilhamento dos eventos por todas as regras e eventos compostos, melhorando assim consideravelmente a utilização de espaço.

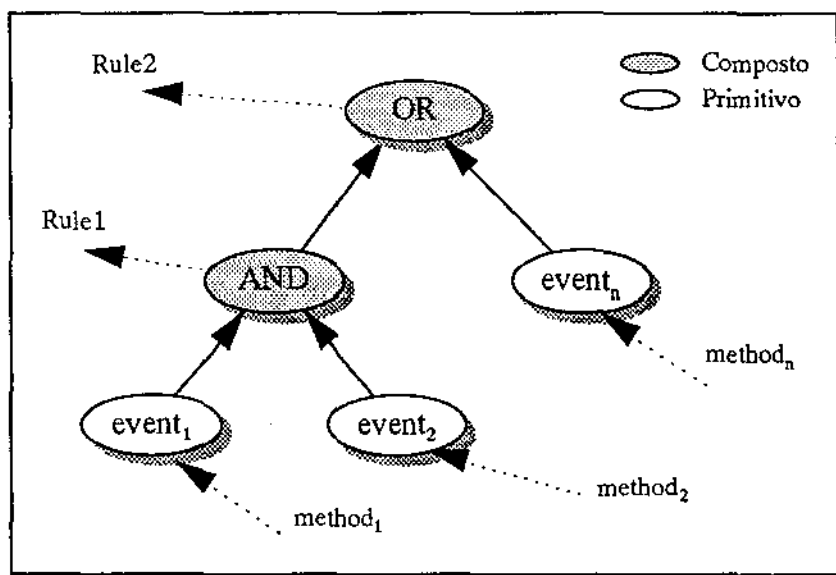


Figura 3.11. Grafo de Eventos.

5.3 MODELO DE EXECUÇÃO

O protótipo suporta três tipos de granularidade de ativação, nos níveis de operação, de transação e sessão. Para cada uma destas existe uma classe Fila que controla a execução e escalonamento das regras. As regras notificadas entram na fila, utilizando o método *AddQueue*, segundo sua granularidade. Imediatamente antes das operações do BD (com exceção de *rollback*) correspondentes a cada granularidade serem executadas, as filas respectivas também são notificadas. Ante esta notificação todas as regras dessa fila são disparadas, a partir do envio da mensagem *Fire* a cada uma delas. Um tratamento especial tem a notificação da operação *rollback*, que quando invocada elimina as regras que correspondem às filas respectivas.

A figura 3.12 apresenta uma visão esquemática do protótipo implementado. As flechas correspondem ao envio de mensagens (respectivas a seus rótulos). A execução de uma regra envolve os seguintes passos (figura 3.12):

- ❶ Um método (que está subscrito) é executado por um objeto reativo, e o evento correspondente é notificado (via **Notify**).
- ❷ O evento notifica a regra associada, enviando a mensagem **Notify** com os parâmetros correspondentes.
- ❸ Dependendo da granularidade de ativação da regra, esta é colocada automaticamente na fila adequada (**AddQueue**).
- ❹ Quando uma operação do banco de dados é executada, esta notifica a fila correspondente a sua granularidade.
- ❺ Todas as regras que estão na fila são executadas, mediante o envio da mensagem **Fire** para cada uma.

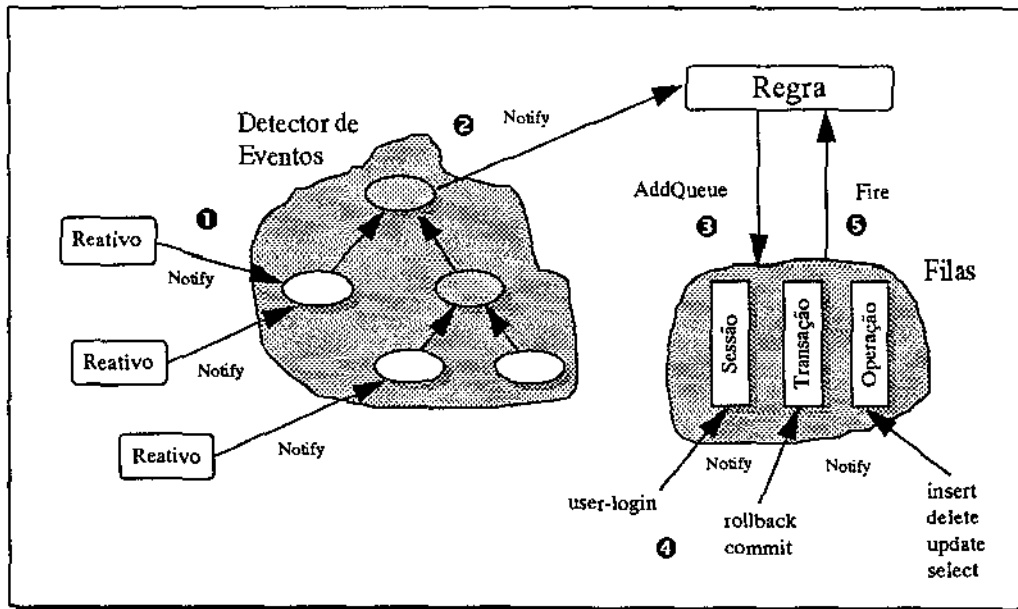


Figura 3.12. Visão global do modelo de execução.

IV

SISTEMAS DE INFORMAÇÃO GEOGRÁFICA

1. CONCEITOS BÁSICOS

Existem numerosas aplicações nas quais dados georeferenciados são necessários, tais como gerenciamento e mapeamento de recursos naturais (minerais, hídricos, ecológico, agrícola); planejamento do uso da terra; determinação de áreas suscetíveis a inundações, incêndios, secas ou terremotos; determinação do censo populacional; planejamento urbano (cadastro imobiliário, transporte, educação, saúde, habitação, segurança); gerenciamento de redes de serviços públicos (água, esgoto, telefone, eletricidade); controle de epidemias.

Dependendo do objetivo de sua utilização, diferentes características podem ser apresentadas de forma distinta. Em adição, dependendo da escala e do contexto, alguns detalhes podem ser omitidos, existindo assim diferentes visualizações/representações para uma mesma pergunta. Por exemplo a resposta para a consulta “*onde está localizada a Unicamp?*”, pode ser apresentada diferentemente, dependendo do contexto. Na figura 4.1 são apresentadas três situações: a parte (a) apresenta o estado de São Paulo no mapa do Brasil; a parte (b) apresenta o município de Campinas, dentro do estado de São Paulo, tendo como referência sua Capital (cidade de São Paulo); e a parte (c) mostra a localização da Unicamp no município de Campinas, juntamente com as principais vias de acesso.

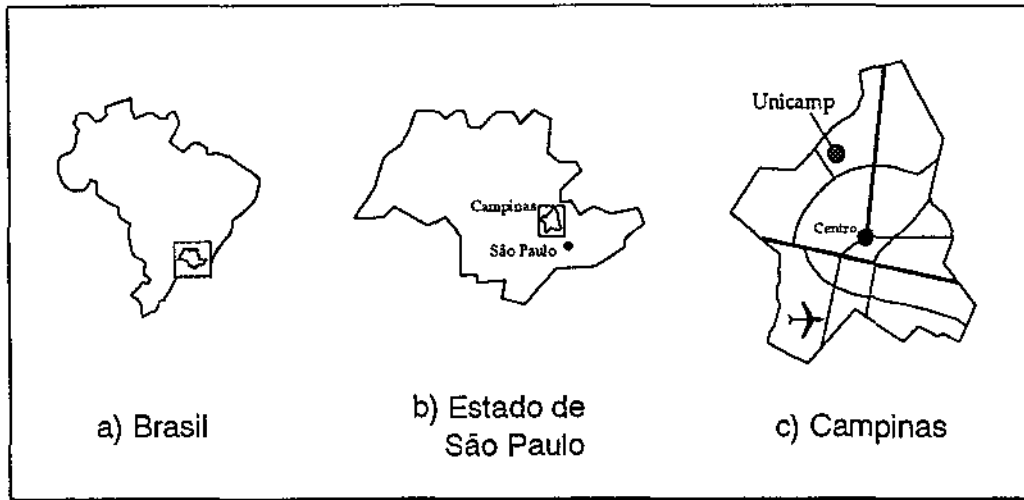


Figura 4.1. Diferentes respostas para a consulta *onde está localizada a Unicamp?*.

1.1 MODELOS DE DADOS

Um modelo de dados é uma coleção de ferramentas conceituais para descrição dos dados, relacionamentos entre os dados, semântica destes e restrições de consistência [EN94]. O propósito de um modelo de dados é fornecer uma maneira formal de representar informações, além de meios para manipular tal representação. Um modelo pode ser visto como uma abstração do mundo real que incorpora somente aquelas propriedades que são relevantes para a aplicação em questão.

Apesar de serem utilizados de forma eficiente em uma enorme gama de aplicações, os modelos utilizados para aplicações de banco de dados não são suficientes para representar a realidade geográfica. Existem várias abordagens para a modelagem de dados geográficos, como por exemplo as encontradas em [PMS93, CFS+94].

1.2 TIPOS DE DADOS

Dados geográficos têm quatro características [MP94]: sua posição geográfica (coordenadas); seus atributos (valores); relações topológicas; e a componente temporal. Uma vez armazenado num SIG, os dados são classificados nas seguintes categorias:

- **Dados convencionais.** Atributos alfanuméricos comumente encontrados em banco de dados convencionais. Representam as características não espaciais que descrevem o fenômeno geográfico.

- **Dados espaciais.** Atributos que descrevem a geometria e localização de um fenômeno. Dados espaciais têm propriedades geométricas e topológicas. Propriedades topológicas descrevem relações entre entidades geográficas (vizinhança, conectividade, intersecção, etc.). As representações dos dados geométricos podem ser em formato vetorial ou matricial (raster).

O formato vetorial é baseado em pontos, linhas e polígonos. No formato matricial, os dados são representados em pixels, ou células, onde a cada célula é associada um conjunto de valores textuais que descrevem as propriedades dos pontos dentro da mesma.

- **Dados com representação pictórica.** Dados em formato de *bitmap*, muitas vezes gerados por sensoriamento remoto (por exemplo, imagens de satélite ou radar) ou por fotografia aérea de uma determinada região geográfica (figura 4.2). Estas imagens são manipuladas por funções de processamento de imagens.



Figura 4.2. Fotografia aérea.

A componente temporal é importante para registrar a evolução e a validade do fenômeno geográfico representado.

1.3 FUNCIONALIDADE

Segundo [ABC+91, Mag91, Cif95], os SIG devem oferecer três funcionalidades distintas:

- **Cartográfica.** Enfoca os aspectos relacionados com o processamento e geração de mapas de maneira precisa e confiável. Suas responsabilidades são a digitalização (conversão de mapas analógicos para a forma digital), visualização e manipulação gráfica de mapas (inserção, alteração, remoção, zoom, etc.), impressão de mapas, entre outras.

- **Banco de dados.** Oferece as capacidades de controle de acesso e segurança, gerenciamento de transações e recuperação ante falhas. Também deve manipular e integrar de forma eficiente tanto os dados convencionais quanto os georeferenciados, precisando desta forma de métodos de armazenamento e consulta não suportados em SGBDs convencionais.

- **Análítica.** Permite interpretar os dados espaciais armazenados no SGBD. Algumas destas funções analíticas são: busca espacial, superposição de polígonos, computação de operações escalares, roteamento, cálculo estatístico, previsões de acontecimentos futuros. Maiores detalhes destas funções podem ser encontradas em [Cit95].

Um SIG é um tipo particular de sistema de informação, e deve fornecer integralmente procedimentos para aquisição, gerenciamento, análise e saída de dados.

- **Aquisição.** Consiste na obtenção dos dados geográficos através de alguma técnica de captura, tal como fotografia, sensoriamento remoto, GPS, digitalização de mapas, conversão de dados. Esta fase é considerada crítica devido ao seu custo (60% a 70% do custo total do projeto [ABC+91]), e à precisão com que estes dados devem ser incorporados. O sistema de informação depende da qualidade dos dados espaciais armazenados. De acordo com a aplicação existe também uma fase de pré-processamento, responsável pela conversão dos dados coletados para um formato apropriado.

- **Gerenciamento dos dados.** Caracterizada principalmente por suportar mecanismos para consulta, alteração, eliminação e inserção de novos dados, geralmente com o auxílio de um SGBD. Algumas destas características já foram mencionadas na seção anterior.

- **Análise.** Responsável pela geração de novas informações baseadas nos dados armazenados. Para isto é necessário um conjunto de funções analíticas, tais como, análise espacial e estatística, cálculo de relacionamentos espaciais.

- **Saída.** Responsável pela visualização das operações realizadas pelo SIG. A saída pode ser obtida utilizando-se diversos meios, tais como listagens estatísticas, gráficos de vários tipos ou mapas, gerados a partir de dispositivos como monitor, plotter ou impressora.

2. MODELO ADOTADO

[CFS+94] apresenta um modelo orientado a objetos para a modelagem de objetos georeferenciados, que permite a separação da especificação lógica das entidades da sua implementação. Uma vantagem desta separação é que os conceitos definidos pelo usuário podem ser mapeados em classes (em um banco de dados orientado a objetos), minimizando os problemas entre a modelagem e a implementação.

Neste modelo, a realidade geográfica é modelada de acordo com quatro camadas: i) *mundo real*, que refere-se à realidade geográfica; ii) *conceitual*, que descreve as entidades em alto nível de abstração; iii) *representação*, que define diferentes representações para uma determinada entidade conceitual; iv) *implementação*, que define a estrutura do banco de dados e os métodos de acesso correspondentes.

A hipótese básica é que o mundo é modelado utilizando as classes georeferenciadas, cujos objetos descrevem características de regiões da superfície da terra. No nível conceitual, o modelo suporta as visões de campos e objetos, distinguindo dentre duas classes básicas: campos geográficos (para a manipulação de variáveis contínuas), e objetos geográficos (para a especificação de entidades identificáveis).

Para o propósito das restrições topológicas (tratadas nesta dissertação), é adotada a visão do mundo como objetos, onde os dados vetoriais representam os objetos espaciais, ignorando a especificação de campos geográficos.

Objetos geográficos (OG) são entidades identificáveis no mundo real e podem ser *elementares*, *compostos* ou *fracos*. Um objeto geográfico *elementar* O_e é uma entidade individualizável da realidade geográfica, que não tem outros objetos geográficos como componentes. As características espaciais são armazenadas no atributo *loc*, que é representado por uma região (figura 4.3.b). A geometria de um objeto geográfico é representada com pontos, linhas e regiões.

$$O_e :: \text{tuple}(\text{tuple}(\langle \text{atributos não espaciais} \rangle), \text{loc})$$

Um objeto geográfico *composto* é um objeto geográfico construído baseado em outros objetos geográficos, cujas localizações devem pertencer à mesma classe geo-referenciada. A localização *loc* é um objeto complexo similar a O_c , exceto que cada objeto geográfico que é componente de O_c é substituído por sua localização.

$$O_c :: \text{tuple}(\text{tuple}(\langle \text{atributos não espaciais} \rangle), \text{set}(\langle \text{objetos geográficos} \rangle), \text{loc})$$

Um objeto geográfico *fraco* contém somente o atributo localização e existe apenas enquanto é parte de um objeto geográfico composto.

Neste trabalho, a parte crucial para expressar relacionamentos espaciais é a componente *localização* de um objeto geográfico. Esta é implementada com a hierarquia de classes apresentada na figura 4.3.a, tendo como raiz a classe Localização, e Ponto, Linha e Região como subclasses, que por sua vez podem ser especializadas.

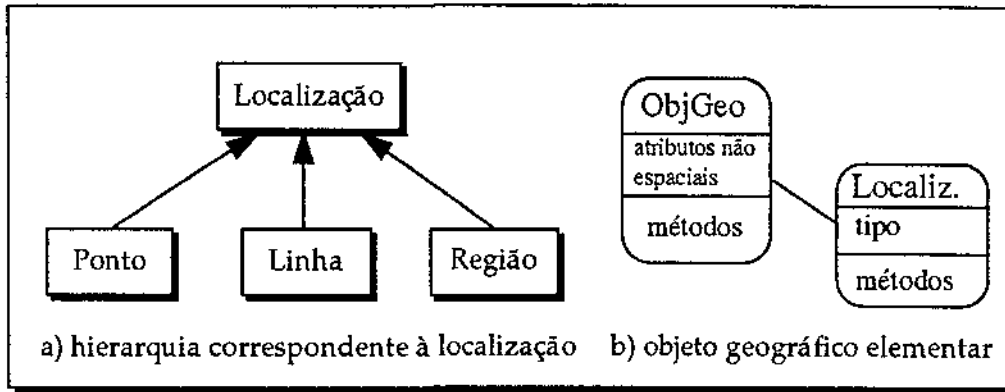


Figura 4.3. Modelo geográfico implementado.

3. OPERAÇÕES SOBRE OBJETOS GEOGRÁFICOS

As operações sobre objetos geográficos podem ser classificadas em consultas ou atualizações (figura 4.4). Consultas e atualizações podem atuar sobre dados convencionais (operações tradicionais) ou sobre a localização. Consultas podem retornar dados já existentes, derivar dados (área, perímetro, comprimento) ou objetos (extremos de uma linha). As operações de atualização sobre localização envolvem a criação (de pontos, linhas e regiões), alteração (por exemplo, ampliação, traslado) e remoção (de pontos, linhas e regiões).

O domínio (ou dimensões) das operações sobre objetos geográficos em duas dimensões é definido como OG-0D, OG-1D, OG-2D ou vazio, onde x D identifica a dimensão x . Estas dimensões correspondem a ponto, linha e região respectivamente. A seguir são descritas estas operações segundo esta classificação, estando restritas apenas à manipulação de dados espaciais, já que operações sobre dados convencionais estão disponíveis em bancos de dados convencionais.

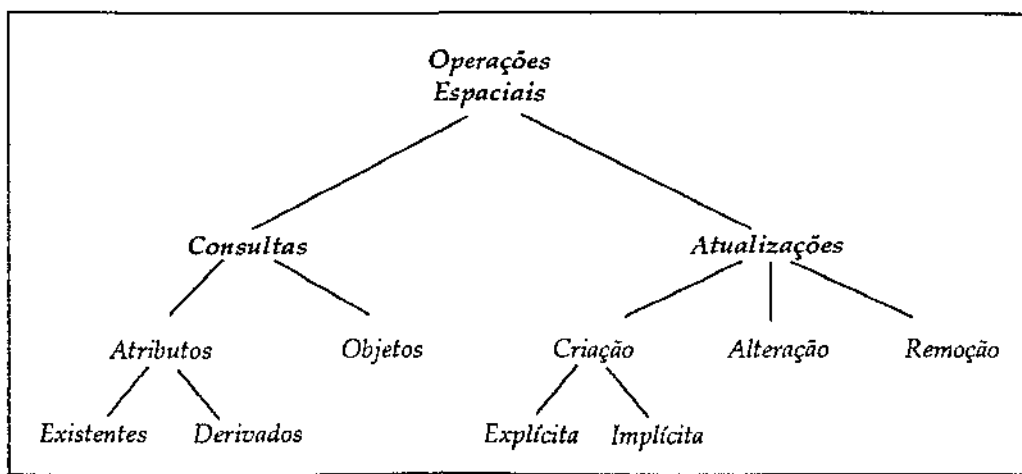


Figura 4.4. Classificação das operações sobre objetos geográficos.

CONSULTAS

• **Atributos existentes.** Operações que retornam a componente correspondente a um atributo espacial. Exemplos são:

- **Abcissa.** Retorna o atributo correspondente à abcissa do ponto em questão.

Abcissa(OG-0D): value

- **Extremos.** Retorna os extremos de um objeto tipo linha.

EndPoints(OG-1D): set(OG-0D)

- **Nós.** Retorna todos os pontos que compõem um objeto de tipo região.

Nodes(OG-2D): set(OG-0D)

- **Linhas.** Retorna todas as linhas que compõem um objeto de tipo região.

Lines(OG-2D): set(OG-1D)

• **Atributos derivados.** Operações que retornam um escalar, que é calculado a partir de dados já existentes.

- **Distância.** Retorna a distância entre dois objetos, calculada de acordo com o tipo destes objetos. Como exemplo, para objetos do tipo região, a distância pode ser calculada considerando o centróide ou a borda, enquanto que para objetos do tipo linha esta pode ser calculada considerando a mediana.

Distance(OG, OG): value

- **Área.** Retorna a área de uma região.

Area(OG-2D): value

- **Perímetro.** Retorna o perímetro um objeto de tipo região.

Perimeter(OG-2D): value

- **Comprimento.** Retorna o comprimento de um objeto de tipo linha.

Length(OG-1D): value

- **Objetos espaciais.** Operações que retornam um conjunto de objetos espaciais.

- **Busca por objetos adjacentes.** Retorna todas os objetos geográficos adjacentes a um determinado objeto.

Adjacent(OG): set(OG)

- **Análise de proximidade.** Gera um objeto geográfico de duas dimensões (*buffer*), cujos limites possuem uma distância fixa *r* em relação à fronteira de um objeto geográfico fonte. A análise de proximidade ao redor de uma região pode ser efetuada segundo duas abordagens: externa ou interna (figura 4.5).

Já ao redor de um ponto, a análise de proximidade pode ser circular ou quadrática (figura 4.6).

SimpleBuffer(OG, distance, Type): set(OG)

Type = {interna, externa}

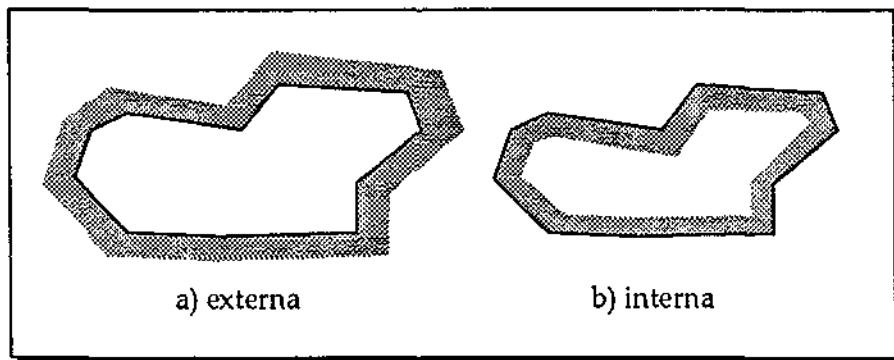


Figura 4.5. Análise de proximidade ao redor de uma região.

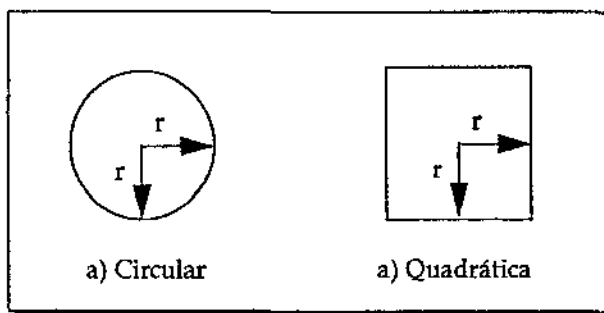


Figura 4.6. Análise de proximidade ao redor de um ponto.

- Busca Topológica. Obtém o conjunto de objetos geográficos que satisfaçam um certo relacionamento topológico em relação a um objeto geográfico fonte. Existem várias propostas nesta área, dentre elas podem ser citadas [CSE94, PS94, PSTE95].

ATUALIZAÇÕES

- **Criação.** Operações que criam objetos. Podem ser classificadas em explícitas e implícitas.

- Explícita. Cria objetos geográficos do tipo ponto, linha e região.

CreatePoint(x, y): OG-0D

CreateLine(set(OG-0D)): OG-1D

CreateRegion(set(OG-0D)): OG-2D

- **Implícita.** Cria objetos geográficos como resultado de operações.
 - **União.** Retorna a união de vários objetos geográficos. O resultado da união pode ser um conjunto de objetos geográficos, caso a operação seja aplicada a elementos espacialmente disjuntos.

Union(set(OG)): set(OG)

- **Intersecção.** Retorna a intersecção de objetos geográficos.

Intersection(set(OG)): set(OG)

- **Diferença.** Retorna a diferença entre objetos geográficos.

Difference(set(OG)): set(OG)

- **Reclassificação.** Agrupa categorias de um tema (ou camada temática), de modo a gerar um novo conjunto de categorias para o tema em questão. As categorias a serem agrupadas podem ser tanto primitivas quanto derivadas. Esta operação exige que sejam especificadas as relações existentes entre as categorias antigas e novas.

Reclass(set(OG), relcateg): set(OG)

- **Superposição.** Existem vários tipos de superposição, mas todos estes possuem em comum o fato de superpor objetos geográficos de duas dimensões de temas distintos, diferindo no modo como as categorias resultantes são geradas. Esta operação apresenta vários modos de operação, os quais são detalhados em [Cif95].

Overlay(set(OG-2D), set(OG-2D), mode): set(OG)

- **Ponto médio.** Retorna o ponto médio de uma linha.

MidPoint(OG-1D): OG-0D

- **Centróide.** Retorna o ponto que é o centróide de uma região.

Centroid(OG-2D): OG-0D

- **Alterações.** Operações que mudam o atributo correspondente à localização ou geometria do objeto geográfico.

MoveTo(OG-0D, lat, long)

Rotate(OG, degree)

Translate(OG, units, direction)

Shrink(OG, factor)

Expand(OG, factor)

AddPoint(OG-2D, OG-0D)

- **Remoção.** Operação que elimina objeto geográfico.

RemovePoint(OG-0D)

RemoveLine(OG-1D)

RemoveRegion(OG-2D)

4. SIG E BANCO DE DADOS ATIVOS

O gerenciador de dados em um SGBD de um SIG não utiliza características ativas. No entanto, há vários tipos de aplicações que lucrariam com o suporte de características ativas, em distintos níveis semânticos. Dentre os inúmeros exemplos podem ser citados:

- *Aquisição e definição dos dados.* A verificação da integridade e o ajuste dos dados podem ser processados durante a carga dos dados [LM94]. Regras podem expressar estes possíveis ajustes, e serem executadas quando necessário.

- *Restrições de integridade espaciais.* São aquelas que estabelecem algum tipo de relacionamento espacial entre objetos geográficos, como exemplo: a *distância* entre dois postes deve ser menor que X ; a *área* de uma fazenda não pode ser maior que Y ; ou não deve existir *interseção* entre os polígonos que representam um lote no cadastro de propriedades. Estes exemplos são apresentados na figura 4.7. No capítulo 5 são descritos com maior detalhes estes relacionamentos. Esta é talvez uma das principais utilizações de bancos de dados ativos em SIG, tendo em vista a atual tecnologia na área.

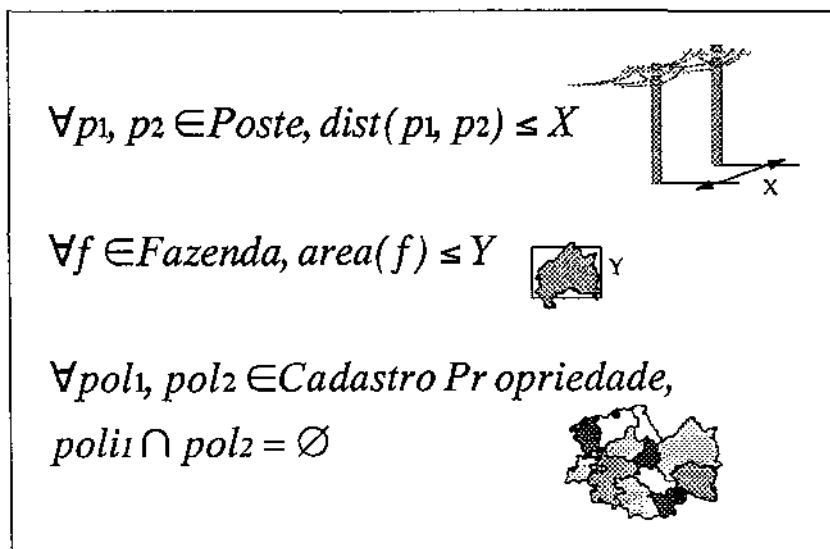


Figura 4.7. Exemplos de restrições de integridade espaciais.

- *Visões complexas.* Uma camada primária é aquela que tem características temáticas de apenas um tipo [PMS93]. Às vezes, é necessário combinar dados de diferentes camadas para criar outra camada, chamada camada derivada. Uma camada derivada pode ser gerada dinamicamente na hora da consulta, ou criada e armazenada como outra camada (materializada). Por exemplo a camada (derivada) clima precisa

combinar dados como temperatura, pressão atmosférica, ventos, umidade, etc., que são diferentes camadas primárias (figura 4.8).

- **Integração de bancos de dados heterogêneos.** Existem propostas de integração de banco de dados heterogêneos utilizando bancos de dados ativos [BBKZ92]. Em [Agu95] é proposto um método para integrar banco de dados heterogêneos que contém informação geo-referenciada, especificamente informação sobre planejamento urbano.

- **Otimização de consultas.** A consulta é uma das funções fundamentais de um SIG. A dimensão ativa pode ser empregada para a dedução de propriedades complexas da informação que fica armazenada no banco de dados e de conhecimento. A otimização é aplicável não só na execução como também na otimização do armazenamento dos dados.

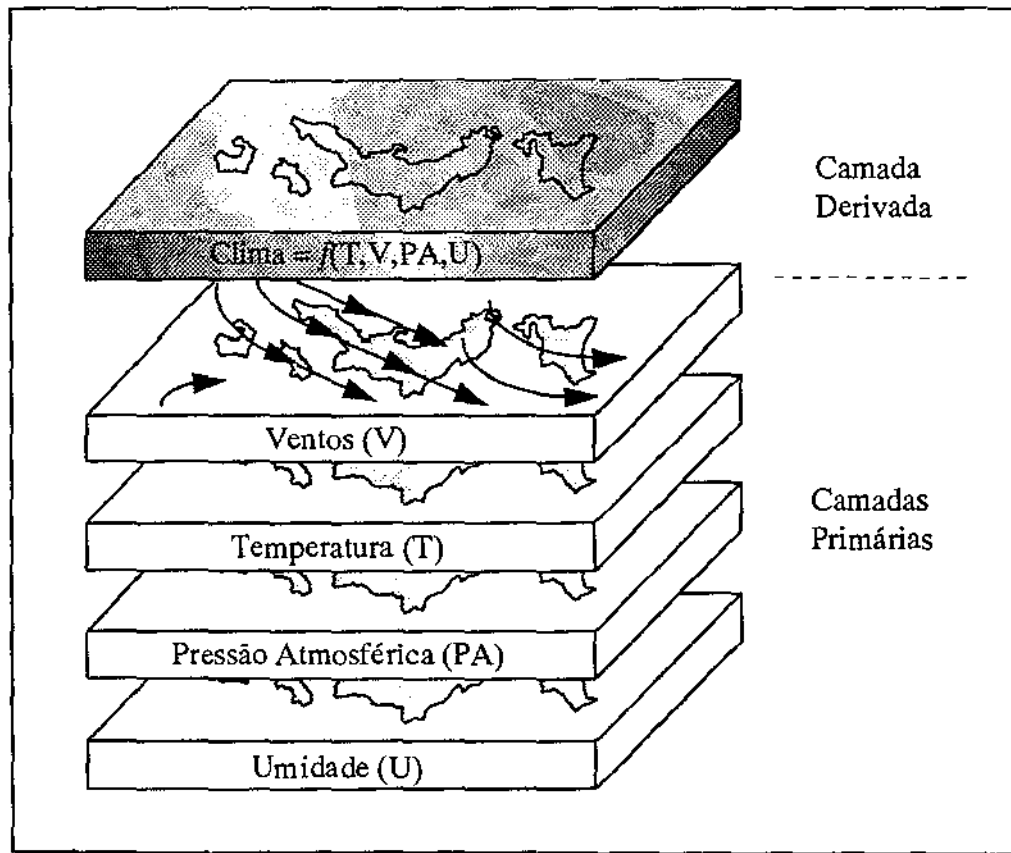


Figura 4.8. Visão conceitual do esquema de camadas.

V RESTRIÇÕES BINÁRIAS DE INTEGRIDADE TOPOLÓGICA

1. RELACIONAMENTOS ESPACIAIS

Relacionamentos espaciais entre objetos geográficos podem ser especificados em uma, duas ou três dimensões. Em [Güt94] os relacionamentos espaciais são classificadas em topológicos, de orientação e métricos. Aqui são estudados apenas relacionamentos topológicos de uma ou duas dimensões.

1.1 RELACIONAMENTOS DE ORIENTAÇÃO

Os relacionamentos de orientação descrevem onde os objetos estão posicionados em relação a outros. Para este fim, é preciso de um marco de referência, um objeto de referência e o objeto em questão. O marco de referência é a orientação que determina a direção na qual o objeto em questão está localizado em relação a um objeto de referência. Estudos sobre sentenças espaciais em linguagem natural revelam três tipos de marcos de referência: intrínseco (a orientação é dada por alguma propriedade herdada do objeto de referência), extrínseco (orientação é imposta por fatores externos) e deítico (orientação é imposta por um ponto de vista externo). Maiores detalhes sobre este tópico podem ser encontradas em [Her94].

Existem várias alternativas para a definição dos relacionamentos de orientação entre dois objetos, dependendo da quantidade de pontos que representam cada objeto. Por exemplo, quando o objeto é representado por um único ponto, pode ser utilizado o centróide ou mediana (em regiões e linhas respectivamente). Por outro lado, quando um objeto é representado por dois pontos, pode ser usado o limite inferior esquerdo e o limite superior direito do MBR (*Minimum Bounding Rectangle*) do objeto em questão.

[PS94] considera quatro primitivas de orientação: norte, sul, leste e oeste. Com base nestas primitivas, são definidos formalmente outros relacionamentos, por exemplo noroeste, nordeste, e mesma_posição. A figura 5.1 apresenta as partições do plano utilizando um único ponto de representação por objeto.

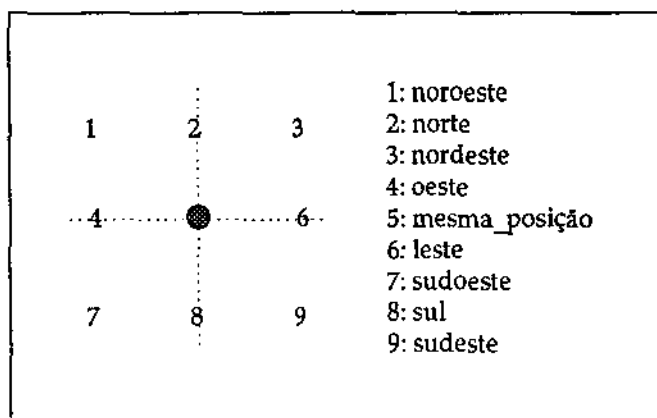


Figura 5.1. Partições do plano considerando um único ponto de representação.

1.2 RELACIONAMENTOS MÉTRICOS (ESCALARES)

Os relacionamentos métricos geram, a partir de um ou mais objetos geográficos, um escalar que representa uma propriedade intrínseca aos objetos geográficos analisados. Estes relacionamentos podem ser classificados em:

- intra-objeto: baseado principalmente no cálculo de área, perímetro e comprimento;
- inter-objeto: baseado no cálculo de um valor escalar usando mais de um objeto geográfico, por exemplo, distância.

No primeiro caso os valores calculados podem ser pré-computados e armazenados. Esta alternativa apresenta como vantagem a diminuição do tempo de resposta da consulta, mas exige maior espaço de armazenamento. No segundo caso é praticamente impossível pré-computar os valores, uma vez que seria necessário manter atualizada uma tabela com os valores de relacionamentos entre conjuntos de objetos, com todas as conseqüências que isto traz.

1.3 RELACIONAMENTOS TOPOLÓGICOS

[EH90] apresenta uma teoria que provê um estudo completo sobre as restrições topológicas binárias entre regiões sem buracos, baseada em conceitos algébricos e na teoria de conjuntos. O estudo compara as interseções das fronteiras (representado pelo símbolo δ) e interiores (representado pelo símbolo \circ) de duas regiões. Os resultados válidos para estas interseções são *vazio* ou *não vazio*. Existem 16 (2^4) possíveis combinações para estes relacionamentos, apresentadas na tabela 5.1. Destas combinações, oito não são válidas e duas são simétricas, resultando assim seis relacionamentos válidos entre regiões: *disjoint*, *in*, *touch*, *equal*, *cover* e *overlap*. Este método é conhecido como método das 4-interseções.

$\delta A \cap \delta B$	$\delta A \cap B^\circ$	$A^\circ \cap \delta B$	$A^\circ \cap B^\circ$	nome do relacionamento
0	0	0	0	A disjoint B
0	0	0	$\neg 0$	
0	0	$\neg 0$	0	
0	0	$\neg 0$	$\neg 0$	A in B
0	$\neg 0$	0	0	
0	$\neg 0$	0	$\neg 0$	A in B
0	$\neg 0$	$\neg 0$	0	
0	$\neg 0$	$\neg 0$	$\neg 0$	
$\neg 0$	0	0	0	A touch B
$\neg 0$	0	0	$\neg 0$	A equal B
$\neg 0$	0	$\neg 0$	0	
$\neg 0$	0	$\neg 0$	$\neg 0$	A cover B
$\neg 0$	$\neg 0$	0	0	
$\neg 0$	$\neg 0$	0	$\neg 0$	A cover B
$\neg 0$	$\neg 0$	$\neg 0$	0	
$\neg 0$	$\neg 0$	$\neg 0$	$\neg 0$	A overlap B

Tabela 5.1. Casos possíveis de relacionamentos topológicos binários utilizando o método das 4-interseções.

Em [Egen91], Egenhofer estendeu as 4-interseções para o método das 9-interseções, que considera as interseções com o exterior (complemento) da região. Na figura 5.2 é descrita a matriz de 3x3 correspondente, onde B^- representa o exterior de B.

$$I_9(A, B) = \begin{pmatrix} A^\circ \cap B^\circ & A^\circ \cap \delta B & A^\circ \cap B^- \\ \delta A \cap B^\circ & \delta A \cap \delta B & \delta A \cap B^- \\ A^- \cap B^\circ & A^- \cap \delta B & A^- \cap B^- \end{pmatrix}$$

Figura 5.2. Representação do método das 9-interseções.

Baseados em [EH90], Hadzilacos e Tryfona [HT92] apresentaram exemplos da aplicação do método das 4-interseções para relacionamentos binários entre qualquer tipo de objeto elementar (ponto, linha e região) obtendo 16 possíveis relacionamentos. Um trabalho similar foi feito por Egenhofer e Herring em [EH92].

[EH90, Egen91, HT92, EH92] não consideram a dimensão da interseção, mas tão somente se esta é vazia ou não. O trabalho de Clementini et al. [CFO93], adotado na dissertação, mostra como resolver o problema. Ele considera a dimensão do resultado das interseções, utilizando o método das 4-interseções. A dimensão da interseção pode ser *vazia* (null), *pontual* (dimensão OG-0D), *linha* (OG-1D) ou *região* (OG-2D), gerando um total de 256 (4^4) relacionamentos, dos quais somente 52 são aplicáveis. [CFO93] mostra que estes

52 relacionamentos podem ser especificados pela combinação de operadores de fronteira (*from*, *to*, δ) e os relacionamentos topológicos mutuamente exclusivos *disjoint*, *touch*, *overlap*, *in* e *cross*.

Os operadores de fronteira *from* e *to* são utilizados para obter os extremos (pontos) de um objeto de tipo linha, enquanto que δ é utilizado para obter uma linha fechada que corresponde à fronteira de um objeto de tipo região.

Um relacionamento *r* é simétrico se e somente se $\langle A, r, B \rangle \Leftrightarrow \langle B, r, A \rangle$. Um relacionamento *r* é transitivo se e somente se $\langle A, r, B \rangle \wedge \langle B, r, C \rangle \Rightarrow \langle A, r, C \rangle$. Todos os relacionamentos são simétricos com exceção do relacionamento *in*, que por sua vez é o único transitivo. A seguir são apresentadas as definições destes cinco relacionamentos.

- O relacionamento *disjoint* é aplicável a todas as situações (figura 5.3).

$$\langle A, disjoint, B \rangle \Leftrightarrow A \cap B = \emptyset$$

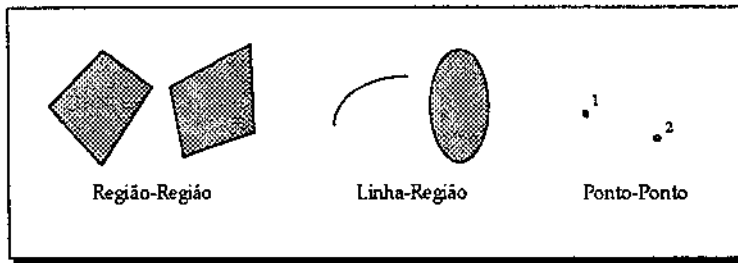


Figura 5.3. Exemplos do relacionamento *disjoint*.

- O relacionamento *overlap* somente é aplicável às situações região-região e linha-linha (figura 5.4).

$$\langle A, overlap, B \rangle \Leftrightarrow (\dim(A^\circ) = \dim(B^\circ) = \dim(A^\circ \cap B^\circ)) \wedge (A \cap B \neq A) \wedge (A \cap B \neq B)$$

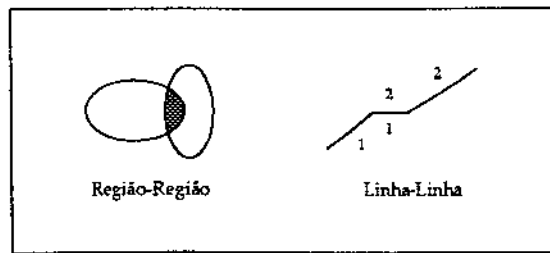


Figura 5.4. Exemplos do relacionamento *overlap*.

- O relacionamento *touch* não é aplicável somente para situações ponto-ponto (figura 5.5).

$$\langle A, touch, B \rangle \Leftrightarrow (A^\circ \cap B^\circ = \emptyset) \wedge (A \cap B \neq \emptyset)$$

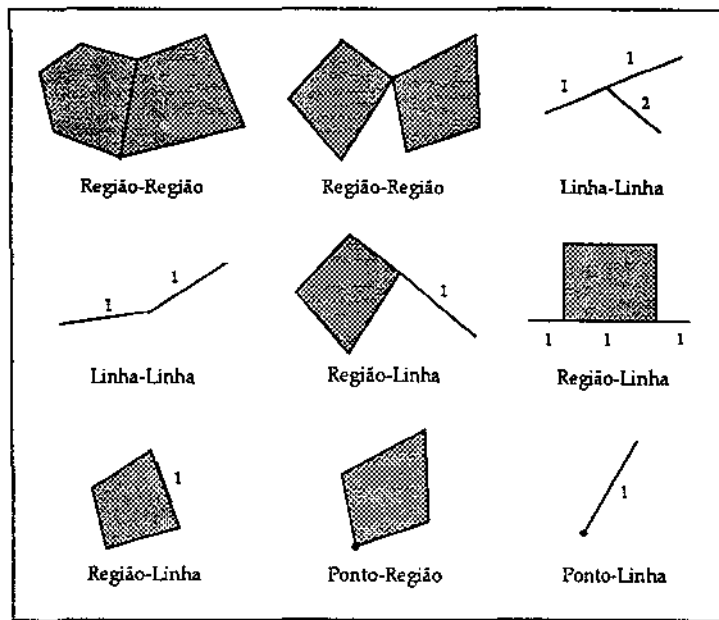


Figura 5.5. Exemplos do relacionamento *touch*.

- O relacionamento *in* é aplicável a todas as situações (figura 5.6).

$$\langle A, in, B \rangle \Leftrightarrow (A \cap B = A) \wedge (A^\circ \cap B^\circ \neq \emptyset)$$

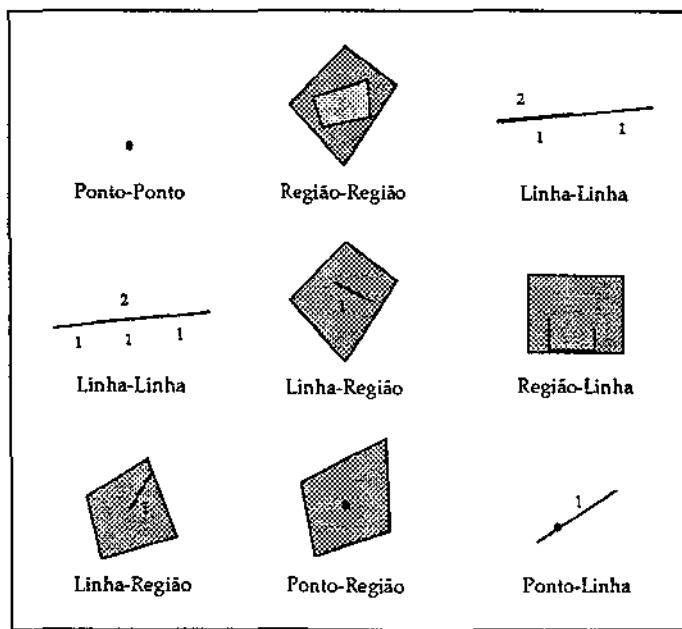


Figura 5.6. Exemplos do relacionamento *in*.

• O relacionamento *cross* somente é aplicável às situações linha-linha e linha-região (figura 5.7).

$$\langle A, \text{cross}, B \rangle \Leftrightarrow (\dim(A^\circ \cap B^\circ) = (\max(\dim(A^\circ), \dim(B^\circ)) - 1)) \\ \wedge (A \cap B \neq A) \wedge (A \cap B \neq B)$$



Figura 5.7. Exemplos do relacionamento *cross*.

2. ESPECIFICAÇÃO DE RESTRIÇÕES TOPOLÓGICAS

Os relacionamentos básicos apresentados na seção anterior podem ser utilizados para a construção de restrições binárias de integridade topológicas complexas. Estas são construídas a partir do conjunto de predicados topológicos binários junto com a conjunção (\wedge), a disjunção (\vee) e os quantificadores existencial (\exists) e universal (\forall).

2.1 PREDICADO TOPOLÓGICO

Nesta dissertação, os predicados topológicos binários são especificados sobre instâncias de classes, expressas como variáveis ligadas a quantificadores (\exists , \forall) e objetos geográficos específicos (*named objects*).

Os relacionamentos entre variáveis e objetos são especificados através dos predicados [CFO93] *disjoint*, *touch*, *overlap*, *in*, *cross* e operadores de fronteira (*from*, *to* e δ).

Além dos relacionamentos topológicos é considerado o predicado de orientação *same_position*, que especifica que dois objetos geográficos ocupam a mesma posição.

Alguns exemplos de predicados topológicos são:

• Para cada elemento *tri* da classe Triângulo, existe pelo menos um elemento *qua* da classe Quadrilátero, que se sobrepõe espacialmente a *tri* (figura 5.8.a).

$$\forall \text{tri} \in \text{Triângulo}, \exists \text{qua} \in \text{Quadrilátero}: \text{overlap}(\text{tri}, \text{qua})$$

• Dados um conjunto de Linhas e Regiões, existe pelo menos um par (*lin*, *reg*) tal que *lin cross reg* (figura 5.8.b).

$$\exists \text{lin} \in \text{Linha}, \exists \text{reg} \in \text{Região}: \text{cross}(\text{lin}, \text{reg})$$

- Os *named objects* paran e so_paulo, ambos pertencentes a classes geogrficas, so adjacentes (figura 5.8.c).

touch(paran, so_paulo)

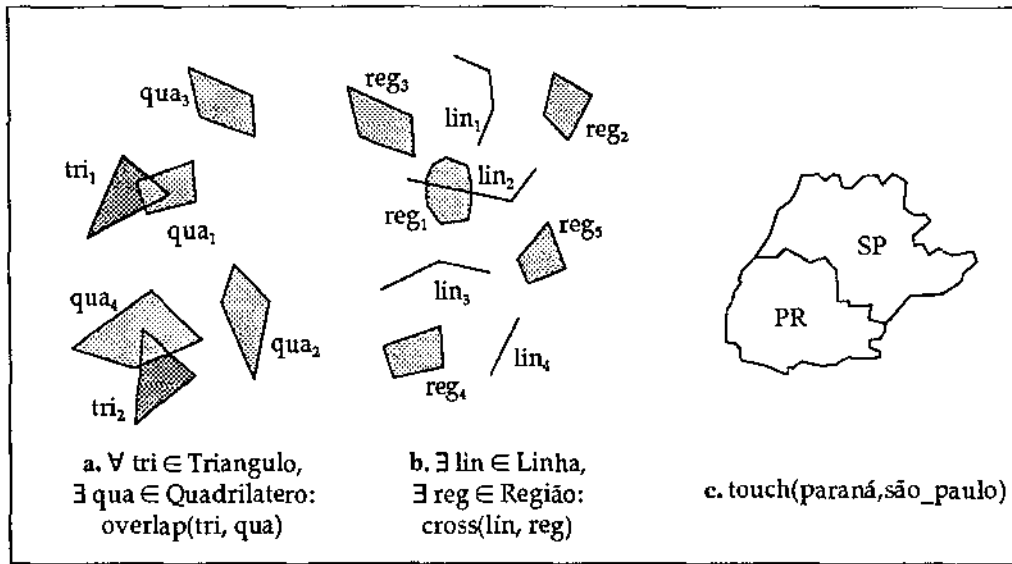


Figura 5.8. Exemplos de predicados topolgicos.

2.2 SENTENA TOPOLGICA

Uma sentena topolgica, tambm chamada restrio topolgica,  definida como:

1. Todo predicado topolgico  uma sentena topolgica.
2. Se P  uma sentena topolgica ento (P) tambm .
3. Se P1 e P2 so sentenas topolgicas ento $P1 \wedge P2$ e $P1 \vee P2$ tambm so.
4. Na ausncia de parntesis a prioridade  \wedge, \vee .
5. Nada mais  uma sentena topolgica.

Como exemplo, pode ser definida uma restrio topolgica descrevendo que para todo *lin* pertencente  classe Linha, existe um objeto *reg* da classe Regio que faz *touch* com *lin*, mas ambos extremos de *lin* no intersectam *reg* (figura 5.9).

$$\forall \text{lin} \in \text{Linha}, \exists \text{reg} \in \text{Regio}: \text{touch}(\text{lin}, \text{reg}) \wedge \text{disjoint}(\text{to}(\text{lin}), \text{reg}) \wedge \text{disjoint}(\text{from}(\text{lin}), \text{reg})$$

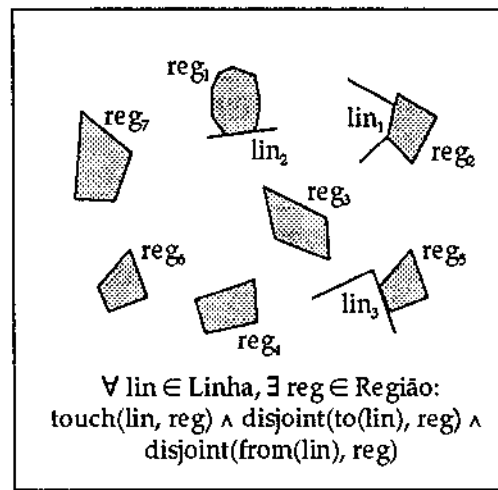


Figura 5.9. Exemplo de sentença topológica.

2.3 LINGUAGEM PARA ESPECIFICAR RESTRIÇÕES TOPOLÓGICAS BINÁRIAS

É apresentada aqui a linguagem de especificação de restrições topológicas binárias para sistemas geográficos baseados em um modelo OO. Esta linguagem permite especificar restrições inter e intra-classe, e inter-objeto. A seguir é apresentada a gramática para esta linguagem utilizando BNF estendido.

<Restrição>	::= <Nome> :: <Universo> : <Predicado>
<Universo>	::= <DefUniverso> , <DefUniverso>
<DefUniverso>	::= <NamedObject> <Quant> <LocalVar> ∈ <Classe>
<Quant>	::= ∀ ∃
<Predicado>	::= <PredTop> <PredTop> <Conector> <Predicado> (<Predicado>)
<Conector>	::= ∧ ∨
<PredTop>	::= <TopRel> (<ObjGeo> , <ObjGeo>)
<TopRel>	::= disjoint touch overlap in cross samepos
<ObjGeo>	::= <OpFronteira> (<var>) <var>
<OpFronteira>	::= from to bound
<var>	::= <LocalVar> <NamedObject>
<Nome>	::= <Identificador>
<NamedObject>	::= <Identificador>
<LocalVar>	::= <Identificador>
<Classe>	::= <Identificador>
<Identificador>	::= <letra> { <letra> <digito> }
<letra>	::= a b c d ... z
<digito>	::= 0 1 2 ... 9

Outros detalhes sobre a linguagem são descritos no apêndice B.

3. TRANSFORMAÇÃO DE RESTRIÇÕES TOPOLÓGICAS EM REGRAS ECA

Como descrito no capítulo 2, os bancos de dados ativos são uma solução adequada e flexível para a manutenção de restrições. Dentro deste contexto, esta seção propõe uma solução para o problema de determinação automática do conjunto de regras que manterão a consistência de uma determinada restrição topológica.

Para tanto, esta dissertação adota as seguintes considerações:

- O banco de dados está em um estado consistente antes de ser executado um conjunto de operações de atualização (descritas dentro de uma transação), e seu estado deve permanecer consistente depois que estas operações foram executadas.
- A análise de consistência deve ser feita antes do *commit* da transação. Isto implica que antes de ser executado o *commit* serão avaliadas/executadas as regras correspondentes à verificação da consistência.

3.1 IDÉIAS CENTRAIS DA TRANSFORMAÇÃO

O objetivo desta transformação é a derivação automática dos eventos e predicados (consultas) necessários para manter uma determinada restrição. Para isto são considerados os seguintes passos:

1. Analisar a restrição R , escrita na linguagem proposta, utilizando o esquema do banco de dados (classes e objetos especificados na restrição).
2. Para cada predicado topológico que compõe a restrição, derivar automaticamente o conjunto de eventos que podem violá-lo.
3. A partir deste predicado, derivar as condições formuladas como consultas.
4. Gerar as regras para manter a restrição correspondente.

O primeiro passo, correspondente à análise sintática, é descrito no apêndice B, enquanto que os demais são descritos nas próximas seções.

3.2 EVENTOS QUE PODEM VIOLAR UM PREDICADO TOPOLÓGICO

A análise das operações que podem violar um predicado tem por objetivo evitar verificações desnecessárias do predicado, melhorando assim o desempenho do sistema.

Nesta dissertação, os únicos eventos considerados são os correspondentes às operações de atualização (*insert*, *update*, *delete*). Exemplos de modificação (*update*) de objetos geográficos que podem violar restrições espaciais são a operação de *move* (translação ou rotação) que altera alguma das coordenadas da localização, ou operações do tipo *expand* ou *shrink* que mudam a geometria do objeto. A tabela 5.2 apresenta o conjunto de eventos que podem violar um predicado conforme o quantificador associado.

$\exists o \in C$	$\forall o \in C$	Named Object O
C.delete o C.update o	C.insert o C.update o	O.insert O.update O.delete

Tabela 5.2. Eventos que podem violar um predicado topológico.

A tabela indica, por exemplo, que se um predicado tem uma variável ligada a um quantificador existencial ($\exists o \in C$), então os eventos que podem violá-lo são *C.delete* e *C.update* sobre os objetos da classe C. Se outra variável do mesmo predicado estiver associada a um quantificador universal ($\forall o \in C$) então atualizações do tipo *insert* ou *update* na classe C podem violá-lo. Desta forma, o predicado *TopRel* da restrição:

$$\forall o_1 \in C_1, \exists o_2 \in C_2: \text{TopRel}(o_1, o_2)$$

pode ser violado se houver modificação em objetos das classes C_1 ou C_2 ; ou se um objeto de C_2 for eliminado; ou se um objeto for inserido em C_1 .

3.3 CONDIÇÕES PARA VERIFICAR UM PREDICADO TOPOLÓGICO

A semântica de um predicado topológico binário varia conforme os quantificadores associados às variáveis. Na seção anterior foram apresentados os conjuntos de eventos que podem violar um predicado topológico, segundo seus quantificadores. A tabela 5.3 mostra, de forma resumida, como determinar automaticamente: i) os eventos passíveis de violar um predicado topológico, e ii) as consultas que verificam a condição correspondente. Os elementos b_i e a_i denotam *named objects*. *TopRel* denota um relacionamento topológico (*disjoint, touch, overlap, in, cross*). Esta tabela é simétrica com respeito à triangulação superior.

Para poder determinar as consultas que verificam o predicado é necessário analisar as variáveis e seus respectivos quantificadores. Como se trata de predicados binários (P) esta análise é feita aos pares, onde a combinação de linhas e colunas indica os tipos de predicado. O elemento (1,1) da tabela 5.3, por exemplo, se refere a um predicado onde ambas variáveis estão ligadas a um quantificador existencial. Neste caso, eventos que violam o predicado podem compreender tanto a atualização de objetos da classe A quanto da classe B. Os eventos são obtidos utilizando a função *ev()* que consulta a entrada correspondente na tabela 5.2. Ainda no mesmo exemplo, *ev*($\exists A$) pode ser uma atualização do tipo *delete* ou *update* (que afeta variáveis ligadas existencialmente). A condição a ser testada envolve o predicado binário P , correspondente à condição da regra. Com estas informações podem ser geradas as regras *<when evento if not(P) then ação>* para manutenção de um predicado topológico binário P .

	$\exists b \in B$		$\forall b \in B$		b_i	
	Evento	Predicado	Evento	Predicado	Evento	Predicado
$\exists a \in A$	ev($\exists a$) ev($\exists b$)	exist1(A, TopRel, B)				
$\forall a \in A$	ev($\forall a$)	exist2(a, TopRel, B)	ev($\forall a$)	all2(a, TopRel, B)		
	ev($\exists b$)	all1(A, TopRel, B)	ev($\forall b$)	all2(b, TopRel, A)		
a_i	ev(a_i)	exist2(a_i , TopRel, B)	ev(a_i)	all2(a_i , TopRel, B)	ev(a_i)	n_o(a_i , TopRel, b_i)
	ev($\exists b$)	n_o(a_i , TopRel, b)	ev($\forall b$)	n_o(a_i , TopRel, b)	ev(b_i)	

Tabela 5.3. Consultas que representam o predicado topológico, segundo a ligação de suas variáveis. Também são descritos os eventos que ativam a avaliação do predicado.

O predicado P , corresponde a uma consulta. Todas estas consultas são expressas em SQL espacial [Egen94], onde o relacionamento topológico binário $TopRel$ é explicitado na cláusula *where*. Estes relacionamentos são aplicáveis aos atributos espaciais (*loc*, *from*, *to* e *bound*). Se não especificado, o atributo *loc* é utilizado na verificação do relacionamento $TopRel$. Caso contrario, esta verificação deve ser feita através dos atributos *from/to/bound*.

Neste trabalho, uma consulta SQL é considerada verdadeira se resulta em uma resposta diferente do conjunto vazio. Cada um dos predicados da tabela 5.3 é resolvido como segue:

- exist1(A, TopRel, B). Verifica a existência de pelo menos um par de objetos $a \in A$, $b \in B$ em que seja verdadeiro o relacionamento ($a TopRel b$).

```
Select A.id from A, B where
A.loc TopRel B.loc
```

- exist2(a, TopRel, B). Dado um objeto $a \in A$, verifica a existência de pelo menos um objeto $b \in B$ tal que ($a TopRel b$).

```
Select B.id from A, B where
A.id=a and A.loc TopRel B.loc
```

- n_o(a, TopRel, b). Verifica a existência do relacionamento $TopRel$ entre os elementos a e b .

```
Select A.id from A, B where
A.id=a and B.id=b and A.loc TopRel B.loc
```

- all1(A, TopRel, B)¹. Verifica a existência de um relacionamento $TopRel$ entre todos os elementos da classe A e todos os elementos da classe B. Isto também pode ser verificado se existe algum elemento da classe A que *não* tem um relacionamento $TopRel$ com algum outro elemento da classe B.

¹ Neste caso, a consulta SQL é verdadeira quando resulta em uma resposta vazia.

Select A.id from A, B where A.loc not *TopRel* B.loc

• $\text{all2}(a, \text{TopRel}, B)^1$. Verifica a existência de um relacionamento *TopRel* entre a e todos os elementos da classe B . Isto também pode ser verificado se *não* existe um relacionamento *TopRel* entre o elemento a e algum elemento da classe B .

Select B.id from A, B where
A.id=a and A.loc not *TopRel* B.loc

Veja que a consulta correspondente a $\text{n_o}(a, \text{TopRel}, b)$ poderia ser expressa simplesmente como:

bool := *TopRel*(a.loc, b.loc)

mas, por questões de uniformidade, todas as consultas são expressas em SQL espacial.

Exemplo

O uso das tabelas para análise de um predicado topológico binário é descrita com o exemplo da figura 5.8.a. A restrição que indica que todo triângulo se sobrepõe a pelo menos um quadrilátero é:

$\forall \text{tri} \in \text{Triângulo}, \exists \text{qua} \in \text{Quadrilátero}: \text{overlap}(\text{tri}, \text{qua})$

Para cada variável (var) utilizada na especificação do predicado é utilizada a tupla Pred_{var} que mantém os eventos ($\text{Pred}_{\text{var}}.\text{evento}$) e a consulta ($\text{Pred}_{\text{var}}.\text{cons}$) correspondentes aos elementos da tabela 5.3.

O par de variáveis (tri, qua) tem associado os quantificadores (\forall, \exists) respectivamente. Isto corresponde à utilização do elemento (2,1) da tabela 5.3. Para obter os eventos que podem violá-lo, as referências à tabela 5.2 são resolvidas (mediante a função $\text{ev}()$), obtendo como resultado:

$\text{Pred}_{\text{tri}}.\text{evento} = \text{ev}(\forall \text{tri}) = \{\text{Triângulo.insert}, \text{Triângulo.update}\}$

$\text{Pred}_{\text{qua}}.\text{evento} = \text{ev}(\exists \text{qua}) = \{\text{Quadrilátero.delete}, \text{Quadrilátero.update}\}$

As consultas correspondentes para cada uma destas variáveis são obtidas da tabela 5.3, resultando em:

$\text{Pred}_{\text{tri}}.\text{cons} = \text{exist2}(\text{tri}, \text{overlap}, \text{Quadrilátero}) =$
Select Quadrilátero.id from Triângulo, Quadrilátero
where Triângulo.id=tri and
Triângulo.loc *overlap* Quadrilátero.loc

$\text{Pred}_{\text{qua}}.\text{cons} = \text{all1}(\text{Triângulo}, \text{overlap}, \text{Quadrilátero}) =$
Select Triângulo.id from Triângulo, Quadrilátero
where Triângulo.loc not *overlap* Quadrilátero.loc

3.4 PREDICADOS TOPOLÓGICOS INTRA-CLASSE

Foram tratados até aqui os predicados topológicos binários entre elementos de classes diferentes. Um predicado topológico que envolve elementos de uma única classe é chamado

de predicado intra-classe. Estes predicados são tratados da mesma forma que os predicados inter-classe, ou seja, consultando as tabelas 5.2 e 5.3, com algumas considerações adicionais para a resolução das consultas.

O uso de *alias* na cláusula *from* (do SQL) mantém a mesma estrutura das consultas. A verificação dos relacionamentos topológicos entre os elementos de uma mesma classe devem excluir a verificação do mesmo elemento contra ele próprio. Para isto, é incluído na seção *where* a verificação correspondente. Por exemplo, para predicados topológicos intra-classe a consulta *exist1()* é resolvida como segue:

```
exist1(A, TopRel, A): Select A.id from A, A as B where
                        A.loc TopRel B.loc and not(A.id =B.id)
```

Esta consulta verifica a existência de pelo menos um relacionamento *TopRel* entre os elementos da classe A, excluindo a verificação do auto-relacionamento.

A resolução das outras consultas (*exist2*, *n_o*, *all1*, *all2*) são tratadas de forma similar.

Exemplo

Seja a restrição que afirma que nunca há interseção entre triângulos, representado como segue:

$$\forall t1, t2 \in \text{Triângulo}: \text{disjoint}(t1, t2)$$

O par de variáveis (*t1*, *t2*) tem associado os quantificadores (\forall , \forall) respectivamente. Segundo estes quantificadores, o elemento (2,2) da tabela 5.3 deve ser utilizado. Como ambas variáveis estão associadas ao mesmo quantificador e à mesma classe, a determinação dos eventos e condições pode ser simplificada analisando somente uma destas variáveis. Como em todos os casos, a referência a eventos é resolvida consultando a tabela 5.2, obtendo como resultado:

$$\text{Pred}_{t1}.\text{evento} = \text{ev}(\forall t1) = \{\text{Triângulo.insert}, \text{Triângulo.update}\}$$

A consulta respectiva é resolvida como segue:

```
Predt1.cons = all2(t1, disjoint, Triângulo) =
Select Triângulo1.id from Triângulo, Triângulo as Triângulo1
where Triângulo.id=p1 and
      Triângulo.loc not disjoint Triângulo1.loc
and not(Triângulo1.id=t1)
```

3.5 TRATAMENTO DO ESTADO INICIAL

O conjunto de estados consistentes para um par de classes (em um banco de dados) inclui o “estado inicial”, que representa o estado onde não existem elementos em ambas classes. Para efeito deste trabalho é importante fazer esta distinção, pois a transição de/para este estado deve ser monitorada com especial atenção, como será descrito nesta seção.

A evolução consistente de um par de classes (em um banco de dados) pode ser representada por uma máquina de estado finitos (figura 5.10). Nesta figura os estados e as

transações são representados por nós e arestas, respectivamente. O estado consistente agrupa todos os estados consistentes possíveis (entre duas classes) com exceção do “estado inicial”.

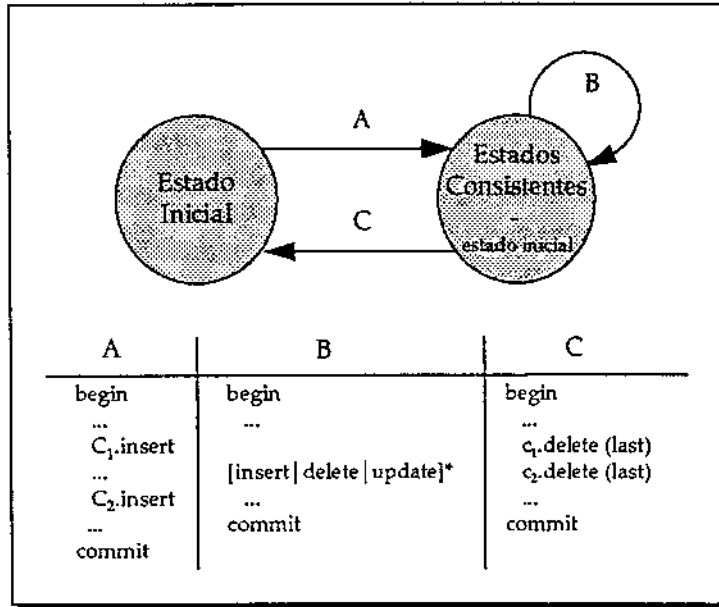


Figura 5.10. Máquina de estados que representa a evolução consistente.

A evolução de um estado consistente para outro, mediante a transição rotulada B, foi tratada nas seções anteriores. Aqui são considerados as transições A e C que representam: i) a evolução do “estado inicial” para o estado consistente e ii) a evolução do estado consistente para o “estado inicial”.

Para o primeiro caso, devem ser monitoradas a primeira operação de *insert* correspondente às classes envolvidas. A única forma de passar do “estado inicial” para o estado consistente é inserindo pelo menos um elemento em cada classe. Esta situação é descrita na figura 5.10 com a flecha rotulada A. É importante lembrar que o evento *insert* é monitorado somente quando estão envolvidos o quantificador universal ou *named objects* (conforme mostra a tabela 5.2). Note que em predicados topológicos que envolvem somente quantificadores existenciais o *insert* (das classes correspondentes) não é monitorado.

Considere o exemplo da figura 5.8.b:

$$\exists \text{lin} \in \text{Linha}, \exists \text{reg} \in \text{Região}: \text{cross}(\text{lin}, \text{reg})$$

Depois da análise do predicado topológico utilizando o elemento (1,1) da tabela 5.3 para a obtenção de eventos e condições, resulta:

$$\begin{aligned} \text{Pred}_{\text{lin}, \text{evento}} &= \{\text{ev}(\exists \text{lin}), \text{ev}(\exists \text{reg})\} \\ &= \{\text{Linha.delete}, \text{Linha.update}, \text{Região.delete}, \text{Região.update}\} \\ \text{Pred}_{\text{lin}, \text{cons}} &= \text{exist1}(\text{Linha}, \text{cross}, \text{Região}) \\ \text{Pred}_{\text{reg}, \text{evento}} &= \{\text{ev}(\exists \text{lin}), \text{ev}(\exists \text{reg})\} \end{aligned}$$

$$\begin{aligned} &= \{ \text{Linha.delete}, \text{Linha.update}, \text{Região.delete}, \text{Região.update} \} \\ \text{Pred}_{\text{reg.cons}} &= \text{exist1}(\text{Linha}, \text{cross}, \text{Região}) \end{aligned}$$

Como não são monitorados os eventos *insert* em ambas classes, se inseridos alguns elementos (em uma ou ambas classes) não é verificado o predicado topológico que diz que existe pelo menos um par de elementos (lin,reg) que tem um relacionamento topológico *cross*.

Para resolver este problema é necessário monitorar a inserção do primeiro elemento no caso de variáveis com ligação a quantificador existencial. Assim, o evento *insert_first*² de ambas classes deve ser monitorado. A consulta respectiva é a mesma que utilizada para os outros eventos.

Para cada variável (var), em um predicado topológico, a tupla $\text{Inicial}_{\text{var}}$ representa os eventos e condições que devem manter a consistência considerando o estado inicial (flechas A e C da figura 5.10, para o primeiro e o segundo caso respectivamente). Para o exemplo apresentado anteriormente, estes eventos e condições são:

$$\begin{aligned} \text{Inicial}_{\text{lin.evento}} &= \{ \text{Linha.insert_first} \} \\ \text{Inicial}_{\text{lin.cons}} &= \text{exist1}(\text{Linha}, \text{cross}, \text{Região}) \\ \text{Inicial}_{\text{reg.evento}} &= \{ \text{Região.insert_first} \} \\ \text{Inicial}_{\text{reg.cons}} &= \text{exist1}(\text{Linha}, \text{cross}, \text{Região}) \end{aligned}$$

Um *delete*, um *update* ou um *insert_first* na classe *Linha* ativarão a verificação da consulta. Esses mesmos eventos, mas da classe *Região*, também ativarão a mesma consulta.

O **segundo caso** representa a situação onde o último elemento de uma classe (envolvida em um predicado topológico) é eliminado, ou seja, quando a operação *delete_last*³ for aplicada. Considere um predicado topológico que envolve as classes A e B. Neste caso deve ser monitorado o evento *A.delete_last*, onde deve ser verificado que também não existam elementos na classe B. Esta situação é identificada na figura 5.10 com a flecha (rotulada C) indo do estado consistente ao “estado inicial”.

Considere o exemplo da figura 5.8.a, que diz que todos os triângulos devem ter pelo menos um quadrilátero sobreposto. É importante destacar que podem existir quadriláteros sem que existam triângulos, mas não pode haver triângulos sem pelo menos um quadrilátero que o sobreponha.

$$\forall \text{tri} \in \text{Triângulo}, \exists \text{qua} \in \text{Quadrilátero}: \text{overlap}(\text{tri}, \text{qua})$$

Os eventos e condições correspondentes são:

$$\begin{aligned} \text{Pred}_{\text{tri.evento}} &= \text{ev}(\forall \text{tri}) = \{ \text{Triângulo.insert}, \text{Triângulo.update} \} \\ \text{Pred}_{\text{tri.cons}} &= \text{exist2}(\text{tri}, \text{overlap}, \text{Quadrilátero}) \\ \text{Pred}_{\text{qua.evento}} &= \text{ev}(\exists \text{qua}) = \{ \text{Quadrilátero.delete}, \text{Quadrilátero.update} \} \end{aligned}$$

² O evento *insert_first* representa a inserção de um elemento quando não existem outros em uma determinada classe.

³ O evento *delete_last* representa a eliminação do último elemento de uma determinada classe.

$$\text{Pred}_{\text{qua.cons}} = \text{all1}(\text{Triângulo}, \text{overlap}, \text{Quadrilátero})$$

Se eliminado o último elemento da classe Triângulo, e ainda existem quadriláteros, o predicado ainda é mantido. Por outro lado, se eliminado o último quadrilátero, não podem existir triângulos. Por esta razão, deve ser monitorado com especial atenção o evento Quadrilátero.*delete_last*, cuja consulta associada (*empty*) deve verificar a inexistência de elementos em ambas classes.

$$\text{Inicial}_{\text{qua.evento}} = \{\text{Quadrilátero.delete_last}\}$$

$$\text{Inicial}_{\text{qua.cons}} = \text{empty}(\text{Quadrilátero}, \text{Triângulo})$$

Esta situação deve ser considerada para todos os predicados onde o quantificador universal (\forall) esteja presente. Então, para um predicado do tipo:

$$\text{quantif}_a a \in A, \text{quantif}_b b \in B: \text{TopRel}(a, b)$$

Se $\text{quantif}_a = \forall$ então

$$\text{Inicial}_b.\text{evento} = \{B.\text{delete_last}\}$$

$$\text{Inicial}_b.\text{cons} = \text{empty}(A, B)$$

Se $\text{quantif}_b = \forall$ então

$$\text{Inicial}_a.\text{evento} = \{A.\text{delete_last}\}$$

$$\text{Inicial}_a.\text{cons} = \text{empty}(A, B)$$

3.6 RESTRIÇÕES TOPOLÓGICAS COMPLEXAS

Restrições podem ter predicados compostos. Neste caso, o tratamento é feito da seguinte forma:

1. A restrição R é dividida em predicados atômicos $\text{Pred}[1], \text{Pred}[2], \dots, \text{Pred}[n]$ sendo criada uma estrutura de tipo árvore que mantém a associação (*and* e *or*) entre estes predicados. Os conectivos lógicos ocupam os nós internos desta árvore.

2. Para cada predicado $\text{Pred}[i]$ que compõe uma restrição topológica composta R :

2.1. Utilizando as tabelas 5.2 e 5.3 é feita uma análise de $\text{Pred}[i]$, que envolve duas variáveis (a e b), sendo obtidos os eventos e as consultas necessárias para sua manutenção ($\text{Pred}[i]_a$ e $\text{Pred}[i]_b$).

2.2. $\text{Pred}[i]$ é analisado considerando as transições de/para o estado inicial, obtendo como resultado $\text{Inicial}[i]_a$ e $\text{Inicial}[i]_b$.

3. O resultado são quatro conjuntos de tuplas ($i=1..n$)

$$G\text{Pred}_a = \{ \langle \text{Pred}[i]_a.\text{evento}, \text{Pred}[i]_a.\text{cons} \rangle \}$$

$$G\text{Pred}_b = \{ \langle \text{Pred}[i]_b.\text{evento}, \text{Pred}[i]_b.\text{cons} \rangle \}$$

$$G\text{Inicial}_a = \{ \langle \text{Inicial}[i]_a.\text{evento}, \text{Inicial}[i]_a.\text{cons} \rangle \}$$

$$G\text{Inicial}_b = \{ \langle \text{Inicial}[i]_b.\text{evento}, \text{Inicial}[i]_b.\text{cons} \rangle \}$$

Cada conjunto de tuplas dá origem a uma árvore ($GPred_a$, $GPred_b$, $GInicial_a$, $GInicial_b$), onde as folhas são elementos das tuplas.

Como exemplo, considere a restrição da figura 5.9:

$$R :: \forall \text{lin} \in \text{Linha}, \exists \text{reg} \in \text{Região}: \\ \text{touch}(\text{lin}, \text{reg}) \wedge \text{disjoint}(\text{to}(\text{lin}), \text{reg}) \wedge \text{disjoint}(\text{from}(\text{lin}), \text{reg})$$

Passo 1. A restrição R é dividida em três predicados atômicos, criando uma estrutura de tipo árvore (figura 5.11):

$$\begin{aligned} \text{Pred}[1] &= \text{touch}(\text{lin}, \text{reg}) \\ \text{Pred}[2] &= \text{disjoint}(\text{to}(\text{lin}), \text{reg}) \\ \text{Pred}[3] &= \text{disjoint}(\text{from}(\text{lin}), \text{reg}) \end{aligned}$$

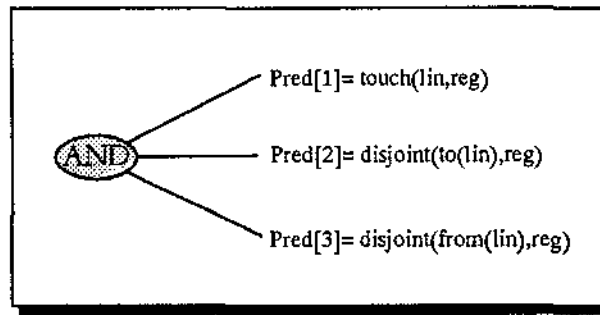


Figura 5.11. Árvore correspondente ao passo 1.

Passo 2. Análise de cada predicado $\text{Pred}[i]$

Passo 2.1. Obtenção dos eventos (tabela 5.2) e consultas utilizando o elemento (2,1) da tabela 5.3 (pois a restrição envolve quantificadores \forall e \exists).

$$\begin{aligned} \text{Pred}[1]_{\text{lin.evento}} &= \text{ev}(\forall \text{lin}) = \{\text{Linha.insert}, \text{Linha.update}\} \\ \text{Pred}[1]_{\text{lin.cons}} &= \text{exist2}(\text{lin}, \text{touch}, \text{Região}) \\ \text{Pred}[1]_{\text{reg.evento}} &= \text{ev}(\exists \text{reg}) = \{\text{Região.delete}, \text{Região.update}\} \\ \text{Pred}[1]_{\text{reg.cons}} &= \text{all1}(\text{Linha}, \text{touch}, \text{Região}) \\ \\ \text{Pred}[2]_{\text{lin.evento}} &= \text{ev}(\forall \text{lin}) = \{\text{Linha.insert}, \text{Linha.update}\} \\ \text{Pred}[2]_{\text{lin.cons}} &= \text{exist2}(\text{lin.to}, \text{disjoint}, \text{Região}) \\ \text{Pred}[2]_{\text{reg.evento}} &= \text{ev}(\exists \text{reg}) = \{\text{Região.delete}, \text{Região.update}\} \\ \text{Pred}[2]_{\text{reg.cons}} &= \text{all1}(\text{Linha.to}, \text{disjoint}, \text{Região}) \\ \\ \text{Pred}[3]_{\text{lin.evento}} &= \text{ev}(\forall \text{lin}) = \{\text{Linha.insert}, \text{Linha.update}\} \\ \text{Pred}[3]_{\text{lin.cons}} &= \text{exist2}(\text{lin.from}, \text{disjoint}, \text{Região}) \\ \text{Pred}[3]_{\text{reg.evento}} &= \text{ev}(\exists \text{reg}) = \{\text{Região.delete}, \text{Região.update}\} \\ \text{Pred}[3]_{\text{reg.cons}} &= \text{all1}(\text{Linha.from}, \text{disjoint}, \text{Região}) \end{aligned}$$

Passo 2.2. Análise do estado inicial.

Inicial[1]_{reg}.evento = {Região.delete_last}

Inicial[1]_{reg}.cons = empty(Linha,Região)

Inicial[2]_{reg}.evento = {Região.delete_last}

Inicial[2]_{reg}.cons = empty(Linha,Região)

Inicial[3]_{reg}.evento = {Região.delete_last}

Inicial[3]_{reg}.cons = empty(Linha,Região)

Passo 3. Preenchimento das estruturas de tipo árvore (figura 5.12).

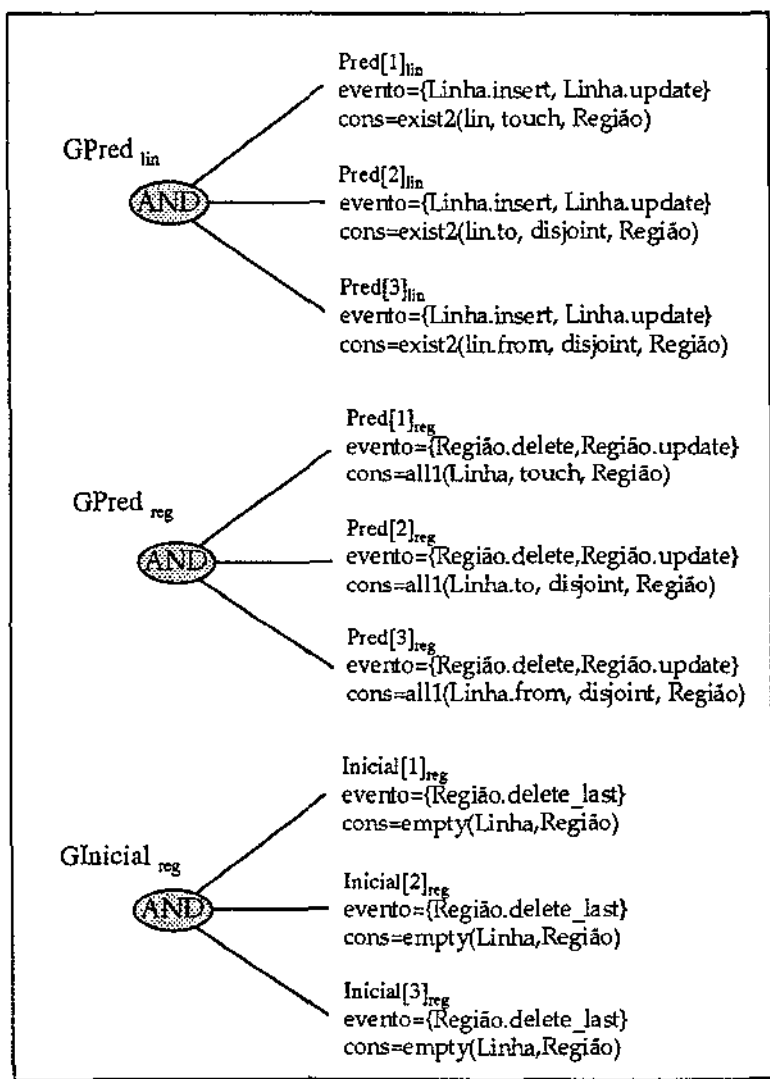


Figura 5.12. Árvores correspondentes ao passo 3.

A árvore correspondente a GInicial_{lin} não tem elementos, e por essa razão não está incluída na figura 5.12.

3.7 GERAÇÃO DE REGRAS

As regras que manterão uma restrição topológica R estão baseadas no resultado do algoritmo apresentado na seção anterior. Para cada estrutura de árvore (G_{pred_a} , G_{pred_b} , $Inicial_a$, $Inicial_b$) é gerada uma regra, e para cada uma delas devem ser obtidos os eventos e condições como segue. Se o predicado é atômico, então G_{pred_a} , G_{pred_b} , $G_{inicial_a}$ e $G_{inicial_b}$ só tem no máximo uma folha cada. Caso contrário, as árvores terão tantas folhas quantos forem os predicados atômicos da restrição.

Eventos. O conjunto de eventos E representa todos os atributos evento (das tuplas) correspondentes às folhas da árvore. Como qualquer um dos eventos em E pode violar a restrição, então a regra deve ter associada um evento composto disjunção (OR) que relacione todos os elementos do conjunto E .

Condição. É feito um percurso *inordem*, consultando o atributo *cons* correspondente às tuplas que estão nas folhas da árvore (somente considerando a cláusula *where*). Assim é gerada a consulta SQL correspondente.

A ação respectiva para todas as regras é o *abort*.

Continuando com o exemplo da seção anterior, são geradas as regras necessárias para a manutenção da restrição R , como apresentadas na figura 5.13.

```

Regra1 = when (Linha.insert(lin) or Linha.update(lin))
           if not( Select Região.id from Linha, Região where Linha.id=lin
                  and Linha.loc touch Região.loc and
                  Linha.to disjoint Região.loc and
                  Linha.from disjoint Região.loc )
           then {abort}

Regra2 = when (Região.delete(reg) or Região.update(reg))
           if ( Select Linha.id from Linha, Região where
                Linha.loc not touch Região.loc and
                Linha.to not disjoint Região.loc and
                Linha.from not disjoint Região.loc )
           then {abort}

Regra3 = when Região.delete_last(reg)
           if not( (Select Linha.id, Região.id from Linha, Região) or
                  not(Select Linha.id, Região.id from Linha, Região) )
           then {abort}

```

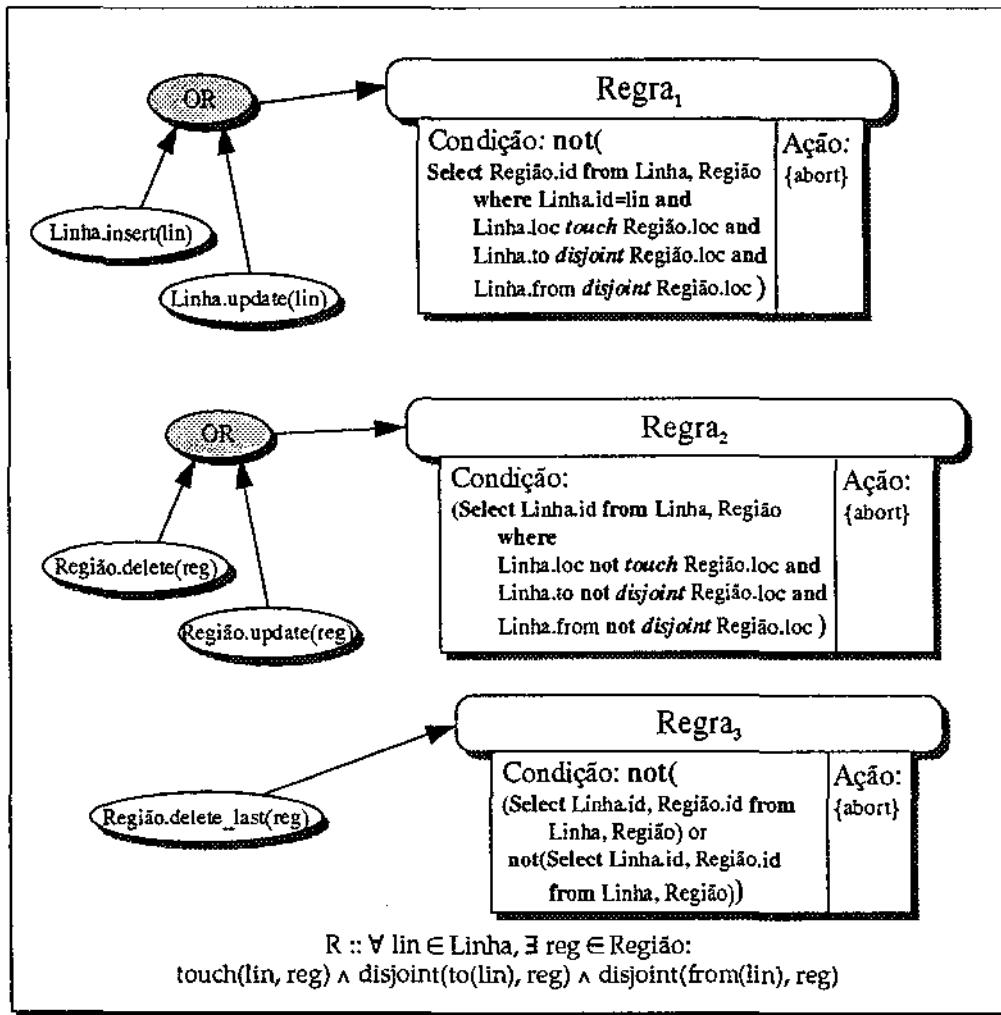


Figura 5.13. Regras E-C-A correspondentes à manutenção da restrição R (figura 5.9).

A condição correspondente ($\text{not}(P)$) é avaliada quando: i) uma linha é inserida ou modificada, ii) uma região é eliminada ou modificada, iii) o último elemento da classe região for eliminado. Se esta condição for verdadeira, então a ação *rollback* (ou *abort*) é executada.

Exemplos da transformação de restrições topológicas em regras são apresentadas no apêndice C.

VI

CONCLUSÕES E FUTURAS EXTENSÕES

1. CONCLUSÕES

Esta dissertação apresentou uma solução para o problema das restrições topológicas binárias em sistemas de informação geográfica, baseada na utilização do paradigma de regras E-C-A. A solução basicamente pode ser dividida em três etapas (figura 7.1):

- Especificação da restrição
- Tradução da restrição em regras E-C-A
- Manutenção da restrição utilizando um banco de dados ativo

Inicialmente, no capítulo 2, foram apresentados os conceitos sobre bancos de dados ativos, como também os principais protótipos nesta área.

No capítulo 3 foram descritos mecanismos para manutenção de restrições de integridade em SGBDs, como também os trabalhos de destaque na especificação e manutenção de restrições (convencionais) utilizando bancos de dados ativos. Ainda neste capítulo, foi descrito o mecanismo ativo implementado que dá suporte às regras E-C-A que mantêm as restrições.

O capítulo 4 apresentou uma visão geral dos SIG, relacionando as funcionalidades com o suporte de características ativas. Foi apresentado o modelo geográfico orientado a objetos utilizado para modelar aplicações geográficas e suas correspondentes restrições topológicas.

Os capítulos anteriores servem como uma base para resolver o problema da manutenção das restrições. No capítulo 5 foi apresentado um estudo sobre os relacionamentos espaciais, dando ênfase aos relacionamentos topológicos binários. Este capítulo descreve também a linguagem de especificação de restrições topológicas binárias e a sua correspondente tradução em regras E-C-A.

Dentre as principais contribuições deste trabalho, pode-se citar:

- estudo detalhado dos relacionamentos topológicos binários, a partir das principais fontes bibliográficas encontradas;
- desenvolvimento de um protótipo de sistema ativo que incorpora um modelo orientado a objetos espacial, eliminando o problema de impedância existente entre SIG e sistemas especialistas;
- definição de algoritmos para verificação de restrições de integridade topológica, com sua implementação no protótipo;
- tratamento integral do problema, desde a especificação até a manutenção.

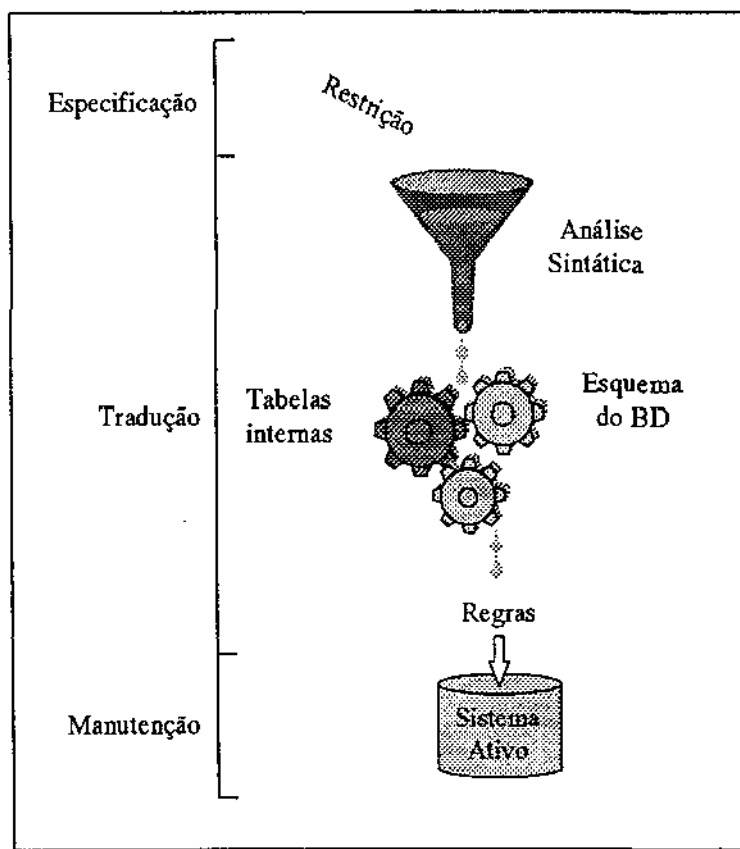


Figura 6.1. Etapas para a resolução do problema das restrições topológicas.

2. FUTURAS EXTENSÕES

As extensões a este trabalho podem ser teóricas ou práticas. Do ponto de vista teórico, podem ser desenvolvidos trabalhos como:

- extensão do estudo para manutenção de restrições topológicas não binárias;

- tratamento de outras restrições espaciais - métricas e de orientação;
- extensão da semântica dos relacionamentos binários para outros tipos de estruturas topológicas (por exemplo, estrutura arco-nó);
- extensão do trabalho para suporte a restrições espaciais em três dimensões.

Do ponto de vista prático, a primeira questão envolve o protótipo desenvolvido, em Smalltalk, que não é um SGBD ativo. Assim seria necessário reproduzir o trabalho em um SGBD real (por exemplo, SAMOS). Além disso, os testes realizados não utilizam dados reais, o que exigiria desenvolvimento de estruturas apropriadas para desempenho adequado. Finalmente alguns dos relacionamentos precisariam de rotinas específicas de geometria computacional que não foram implementadas.

APÊNDICE A

IMPLEMENTAÇÃO DO MODELO GEOGRÁFICO ADOTADO

A figura A.1 apresenta as classes e seus relacionamentos correspondentes ao modelo geográfico orientado a objetos adotado (descrito no capítulo 4).

A classe *GLocation* representa a localização/geometria de um objeto geográfico. Nela estão definidos os métodos que verificam um determinado relacionamento topológico (*disjoint*, *touch*, *overlap*, *in*, *cross*). *GLocation* é especializada em três outras classes:

- *Ponto*. Representado por um par de atributos (x, y) que indica seu posicionamento.

- *Linha*. Uma linha é uma lista ordenada de pontos, representando segmentos de reta. Para esta classe são incluídos os métodos *from* e *to*, que retornam os extremos de uma determinada linha.

- *Região*. Só são considerados polígonos convexos simples. Representada por uma lista ordenada de pontos, onde o primeiro ponto coincide com o último. É incluído o método *bound* que retorna uma linha que corresponde ao perímetro da região.

Busca Topológica

Devido à inexistência de uma linguagem de consulta espacial no protótipo, foram implementados os algoritmos de busca topológica necessários para este trabalho. A preocupação na escolha das estruturas de dados foi sobretudo guiada pela simplicidade, já que a implementação eficiente de estruturas de dados otimizadas foge aos objetivos do protótipo implementado.

Para facilitar a compreensão e implementação dos algoritmos, os relacionamentos topológicos binários são divididos em dois grupos: i) aqueles relacionamentos que implicam que dois objetos geográficos não tem pontos em comum ($TopRel_1 = \{disjoint\}$) e ii) aqueles que tem pontos em comum ($TopRel_2 = \{touch, overlap, in, cross\}$).

Para minimizar os custos (com relação ao tempo de execução) estes algoritmos verificam primeiro se os MBR dos objetos em questão se intersectam ou não. Os algoritmos de geometria computacional se aplicam somente aos objetos cujos relacionamentos entre MBRs indica a sua necessidade.

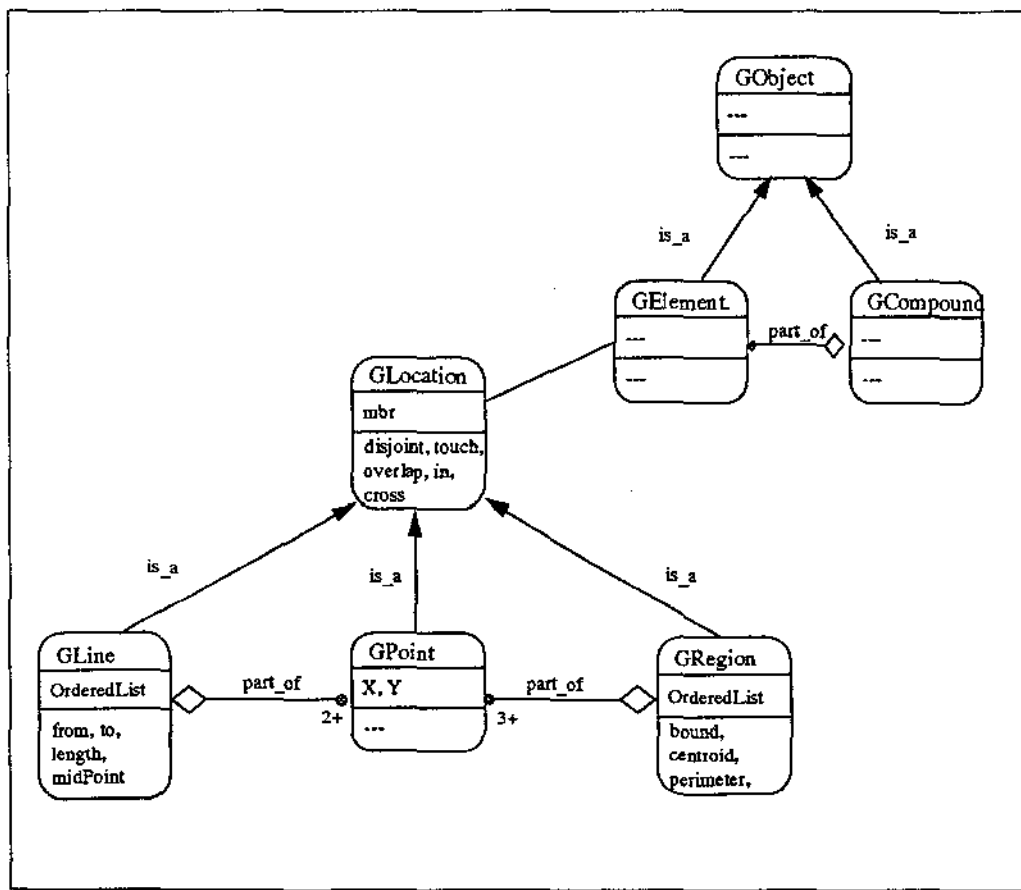


Figura A.1. Classes e relacionamentos correspondentes ao modelo adotado.

Os nomes dos algoritmos correspondem aos apresentados na tabela 5.3 do capítulo 5.

Para verificar a existência (exist2) ou não (all2) do relacionamento topológico do primeiro grupo (*disjoint*) são utilizados os algoritmos das figuras A.2 e A.3, respectivamente.

```

exist21(a, TopRel1, B):                                     /* disjoint */
  R := {};
  mbr_a := mbr(a.loc);
  foreach b in B
    if not intersect(mbr_a, mbr(b.loc)) then
      R := R add b;
    else
      if disjoint(a.loc, b.loc) then
        R := R add b;
  return R;

```

Figura A.2. Algoritmo que verifica a existência de um relacionamento topológico *disjoint* entre um objeto geográfico fonte (a) e um conjunto de objetos geográficos (B).

```

all21(a, not TopRel1, B):                                 /* not disjoint */
  R := {};
  mbr_a := mbr(a.loc);
  foreach b in B
    if intersect(mbr_a, mbr(b.loc)) then
      if not disjoint(a.loc, b.loc) then
        R := R add b;
  return R;

```

Figura A.3. Algoritmo que verifica a inexistência de um relacionamento topológico *disjoint* entre um determinado objeto (a) e um conjunto de objetos (B).

Para verificar a existência (*exist2*) ou não (*all2*) dos relacionamentos topológicos do segundo grupo (*touch*, *overlap*, *in*, *cross*) são utilizados os algoritmos das figuras A.4 e A.5.

```

exist22(a, TopRel2, B):                                   /* [touch, overlap, in, cross] */
  R := {};
  mbr_a := mbr(a.loc);
  foreach b in B
    if intersect(mbr_a, mbr(b.loc)) then
      if TopRel2(a.loc, b.loc) then
        R := R add b;
  return R;

```

Figura A.4. Algoritmo que verifica a existência de um relacionamento TopRel₂ entre um determinado objeto geográfico (a) e um conjunto de objetos geográficos (B).

```

all22(a, not TopRel2, B):                               /* not [touch, overlap, in, cross] */
  R := {};
  mbr_a := mbr(a.loc);
  foreach b in B
    if not intersect(mbr_a, mbr(b.loc)) then
      R := R add b;
    else
      if not TopRel2(a.loc, b.loc) then
        R := R add b;
  return R;

```

Figura A.5. Algoritmo que verifica a inexistência de um relacionamento TopRel₂ entre um determinado objeto geográfico (a) e um conjunto de objetos geográficos (B).

Os algoritmos exist2 e all2 verificam a existência ou não de um determinado relacionamento em relação a um objeto geográfico fonte (para os relacionamentos de ambos grupos). Além disso, estes algoritmos obtêm o conjunto de objetos geográficos que satisfazem o relacionamento correspondente.

Os algoritmos que verificam a existência (exist1) ou não (all1) de relacionamentos topológicos entre dois conjuntos de objetos geográficos baseiam-se nos algoritmos correspondentes, apresentados anteriormente.

```

exist1(A, TopRel, B):
  R := {};
  foreach a in A
    R := R add exist2(a, TopRel, B);
  return R;

```

Figura A.6. Algoritmo que verifica a existência de um relacionamento TopRel entre um par de conjuntos de objetos geográficos (A e B).

Os algoritmos de geometria computacional utilizados (e adaptados) neste protótipo pertencem à biblioteca *The Workbench for Computational Geometry* desenvolvidos na Universidade de Carleton, Canada.

APÊNDICE B

LINGUAGEM PARA ESPECIFICAR RESTRIÇÕES TOPOLÓGICAS

Neste apêndice é descrita a gramática e os diagramas sintáticos correspondentes à linguagem de especificação de restrições topológicas binárias para sistemas geográficos em um modelo OO. Esta linguagem permite especificar restrições inter e intra-classe, e inter-objeto.

A seguir é apresentada a gramática para esta linguagem utilizando BNF estendido. Os não terminais da gramática são nomes colocados entre parênteses angulares.

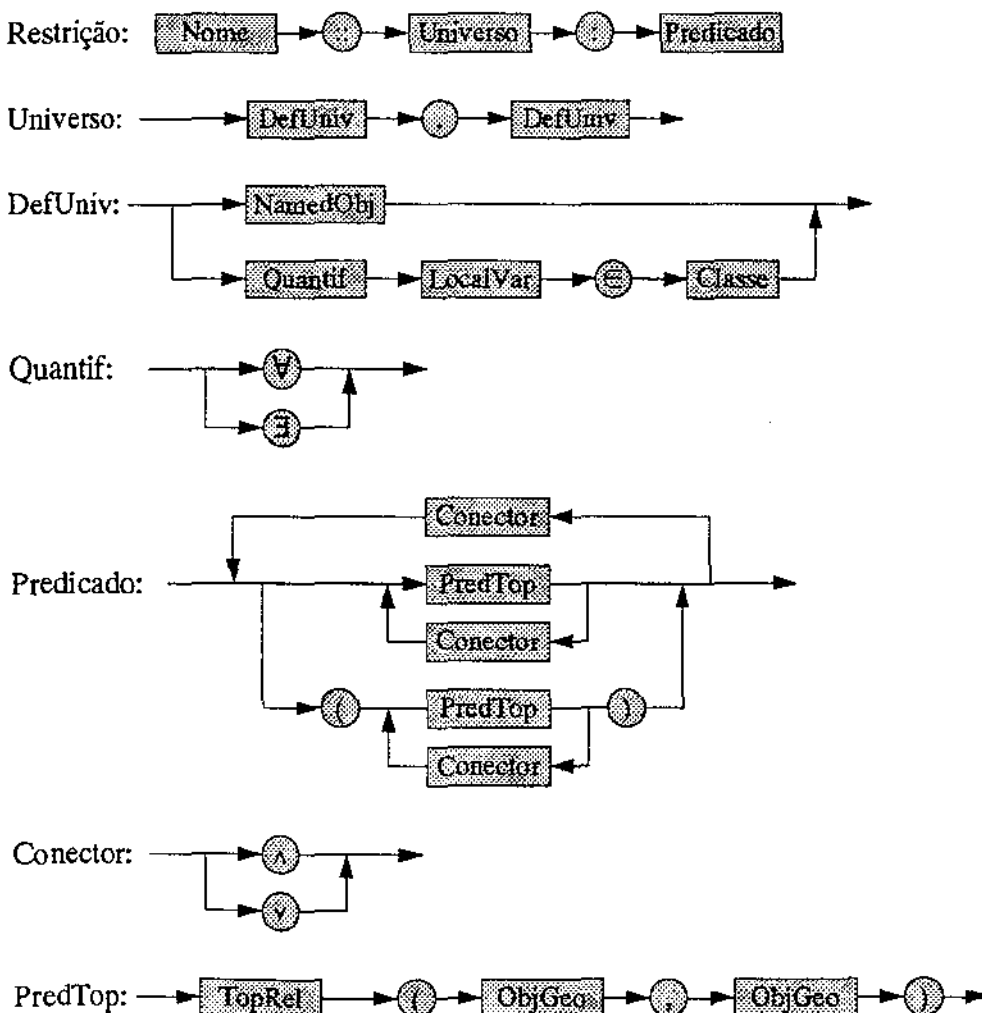
<Restrição>	::= <Nome> :: <Universo> : <Predicado>
<Universo>	::= <DefUniverso> , <DefUniverso>
<DefUniverso>	::= <NamedObject> <Quant> <LocalVar> ∈ <Classe>
<Quant>	::= ∀ ∃
<Predicado>	::= <PredTop> <PredTop> <Conector> <Predicado> (<Predicado>)
<Conector>	::= ∧ ∨
<PredTop>	::= <TopRel> (<ObjGeo> , <ObjGeo>)
<TopRel>	::= disjoint touch overlap in cross samepos
<ObjGeo>	::= <OpFronteira> (<var>) <var>
<OpFronteira>	::= from to bound
<var>	::= <LocalVar> <NamedObject>
<Nome>	::= <Identificador>

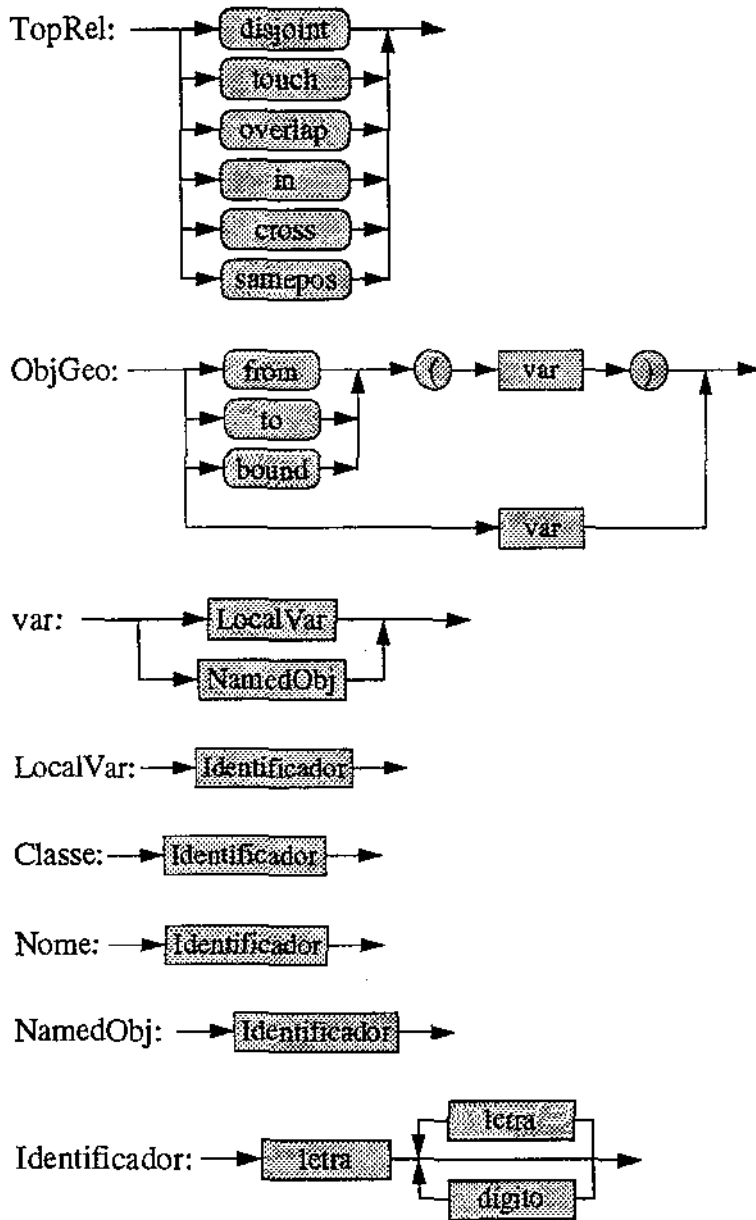
```

<NamedObject> ::= <Identificador>
<LocalVar>    ::= <Identificador>
<Classe>     ::= <Identificador>
<Identificador> ::= <letra> { <letra> | <dígito> }
<letra>      ::= a | b | c | d | ... | z
<dígito>     ::= 0 | 1 | 2 | ... | 9
    
```

Foi utilizado o utilitário **T-gen** para etapa de análise sintática. Este utilitário foi obtido do repositório de *software* para Smalltalk da Universidade de Illinois em Urbana-Champaign, Estados Unidos de América.

A seguir apresentam-se os diagramas sintáticos da linguagem proposta para especificar restrições topológicas binárias.





APÊNDICE C

EXEMPLOS DE RESTRIÇÕES TOPOLÓGICAS

Este apêndice mostra exemplos de como transformar restrições topológicas em regras E-C-A, segundo o algoritmo apresentado no capítulo 5. Algumas destas restrições foram obtidas do Projeto SAGRE do CPqD-Telebrás. Os demais exemplos demonstram capacidades adicionais das restrições topológicas propostas.

1. PROJETO SAGRE

A figura C.1 apresenta a hierarquia de classes geográficas que modelam parte da rede telefônica externa do SAGRE. As linhas representam o relacionamento *é_um* (*is_a*) e as caixas representam as classes. Por exemplo, um ponto de acesso pode ser aéreo ou subterrâneo; pontos de acesso aéreo podem ser postes, emendas, armários ou terminações. Os equipamentos de terceiros podem ser classificados em: transformadores, bancos de capacitores, para raios, subidas de alta tensão, descidas de alta tensão, entre outras.

Baseados nestas classes foram definidas restrições topológicas binárias, segundo o relatório [CPqD95].

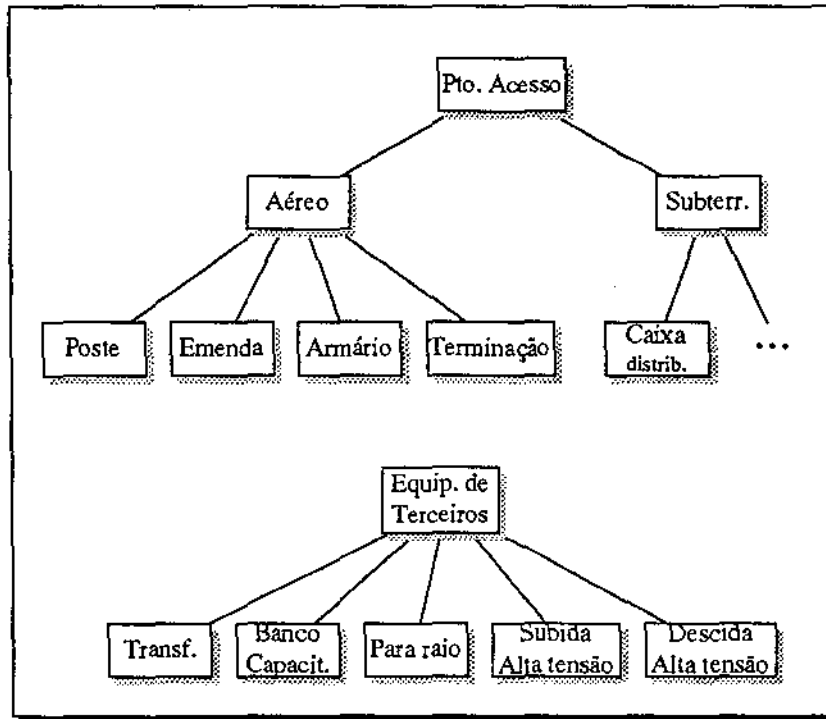


Figura C.1. Hierarquia de classes de algumas entidades da Telebrás.

1.1 CABOS SUBTERRÂNEOS X DUTOS

Em todos os dutos deve existir pelo menos um cabo Subterrâneo (figura C.2).

$$CaboSubEmDuto :: \forall \text{duto} \in \text{Duto}, \exists \text{ca} \in \text{CaboSub}: \text{in}(\text{duto}, \text{ca})$$

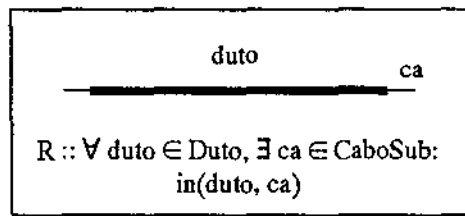


Figura C.2. Em todos os dutos há pelo menos um cabo.

Passo 1. A restrição *CaboSubEmDuto* tem um predicado.

$$\text{Pred}[1] = \text{in}(\text{duto}, \text{ca})$$

Passo 2. Análise do Predicado $\text{Pred}[1]$.

Passo 2.1. Obtenção dos eventos (tabela 5.2) e consultas utilizando o elemento (2,1) da tabela 5.3.

$$\text{Pred}[1]_{\text{duto.evento}} = \text{ev}(\forall \text{duto}) = \{\text{Duto.insert}, \text{Duto.update}\}$$

$$\text{Pred}[1]_{\text{duto.cons}} = \text{exist2}(\text{duto}, \text{in}, \text{CaboSub})$$

$$\begin{aligned} \text{Pred}[1]_{ca}.\text{evento} &= \text{ev}(\exists ca) = \{\text{CaboSub.delete}, \text{CaboSub.update}\} \\ \text{Pred}[1]_{ca}.\text{cons} &= \text{all1}(\text{Duto}, \text{in}, \text{CaboSub}) \end{aligned}$$

Passo 2.2. Análise do estado inicial.

$$\begin{aligned} \text{Inicial}[1]_{ca}.\text{evento} &= \{\text{CaboSub.delete_last}\} \\ \text{Inicial}[1]_{ca}.\text{cons} &= \text{empty}(\text{Duto}, \text{CaboSub}) \end{aligned}$$

Passo 3. Preenchimento das estruturas de tipo árvore.

Como não existem conectores lógicos, cada árvore tem apenas um elemento:

$$\begin{aligned} \text{GPred}_{\text{duto}} &= \text{Pred}[1]_{\text{duto}} \\ \text{GPred}_{ca} &= \text{Pred}[1]_{ca} \\ \text{GInicial}_{\text{duto}} &= \text{Inicial}[1]_{\text{duto}} = \{\} \\ \text{GInicial}_{ca} &= \text{Inicial}[1]_{ca} \end{aligned}$$

Geração das Regras

$$\begin{aligned} \text{Regra}_1 &= \text{when } (\text{Duto.insert}(\text{duto}) \text{ or } \text{Duto.update}(\text{duto})) \\ &\quad \text{if not(Select CaboSub.id from Duto, CaboSub where} \\ &\quad \quad \text{Duto.id=duto and} \\ &\quad \quad \text{Duto.loc in CaboSub.loc)} \\ &\quad \text{then \{abort\}} \\ \text{Regra}_2 &= \text{when } (\text{CaboSub.delete}(ca) \text{ or } \text{CaboSub.update}(ca)) \\ &\quad \text{if (Select Duto.id from Duto, CaboSub where} \\ &\quad \quad \text{Duto.loc not in CaboSub.loc)} \\ &\quad \text{then \{abort\}} \\ \text{Regra}_3 &= \text{when CaboSub.delete_last}(ca) \\ &\quad \text{if not((Select Duto.id, CaboSub.id from Duto, CaboSub) or} \\ &\quad \quad \text{not(Select Duto.id, CaboSub.id from Duto, CaboSub))} \\ &\quad \text{then \{abort\}} \end{aligned}$$

1.2 POSTE X EQUIPAMENTOS DE TERCEIROS

Existe pelo menos um poste que tem um equipamento de terceiros.

$$\begin{aligned} \text{EqTerceirosEmPoste} &:: \exists \text{poste} \in \text{Poste}, \exists \text{eqt} \in \text{EqTerceiros:} \\ &\quad \text{touch}(\text{poste}, \text{eqt}) \end{aligned}$$

Passo 1. A restrição *EqTerceirosEmPoste* tem um predicado.

$$\text{Pred}[1] = \text{touch}(\text{poste}, \text{eqt})$$

Passo 2. Análise do predicado $\text{Pred}[1]$.

Passo 2.1. Obtenção dos eventos (tabela 5.2) e consultas utilizando o elemento (1,1) da tabela 5.3.

$$\begin{aligned} \text{Pred}[1]_{\text{poste.evento}} &= \{ \text{ev}(\exists \text{poste}), \text{ev}(\exists \text{eqt}) \} \\ &= \{ \text{Poste.delete}, \text{Poste.update}, \\ &\quad \text{EqTerceiros.delete}, \text{EqTerceiros.update} \} \\ \text{Pred}[1]_{\text{poste.cons}} &= \text{exist1}(\text{Poste}, \textit{touch}, \text{EqTerceiros}) \\ \text{Pred}[1]_{\text{eqt.evento}} &= \{ \text{ev}(\exists \text{poste}), \text{ev}(\exists \text{eqt}) \} \\ &= \{ \text{Poste.delete}, \text{Poste.update}, \\ &\quad \text{EqTerceiros.delete}, \text{EqTerceiros.update} \} \\ \text{Pred}[1]_{\text{eqt.cons}} &= \text{exist1}(\text{Poste}, \textit{touch}, \text{EqTerceiros}) \end{aligned}$$

Passo 2.2. Análise do estado inicial.

$$\begin{aligned} \text{Inicial}[1]_{\text{poste.evento}} &= \{ \text{Poste.insert_first} \} \\ \text{Inicial}[1]_{\text{poste.cons}} &= \text{exist1}(\text{Poste}, \textit{touch}, \text{EqTerceiros}) \\ \text{Inicial}[1]_{\text{eqt.evento}} &= \{ \text{EqTerceiros.insert_first} \} \\ \text{Inicial}[1]_{\text{eqt.cons}} &= \text{exist1}(\text{Poste}, \textit{touch}, \text{EqTerceiros}) \end{aligned}$$

Passo 3. Preenchimento das estruturas de tipo árvore.

Como não existem conectores lógicos, em cada árvore existe apenas um elemento.

$$\begin{aligned} \text{GPred}_{\text{poste}} &= \text{Pred}[1]_{\text{poste}} \\ \text{GPred}_{\text{eqt}} &= \text{Pred}[1]_{\text{eqt}} \\ \text{GInicial}_{\text{poste}} &= \text{Inicial}[1]_{\text{poste}} \\ \text{GInicial}_{\text{eqt}} &= \text{Inicial}[1]_{\text{eqt}} \end{aligned}$$

Geração das Regras

$$\begin{aligned} \text{Regra}_1 &= \textit{when} (\text{Poste.delete}(\text{poste}) \textit{ or } \text{Poste.update}(\text{poste}) \textit{ or} \\ &\quad \text{EqTerceiros.delete}(\text{eqt}) \textit{ or } \text{EqTerceiros.update}(\text{eqt})) \\ &\textit{ if not} (\text{Select Poste.id from Poste, EqTerceiros where} \\ &\quad \text{Poste.loc } \textit{touch} \text{ EqTerceiros.loc }) \\ &\textit{ then } \{ \text{abort} \} \\ \text{Regra}_2 &= \textit{when} (\text{Poste.delete}(\text{poste}) \textit{ or } \text{Poste.update}(\text{poste}) \textit{ or} \\ &\quad \text{EqTerceiros.delete}(\text{eqt}) \textit{ or } \text{EqTerceiros.update}(\text{eqt})) \\ &\textit{ if not} (\text{Select Poste.id from Poste, EqTerceiros where} \\ &\quad \text{Poste.loc } \textit{touch} \text{ EqTerceiros.loc }) \\ &\textit{ then } \{ \text{abort} \} \\ \text{Regra}_3 &= \textit{when} (\text{Poste.insert_first}(\text{poste})) \\ &\textit{ if not} (\text{Select Poste.id from Poste, EqTerceiros where} \\ &\quad \text{Poste.loc } \textit{touch} \text{ EqTerceiros.loc }) \\ &\textit{ then } \{ \text{abort} \} \end{aligned}$$

```

Regra4 = when ( EqTerceiros.insert_first(eqt) )
           if not( Select Poste.id from Poste, EqTerceiros where
                   Poste.loc touch EqTerceiros.loc )
           then {abort}

```

Note que as quatro regras tem a mesma as condição, então os eventos correspondentes podem ser compostos utilizando o conector **OR**, obtendo como resultado uma única regra:

```

Regra = when ( Poste.delete(poste) or Poste.update(poste) or
              EqTerceiros.delete(eqt) or EqTerceiros.update(eqt) or
              Poste.insert_first(poste) or EqTerceiros.insert_first(eqt))
          if not( Select Poste.id from Poste, EqTerceiros where
                  Poste.loc touch EqTerceiros.loc )
          then {abort}

```

1.3 CABO AÉREO X POSTE

Um cabo aéreo tem pelo menos um poste associado (figura C.3).

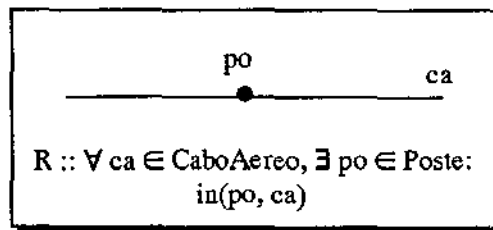
$$\text{CaboAexPoste} :: \forall ca \in \text{CaboAéreo}, \exists po \in \text{Poste}: \text{in}(po, ca)$$


Figura C.3. Existe pelo menos um poste associado a um cabo aéreo.

Passo 1. A restrição *CaboAexPoste* tem um predicado.

$$\text{Pred}[1] = \text{in}(po, ca)$$

Passo 2. Análise do Predicado $\text{Pred}[1]$

Passo 2.1. Obtenção dos eventos (tabela 5.2) e consultas utilizando o elemento (2,1) da tabela 5.3.

$$\text{Pred}[1]_{po.\text{evento}} = \text{ev}(\forall po) = \{\text{Poste.insert}, \text{Poste.update}\}$$

$$\text{Pred}[1]_{po.\text{cons}} = \text{exist2}(po, \text{in}, \text{CaboAéreo})$$

$$\text{Pred}[1]_{ca.\text{evento}} = \text{ev}(\exists ca) = \{\text{CaboAéreo.delete}, \text{CaboAéreo.update}\}$$

$$\text{Pred}[1]_{ca.\text{cons}} = \text{all1}(\text{Poste}, \text{in}, \text{CaboAéreo})$$

Passo 2.2. Análise do estado inicial.

$$\text{Inicial}[1]_{ca.\text{evento}} = \{\text{CaboAéreo.delete_last}\}$$

$$\text{Inicial}[1]_{ca.\text{cons}} = \text{empty}(\text{Poste}, \text{CaboAéreo})$$

Passo 3. Preenchimento das estruturas de tipo árvore.

Como não existem conectores lógicos, cada árvore tem apenas um elemento:

$$\begin{aligned} \text{GPred}_{po} &= \text{Pred}[1]_{po} \\ \text{GPred}_{ca} &= \text{Pred}[1]_{ca} \\ \text{GInicial}_{po} &= \text{Inicial}[1]_{po} \\ \text{GInicial}_{ca} &= \text{Inicial}[1]_{ca} = \{ \} \end{aligned}$$

Geração das Regras

$$\begin{aligned} \text{Regra}_1 &= \text{when (Poste.insert(po) or Poste.update(po))} \\ &\quad \text{if not(Select CaboAéreo.id from Poste, CaboAéreo where} \\ &\quad\quad \text{Poste.id=po and Poste.loc in CaboAéreo.loc)} \\ &\quad \text{then \{abort\}} \\ \text{Regra}_2 &= \text{when (CaboAéreo.delete(ca) or CaboAéreo.update(ca))} \\ &\quad \text{if (Select Poste.id from Poste, CaboAéreo where} \\ &\quad\quad \text{Poste.loc not in CaboAéreo.loc)} \\ &\quad \text{then \{abort\}} \\ \text{Regra}_3 &= \text{when CaboAéreo.delete_last(ca)} \\ &\quad \text{if not((Select Poste.id, CaboAéreo.id from Poste, CaboAéreo) or} \\ &\quad\quad \text{not(Select Poste.id, CaboAéreo.id from Poste, CaboAéreo))} \\ &\quad \text{then \{abort\}} \end{aligned}$$

1.4 TRANSFORMADOR X EMENDA

Transformadores não podem ser colocados em emendas.

$$\begin{aligned} \text{TransfEmenda} &:: \forall tra \in \text{Transf}, \forall em \in \text{Emenda} : \\ &\quad \text{disjoint}(tra, em) \end{aligned}$$

Passo 1. A restrição *TransfEmenda* tem um predicado.

$$\text{Pred}[1] = \text{disjoint}(tra, em)$$

Passo 2. Análise do Predicado $\text{Pred}[1]$

Passo 2.1. Obtenção dos eventos (tabela 5.2) e consultas utilizando o elemento (2,2) da tabela 5.3.

$$\begin{aligned} \text{Pred}[1]_{tra.\text{evento}} &= \text{ev}(\forall tra) = \{\text{Transf.insert}, \text{Transf.update}\} \\ \text{Pred}[1]_{tra.\text{cons}} &= \text{all2}(tra, \text{disjoint}, \text{Emenda}) \\ \text{Pred}[1]_{em.\text{evento}} &= \text{ev}(\forall em) = \{\text{Emenda.insert}, \text{Emenda.update}\} \\ \text{Pred}[1]_{em.\text{cons}} &= \text{all2}(em, \text{disjoint}, \text{Transf}) \end{aligned}$$

Passo 2.2. Análise do estado inicial.

```

Inicial[1]tra.evento = {Transf.delete_last}
Inicial[1]tra.cons   = empty(Transf,Emenda)

Inicial[1]em.evento = {Emenda.delete_last}
Inicial[1]em.cons   = empty(Transf,Emenda)

```

Passo 3. Preenchimento das estruturas de tipo árvore.

Como não existem conectores lógicos, cada árvore tem apenas um elemento:

```

GPredtra = Pred[1]tra
GPredem = Pred[1]em
GInicialtra = Inicial[1]tra
GInicialem = Inicial[1]em

```

Geração das Regras

```

Regra1 = when (Transf.insert(tra) or Transf.update(tra))
          if not( Select Emenda.id from Transf, Emenda where
                  Transf.id=tra and
                  Transf.loc not disjoint Emenda.loc )
          then {abort}

Regra2 = when (Emenda.insert(em) or Emenda.update(em))
          if not( Select Transf.id from Transf, Emenda where
                  Emenda.id=em and
                  Transf.loc not disjoint Emenda.loc )
          then {abort}

Regra3 = when Emenda.delete_last(em)
          if not( (Select Emenda.id, Transf.id from Emenda, Transf) or
                  not(Select Emenda.id, Transf.id from Emenda, Transf) )
          then {abort}

Regra4 = when Transf.delete_last(tra)
          if not( (Select Emenda.id, Transf.id from Emenda, Transf) or
                  not(Select Emenda.id, Transf.id from Emenda, Transf) )
          then {abort}

```

2. OUTROS EXEMPLOS

Nesta seção são apresentados exemplos de restrições topológicas inter-objeto, intra-classe e inter-classe.

2.1 INTER OBJETO

No estado de São Paulo existe pelo menos um rio.

ExistemRios :: $sp, \exists r \in \text{Rio}: \text{cross}(r, sp) \text{ or } \text{in}(r, sp)$

Passo 1. A restrição *ExistemRios* é dividida em dois predicados.

$\text{Pred}[1]_r = \text{cross}(r, sp)$

$\text{Pred}[2]_r = \text{in}(r, sp)$

Passo 2. Análise de cada predicado $\text{Pred}[i]$

Passo 2.1. Obtenção dos eventos (tabela 5.2) e consultas utilizando o elemento (3,1) da tabela 5.3.

$\text{Pred}[1]_r.\text{evento} = \{\text{ev}(\exists \text{Rio})\} = \{\text{Rio.delete}, \text{Rio.update}\}$

$\text{Pred}[1]_r.\text{cons} = n_o(r, \text{cross}, sp)$

$\text{Pred}[1]_{sp}.\text{evento} = \{\text{ev}(sp)\} = \{\text{Estado.insert}(sp), \text{Estado.delete}(sp),$
 $\text{Estado.update}(sp)\}$

$\text{Pred}[1]_{sp}.\text{cons} = \text{exist2}(\text{Rio}, \text{cross}, sp)$

$\text{Pred}[2]_r.\text{evento} = \{\text{ev}(\exists \text{Rio})\} = \{\text{Rio.delete}, \text{Rio.update}\}$

$\text{Pred}[2]_r.\text{cons} = n_o(r, \text{in}, sp)$

$\text{Pred}[2]_{sp}.\text{evento} = \{\text{ev}(sp)\} = \{\text{Estado.insert}(sp), \text{Estado.delete}(sp),$
 $\text{Estado.update}(sp)\}$

$\text{Pred}[2]_{sp}.\text{cons} = \text{exist2}(\text{Rio}, \text{in}, sp)$

Passo 2.2. Análise do estado inicial.

$\text{Inicial}[1]_r = \{\}$

$\text{Inicial}[1]_{sp} = \{\}$

$\text{Inicial}[2]_r = \{\}$

$\text{Inicial}[2]_{sp} = \{\}$

Passo 3. Preenchimento das estruturas de tipo árvore (figura C.4).

$\text{GPred}_{sp} = \{ \text{Pred}[1]_{sp}, \text{Pred}[2]_{sp} \}$

$\text{GPred}_r = \{ \text{Pred}[1]_r, \text{Pred}[2]_r \}$

$\text{GInicial}_r = \{\}$

$\text{GInicial}_{sp} = \{\}$

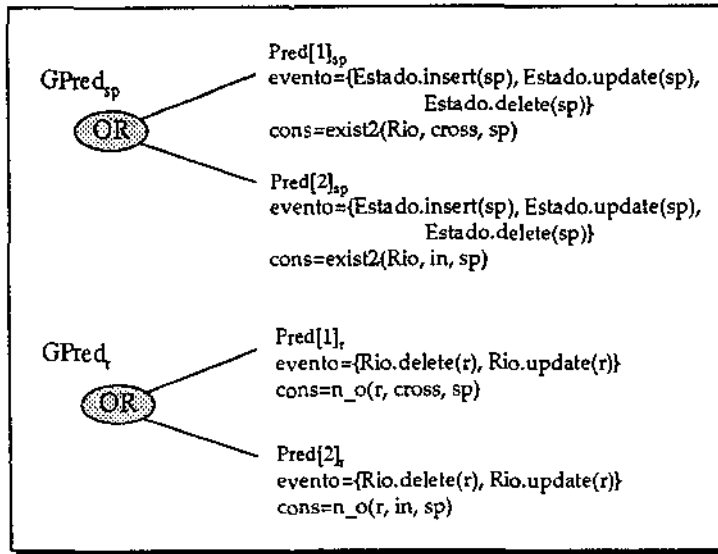


Figura C.4. Estruturas correspondentes ao exemplo 2.1.

Geração das Regras

```

    Regra1 = when ( Estado.insert(sp) or Estado.delete(sp) or
                    Estado.update(sp) )
              if not( Select Rio.id from Rio, Estado where
                      Estado.id=sp and
                      (Rio.loc cross Estado.loc or Rio.loc in Estado.loc) )
              then {abort}

    Regra2 = when ( Rio.delete(r) or Rio.update(r) )
              if not( Select Rio.id from Rio, Estado where
                      Estado.id=sp and Rio.id=r and
                      (Rio.loc cross Estado.loc or Rio.loc in Estado.loc) )
              then {abort}
    
```

2.2 INTRA CLASSE

Não ha interseção entre triângulos (figura C.5).

$$TriDisjuntos :: \forall t1, t2 \in Triângulo: disjoint(t1, t2)$$

Como ambas variáveis estão associadas ao mesmo quantificador e à mesma classe, a determinação dos eventos e condições pode ser simplificada analisando apenas uma destas variáveis.

Passo 1. A restrição *TriDisjuntos* tem um predicado.

$$Pred[1] = disjoint(t1, t2)$$

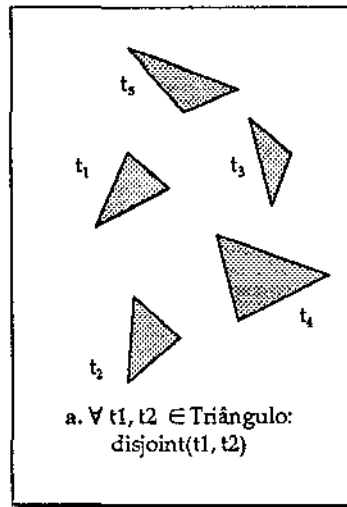


Figura C.5. Exemplo de restrição intra-classe.

Passo 2. Análise do Predicado $\text{Pred}[1]$.

Passo 2.1. Obtenção dos eventos (tabela 5.2) e consultas utilizando o elemento (2,2) da tabela 5.3.

$\text{Pred}_{t1}.\text{evento} = \text{ev}(\forall t1) = \{\text{Triângulo.insert}, \text{Triângulo.update}\}$
 $\text{Pred}_{t1}.\text{cons} = \text{all2}(t1, \text{disjoint}, \text{Triângulo})$

Passo 2.2. Análise do estado inicial.

$\text{Inicial}[1]_{t1}.\text{evento} = \{\text{Triângulo.delete_last}\}$
 $\text{Inicial}[1]_{t1}.\text{cons} = \text{empty}(\text{Triângulo}, \text{Triângulo})$

Passo 3. Preenchimento das estruturas de tipo árvore.

Como não existem conectores lógicos, cada árvore tem apenas um elemento:

$\text{GPred}_{t1} = \text{Pred}[1]_{t1}$
 $\text{GINicial}_{t1} = \text{Inicial}[1]_{t1}$

Geração das Regras

$\text{Regra}_1 = \text{when} (\text{Transf.insert}(t1) \text{ or } \text{Transf.update}(t1))$
 $\text{if} (\text{Select Triângulo1.id from Triângulo, Triângulo as Triângulo1 where Triângulo.id=t1 and not(Triângulo1.id=t1) and Triângulo.loc not disjoint Triângulo1.loc}$
 $\text{then } \{\text{abort}\}$

$\text{Regra}_2 = \text{when Triângulo.delete_last}$
 $\text{if not} ((\text{Select Triângulo.id from Triângulo}) \text{ or not}(\text{Select Triângulo.id from Triângulo}))$
 $\text{then } \{\text{abort}\}$

2.3 INTER CLASSE

Para cada elemento *tri* da classe Triângulo, existe pelo menos um elemento *qua* da classe Quadrilátero, que se sobrepõe espacialmente a *tri* (figura C.6).

$$TriSobreQua :: \forall tri \in \text{Triângulo}, \exists qua \in \text{Quadrilátero}: \text{overlap}(tri, qua)$$

Passo 1. A restrição *TriSobreQua* tem um predicado.

$$\text{Pred}[1] = \text{overlap}(tri, qua)$$

Passo 2. Análise do Predicado Pred[1]

Passo 2.1. Obtenção dos eventos (tabela 5.2) e consultas utilizando o elemento (2,1) da tabela 5.3.

$$\begin{aligned} \text{Pred}_{tri}.\text{evento} &= \text{ev}(\forall tri) = \{\text{Triângulo.insert}, \text{Triângulo.update}\} \\ \text{Pred}_{tri}.\text{cons} &= \text{exist2}(tri, \text{overlap}, \text{Quadrilátero}) \\ \text{Pred}_{qua}.\text{evento} &= \text{ev}(\exists qua) = \{\text{Quadrilátero.delete}, \text{Quadrilátero.update}\} \\ \text{Pred}_{qua}.\text{cons} &= \text{all1}(\text{Triângulo}, \text{overlap}, \text{Quadrilátero}) \end{aligned}$$

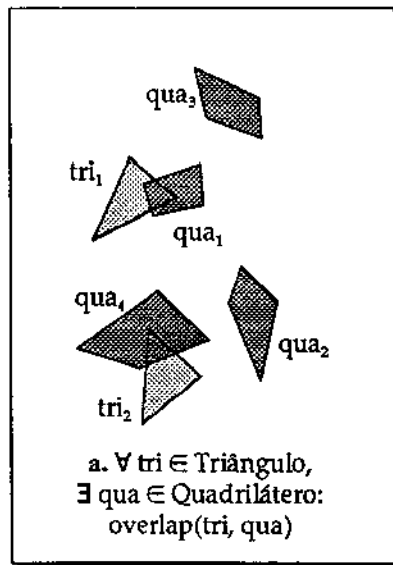


Figura C.6. Exemplo de restrição inter-classe.

Passo 2.2. Análise do estado inicial.

$$\text{Inicial}_{qua}.\text{evento} = \{\text{Quadrilátero.delete_last}\}$$

$$\text{Inicial}_{qua}.\text{cons} = \text{empty}(\text{Quadrilátero}, \text{Triângulo})$$

Passo 3. Preenchimento das estruturas de tipo árvore.

Como não existem conectores lógicos, cada árvore tem apenas um elemento.

$GPred_{tri} = Pred[1]_{tri}$
 $GPred_{qua} = Pred[1]_{qua}$
 $GINicial_{tri} = Inicial[1]_{tri} = \{\}$
 $GINicial_{qua} = Inicial[1]_{qua}$

Geração das Regras

Regra₁ = *when* (Triângulo.insert(tri) or Triângulo.update(tri))
if not (**Select** Quadrilátero.id from Triângulo, Quadrilátero **where**
Triângulo.id=tri and
Triângulo.loc *overlap* Quadrilátero.loc)
then {abort}

Regra₂ = *when* (Quadrilátero.delete(qua) or Quadrilátero.update(qua))
if (**Select** Quadrilátero.id from Triângulo, Quadrilátero **where**
Triângulo.loc *not overlap* Quadrilátero.loc)
then {abort}

Regra₃ = *when* (Quadrilátero.delete_last)
if not (**Select** Quadrilátero.id from Triângulo, Quadrilátero) or
not(**Select** Quadrilátero.id from Triângulo, Quadrilátero))
then {abort}

BIBLIOGRAFIA

- [ABC+91] J. Antenucci, K. Brown, P. Croswell and M. Kevany. *Geographic Information Systems: A Guide to the Technology*. Van Nostrand Reinhold, New York, 1991.
- [ABD+92] Atkinson, Bancilhon, DeWitt, Dittrich, Maier and Zdonik. *The Object-Oriented Database System Manifesto*. In Building an Object-Oriented Database System, F. Bancilhon, C. Delobel and P. Kandellakis (editors), Morgan Kaufmann, 1992.
- [AD90] M. Armstrong and P. Densham. *Database Organization Strategies for Spatial Decision Support Systems*. International Journal of Geographical Information Systems, 4(1):3-30, 1990.
- [Agu95] C. D. Aguiar. *Integração de Sistemas de Banco de Dados Heterogêneos em Aplicações de Planejamento Urbano*. Master's Thesis, DCC - IMECC - UNICAMP, March 1995.
- [And92] M. J. Andrade. *Manutenção de Restrições de Integridade em Banco de Dados Orientados a Objeto*. Master's Thesis, DCC - IMECC - UNICAMP, March 1992.
- [AWP93] A. Abdelmoty, M. Williams, and N. Paton. *Deduction and Deductive Databases for geographic Data Handling*. In Proceedings of the 3rd International Symposium on Spatial Databases (SSD '93), pages 443-464, 1993.
- [AYA+92] D. Abel, S. Yap, R. Ackland, M. Cameron, D. Smith, and G. Walker. *Environmental Decision Support Systems Project: an Exploration of Alternative Architectures for Geographical Information Systems*. International Journal of Geographical Information Systems, 6(3):193-204, 1992.

- [BBKZ92] A. Buschmann, H. Branding, T. Kudrass, and J. Zimmermann. *REACH: A REal-Time, Active and Heterogeneous Mediator System*. In IEEE Bulletin of the Technical Committee on Data Engineering. Special Issue in Active Databases, 15(1-4), pages 44-47, December 1992.
- [Bee89] C. Beeri. *Formal Models for Object-Oriented Databases*. In Proceedings of the 1st International Conference on Deductive and Object-Oriented Databases (DOOD '89), pages 370-395, 1989.
- [Ber94] M. Berndtsson. *Reactive Object-Oriented Databases and CIM*. In Proceedings of the 5th International Conference on Database and Expert Systems Applications (DEXA '94), pages 769-728, Athens, Greece, September 1994.
- [BL92] M. Berndtsson and B. Lings. *On Developing Reactive Object-Oriented Databases*. IEEE Bulletin of the Technical Committee on Data Engineering, 15(1-4), pages 31-34, December 1992.
- [BR84] M. Brodie and D. Ridjanovic. *On the design and specification of database transactions*. In On Conceptual Modeling, M. Brodie, J. Mylopoulos and J. Schmidt (editors), pages 277-306, Springer, New York, 1984.
- [Buch94] A. Buchmann. *Active Databases*. Tutorial Notes. IX School of Computing, Recife, Brazil, July 1994.
- [CAM93] S. Chakravathy, E. Anwar and L. Maugis. *Design and Implementation of Active Capability for an Object-Oriented Database*. Technical Report UF-CIS-TR-93-001, CIS Department, University of Florida, September 1993.
- [CBB+89] S. Chakravarthy, B. Blaustein, A. Buchman, et. al. *HiPAC: A Research Project in Active, Time-Constrained Database Management*. Final Technical Report XAIT-89-02 (187), July 1989.
- [CFM+86] M. Carey, D. Frank, M. Muralkrisna, D. DeWitt, G. Graefe, J. Richardson and E. Shekita. *The Architecture of the EXODUS Extensible DBMS*. In Readings in Database Systems, M. Stonebraker (editor), pages 488-502, Morgan-Kaufmann Publishers, San Mateo, California, 1988.
- [CFO93] E. Clementini, P. Di Felice and P. van Oosterom. *A Small Set of Formal Topological Relationships Suitable for End-User Interaction*. In Proceedings of the 3rd International Symposium on Large Spatial Databases, pages 277-295, 1993. Also in LNCS 692.
- [CFPW94] S. Ceri, P. Fraternali, S. Paraboschi and J. Widom. *Active Database Systems*. In PUC-Rio DB Workshop. On New Database Research Challenges, pages 35-52, Rio de Janeiro, Brazil, September 1994.
-

- [CFS+94] G. Camara, U. Freitas, R. Souza, M. Casanova, A. Hemerley and C. Medeiros. *A model to cultivate objects and manipulate fields*. In Proceedings of the 2nd International ACM Workshop on Advances in GIS, pages 20-28, 1994.
- [CHS92] S. Chakravarthy, E. Hanson and S. Su. *Active Database/Knowledge Base Research at the University of Florida*. IEEE Bulletin of the Technical Committee on Data Engineering, 15(1-4), pages 35-39, December 1992.
- [Cif95] R. Ciferri. *Um Benchmark voltado à Análise de Sistemas de Informação Geográfica*. Master's Thesis, DCC - IMECC - UNICAMP, June 1995.
- [CKTB94] S. Chakravathy, V. Krishnaprasad, Z. Tamizuddin and R. Badani. *ECA Rule Integration into an OODBMS: Architecture and Implementation*. Technical Report UF-CIS-TR-94-023, University of Florida, May 1994.
- [CM91] S. Chakravarthy and D. Mishra. *An Event Specification Language (Snoop) for Active Databases and its Detection*. Technical Report UF-CIS TR-91-23, CIS Department, University of Florida, September 1991.
- [CM93] S. Chakravarthy and D. Mishra. *Snoop: an Expressive Event Specification Language for Active Databases*. Technical Report UF-CIS-TR-93-007, University of Florida, March 1993.
- [CN90] S. Chakravarthy and S. Nesson. *Making an Object-Oriented DBMS Active: Design, Implementation and Evaluation of a Prototype*. In Proceedings of the International Conference on Extending Database Technology, Venice, March 1990.
- [CPqD95] Relatório Técnico. Projeto SAGRE. CPqD - Telebrás, 1995.
- [CS94] R. Chandra and A. Segev. *Active Databases for Financial Applications*. In Proceedings of the Fourth International Workshop on Research in Data Engineering: Active Database Systems (RIDE '94), Houston, Texas, February 1994.
- [CSE94] E. Clementini, J. Sharma and M. Egenhofer. *Modeling Topological Spatial Relations: Strategies for Query Processing*. In Computers & Graphics, 18(6):815-822, 1994.
- [DBB+88] U. Dayal, B. Blaustein, A. Buchmann, S. Chakravarthy et al. *The HiPAC Project: Combining Active Databases and Timing Constraints*. ACM SIGMOD Record, 17(1), pages 51-70, March 1988.
-

- [DBM88] U. Dayal, A. Buchmann and D. McCarthy. *Rules are Objects too: A Knowledge Model for an Active, Object-Oriented Database System*. In Proceedings of the 2nd International Workshop on Object Oriented Database Systems, K. Dittrich (editor), pages 129-143, September 1988. Also in LNCS 334.
- [DF92] J. Doerschler and H. Freeman. *A Rule-Based System for Dense-Map Name Placement*. Communications of the ACM, 35(1):68-79, January 1992.
- [DHL90] U. Dayal, M. Hsu and R. Ladin. *Organizing Long-Running Activities with Triggers and Transactions*. In Proceedings of the International Conference on Management of Data (SIGMOD '90), Atlantic City, NJ, May 1990.
- [DHL91] U. Dayal, M. Hsu and R. Ladin. *A Transactional Model for Long-Running Activities*. In Proceedings 17th International Conference on Very Large Data Bases (VLDB '91), pages 113-122, Barcelona, Spain, September 1991.
- [DHW95] U. Dayal, E. Hanson and J. Widom. *Active Database Systems*. In Modern Database System: The Object Model, Interoperability, and Beyond, W. Kim (editor), pages 434-456, ACM Press, New York, 1995.
- [DPG91] O. Diaz, N. Paton and P. Gray. *Rule Management in Object-Oriented Databases: A Unified Approach*. In Proceedings 17th International Conference on Very Large Data Bases (VLDB '91), Barcelona, Spain, September 1991.
- [Egen91] M. Egenhofer. *Reasoning about Binary Topological Relations*. In Proceedings of the 2nd Symposium (SSD '91): "Advances in Spatial Databases", O. Gunther, H. Schek (editors), pages 143-180, Zurich, Switzerland, August 1991. Also in LNCS 525.
- [Egen94] M. Egenhofer. *Spatial SQL: A Query and Presentation Language*. IEEE Transactions on Knowledge and Data Engineering, 6(1):86-95, February 1994.
- [EH90] M. Egenhofer and J. Herring. *A Mathematical Framework for the Definitions of Topological Relationships*. In Proceedings of the Fourth International Symposium on Spatial Data Handling, Zurich, Switzerland 1990.
- [EH92] M. Egenhofer and J. Herring. *Categorizing Binary Topological Relationships between Regions, Lines and Points in Geographic Databases*. Technical Report, Department of Surveying Engineering, University of Maine, Orono, ME, 1992.
- [EN94] R. Elmasri and S. Navathe. *Fundamentals of Database Systems*. Second Edition. Benjamin Cummings, Redwood City, USA, 1994.
-

- [For82] C. Forgy. *RETE: A Fast Algorithm for the many Pattern/many Object Pattern Match Problem*. Artificial Intelligence, Vol. 19, pages 17-37, 1982.
- [GD92] S. Gatzju and K. Dittrich. *SAMOS: an Active Object-Oriented Database System*. IEEE Bulletin of the Technical Committee on Data Engineering, 15(1-4), pages 23-26, December 1992.
- [GD93] S. Gatzju and K. Dittrich. *Events in an Active Object Oriented Database System*. In Proceedings of the International Workshop on Rules in Database Systems, pages 23-29, August 1993.
- [GD94] S. Gatzju and K. Dittrich. *Detecting Composite Events in Active Database Systems Using Petri Nets*. In Proceedings of the 4th International Workshop on Research Issues in Data Engineering: Active Database Systems (RIDE '94), Houston, Texas, February 1994.
- [GJ91] N.H. Gehani and H.V. Jagadish. *Ode as an Active Database: Constraints and Triggers*. In Proceedings 17th International Conference on Very Large Data Bases (VLDB '91), pages 327-336, Barcelona, Spain, September 1991.
- [GJ92] N. Gehani and H. Jagadish. *Active Database Facilities in Ode*. IEEE Bulletin of the Technical Committee on Data Engineering, 15(1-4), pages 19-22, December 1992.
- [GJ92a] N.H. Gehani and H.V. Jagadish. *Event Specification in an Object-Oriented Database*. In Proceedings of the International Conference on Management of Data (SIGMOD '92), pages 81-90, San Diego, June 1992.
- [GJS92a] N. Gehani, H. Jagadish and O. Shmueli. *Compose: a System for Composite Event Specification and Detection*. Technical Report AT&T Bell Laboratories, December 1992.
- [GJS92b] N. Gehani, H. Jagadish and O. Shmueli. *Event Specification in an Active Object Oriented Database*. In Proceedings of the International Conference of Management of Data (SIGMOD '92), pages 81-90, 1992.
- [GMR91] M. Goochild, D. Maguire and D. Rhind, editors. *Geographic Information Systems: Principles and Applications*. Longman, Scientific & Technical, 1991.
- [Güt94] R. Güting. *An Introduction to Spatial Databases*. In The VLDB Journal, 3(4):357-400, October 1994.
- [Han89] E. Hanson. *An Initial Report on the Design of Ariel: A DBMS with an Integrated Production Rule System*. ACM SIGMOD Record, 18(3), pages 12-19, September 1989.
-

- [HC+90] E. Hanson, M. Chaabouni, C. Kim and Y. Wang. *A Predicate Matching Algorithm for Database Rule Systems*. In Proceedings of the International Conference on Management of Data (SIGMOD '90), May 1990.
- [Her94] D. Hernández. *Qualitative Representation of Spatial Knowledge*. Lecture Notes in Artificial Intelligence (Subseries of LNCS) No. 804. Springer-Verlag, 1994.
- [HLM88] M. Hsu, R. Ladin and D. McCarthy. *An Execution Model for Active Data Base Management Systems*. In Proceedings of the 3rd International Conference on Data and Knowledge Bases, Jerusalem, June 1988.
- [HT92] T. Hadzilacos and N. Tryfona. *A Model for Expressing Topological Integrity Constraints in Geographic Databases*. In Proceedings of the International Conference GIS: "Theories and Methods of Spatio-temporal Reasoning in Geographic Space". A.U. Frank, I. Campari, and U. Formentini (editors), Pisa, Italy, September 1992. Also in LNCS 639.
- [KWB+93] C. Kontoes, G. Wilkinson, A. Burrill, S. Goffredo, and J. Megier. *An Experimental System for the Integration of GIS Data in Knowledge-based Image Analysis for Remote Sensing of Agriculture*. International Journal of Geographical Information Systems, 7(3):247-263, 1993.
- [LL93] Y. Leung and K. S. Leung. *An Intelligent Expert System Shell for Knowledge-based Geographical Information Systems: 1 - The Tools*. International Journal of Geographical Information Systems, 7(1):189-200, 1993.
- [LM94] R. Laurini and F. Milleret-Raffort. *Topological Reorganization of Inconsistent Geographical Databases: A Step Towards their Certification*. Comput. & Graphics, Vol. 18, No. 6, pages 803-813, 1994.
- [LT92] T. Ling and P. Teo. *On Rules and Integrity Constraints in Database Systems*. Information and Software Technology, 34(3), March 1992.
- [Mag91] D. Maguire. *An Overview and Definition of GIS*. Chapter 1, Volume 1, pages 9-20, In M. Goochild, D. Maguire and D. Rhind [GMR91], 1991.
- [Mor84] M. Morgestern. *Constraint Equations: Declarative Expression of Constraints with Automatics Enforcement*. In Proceedings of the 10th International Conference on Very Large Data Base (VLDB '84), pages 291-300, Singapore, August 1984.
- [MP91] C. Medeiros and P. Pfeffer. *Object Integrity Using Rules*. In Proceedings of the European Conference on Object Oriented Programming (ECOOP '91), pages 219-230, Geneva, Switzerland, July 1991. Also in LNCS 512.
-

- [MP94] C. Bauzer Medeiros and F. Pires. *Databases for GIS*. ACM SIGMOD Record, 23(1):107-115, March 1994.
- [PMP93] N. Pissinou, K. Makki, and E. Park. *Towards the Design and Development of a New Architecture for Geographic Information Systems*. In Proceedings of the 2nd International Conference on Information and Knowledge Management (CIKM '93), pages 565-573, 1993.
- [PMS93] F. Pires, C. Bauzer Medeiros and A. Silva. *Modeling Geographic Information Systems using an Object Oriented Framework*. In Proceedings of the XIII International Conference of the Chilean Computer Society. La Serena, Chile, October 1993.
- [PS94] D. Papadias and T. Sellis. *Qualitative Representation of Spatial Knowledge in Two-Dimensional Space*. In The VLDB Journal, 3(4):479-516, October 1994.
- [PSTE95] D. Papadias, T. Sellis, Y. Theodoridis and M. Egenhofer. *Topological Relations in the World of Minimum Bounding Rectangles: A Study with R-trees*. In Proceedings of the International Conference of management of Data (SIGMOD '95), pages 92-103, 1995.
- [SA93] A. Subramanian and N. Adam. *Applying OOAD in the Design and Implementation of an Intelligent Geographic Information System*. In Advanced Database Systems, LNCS 759, pages 127-150, Springer Verlag.
- [Sam94] P. R. Falcone Sampaio. *Restrições Dinâmicas em Bancos de Dados Ativos Orientados a Objetos*. Master's Thesis, DCC - IMECC - UNICAMP, December 1994.
- [SHH88] M. Stonebraker, E. Hanson and C. Hong. *The Design of the Postgres Rule System*. In Readings in Database Systems, M. Stonebraker (editor), Morgan-Kaufmann, 1988.
- [SHP88] M. Stonebraker, E. Hanson and S. Potamianos. *The Postgres Rule Manager*. IEEE Transactions on Software Engineering, 14(7):897-907, 1988.
- [SHP89] M. Stonebraker, M. Hearst and S. Potamianos. *A Commentary on the Postgres Rule Manager*. SIGMOD Record, 18(3), September 1989.
- [SJGP90] M. Stonebraker, A. Jhingran, J. Goh and S. Potamianos. *On Rules, Procedures, Caching and Views in Data Base Systems*. In Proceedings of the International Conference on Management of Data (SIGMOD '90), pages 281-290, Atlantic City, May 1990.
-

- [SKD95] E. Simon and A. Kotz-Dittrich. *Promises and Realities of Active Database Systems*. In Proceedings of the 21st International Conference on Very Large Data Bases (VLDB '95), pages 642-653, Zurich, Switzerland, 1995.
- [SKdM92] E. Simon, J. Kiernan and D. de Maindreville. *Implementing High Level Active Rules on Top of a Relational DBMS*. In Proceedings of the 18th International Conference on Very Large Data Bases (VLDB '92), pages 315-326, Vancouver, Canada, 1992.
- [SPAM91] U. Schreier, H. Pirahesh, R. Agrawal and C. Mohan. *Alert: An Architecture for Transforming a Passive DBMS into an Active DBMS*. In Proceedings of the 17th International Conference on Very Large Data Bases (VLDB '91), pages 469-478, Barcelona, Spain, September 1991.
- [SR86] M. Stonebraker and L. Rowe. *The Design of POSTGRES*. In Proceedings of the International Conference on Management of Data (SIGMOD '86), Washington, DC, June 1986.
- [SR88] T. Sellis and L. Raschid. *Implementing Large Production Rules System in a DBMS Environment: Concepts and Algorithms*. In Proceedings of the International Conference on Management of Data (SIGMOD '88), pages 281-290, Chicago, June 88.
- [SR93] A. Srinivasan and J. Richards. *Analysis of GIS spatial data using knowledge-based methods*. In International Journal on Geographical Information Systems, 7(6):479-500, 1993.
- [SRD+91] A. Skidmore, P. Ryan, W. Dawes, D. Short, and E. O'Loughlin. *Use of an Expert System to Map Forest Soils from a Geographical Information System*. International Journal of Geographical Information Systems, 5(4):431-446, 1991.
- [SZ91] A. Segev and J. Zhao. *Data Management for Large Rule Systems*. In Proceedings of the 17th International Conference on Very Large Data Bases (VLDB '91), pages 297-307, Barcelona, Spain, September 1991.
- [Tan95] A. Tanaka. *Bancos de Dados Ativos*. Tutorial Notes. In SBBB '95, Recife, Brazil, October 1995.
- [UD89] S. Urban and L. Delcambre. *Constraint Analysis for Specifying Perspectives of Class Objects*. In Proceedings of the IEEE Data Engineering Conference, pages 10-17, 1989.
- [UD90] S. Urban and L. Delcambre. *Constraint Analysis: A Design Process for Specifying Operations on Objects*. IEEE Transactions on Knowledge and Data Engineering, 2(4):391-400, December 1990.
-

- [UKN92] S. Urban, A. Karadniece and R. Nannapaneni. *The Implementation and Evaluation of Integrity Maintenance Rules in an Object-Oriented Database*. In Proceedings of the IEEE Data Engineering Conference, pages 565-572, 1992.
- [Ull82] J. Ullman. *Principles of Database Systems*. Computer Science Press, 1982.
- [VK93] M. Van der Voort and M. Kersten. *Facets of Database Triggers*. Internal Report of CWI, 1009 AB Amsterdam, Netherlands, April 1993.
- [WCY89] J. Wu, T. Chen, and L. Yang. *QPF: a Versatile Query Language for Knowledge-based Geographical Information System*. International Journal of Geographical Information Systems, 3(1):51-58, 1989.
- [Web90] C. Webster. *Rule-based Spatial Search*. International Journal of Geographical Information Systems, 4(3):241-260, 1990.
- [WF90] J. Widom and S. Finkelstein. *Set Oriented Production Rules in Relational Database Systems*. In Proceedings of the International Conference on Management of Data (SIGMOD '90), pages 259-270, Atlantic City, NJ, May 1990.
- [Wid92] J. Widom. *The Starburst Rule System: Language Design, Implementation, and Applications*. IEEE Bulletin of the Technical Committee on Data Engineering, 15(1-4), pages 15-18, December 1992.
- [Wid93] J. Widom. *Deductive and Active Databases: Two Paradigms or Ends of a Spectrum?*. IBM Almaden Research Report, RJ9268 (81832), March 1993.
- [YS91] Z. Yang and D. Sharpe. *Design of Buffer Zones for Conservation Areas and a Prototype Spatial Decision Support System (SDSS)*. In Proceedings GIS/LIS '91, Volume 1, pages 60-70, 1991.
- [ZJ93] O. Zukunft and H. Jasper. *Integrating Semantic Integrity Constraints into an Object-Oriented Database Management System: A Pragmatic Approach*. Technical Report TR-IS-AIS-93-02, Fachbereich Informatik, Universität Oldenburg, December 1993.
-