UERJ

Dissertação de Mestrado em Engenharia de Computação
Área de Concentração em Geomática

ASPECTUML
APLICADA À CRIAÇÃO DE SISTEMAS
DE INFORMAÇÃO GEOGRÁFICA

Autor: Heitor Roméro Cajaty

Orientador: Orlando Bernardo Filho
Co-orientadora: Neide dos Santos

Programa de Pós Graduação em Engenharia de Computação
Área de Concentração em Geomática

Faculdade de Engenharia

ASPECTUML
APLICADA À CRIAÇÃO DE SISTEMAS
DE INFORMAÇÃO GEOGRÁFICA

Heitor Roméro Cajaty

Dissertação submetida ao corpo docente da Faculdade de Engenharia da Universidade do Estado do Rio de Janeiro – UERJ, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Computação – área de concentração Geomática.

Orientador: Professor Orlando Bernardo Filho D.Sc.

Co-Orientadora: Professora Neide Santos D.Sc.

Programa de Pós Graduação em Engenharia de Computação – Área de Concentração Geomática

Rio de Janeiro
Maio – 2006

CAJATY, HEITOR ROMÉRO
AspectUML aplicada à criação de Sistemas de Informação Geográfica [Rio de janeiro] 2006
xv, 107 p. 29,7 cm (FEN/UERJ, M.Sc., Programa de Pós Graduação em Engenharia de Computação – Área de Concentração Geomática, 2006)
Dissertação – Universidade do Estado do Rio de Janeiro – UERJ
1. Engenharia de Software.
2. Orientação a Aspectos.
3. Orientação a Objetos.
4. Sistemas de Informação Geográfica
5. Modelo de desenvolvimento de SIGs.
6. UML
I. FEN/UERJ II. Titulo

FOLHA DE JULGAMENTO

Título: AspectUML aplicada à criação de Sistemas de Informação Geográfica

Candidato: Heitor Roméro Cajaty

Programa de Pós-Graduação em Engenharia da Computação – Área de Concentração Geomática

Data da Defesa: 04/05/2006

Aprovada por:

_____

Orientador: Prof. Orlando Bernardo Filho, D.Sc., UERJ

_____

Co-orientador: Prof. Neide Santos, D.Sc., UERJ

_____

Prof. Vera Maria Benjamin Werneck, Phd., UERJ

_____

Prof. Sergio Palma da Justa Medeiros, D.Sc., UFRJ

Aos meus avós

Aos meus pais

À minha irmã

À minha madrinha

AGRADECIMENTOS

Aos meus avós, Glaucus, Maria Déa e Maria José pelo carinho e dedicação.

À minha madrinha e ao meu padrinho, Josephina e Antônio Heitor, pela proteção e dedicação.

Aos meus pais, Pedro Paulo e Lucia, pelo amor, carinho, dedicação, educação e suporte, sem os quais não seria possível alcançar este objetivo.

À minha querida irmã Paula, pela paciência, ajuda, suporte e compreensão em todos os momentos.

À Universidade do Estado do Rio de Janeiro, por me dar a oportunidade de realizar este curso e a graduação, sem qualquer ônus.

Ao meu orientador Orlando, pelo otimismo e perseverança com que conduziu este trabalho.

À minha co-orientadora, Neide, pela confiança e ensinamentos, sem os quais não conseguiria concluir este trabalho.

Enfim, a todos que colaboraram direta ou indiretamente para a conclusão deste trabalho.

Resumo da Dissertação apresentada à FEN/UERJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

ASPECTUML APLICADA À CRIAÇÃO DE SISTEMAS DE INFORMAÇÃO GEOGRÁFICA

Heitor Roméro Cajaty

Maio/2006

Orientadores: Orlando Bernardo Filho, D.Sc., UERJ.
Neide Santos, D.Sc., UERJ.

Programa de Pós-Graduação em Engenharia da Computação – Área de Concentração Geomática

A programação orientada a objetos é um importante paradigma e foi criada com a finalidade de trazer soluções à implementação, tentando facilitar o reuso e a manutenção do código, entretanto, não é suficientemente capaz de prover meios para separação dos interesses comuns no sistema. Com efeito, surgiu a programação orientada a aspectos (POA), introduzindo o conceito de aspectos e permitindo a separação dos requisitos transversais do sistema, afetando o desempenho ou semântica dos componentes, tratamento de exceções, consistência de dados, segurança etc. O presente trabalho objetiva estudar a programação orientada a aspectos para apresentar um novo processo de desenvolvimento de sistemas de informação geográfica usando a orientação a aspectos, como também criar uma extensão para a linguagem de modelagem unificada (UML), voltada para a representação dos aspectos (AspectUML), além de um sistema *web* (GD-SIGPOA) para o gerenciamento do desenvolvimento de SIGs usando a POA, todos objetivando auxiliar o analista de sistemas a visualizar a comunicação entre os objetos e os aspectos do projeto.

Abstract of Dissertation presented to FEN/UERJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

ASPECTUML ON GEOGRAPHICAL INFORMATION SYSTEMS

Heitor Roméro Cajaty

May/2006

Advisors: Orlando Bernardo Filho, D.Sc., UERJ.
Neide Santos, D.Sc., UERJ.

Program Computing Engineering - Geomatic

Object-oriented programming is an important paradigm and was created with the purpose to bring implementation solutions, trying to make reuse and code maintenance easier, however, it is not sufficiently capable to provide ways for separating common interests in the system. Thus, appeared the aspect-oriented programming (AOP), introducing the aspects concept and making it possible to separate crosscut concerns from components, affecting the performance or components semantic, exceptions treatment, data consistency, security etc. The present work aims to provide the study of aspect oriented programming to develop geographical information systems, proposing a new development process for GIS using AOP, a new extension for the Unified Modeling Language (AspectUML) and a web application (GD-SIGPOA) to manage the development of GIS using AOP, all of them assisting the systems engineer to visualize the communication between objects and aspects of the project.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABELAS

LISTA DE SIGLAS

AM/FM	– *Automated Mapping/Facility Management*
CAD	– *Computer-aided Design*
CADD	– *Computer-aided Design and Drafting*
HPGL	– *Hewlett-Packard Graphics Language*
JSP	– *Java Server Pages*
MADS	– *Modeling of Aplication Data with Spatio-Temporal features*
OO	– Orientação a Objetos
POA	– Programação Orientada a Aspectos
SGBD	– Sistema Gerenciador de Banco de Dados
SIG	– Sistema de Informação Geográfica
UML	– Linguagem de Modelagem Unificada (*Unified Modeling Language*)

CAPÍTULO I – INTRODUÇÃO

I.1 – APRESENTAÇÃO

Sistemas de informação geográfica são repositórios de informações associadas a uma determinada coordenada georreferenciada, que permitem a análise e visualização desses dados geográficos.

Com efeito, são comumente desenvolvidos utilizando a programação orientada a objetos, linguagem ineficaz para a implementação dos interesses multidimensionais do sistema, isto é, seus requisitos transversais.

Assim, para solucionar este problema, surgiu a linguagem orientada a aspectos, que permite a separação dos interesses multidimensionais do sistema de seus componentes funcionais, evitando a dispersão de código e o entrelaçamento desse, o que facilita o reuso e diminui o custo do desenvolvimento e das manutenções do *software*.

Por oportuno, antes do início do desenvolvimento de um sistema orientado a objetos, geralmente, o analista de *software* modela-o utilizando a linguagem de modelagem unificada (UML[1]). Entretanto, para a implementação do mesmo projeto utilizando a programação orientada a aspectos, devem ser integradas novas notações aos diagramas da UML para a representação dos aspectos

Portanto, o presente trabalho estudará os sistemas de informação geográfica e a programação orientada a aspectos, representada pela AspectJ[2], uma extensão do Java. Além disso, abordará a Linguagem de Modelagem Unificada (UML) e a *Java Server Pages* (JSP), objetivando a criação de um processo de implementação de sistemas de informação geográfica orientados a aspectos.

Para auxiliar o desenvolvimento de SIGs utilizando a programação orientada a aspectos foi criada a AspectUML, uma extensão da UML para a representação dos aspectos, bem como um sistema *web* para o acompanhamento do projeto e sua documentação.

[1] A *Unified Modeling Language* (ou linguagem de modelagem unificada) é uma linguagem de modelagem não proprietária, que permite aos desenvolvedores visualizarem os produtos de seu trabalho em diagramas padronizados.
[2] O AspectJ é uma extensão orientada a aspecto para a linguagem de programação Java.

I.2 – OBJETIVO

O objetivo desta dissertação de mestrado é apresentar um processo de desenvolvimento de sistemas de informação geográfica (SIGs) usando a programação orientada a aspectos.

O aludido processo visa facilitar a implementação de sistemas de informação geográfica com os benefícios do paradigma da orientação a aspectos, isto é, da modularização dos interesses multidimensionais do sistema (requisitos transversais), que, com a orientação a objetos, ficam dispersos e entrelaçados no código, gerando a perda de desempenho e altos custos no desenvolvimento e manutenção do *software.*

Para alcançar tais objetivos e auxiliar o mencionado processo propõe-se a utilização da AspectUML e de um sistema *web* para o acompanhamento do desenvolvimento de SIGs orientados a aspectos, ambos criados neste trabalho.

I.3 – ORGANIZAÇÃO

Esta dissertação está dividida em 6 (seis) capítulos e 4 (quatro) apêndices. No próximo capítulo, realiza-se um estudo sobre todos os conceitos e tecnologias necessários para o perfeito entendimento do escopo deste trabalho, definindo-se, por exemplo, o que são os sistemas de informação geográfica (SIG), a programação orientada a objetos e a aspectos, *Unified Modelling Language* (UML) e *Java Server Pages* (JSP).

No terceiro capítulo, apresenta-se um processo de implementação de SIGs, criado para auxiliar o desenvolvimento de projetos de sistemas de informações geográficas, que poderá variar conforme o tamanho da organização, a extensão das aplicações de SIG, o nível de detalhamento e a equipe comprometida com o projeto.

O quarto capítulo aborda uma análise detalhada sobre a utilização da programação orientada a aspectos no desenvolvimento de SIGs e seus benefícios. Além disso, propõe-se a AspectUML, uma extensão da UML para a representação dos aspectos, que auxiliará na modelagem e no desenvolvimento dos SIGs, como também de quaisquer outros sistemas orientados a aspectos. Por oportuno, adota-se um estudo de caso para exemplificar o uso da AspectUML e do processo de implementação de SIGs criados neste trabalho.

O quinto capítulo trata do sistema *web* desenvolvido em JSP para auxiliar o desenvolvimento, a manutenção e a documentação de sistemas de informação geográfica usando a programação orientada a aspectos.

CAPÍTULO II – FUNDAMENTOS TEÓRICOS

II.1 – INTRODUÇÃO

Este capítulo tem por objetivo proporcionar uma visão geral de todos os conceitos necessários para o perfeito entendimento do escopo deste trabalho.

Inicialmente, apresentam-se os sistemas de informação geográfica, sua evolução no tempo, implantação e aplicações. Abordam-se também os modelos conceituais de dados, as programações orientadas a objetos e a aspectos, além da Linguagem de Modelagem Unificada (UML), a *Java Server Pages* (JSP) e o banco de dados MySQL.

II.2 – SISTEMAS DE INFORMAÇÃO GEOGRÁFICA

Sistemas de informação geográfica (abreviadamente SIGs e, em inglês, *GIS - Geographical Information Systems*) são sistemas de informação construídos especialmente para armazenar, analisar e manipular dados geográficos, ou seja, dados que representam objetos e fenômenos em que a localização geográfica é uma característica inerente e indispensável para tratá-los.

Os primeiros SIGs foram desenvolvidos no Canadá, nos anos 60, como parte de um plano estratégico governamental para criar um inventário automatizado de recursos naturais. Na década de 80, seu uso foi difundido com a incorporação de funções espaciais devido à popularização e ao barateamento das estações de trabalho e *softwares* de banco de dados. Atualmente, as aplicações SIG incorporaram novas tecnologias, como sistemas especialistas e técnicas de orientação a objetos.

SIGs suportam diferentes tipos de dados e aplicações, de diversas áreas do conhecimento. Por exemplo, o gerenciamento de serviços de utilidade pública, demografia, cartografia, monitoramento costeiro, otimização de tráfego, controle de epidemias e planejamento urbano. A utilização destes sistemas facilita a integração de dados oriundos de fontes heterogêneas, de forma transparente para os usuários finais, os quais não se restringem a especialistas de um domínio específico, mas englobam cientistas, técnicos, gerentes, funcionários de uma empresa e o público em geral.

Devido à sua ampla gama de aplicações, há diferentes formas de se caracterizar SIGs. As definições de SIGs refletem a multiplicidade de usos e visões possíveis desta tecnologia e apontam para uma perspectiva interdisciplinar de sua utilização, isto é, cada tipo de definição

prioriza um aspecto diferente, por exemplo, o enfoque de banco de dados define SIG como um SGBD não convencional, geográfico, que garante o gerenciamento de dados geográficos.

Segundo o *Federal Interangency Coordinating Comitee (FICC),* os principais objetivos dos SIGs são capturar, gerenciar, manipular, analisar, modelar e exibir dados espacialmente referenciados para resolver problemas complexos de planejamento e gestão. Por outro lado, os SIGs podem ser caracterizados como sistemas de informação especialistas[3].

Em uma visão genérica, podemos considerar que um SIG tem os seguintes componentes: interface com o usuário, entrada e integração de dados, funções de processamento, visualização e plotagem, e, armazenamento e recuperação de dados.

A figura 2.1, adaptada de (CÂMARA, *et al.*, 1996), demonstra a visão genérica de um SIG e o relacionamento entre seus componentes.

Estes componentes relacionam-se de uma forma hierárquica, isto é, no nível mais interno do sistema, um SGBD oferece armazenamento e recuperação de dados espaciais e seus atributos. No nível próximo ao usuário, a interface homem-máquina define como o sistema é operado e controlado. No nível intermediário, um SIG deve possuir mecanismos de processamento de dados espaciais, ou seja, a entrada, edição, análise, visualização e saída.

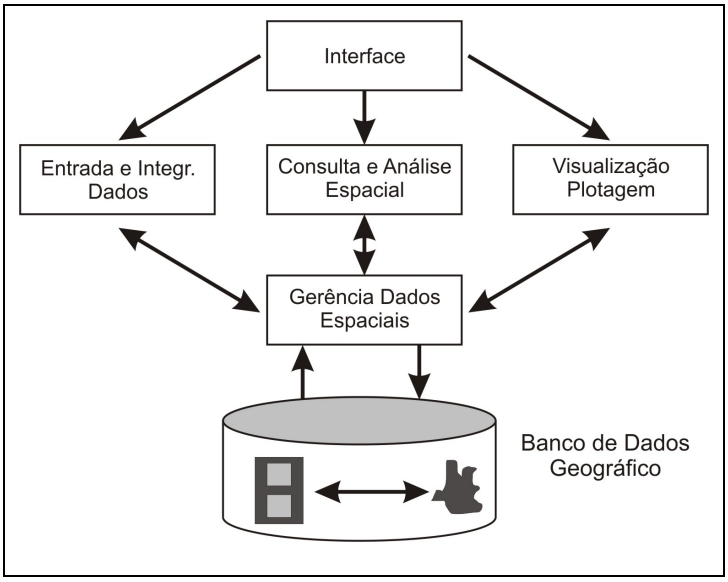


Figura 2.1 – Arquitetura de SIGs

[3] Sistemas especialistas – são sistemas que empregam o conhecimento humano para resolver problemas que requerem a presença de um especialista. O professor Edward Feigenbaum, da Universidade de Stanford, um dos principais pesquisadores em sistemas especialistas, definiu um SE como: *"(...)um programa inteligente de computador que usa conhecimento e procedimentos inferenciais, para resolver problemas que são bastante difíceis de forma a requererem, para sua solução, muita perícia humana. O conhecimento necessário para atuar a esse nível, mais os procedimentos inferenciais empregados, pode considerar-se um modelo da perícia aos melhores profissionais do ramo. O conhecimento de um sistema especialista consiste em fatos e heurísticas. Os fatos constituem um corpo de informação que é largamente compartilhado, publicamente disponível e geralmente aceito pelos especialistas em um campo. As heurísticas são em sua maioria privadas, regras pouco discutidas de bom discernimento (regras do raciocínio plausível, regras de boa conjectura), que caracterizam a tomada de decisão a nível de especialista na área. O nível de desempenho de um sistema especialista é função principalmente do tamanho e da qualidade do banco de conhecimento que possui*" (HARMON, 1988).

Além disso, geralmente estas funções de processamento do SIG operam sobre dados previamente selecionados pelo usuário. A relação entre as funções de processamento do SIG e seus dados é feita através de mecanismos de seleção e consulta, que definem restrições sobre o conjunto de dados, espaciais ou não.

Segundo (CÂMARA *et al.*, 1996), as funções de processamento são naturalmente dependentes dos tipos de dados envolvidos. A análise geográfica engloba funções como superposição, ponderação, medidas (área, perímetro), mapas de distância, tabulação cruzada, dentre outras. O processamento digital de imagens envolve funções como retificação, contraste, filtragem, realce e classificação. Modelos numéricos de terreno permitem a geração de mapas de declividade e aspecto, cálculo de volumes, análise de perfis, além da própria geração do modelo a partir de pontos esparsos ou linhas, entre outras funções. Operações sobre redes incluem caminhos ótimos, caminhos críticos e ligação topológica.

Os ambientes de visualização de um sistema são conseqüência do paradigma adotado para a interface. Quanto à produção cartográfica, alguns sistemas dispõem de recursos altamente sofisticados de apresentação gráfica, englobando a definição de uma área de plotagem, colocação de legendas, textos explicativos e notas de crédito. Já o aparecimento de padrões, como o PostScript e o HPGL[4], facilitam o desenvolvimento de funções de plotagem.

Os dados de um SIG são geralmente organizados sob a forma de bancos de dados geográficos. No passado, os dados geográficos eram armazenados em arquivos internos. Esta solução vem sendo substituída nos últimos anos pelo uso cada vez maior dos SGBDs.

II.2.1 – EVOLUÇÃO DOS SISTEMAS DE INFORMAÇÃO GEOGRÁFICA

Desde o primeiro sistema de informações geográficas até os dias de hoje, os SIGs passaram por muitas mudanças.

Com efeito, visando facilitar o entendimento da evolução destes, (CÂMARA, 1995) considerou a existência de três gerações.

A primeira geração, de 1980 a 1990, era baseada em CAD cartográfico, herdando a tradição de Cartografia, com suporte de bancos de dados limitados, cujo paradigma típico de trabalho era o mapa. Eram utilizados principalmente em projetos isolados, por isso chamados no inglês de *project-oriented GIS*, não se preocupando com a geração de arquivos digitais de dados.

[4] HPGL – *Hewlett-Packard Graphics Language* – um conjunto de comandos para controlar *plotters* e impressoras.

A segunda, de 1990 a 1997, foi concebida para uso em ambientes cliente-servidor, principalmente em empresas, utilizando bancos de dados relacionais e pacotes para tratamento de imagens. Esta geração ficou conhecida como *enterprise-oriented GIS*.

A terceira, de 1997 até os dias de hoje, caracteriza-se por ser baseada em bibliotecas digitais geográficas, com o gerenciamento de grandes bases de dados geográficos, acessíveis através de redes locais e remotas, públicas e privadas. O crescimento dos repositórios de dados e a necessidade do compartilhamento destes com outras instituições requer a utilização de tecnologias de bancos de dados distribuídos, seguindo requisitos de interoperabilidade, para permitir o acesso das informações por SIGs distintos. Assim, em face da troca de informações entre instituições da sociedade, esta geração ficou conhecida como *society-oriented GIS*.

A seguir se encontra a tabela 2.1, extraída de (CÂMARA *et al.*, 1996), que resume a evolução dos SIGs:

Tabela 2.1 – Evolução dos SIGs

Evolução dos SIGs	**1ª Geração** (1980 - 1990)	**2ª Geração** (1990 - 1997)	**3ª Geração** (1997 - ?)
Tecnologia	CAD, cartografia	BD, imagens	sist. distribuídos
Uso Principal	desenho de mapas	análise espacial	centro de dados
Ambiente	projetos isolados	cliente-servidor	multi-servidores
Sistemas	pacotes separados	sistema integrado	interoperabilidade

Por outro lado, segundo (KORTE, 1997), os SIGs evoluíram dentro da categoria de sistemas especialistas, aprimorando as capacidades tecnológicas de sistemas como CADD (*computer-aided design and draft*) e AM/FM (*automated mapping/facility manegement*), com relação à manipulação e análise de dados espaciais. Ainda utilizam outras tecnologias, como por exemplo, a dos Sistemas Gerenciadores de Banco de Dados - SGBDs.

II.2.1.1 – CADD

Computer-aided design and drafting (CADD) é uma tecnologia utilizada para a produção de mapas digitais, que vem substituindo efetivamente, os mapas produzidos manualmente.

Os mapas digitais oferecem inúmeras vantagens. A principal delas é a facilidade de modificar, corrigir ou atualizar os dados no mapa, sem a necessidade de refazê-los por completo, diminuindo, portanto, o custo de sua produção.

Como se demonstra nas três figuras a seguir (2.2, 2.3 e 2.4), todas adaptadas de (KORTE, 1997), os elementos de dados dos sistemas CADD incluem todos os gráficos necessários para desenhar um mapa: linhas, textos, símbolos etc., todos referenciados a um sistema de coordenadas planas e organizados em camadas (*layers*) ou níveis de informação. Geralmente, as camadas são utilizadas para organizar as características do mapa por temas como, por exemplo, rios, estradas, vegetação, ou então por tipos de dados, como símbolos ou textos.

Figura 2.2 - Elementos de um sistema CADD

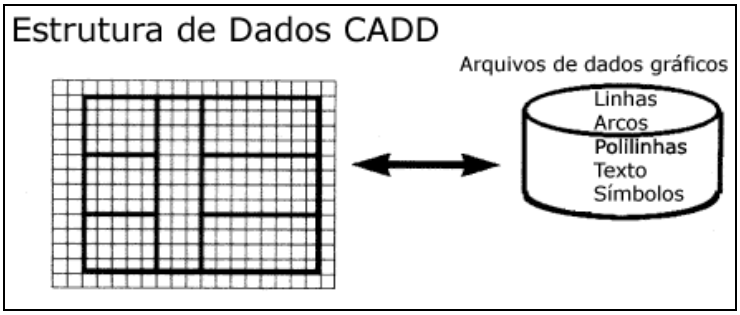


Figura 2.3 - Estrutura de dados de um sistema CADD

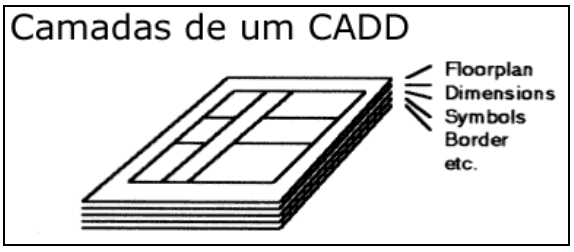


Figura 2.4 - Camadas de um sistema CADD

Além disso, os mapas digitais ainda possuem uma tabela onde são armazenados os registros das camadas e algumas características como nome, cor, tipo de linha e certas características-chave dos elementos nelas representados, como identificação e tipo de representação (linhas, textos ou símbolos). A tabela 2.2, adaptada de (KORTE, 1997), exemplifica as camadas e elementos presentes em um mapa digital:

Tabela 2.2 - Registro de dados de um sistema CADD

Registro	Tipo	Camada	Cor	Estilo	Grossura	Fonte	Nome	Geometria
1	Linha	Estrada	Vermelho	Tracejada	1	-	-	XY,XY
2	Linha	Rios	Azul	Cheia	2	-	-	XY,XY
3	Texto	Texto	Preto	-	1	-	Cais	XY
4	Símbolo	Símbolo	Verde	-	2	Times	Seta	XY

Além disso, os sistemas CADD oferecem outras vantagens, como a facilidade de organizar, armazenar e recuperar dados. Por estas razões, muitos engenheiros civis usam sistemas CADD para armazenar e manipular dados utilizados no processo de produção de mapas.

Entretanto, os sistemas CADD não podem ser utilizados para realizar análises com os dados do mapa, pois, embora seus elementos estejam organizados em camadas e encontrem-se referenciados a um mesmo sistema de coordenadas, não é possível informar como eles se relacionam no espaço. Por exemplo, em um sistema CADD, dois rios pertencentes a uma mesma camada se juntam em um ponto, porém essa informação não pode ser obtida através das tabelas de dados do sistema; uma linha pode pertencer a uma camada chamada "recursos hídricos", porém o banco de dados não reconhece como essa linha está conectada para formar uma linha poligonal fechada (que define uma área ou uma superfície) sem que um processamento adicional seja executado.

Geralmente, gerentes e responsáveis pelo planejamento de recursos perguntam questões que requerem análise de relações espaciais: O que está por perto? Quantos destes têm nesta área? Quais áreas são deste mesmo tipo e daquele tipo?

Os sistemas CADD não são capazes de responder a estas perguntas, pois as relações espaciais não estão definidas na sua estrutura de dados.

II.2.1.2 – AM/FM

O *Automed Mapping / Facility Management (AM/FM)* é um sistema baseado na tecnologia CADD, normalmente utilizado para mapear os componentes físicos de instalações industriais. Por exemplo, uma usina elétrica nuclear pode utilizar um sistema AM/FM para armazenar os locais e atributos das linhas de força, pólos, transformadores, locais de alta periculosidade ou com acesso restrito etc., sendo capaz de referenciar objetos espacialmente em um sistema de coordenadas, além de organizá-los em camadas, da mesma forma que nos sistemas CADD. Entretanto, os sistemas AM/FM são capazes de definir relacionamentos de

rede entre seus componentes, o que não era possível no CADD. A figura 2.5, a seguir, adaptada de (KORTE, 1997), representa a estrutura de dados de um AM/FM.

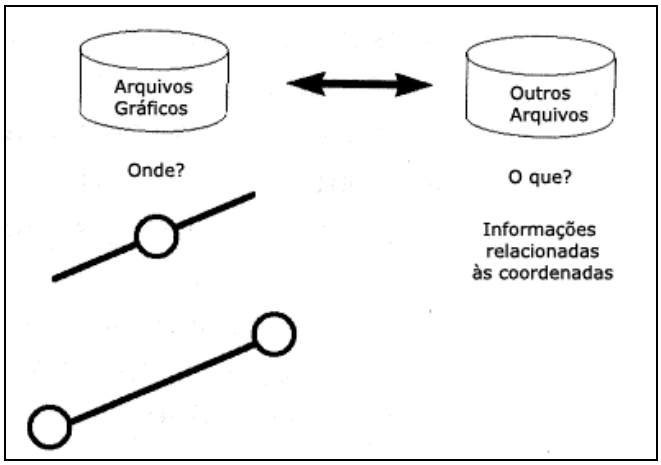


Figura 2.5 – Estrutura de dados de um sistema AM/FM

A rede identifica que componentes estão conectados a outros, e estas informações freqüentemente são armazenadas em uma tabela separada no banco de dados.

Além disso, outra importante característica dos sistemas AM/FM é que os atributos dos elementos representados nos sistemas, isto é, suas características, como dimensões, materiais, capacidade etc., ficam em uma outra tabela da base de dados, na qual os registros relacionam-se aos elementos gráficos do mapa digital por uma chave de identificação única.

II.2.1.3 – SIGs

Sistemas de informação geográfica suportam melhor aplicações de análise de dados geográficos, sendo similares aos sistemas CADD e AM/FM, com relação à referência dos elementos gráficos ao sistema de coordenadas X-Y e a separação das características do mapa digital por camadas ou temas. Em adição aos elementos gráficos, um SIG também armazena atributos, associados a esses elementos.

Por exemplo, em um SIG utilizado para controle municipal de pagamento de IPTU, imóveis são representados por áreas, e suas descrições devem incluir o número do lote, nome do proprietário, tamanho, zoneamento etc., sendo que estes atributos são armazenados em uma tabela distinta da dos elementos gráficos, como se visualiza nas figuras (2.6, 2.7 e 2.8) adiante, todas adaptadas de (KORTE, 1997):

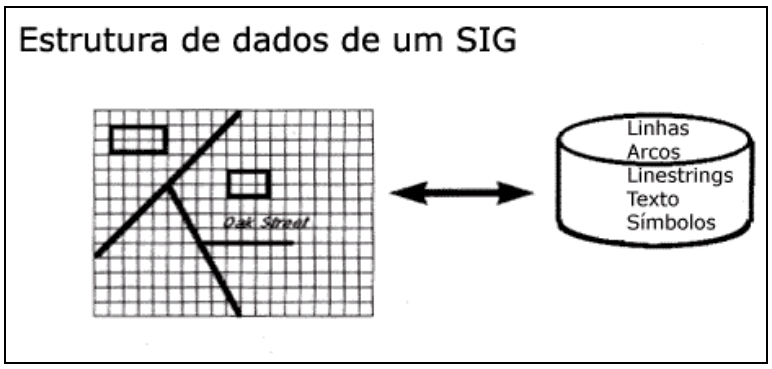


Figura 2.6 – Estrutura de dados de um SIG

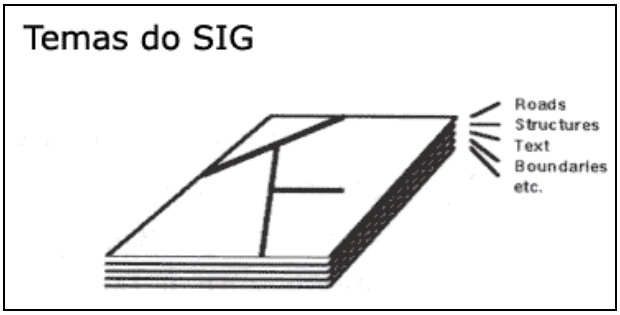


Figura 2.7 – Temas do SIG

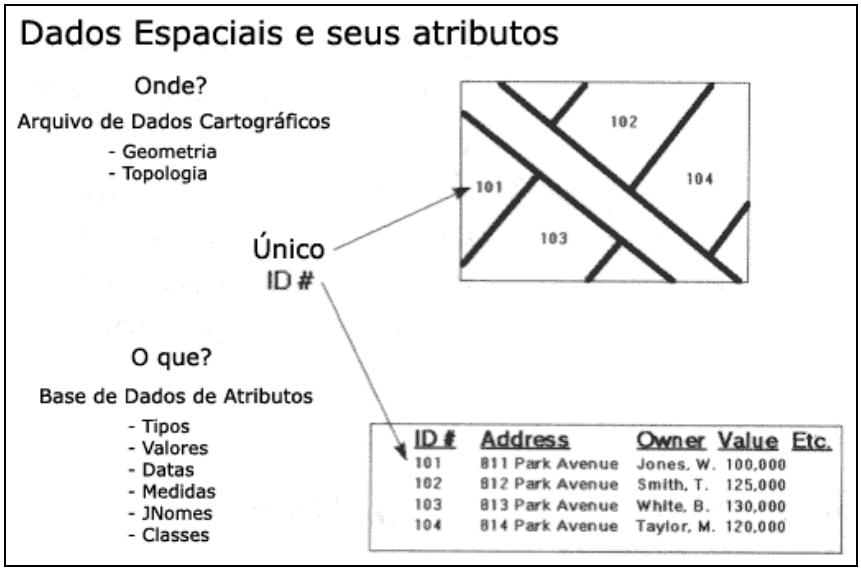


Figura 2.8 – Dados espaciais e seus atributos

Por outro lado, SIGs diferenciam-se dos sistemas CADD e AM/FM na forma como são feitos os relacionamentos entre os elementos espaciais. Essa convenção, conhecida como topologia de dados, vai além da mera descrição das características de localização e geometria do mapa. A topologia também descreve como características lineares do mapa são conectadas, como áreas são limitadas, e quais áreas são contíguas.

Para definir a topologia do mapa, o SIG usa uma estrutura especial na base de dados. Assim como nos sistemas CADD, todos os elementos do mapa estão relacionados a um sistema de coordenadas geográficas. Entretanto, diferentemente do sistema CADD, o qual define o mapa utilizando linhas ou símbolos, um SIG utiliza nós, linhas e áreas (ou pontos, arcos e polígonos). A figura a seguir, adaptada de (KORTE, 1997), ilustra os elementos básicos de um SIG.

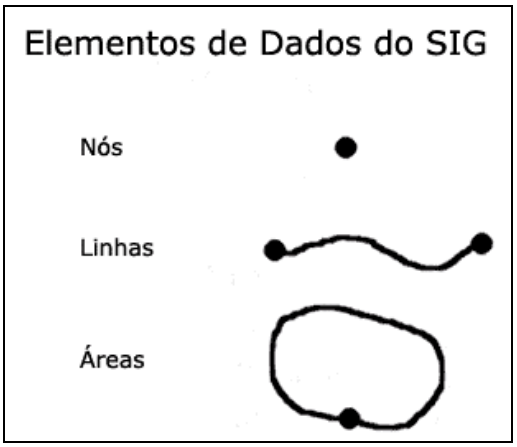


Figura 2.9 – Os elementos básicos de um SIG

Nós representam pontos de intersecção e os finais de linhas. Cada nó é exclusivamente numerado e localizado através de um par X-Y de valores de coordenadas geográficas.

Linhas são também exclusivamente numeradas. Suas geometrias são descritas por uma série de pares de coordenadas. Uma linha reta é definida somente por um par de coordenadas, representando seus pontos de início e fim, e pares de coordenadas adicionais podem ser necessários para representar características curvilíneas. Quanto mais pares de coordenadas são usados, mais precisa se torna a definição geométrica da linha.

A figura 2.10 e as tabelas a seguir, todas adaptadas de (KORTE, 1997), representam um mapa topológico e os registros de dados do SIG correspondentes.

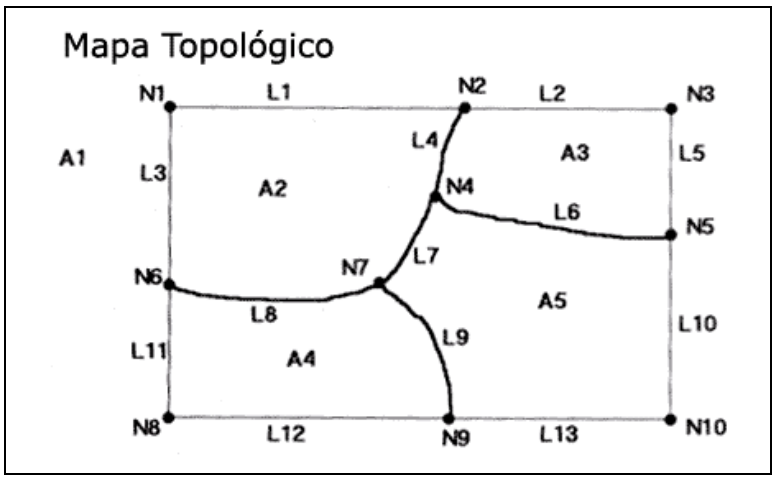


Figura 2.10 – Mapa topológico

Tabelas 2.3, 2.4 e 2.5 – Registros de dados do SIG

Tabela de Nós	
N1	XY
N2	XY
N3	XY
N4	XY
N5	XY
N6	XY
N7	XY
N8	XY
N9	XY
N10	XY

Tabela de Áreas	
A1	XY
A2	XY
A3	XY
A4	XY
A5	XY

Tabela de Linhas					
Linha	Nó Inicial	Nó final	Área a esq.	Área a dir.	Geometria
L1	N1	N2	A1	A2	XY, XY
L2	N2	N3	A1	A3	XY, XY
L3	N1	N6	A2	A1	XY, XY
L4	N2	N4	A3	A2	XY...XY
L5	N3	N5	A1	A3	XY, XY
L6	N4	N5	A3	A5	XY...XY
L7	N4	N7	A5	A2	XY...XY
L8	N6	N7	A2	A4	XY...XY
L9	N7	N9	A5	A4	XY...XY
L10	N5	N10	A1	A5	XY, XY
L11	N6	N8	A4	A1	XY, XY
L12	N8	N9	A4	A1	XY, XY
L13	N9	N10	A5	A1	XY, XY

Através de tabelas como essas, o *software* SIG pode iniciar em qualquer ponto e rapidamente determinar quais linhas estão conectadas, quais criam limites de uma área, e quais áreas são adjacentes. Essa capacidade é a base da análise espacial nos SIGs.

Em adição aos dados geométricos e espaciais nas tabelas anteriores, um SIG também contém dados de atributos, que estão associados a elementos topológicos (nós, linhas e áreas), fornecendo informações sobre eles. Por exemplo, em um SIG usado para controle de pagamento de IPTU, cada imóvel é definido como uma área, e seus atributos devem incluir o número do lote, nome do proprietário, tamanho, zoneamento etc.

Abordou-se o formato vetorial de SIGs, isto é, aqueles representados por pontos, linhas e polígonos. Há, ainda, outra forma de representar o espaço geográfico que é a matricial.

Na forma matricial o espaço geográfico é representado por uma matriz P(m,n) de elementos indexados, onde a cada elemento é associado um valor, que corresponde ao valor encontrado para um determinado fenômeno na localização daquele elemento ou um código de classe a qual este pertença. Entretanto, esta representação considera que o espaço geográfico foi projetado em uma superfície plana, subdividida em um número m x n de elementos, de modo que cada um deles corresponda a uma quota parte do terreno em estudo. A relação entre o tamanho do elemento e a porção do terreno representado é chamada de resolução.

A seguir encontram-se duas figuras de um mesmo território reproduzido na forma matricial, mas com resoluções distintas. O mapa da esquerda possui uma resolução quatro vezes menor que a do mapa da direita. Logo, as áreas dos elementos do primeiro mapa são quatro vezes maiores que as áreas dos elementos do segundo. Além disso, embora o mapa da direita seja capaz de representar o mesmo território com mais detalhes, a sua carga de informação é quatro vezes maior que a do mapa da esquerda.

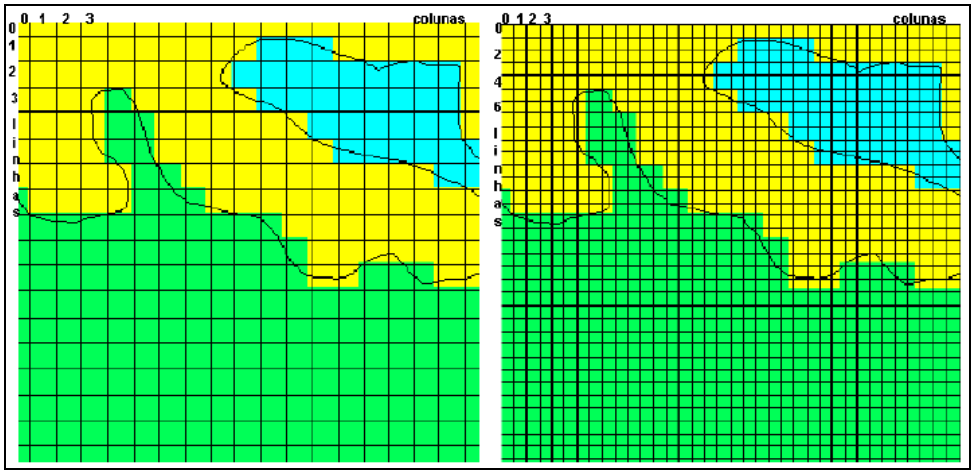


Figura 2.11 - Representações matriciais do mesmo território com resoluções diferentes

Finalmente, os SIGs são desenvolvidos para permitir análises rotineiras de dados espaciais e seus atributos ao mesmo tempo. O usuário está apto a procurar pelo atributo e relacioná-lo ao dado espacial e vice-versa. Por exemplo, no sistema de controle de pagamento do IPTU, o prefeito poderia perguntar onde estão os lotes residenciais da Barra da Tijuca que são maiores do que 800 m^2. O SIG pode responder tanto listando o número dos lotes, ou plotando sua localização no mapa da cidade. Esta é a capacidade que diferencia os SIGs dos CADD e AM/FM.

Todavia, deve-se observar que este tipo de mapa não representa diretamente os elementos da paisagem de um território, pois estes são melhor reproduzidos no formato vetorial. Por outro lado, a representação matricial se adapta melhor à modelagem de fenômenos contínuos no espaço como, por exemplo, dados meteorológicos, tipos de solos,

cobertura vegetal etc. A seguir encontra-se um comparativo entre os formatos de representação vetorial e matricial.

Tabela 2.6 - Comparativo: representação vetorial X representação matricial

Aspecto	Representação Vetorial	Representação Matricial
Relações espaciais entre objetos	Relacionamentos topológicos entre objetos disponíveis.	Relacionamentos espaciais devem ser inferidos.
Ligação com banco de dados	Facilita a associação de atributos a elementos gráficos.	Associa atributos apenas a classes do mapa.
Análise, Simulação e Modelagem	Representação indireta de fenômenos contínuos. Álgebra de mapas é limitada	Representa melhor fenômenos com variação contínua no espaço. Simulação e modelagem mais fáceis.
Escalas de trabalho	Adequado tanto a grandes quanto a pequenas escalas.	Mais adequado para pequenas escalas (1:25.000 e menores)
Algoritmos	Problemas com erros geométricos.	Processsamento mais rápido e eficiente.
Armazenamento	Por coordenadas (mais eficiente).	Por matrizes.

II.2.2 – IMPLANTAÇÃO DE UM SIG

O processo de implantação de um SIG divide-se em três grandes fases: a modelagem do mundo real, a criação do banco de dados geográfico e a operação.

A) Modelagem do mundo real

A modelagem do mundo real abrange a modelagem de processos e de dados, consistindo na seleção de fenômenos e entidades de interesse, generalizando-os e abstraindo-os. Diferentes temas, isto é, conjuntos de fenômenos, podem ser escolhidos para descrever visões distintas do mundo, para uma mesma região em um determinado instante.

B) Banco de Dados Geográficos

Um banco de dados geográficos é um repositório da informação coletada empiricamente sobre os fenômenos do mundo real (EGENHOFER, 1995). A criação de uma base de dados geográficos exige diversas etapas: a coleta dos dados referentes aos fenômenos de interesse identificados na modelagem; verificação e correção dos dados coletados; e georreferenciamento dos dados.

C) Operação

A operação refere-se tanto ao uso do SIG, quanto ao desenvolvimento de aplicações específicas por parte dos usuários a partir dos dados armazenados, reconstruindo visões (particulares) da realidade.

II.2.3 – APLICAÇÕES DE UM SIG

A evolução dos dispositivos de coleta e as facilidades computacionais em geral estão cada vez mais propiciando a ampliação do domínio de aplicações em SIG.

Os SIGs permitem que um fenômeno geográfico possa ser analisado de forma e precisão diferentes dependendo do objetivo da aplicação. Assim, um conjunto de dados armazenados em uma base de dados espaciais poderá ter tratamentos distintos.

Entretanto, cada aplicação requer a manipulação de fenômenos geográficos distintos, associados a diferentes características e propriedades que variam no espaço e no tempo.

Além disso, os usuários de um SIG têm perfis diversos, desde cientistas em um determinado domínio do conhecimento até técnicos ou especialistas em administração e planejamento urbano.

Assim, SIGs podem ser utilizados para administrar regiões como, por exemplo, dar suporte a um governo municipal, utilizando-se em cinco áreas gerais de aplicação: mapeamento de impostos (IPTU), planejamento e uso da terra, gerenciamento de escolas na região, obras públicas e engenharia, atendimentos de emergência etc.

II.3 – MODELOS CONCEITUAIS DE DADOS

A modelagem conceitual proporciona vantagens importantes para as aplicações que manipulam dados espaço-temporais. Por meio dela, o usuário pode expressar seu conhecimento sobre a aplicação, usando definições próximas de sua realidade, independentemente dos conceitos computacionais.

Por sua vez, a maioria dos modelos conceituais de dados pressupõe a definição de relacionamentos espaciais, tanto os métricos como os topológicos e de orientação. Entretanto, somente alguns deles prevêem restrições espaciais, dentre os quais se encontram o MADS e o OMT-G.

Além disso, para auxiliar o projetista na modelagem conceitual de dados, (LISBOA, 1999) definiu um *framework* conceitual que fornece um diagrama de classes básicas, utilizando a notação gráfica dos diagramas de classes da UML – o GeoFrame.

II.3.1 – MADS

O *Modeling of Aplication Data with Spatio-Temporal features* (MADS) é um projeto de modelo de dados conceitual desenvolvido pela *University of Lausanne* e o *Laboratoriere*

de Base de Données, da Escola Politécnica Federal de *Lausanne* na Suíça, elaborado com base nas características comuns encontradas em aplicações práticas estudadas. Oferece duas categorias de relacionamentos espaciais: as agregações espaciais e os relacionamentos topológicos. Aquelas correspondem às exigências mais comuns das aplicações geográficas.

Com efeito, os relacionamentos topológicos oferecidos pelo modelo incluem *disjoint*, *touches*, *crosses*, *overlaps* e *contains*. Segundo (PARENT, 1998), a agregação espacial é muito comum entre as aplicações espaço-temporais, sendo uma ligação binária direcionada, do tipo objeto composto para o tipo componente.

Além disso, é comum que alguns atributos do objeto composto ou componente estejam relacionados. Essas relações podem ser expressas por atributos derivados como, por exemplo, o total arrecadado de IPTU pela Prefeitura ser igual a soma do arrecadado nos distritos de um município. Também podem ser representadas por restrições de integridade como, por exemplo, as áreas dos distritos de um município precisam estar conectadas.

II.3.2 – GEO-OMT E OMT-G

O modelo de dados geográficos Geo-OMT foi proposto por (BORGES, 1997) e aprimorado em 1999 (BORGES, 1999), quando passou a ser denominado OMT-G, oferecendo suporte a um extenso conjunto de relacionamentos espaciais, que inclui, além dos relacionamentos topológicos *disjoint*, *touches*, *overlaps*, *equal*, *inside*, *contains*, *covers*, *coveredBy* e *crosses*, alguns tipos de relacionamentos métricos e de ordem como, por exemplo, perto de, acima/abaixo, sobre/sob, entre, coincide, em frente a, à esquerda e à direita.

Em 1999, o modelo passou a oferecer suporte a inúmeros tipos de restrições espaciais como, por exemplo, regras de dependência espacial, regras de disjunção e regras de conectividade. As regras de dependência espacial são restrições impostas pela existência de objetos agregados, onde a natureza do objeto agregado depende da existência de outro objeto geométrico, um sub-objeto. Segundo (BORGES, 1999), as citadas regras correspondem às primitivas espaciais de subdivisão, união e continência.

As regras de conectividade geralmente são garantidas pelos SIGs. Por exemplo, no caso de uma rede de esgoto, a conexão entre um nó e o segmento é garantida automaticamente pelo sistema (BORGES, 1997).

Por sua vez, as regras de disjunção são importantes na manutenção da integridade em relação à entrada de dados. Por exemplo, um trecho de rua é disjunto de uma edificação, o que implica na impossibilidade de existência de um trecho que cruze uma edificação.

II.3.3 – GEOFRAME

O GeoFrame é um *framework* conceitual que fornece um diagrama de classes básicas para auxiliar o projetista na modelagem conceitual de dados geográficos, utilizando a notação gráfica do diagrama de classes da UML. A figura 2.12, a seguir, extraída de (LISBOA, 2000), representa o GeoFrame.

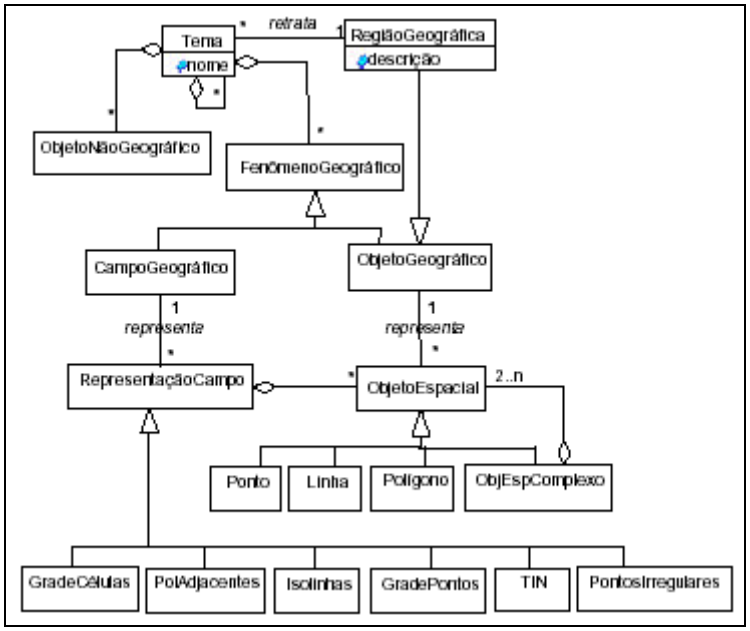


Fig. 2.12 – Diagrama de classes do GeoFrame

Um *framework* é um conjunto de classes inter-relacionadas que constitui uma implementação inacabada para um conjunto de aplicações de mesmo domínio, possibilitando a reutilização do projeto e do código implementado (SILVA, 1998).

As classes *Tema* e *RegiãoGeográfica* formam a base de qualquer aplicação geográfica. Cada uma dessas aplicações tem como objetivo gerenciar e manipular um conjunto de dados de uma determinada região de interesse, a qual constitui um banco de dados geográficos. Assim, para cada região geográfica podem ser especificados diversos temas.

Considerando-se, como exemplo, que o estado do Rio de Janeiro fosse a região de interesse, temas diversos como hidrografia, malha viária, edificações e vegetação, poderiam ser definidos.

Por outro lado, em um banco de dados geográficos podem existir dados convencionais e dados georreferenciados. Os dados convencionais, que não tem representação no espaço, são representados como instâncias das subclasses de *ObjetoNãoGeográfico,* que podem ou não estar associadas a um fenômeno geográfico.

Ressalta-se que fenômenos geográficos são percebidos na realidade, segundo as visões dicotômicas de campo e de objeto (LISBOA, 1999).

As classes *CampoGeográfico* e *ObjetoGeográfico* especializam a classe *FenômenoGeográfico*, permitindo ao projetista especificar de forma distinta, porém integrada, os campos e os objetos geográficos, respectivamente.

A classe abstrata *ObjetoGeográfico* generaliza todas as classes do domínio da aplicação que são percebidas na visão de objeto, ou seja, os elementos geográficos que podem ser individualizados e que possuem identidade própria, cujas características podem ser descritas através de atributos como, por exemplo, rios, rodovias e edificações. Sua representação espacial é dada pelas subclasses de *ObjetoEspacial*, que generalizam as formas de representação espacial dos objetos geográficos. Contudo, as subclasses são instanciadas por objetos espaciais com a forma de uma geometria primitiva do tipo ponto, linha ou polígono, representadas, respectivamente, pelas classes *Ponto*, *Linha* e *Polígono*.

A classe *ObjEspComplexo* representa objetos complexos como, por exemplo, um arquipélago.

A classe *CampoGeográfico*, por sua vez, generaliza os fenômenos que se enquadram na visão de campo. Cabe ressaltar que campos geográficos são modelados como funções sobre variáveis como, por exemplo, a temperatura, e, sua representação espacial é abstraída de forma diferente dos aspectos espaciais de um objeto geográfico.

A classe *RepresentaçãoCampo* generaliza todos os objetos na visão de campo, cujas instâncias são expressas como funções sobre uma variável.

Utilizam-se seis subclasses para representar espacialmente os fenômenos geográficos notados na visão de campo: *GradeCélulas*, *PolAdjacentes*, *Isolinhas*, *GradePontos*, *TIN* e *PontosIrregulares.*

Logo, cada classe identificada no domínio da aplicação deve ser modelada como uma subclasse de *ObjetoNãoGeográfico*, *CampoGeográfico* ou *ObjetoGeográfico*.

Além disso, (LISBOA, 2000) definiu um estereótipo para cada uma das aludidas classes, visando substituir os relacionamentos de generalização entre as classes do domínio da aplicação e as classes do Geoframe, facilitando a visualização do diagrama devido ao grande número de ligações.

O primeiro conjunto da figura abaixo apresenta os estereótipos para cada uma dessas classes. Os fenômenos geográficos percebidos na visão de objeto e de campo podem ser representados por múltiplas subclasses de *ObjetoEspacial* e *RepresentaçãoCampo*, respectivamente. Entretanto, para substituir as associações que resultam da modelagem da

representação espacial dos objetos e campos geográficos, (LISBOA, 2000) apresenta outro conjunto de estereótipos, ilustrados pela segunda e terceira colunas da figura a seguir.

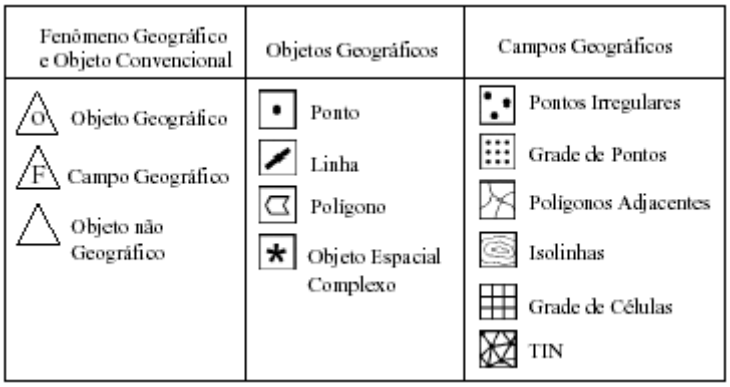


Figura 2.13 – Estereótipos para generalização

II.4 – ORIENTAÇÃO A OBJETOS

A orientação a objetos pressupõe uma organização de *software* em termos de coleções de objetos discretos, incorporando estrutura e comportamento próprios. É um paradigma essencialmente diferente do desenvolvimento tradicional de *software*, o estrutural, no qual rotinas são desenvolvidas e fracamente acopladas.

Um objeto é uma entidade do mundo real que tem uma identidade, possui um agrupamento de características e ações. Objetos podem representar entidades concretas (um arquivo no computador, um carro etc.) ou entidades conceituais (uma estratégia de jogo, uma política de mudanças de temperaturas etc.). Cada objeto ter sua identidade significa que dois objetos são distintos mesmo que eles apresentem exatamente as mesmas características.

Embora os objetos tenham existência própria no mundo real, em termos de linguagem de programação um objeto necessita de um mecanismo de identificação. Este mecanismo deve proporcionar uma identificação única, uniforme e independente do conteúdo do objeto, permitindo a criação de coleções de objetos, as quais são também objetos em si.

A estrutura de um objeto é representada em termos de atributos. Seu comportamento é representado por um conjunto de operações que podem ser executadas sobre ele. Objetos que possuem a mesma estrutura e idêntico comportamento são agrupados em classes.

Uma classe é um conjunto (possivelmente infinito) de objetos parecidos, ou seja, a criação de uma estrutura geral de onde objetos podem ser derivados (MATOS, 2002), sendo, também, uma abstração que descreve propriedades importantes para uma aplicação e simplesmente ignora o resto.

Cada objeto é chamado uma instância de uma classe. E, cada instância tem seus próprios valores para cada atributo, mas dividem os nomes dos atributos e métodos com as outras instâncias da classe. Implicitamente, cada objeto contém uma referência para sua própria classe, isto é, ele sabe sua origem.

Herança é o mecanismo do paradigma de orientação a objetos, baseado em um relacionamento hierárquico, que permite compartilhar atributos e operações entre classes, isto é, uma classe pode ser definida de forma genérica e depois refinada sucessivamente, em termos de subclasses ou classes derivadas. Cada subclasse incorpora, ou herda, todas as propriedades de sua superclasse (ou classe mãe) e adiciona suas propriedades únicas e particulares. As propriedades da superclasse não precisam ser repetidas em cada classe derivada.

O conceito de polimorfismo possibilita que uma mesma operação se comporte de forma diferente em classes diferentes. Por exemplo, a operação mover quando aplicada a uma peça de um jogo de xadrez tem um comportamento distinto do que quando aplicada em uma janela de um sistema de interfaces. Um método é uma implementação específica de uma operação para uma certa classe.

Polimorfismo também implica que uma operação de uma mesma classe possa ser implementada por mais de um método. O usuário não precisa saber quantas implementações existem para uma operação, ou explicitar qual método deve ser utilizado: a linguagem de programação deve ser capaz de selecionar o método correto a partir do nome da operação, classe do objeto e argumentos para a operação. Desta forma, novas classes podem ser adicionadas sem a necessidade de modificação do código existente, pois cada classe apenas define os seus métodos e atributos.

II.5 – ORIENTAÇÃO A ASPECTOS

Normalmente, antes de se iniciar o desenvolvimento de um sistema, na fase de análise de requisitos, definem-se seus requisitos funcionais (funcionalidades que o sistema deve oferecer ao usuário) e seus requisitos não funcionais (propriedades que expressam condições de comportamento e restrições acerca dessas funcionalidades). Estas propriedades, normalmente, afetam várias partes do sistema e podem dificultar seu desenvolvimento e manutenção. Em contrapartida, quando um sistema é melhor modulado, estas tarefas são realizadas de maneira mais clara.

A orientação a objetos é um paradigma importante, pois permite a melhor modularização do sistema. Ela foi criada com a finalidade de trazer soluções à

implementação, facilitando o reuso e a manutenção do código. Entretanto, para que os requisitos estipulados inicialmente sejam alcançados com eficácia, os critérios em relação à qualidade do desenvolvimento de um sistema aumentam cada vez mais, tornando os atuais paradigmas limitados para a implementação de sistemas mais complexos.

Nos sistemas de informação geográfica existem propriedades que não podem ser isoladas em uma única unidade de função, pois participam de várias unidades no sistema. Isso se torna mais evidente quando queremos cadastrar alguma informação relacionada a um ponto na base de dados do sistema, eis que a cada novo cadastro precisamos validar as informações contidas nos campos, para verificar se são consistentes.

Dessa forma, o código que implementa a validação "atravessa" o código que implementa a funcionalidade do cadastro. A linguagem orientada a aspectos define isso como interesses multidimensionais (*crosscut concern*).

Logo, pelo fato da orientação a objetos não ser suficientemente capaz de prover meios para separação dos interesses comuns no sistema, surgiu a programação orientada a aspectos, introduzindo o conceito de aspectos, que são propriedades que afetam a performance ou semântica dos componentes como, por exemplo, o tratamento de exceções, a consistência de dados, a segurança etc.

II.5.1 – REQUISITOS TRANSVERSAIS, ENTRELAÇAMENTO E DISPERSÃO DE CÓDIGO

Antes de iniciar o desenvolvimento de um sistema, devemos estabelecer seu objetivo geral, além de enumerar todos seus requisitos, bem como suas restrições.

Um requisito é uma consideração específica que deve ser satisfeita para cumprir o objetivo geral do sistema. Segundo (LADDAD, 2003) existem duas categorias de classificação para os requisitos do sistema: requisitos principais (*core concern*) e requisitos transversais (*crosscutting concern*). Um requisito transversal utiliza propriedades presentes em múltiplos módulos do sistema, enquanto que o principal é aquele que lida com a lógica do negócio propriamente dita.

Um sistema de informação geográfica de uma empresa de grande porte, por exemplo, deve se preocupar com diversos requisitos transversais, quais sejam a autorização, *logging*, persistência de dados, gerenciamento de armazenamento e outros. Tais requisitos estão distribuídos em vários módulos do sistema. Por exemplo, sempre que um módulo necessitar gravar alguma informação referente ao *log*, este será afetado por esse requisito. A figura 2.14, a seguir, adaptada de (LADDAD, 2003), demonstra os requisitos transversais de *logging* e de persistência presentes em vários módulos do sistema.

Como se pode notar da citada figura, os códigos de *logging* e de persistência estão dispersos por inúmeros módulos do sistema, caracterizando a dispersão de código ou *scattering code*.

Ademais, os requisitos de *logging*, persistência e a própria lógica do negócio encontram-se entrelaçados no mesmo componente, o qual, a princípio, deveria cuidar apenas da lógica negocial do sistema. Este entrelaçamento gera, portanto, um acoplamento indesejado, também conhecido como entrelaçamento de código ou, no inglês, *tangled code*.

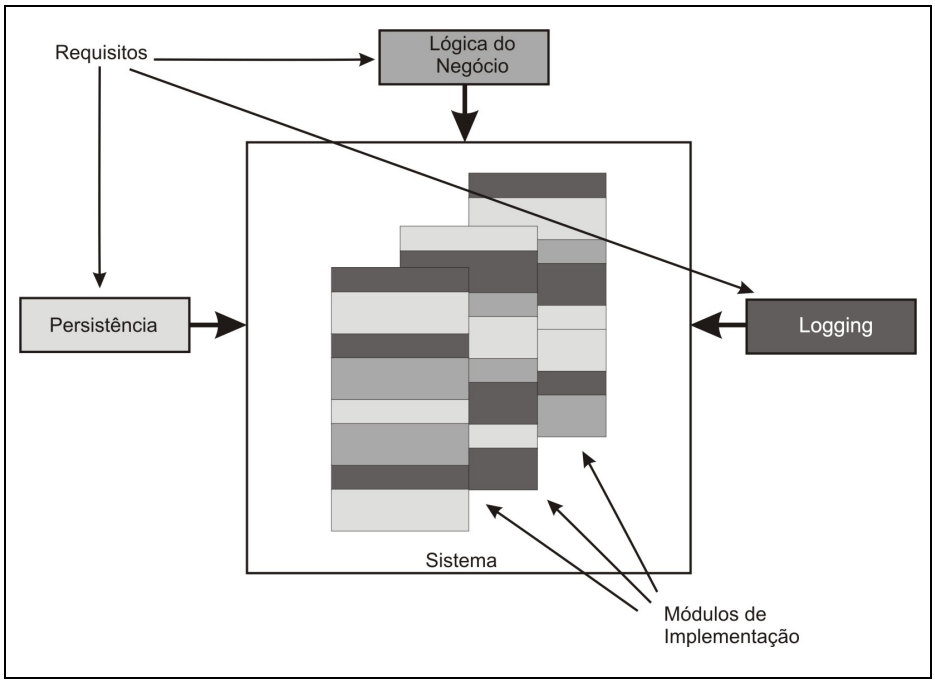


Figura 2.14. Requisitos Transversais no Sistema

Uma solução para problemas de acoplamentos indesejados seria a programação orientada a aspectos.

II.5.2 – CONCEITUAÇÃO

A programação orientada a objetos (POO) proporcionou à Computação a possibilidade de modelar sistemas conforme estes são vistos no mundo real, diminuindo os custos advindos das inúmeras reuniões de análise de requisitos. Dessa forma, cada objeto deve permitir o encapsulamento de dados e seus métodos para o cumprimento de determinados requisitos.

Tais requisitos são capturados e abstraídos em unidades denominadas classes, que são protótipos (tipos) para os objetos de um sistema, ou seja, o objeto é uma instância de uma classe.

Porém, geralmente existem determinados requisitos no sistema que, por sua própria natureza, estão distribuídos em diversos componentes. Os referidos requisitos são denominados requisitos transversais ou, no inglês, *crosscutting concerns* (KICZALES *et. al.*, 1997).

Os exemplos mais comuns de requisitos transversais estão ligados diretamente à segurança dos sistemas, como por exemplo, as funcionalidades de *logging*, autorização, persistência de dados, transações etc.

Ocorre que a implementação destas funcionalidades transversais utilizando a programação orientada a objetos leva a duas condições adversas: o entrelaçamento (*tangled code*) e a dispersão (*code scattering*) de código. O entrelaçamento de código é caracterizado por englobar em uma mesma unidade funcional código referente a diversos requisitos. Já a dispersão de código pode ser notada quando um determinado requisito do sistema encontra-se espalhado por diversas unidades funcionais.

Sistemas com entrelaçamento de código e dispersão geralmente possuem problemas de manutenção; forte acoplamento entre os objetos modelados; baixa reutilização de código; e, baixa rastreabilidade do código.

A programação orientada a aspectos (POA) é um novo paradigma de programação que objetiva separar totalmente os requisitos transversais dos funcionais no sistema, propondo que estes sejam implementados em uma unidade de programação nova, chamada aspecto. Esta nova unidade seria incorporada quando necessária aos objetos do sistema que implementam os requisitos funcionais.

A implementação separada dos aspectos dos objetos possibilita maior reuso do código, facilita a manutenção e provê melhor performance. A seguir abordaremos a linguagem AspectJ, uma extensão para Java que permite implementar os interesses multidimensionais.

II.5.3 – ASPECTJ

Conforme visto, a idéia central da linguagem de programação orientada a aspectos é a de proporcionar a separação dos requisitos transversais nos sistemas complexos, aqueles que a linguagem orientada a objetos é incapaz de modular, dos outros componentes do sistema.

A linguagem AspectJ é uma simples e prática extensão para Java, com suporte para a implementação de requisitos transversais, que torna possível definir uma implementação adicional, a qual pode ser executada em um ponto bem definido na execução do programa. Este método de programação é conhecido no inglês como *dynamic crosscutting mechanism*.

Os principais componentes de uma linguagem orientada a aspectos são:

- Pontos de junção (*join points*)
- Pontos de corte (*pointcuts*)
- Comportamentos (*advices)*
- Aspectos, componentes encapsulados, como as classes.

II.5.3.1 – PONTOS DE JUNÇÃO (*JOIN POINTS*)

Pontos de junção ou, no inglês, *join points*, são pontos bem definidos na execução do programa onde um requisito vai atravessar (*crosscut*) a aplicação (GRADECKI, 2003). Os pontos de junção podem ser chamadas aos métodos, ao construtor, tratamento de exceções, ou outros pontos na execução de um programa.

Por exemplo, se um dos requisitos de um SIG, criado utilizando linguagem orientada a aspectos, for o *log* de todas as consultas feitas a base de dados, para facilitar o desenvolvimento do sistema, uma classe de transações será criada para gerenciar toda a comunicação com a base de dados dos componentes da camada negocial da aplicação. Com o componente transacional, um método chamado atualizarTabelas() cuidará de todas as atualizações na base de dados. Para implementar totalmente o requisito transversal é necessário adicionar um código para o método, registrar o momento que o método foi chamado pela primeira vez, além de incluir outro no final para registrar o momento e adicionar um *flag* de sucesso ao *log*. Assim, o ponto de junção para a implementação é o nome do método junto (possivelmente) com o nome da classe.

A seguir se encontra um ponto de junção:

DBTrans.atualizarTabelas(String);

A sintaxe exata pode variar de linguagem para linguagem, mas o objetivo do ponto de junção é combinar pontos de execução bem definidos.

II.5.3.2 – PONTOS DE CORTE (*POINTCUTS*)

No item acima se definiu o conceito de pontos de junção como um ponto de execução bem definido na aplicação. Dessa forma, é necessário um construtor que fale para a linguagem orientada a objetos quando esta deve executar o ponto de junção. Por exemplo, pode-se querer que a linguagem orientada a aspectos execute o ponto de junção somente quando ele for usado em uma chamada (*call*) de um objeto para outro ou possivelmente uma

chamada dentro do mesmo objeto. Para solucionar esta situação, pode-se definir um designador chamado *call()* que receberá um ponto de junção como parâmetro:

call(DBTrans.atualizarTabelas(String))

O designador informa a linguagem orientada a aspectos que o ponto de junção *DBTrans.atualizarTabelas(String)* deve ser combinado somente quando for parte de uma chamada de método.

Em alguns casos, pode-se usar múltiplos designadores para agrupar os pontos de junção. Os principais encontram-se na tabela 2.7, extraída de (MONTEIRO *et al.*, 2003):

Tabela 2.7 - Principais designadores em AspectJ

Designador	Características
Call (Signature)	Invocação do método / construtor identificado por assinatura
Execution (Signature)	Execução do método / construtor identificado por assinatura
Get (Signature)	Acesso a atributo identificado por assinatura
Set (Signature)	Atribuição do atributo identificado por assinatura
This (Type pattern)	Objeto em execução é instância do padrão tipo
Target (Type pattern)	Objeto de destino é instância do padrão tipo
Args (Type pattern)	Os argumentos são instância do padrão tipo
Within (Type pattern)	O código em execução está definido em padrão tipo

De qualquer maneira, uma outra construção chamada ponto de corte é tipicamente usada para agrupar os pontos de junção. Um ponto de corte pode ser nomeado ou anônimo, assim como uma classe. Por exemplo, adiante, o ponto de corte é chamado *atualizarTabela()*. Ele possui um único designador para todos os chamados ao ponto de junção definido:

```
Pointcut atualizarTabela()
call(public String DBTrans.atualizarTabelas(String))
```

O ponto de corte é usado nas estruturas chamadas *advice*, descritas a seguir.

II.5.3.3 – COMPORTAMENTOS (*ADVICES*)

Na maioria das especificações da programação orientada a aspectos, o código de comportamento (*advice)* pode ser executado em três diferentes momentos quando um *join point* é executado: antes, durante e depois. Em cada um dos casos, um ponto de corte (*pointcut*) deve ser acionado (*triggered*) antes que qualquer código do *advice* seja executado. A seguir se encontra um exemplo do uso do *before advice*:

```
Before(String s) : atualizaTabela(s) {
        System.out.println("Parâmetro passado – " + s);
}
```

Uma vez que o ponto de corte é acionado, o código apropriado do comportamento é executado. No caso do exemplo anterior, o código do *advice* é executado antes da execução do ponto de junção. O argumento *string* é passado para o código para ser usado se necessário.

Na maioria dos sistemas de programação orientada a aspectos, tem-se acesso ao objeto associado ao ponto de junção, assim como a outras informações específicas para o próprio ponto de junção.

II.5.3.4 – ASPECTOS (*ASPECTS*)

Com a utilização da programação orientada a aspectos, pode-se reduzir o entrelaçamento de código e sua desorganização, pois a sintaxe do aspecto foi desenvolvida para tratar o encapsulamento dos pontos de junção, pontos de corte e *advices*.

Aspectos são criados de forma muito semelhante às classes e permitem o completo encapsulamento do código de um requisito transversal em particular (GRADECKI, 2003). A seguir encontra-se encontra um exemplo de aspecto:

```
public aspect AspectoTabela {
      pointcut atualizaTabela(String s) :
      call(public String DBTrans.atualizaTabelas(String) &&
      args(s);
      before(String s) : atualizaTabela(s) {
          System.out.println("Parâmetro passado – " + s); }
}
```

O aspecto *AspectoTabela* é um objeto que implementa um requisito transversal relacionado ao método *atualizaTabela()*. Toda a funcionalidade requerida para este requisito está encapsulada na sua própria estrutura.

Uma vez que um requisito transversal é escrito em AspectJ, uma boa parte do trabalho ainda precisa ser realizada para carregar a aplicação primária e os requisitos transversais como um sistema completo. Esta tarefa de integrar o código do requisito transversal e o da aplicação primária é chamada combinação (*weaving)*.

A tabela a seguir lista os diferentes tipos de combinação:

Tabela 2.8 - Diferentes tipos de combinação (*weaving*)

Tipo	Descrição	Ferramenta utilizada
Tempo de Compilação	O código fonte das linguagens primária e de aspectos é combinado antes de ser colocado na fase de compilação, na qual o byte code é produzido. O AspectJ 1.0.x usa esta forma de combinação.	Compilador
Tempo de Ligação	A combinação ocorre após os códigos das linguagens primária e de aspectos foi transformado em byte code. O AspectJ 1.1.x usa esta forma de combinação.	*Linker*
Tempo de Carregamento	A combinação ocorre quando as classes são carregadas pelo classloader. Finalmente a combinação ocorre no nível do byte-code.	*Classloader* do Java
Tempo de Execução	A máquina virtual é responsável por pela detecção dos pontos de junção e pelo carregamento e execução de aspectos.	Máquina Virtual

Por oportuno, cabe ressaltar que antes do desenvolvimento de qualquer sistema orientado a objetos, isto é, da fase de sua implementação, precede-se a fase de modelagem, recomendada pela engenharia de *software*, e atualmente executada com o uso da UML (*Unified Modelling Language*), que será abordada a seguir.

II.6 – UML

A Linguagem de Modelagem Unificada (UML), ou, no inglês, *Unified Modelling Language*, criada pelo *Object Management Group* (OMG), é uma notação para especificar, documentar e visualizar modelos de sistemas de *software* orientados a objetos.

Atualmente, é um padrão da indústria de *software* para a descrição gráfica dos sistemas, sendo composta por inúmeros elementos que representam as diferentes partes de um sistema.

Os elementos UML são usados para criar diagramas, que representam um determinado módulo ou um ponto de vista do sistema. Os mais utilizados são: a) diagrama de casos de uso; b) diagrama de classes; c) diagrama de pacotes; d) diagrama de seqüência; e) diagrama de colaboração; f) diagrama de estados; g) diagrama de atividade; h) diagrama de componentes; e, i) diagrama de distribuição.

II.6.1 – DIFERENTES VISÕES DO SISTEMA

A UML permite ao analista representar o sistema seguindo diferentes visões. Cada uma delas possui uma grande dependência das outras, garantindo assim a coerência e a completeza do modelo, e, conseqüentemente, do *software*. A relação entre os diagramas não é assegurada pela UML, mas deve ser garantida pelo projetista.

Os diagramas podem ser agrupados segundo aspectos da visão que proporcionam em:

- Diagrama de comportamento externo: dão uma visão externa do sistema e dos objetivos que os atores externos têm do sistema;

- Diagramas estruturais: dão uma visão estática da estrutura de suporte do sistema, sobre a qual ele será construído;

- Diagramas de comportamento interno: tratam dos processos que ocorrem entre as estruturas que compõem o sistema e dão uma visão da dinâmica interna do sistema;

- Diagramas de implementação: descrevem como estas estruturas são implementadas em *software* e *hardware*.

A seguir se demonstra como os diagramas podem ser integrados para formar um sistema:

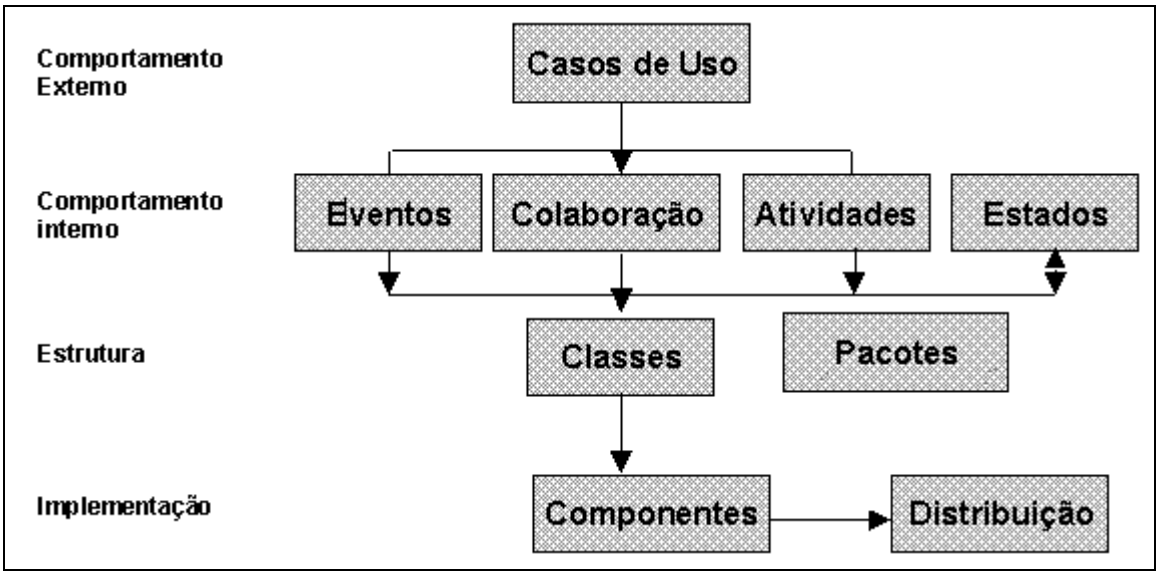


Figura 2.15. Relações entre os diagramas da UML

II.6.2 – DIAGRAMAS DE CASOS DE USO

O diagrama de casos de uso é utilizado para descrever e definir o conjunto de requisitos funcionais do sistema, a partir do ponto de vista do usuário. Assim, é o ponto de partida para todo o processo de modelagem. O diagrama de caso de uso e composto de atores (externos) e casos de uso, onde segundo (BARROS, 2000), *"(...)um ator é conectado a um ou mais casos de uso através de associações, e tanto atores quanto casos de uso podem possuir*

relacionamentos de generalização que definem um comportamento comum de herança em superclasses especializadas em subclasses(...)"

Assim, um caso de uso descreve um objetivo que um ator externo ao sistema tem com este sistema. Por oportuno, um ator pode ser ou não um elemento humano que interage com o sistema.

O diagrama de casos de uso representa graficamente esta interação, definindo o contexto do sistema. Os atores são descritos por simplificadas representações de uma figura humana, enquanto os casos de uso são elipses contendo cada uma o nome de um caso de uso.

Os atores se comunicam com os casos de uso, o que é representado por uma linha unindo os dois elementos. Opcionalmente, uma seta pode representar o fluxo principal de informação nesta interação e ajudar a leitura do caso de uso.

Adiante se visualiza um exemplo de diagrama de casos de uso:

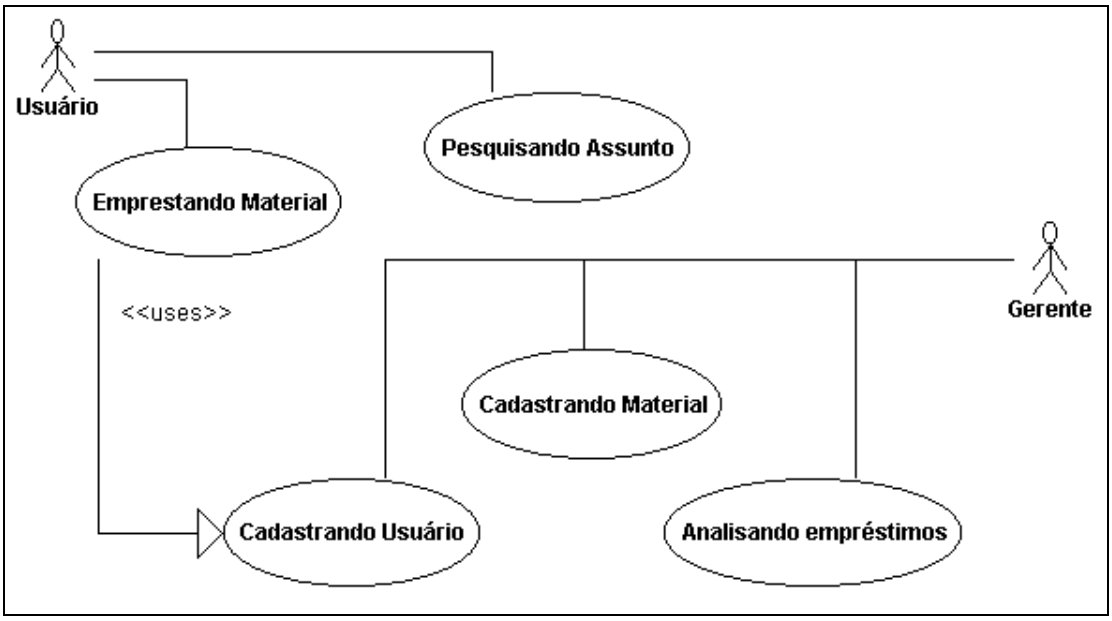


Figura 2.16. Exemplo de diagrama de casos de uso

Uma vez que os casos de uso representam um objetivo do ator, é comum dar como nome aos casos de uso frases verbais curtas no infinitivo (Emprestar Material) ou no gerúndio (Emprestando Material) onde o sujeito é normalmente o ator.

Cada caso de uso deve receber uma descrição textual que permita o entendimento de seu objetivo. Esta descrição pode ser detalhada em cenários, que são instâncias de um caso de uso, isto é, uma situação onde o ator utilizou o sistema para conseguir atingir o objetivo do caso de uso. Um cenário pode ser considerado otimista se o ator obteve sucesso no seu objetivo, pode ser pessimista se o ator não conseguiu atingi-lo e ocorreu uma situação de exceção, ou, o cenário pode ser alternativo, quando em uma situação de exceção, o ator optou

por caminhos alternativos. Assim para cada caso de uso pode se descrever um texto com: cenários otimistas, pessimistas e alternativos.

II.6.3 – DIAGRAMAS DE CLASSE

Os diagramas de classe descrevem as classes que formam a estrutura do sistema e seus relacionamentos. As relações entre elas podem ser associações, agregações ou heranças. As classes possuem além de um nome, os atributos e as operações que desempenham para o sistema. Uma relação indica um tipo de dependência entre as classes, essa dependência pode ser forte, como no caso da herança ou da agregação, ou mais fraca, como no caso da associação, mas indicam que as classes relacionadas cooperam de alguma forma para cumprir um objetivo para o sistema.

A UML, por ser uma linguagem de descrição, permite diferentes níveis de abstração aos diagramas, dependendo da etapa do desenvolvimento do sistema em que se encontram. Assim, os diagramas de classe podem exibir, nas fases iniciais da análise, apenas o nome das classes e, em uma fase seguinte, os atributos e operações, como na figura 2.17. Por fim, em fases avançadas do projeto pode-se mostrar os tipos dos atributos, a visibilidade, a multiplicidade das relações e diversas restrições. Existem elementos na UML para todas estas representações.

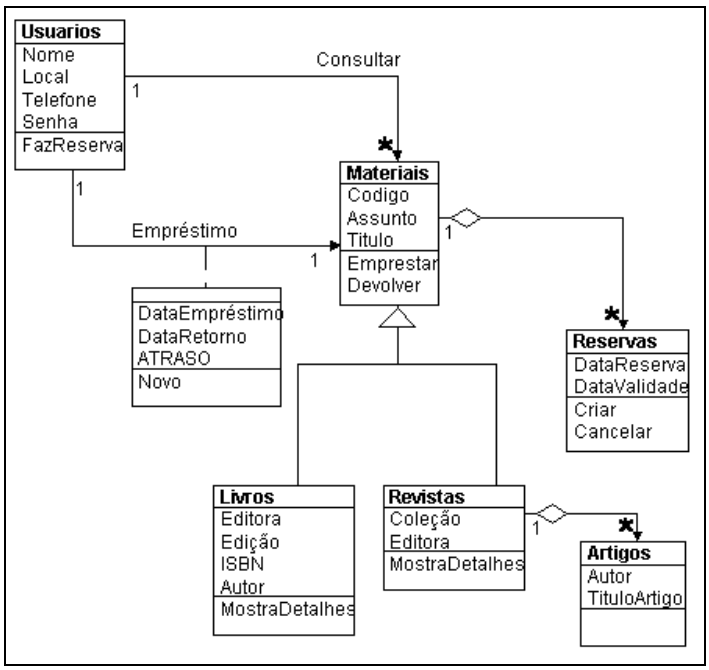


Figura 2.17. Exemplo de diagrama de classes

Ao final do processo de modelagem, o diagrama de classes pode ser traduzido em uma estrutura de código que servirá de base para a implementação dos sistemas. Ressalte-se, no entanto, a inexistência no diagrama de classes de informações sobre os algoritmos que serão utilizados nas operações, além da dinâmica do sistema, uma vez que não há elementos sobre o processo ou sobre a seqüência de processamento neste modelo. Estas informações são representadas em outros diagramas, nos de seqüência de eventos ou diagramas de estado.

II.6.4 – DIAGRAMAS DE PACOTE

Normalmente, um diagrama de classes pode ser exageradamente grande para representar todo um sistema. Por isso, é conveniente usar-se um elemento para organizar os subsistemas do modelo, os diagramas de pacote. Um pacote representa um grupo de classes (ou outros elementos) que se relaciona com outros pacotes através de uma relação de dependência.

Assim, um diagrama de pacotes pode ser utilizado em qualquer fase do processo de modelagem e visa organizar os modelos.

Na figura a seguir, o pacote de classes das janelas, que cuida da interface da aplicação, dependente funcionalmente das classes de negócio para cumprir suas atividades.

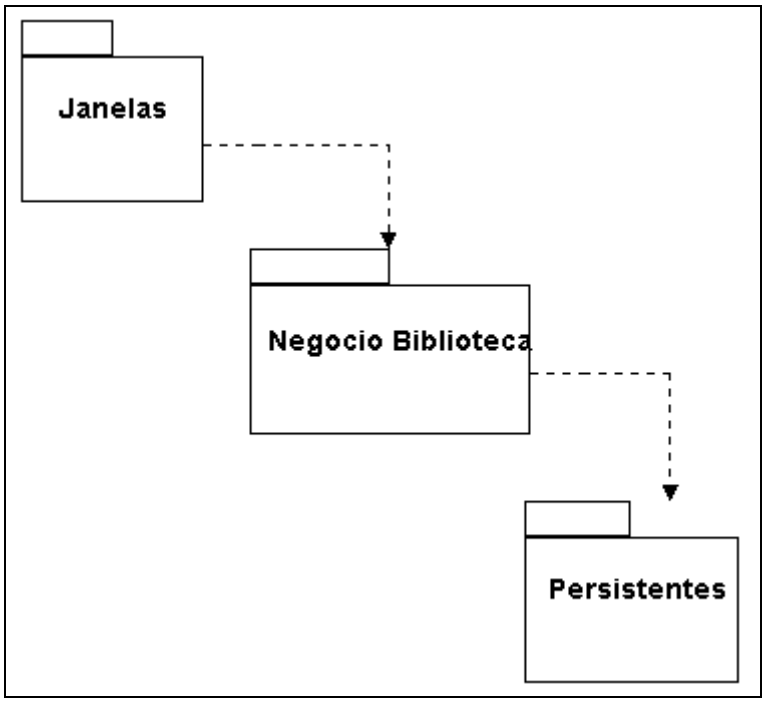


Figura 2.18. Exemplo de diagrama de pacotes

II.6.5 – DIAGRAMAS DE SEQÜÊNCIA DE EVENTOS

Os casos de uso, como visto, representam conjuntos de cenários que descrevem os diferentes processos que ocorrem no sistema.

Por sua vez, o diagrama de seqüência "*é um diagrama de interação que dá ênfase à ordenação temporal de mensagens*" (BOOCH *et al*, 2000), permitindo modelar estes processos com a troca de mensagens (eventos) entre os objetos do sistema, os quais são representados por linhas verticais, e suas mensagens por setas, que partem do objeto que invoca um outro objeto. As setas podem ser cheias para indicar uma mensagem de chamada ou tracejadas para indicar uma mensagem de retorno.

A figura 2.19, a seguir, mostra um exemplo deste diagrama, onde se pode observar que o tempo segue o eixo vertical, de cima para baixo.

Além disso, devem ser desenhados tantos diagramas de seqüência quantos cenários foram levantados no diagrama de casos de uso.

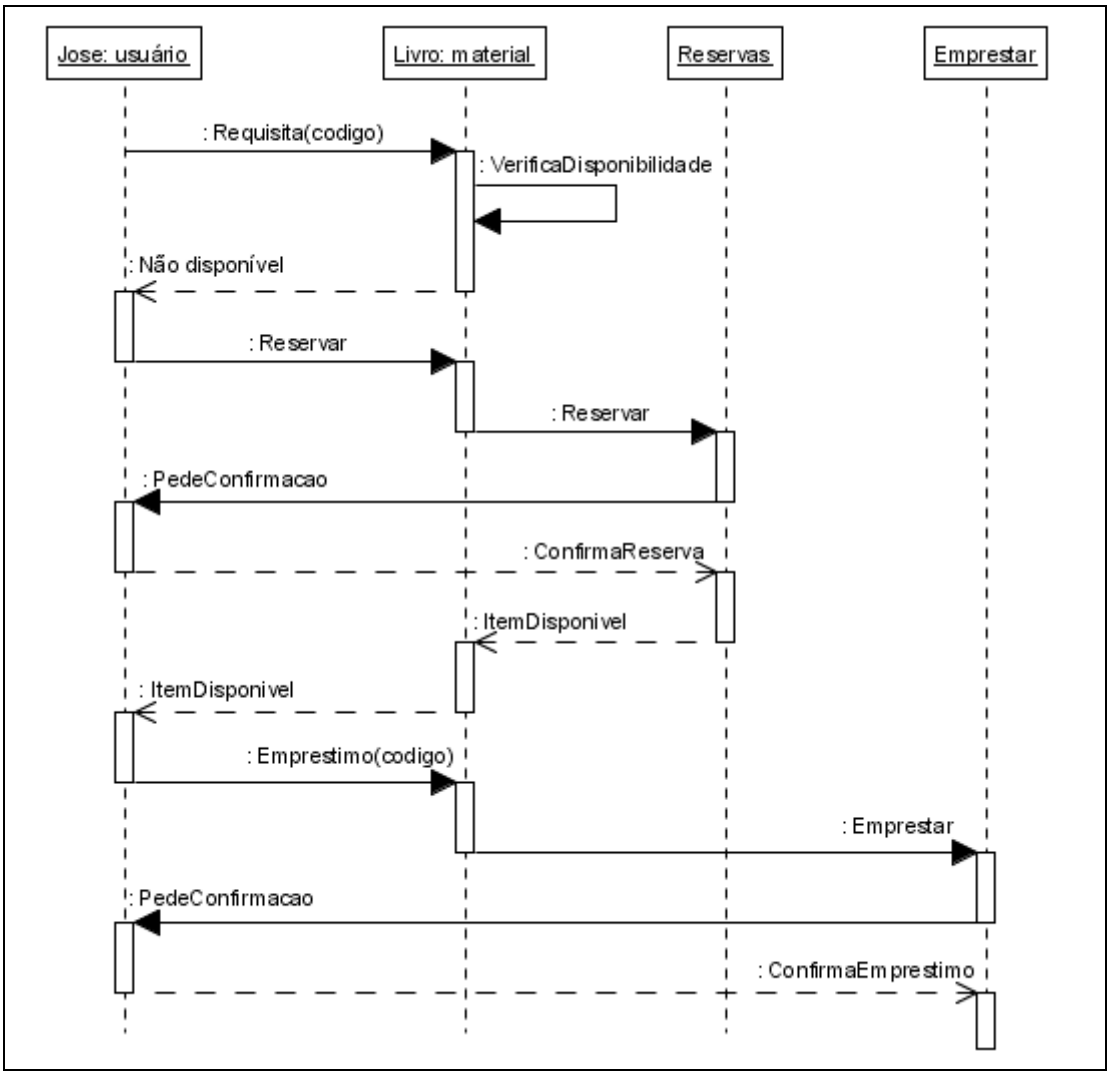


Figura 2.19. Exemplo de diagrama de seqüência de eventos

Cada mensagem no diagrama de seqüência de eventos corresponde a uma operação no diagrama de classes.

II.6.6 – DIAGRAMA DE COLABORAÇÃO

Os diagramas de colaboração possuem, essencialmente, a mesma informação que um diagrama de seqüência de eventos, entretanto, esta é apresentada de uma outra forma, uma vez que, além de mostrar a troca de mensagens entre as classes, também apresenta o relacionamento entre elas, o qual serve de caminho para as mensagens.

A figura 2.20, a seguir, representa um diagrama de colaboração, no qual a numeração foi utilizada para facilitar a leitura e identificação da ordem em que as mensagens são trocadas. A numeração decimal indica a seqüência de uma mensagem e os símbolos -> e <- sua direção. Os diagramas de colaboração mostram o processo que não aparece nos diagramas de classe.

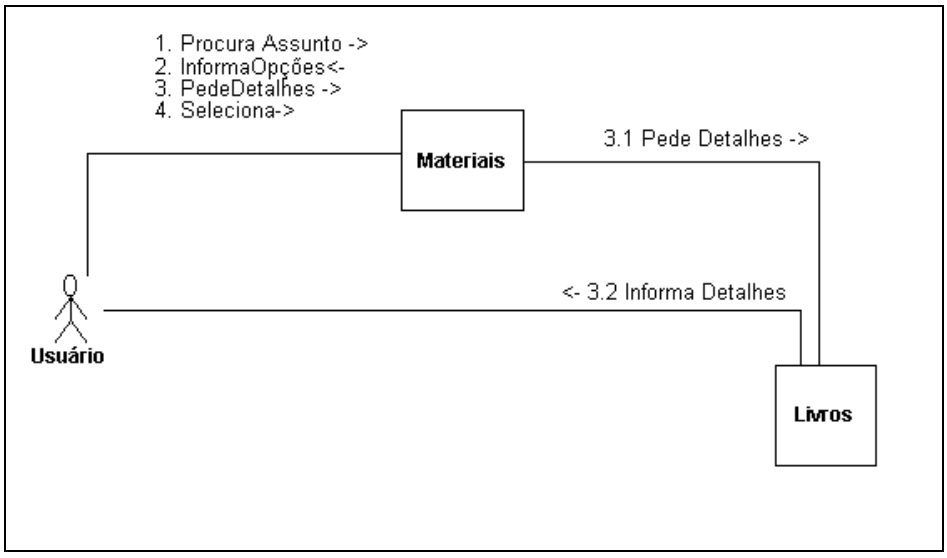


Figura 2.20. Exemplo de diagrama de colaboração

II.6.7 – DIAGRAMA DE ESTADOS

Os diagramas de transição de estados ou diagramas de atividade representam a dinâmica interna de uma classe. Apenas os eventos e estados de uma única classe são apresentados neste diagrama. Entende-se por eventos os fatos que ocorrem em uma classe, provocados por elementos externos (mensagens) ou internos como condições internas da

classe que provocam uma troca de estado. Uma classe pode ter vários estados, caracterizados por situações em que a classe se encontra.

A figura 2.21, a seguir, exemplifica um diagrama de estados.

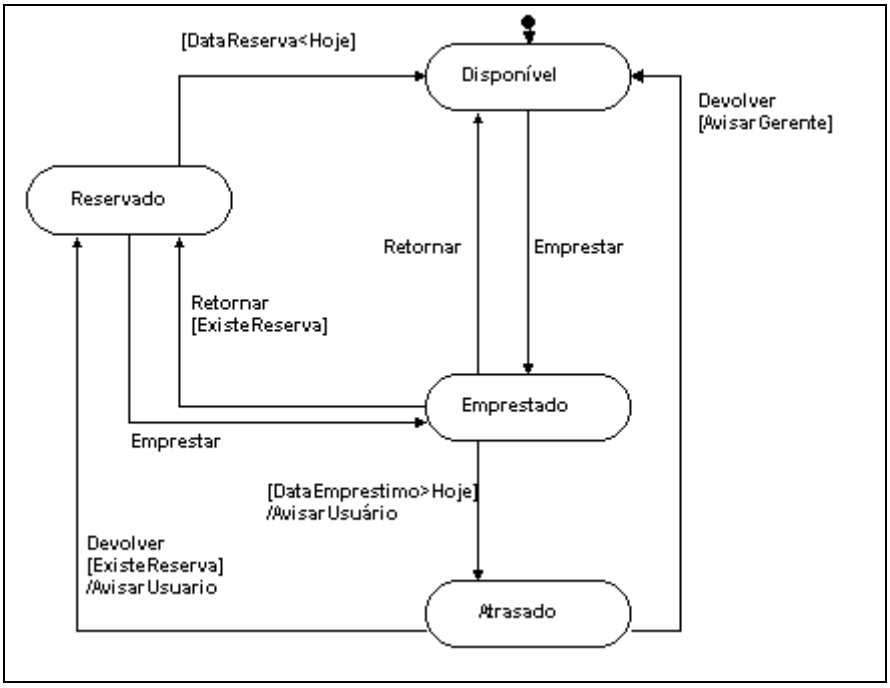


Figura 2.21. Exemplo de diagrama de estados

Analogamente ao diagrama de classes, é possível se extrair um código executável de um diagrama de estados, em uma linguagem de programação. Identificando condições, laços de repetição e chamadas de operações internas ou externas (mensagens) nos estados de espera e nas transições de estado (eventos).

II.6.8 – DIAGRAMA DE ATIVIDADE

Os diagramas de atividade são definidos por (BARROS, 2000) como *"(...) uma maneira alternativa de se mostrar interações, com possibilidade de expressar como as ações são executadas, o que elas fazem (mudanças dos estados dos objetos), quando elas são executadas (seqüência das ações), e onde elas acontecem.*"

A figura 2.22, adiante, exemplifica um diagrama de atividades.

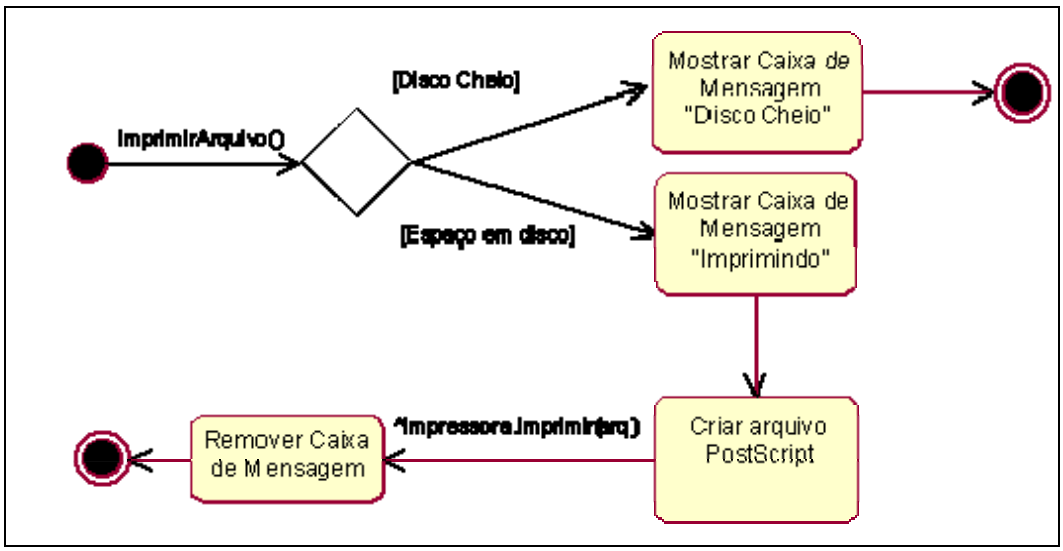


Figura 2.22 – Exemplo de diagrama de atividade

II.6.9 – DIAGRAMA DE COMPONENTES

Os diagramas de componentes representam os elementos reutilizáveis de *software* e sua interdependência. Um componente é formado por um conjunto de classes que se encontram implementadas nele, as quais podem depender das classes de um outro componente. O diagrama de componentes mostra esta dependência. No diagrama de componentes também é possível mostrar a configuração de um sistema de *software,* reproduzindo, graficamente, a dependência entre os diversos arquivos que compõem o sistema, como na figura 2.23, a seguir.

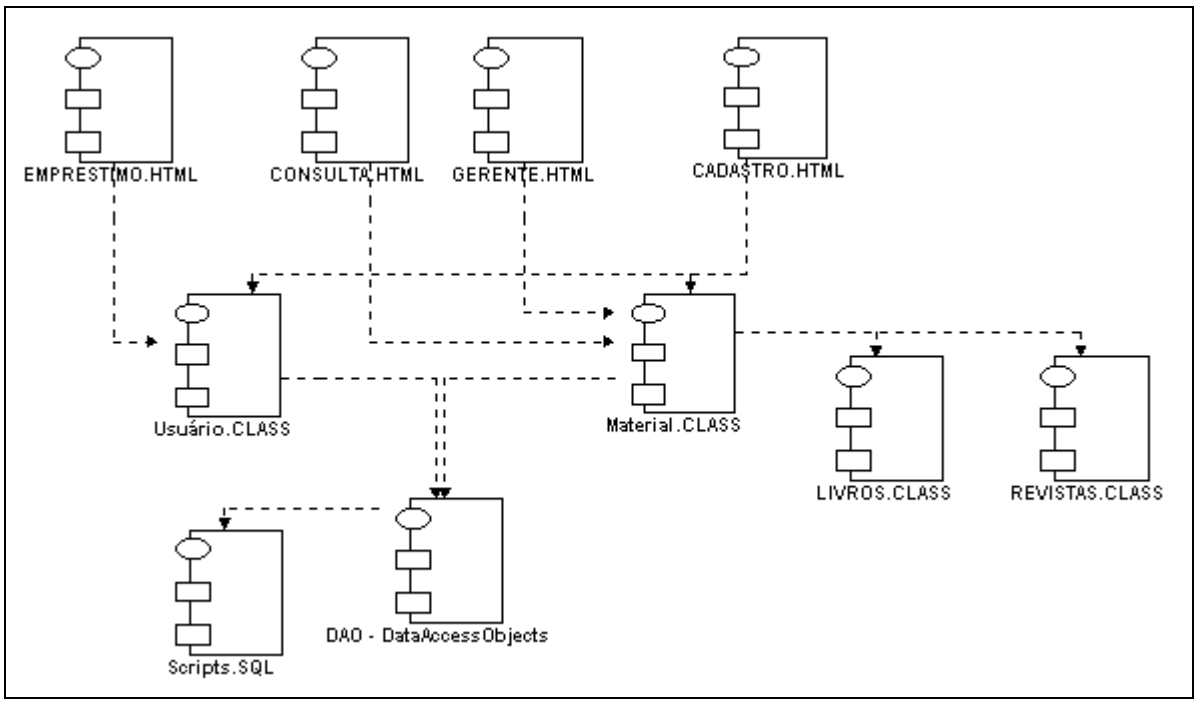


Figura 2.23. Exemplo de diagrama de componentes

II.6.10 – DIAGRAMA DE DISTRIBUIÇÃO

Os diagramas de distribuição mostram a distribuição de *hardware* do sistema, identificando os servidores como nós do diagrama e a rede que relaciona os nós. Os componentes de *software* vão estar mapeados nestes nós. A seguir, na figura 2.24, tem-se um exemplo deste diagrama:

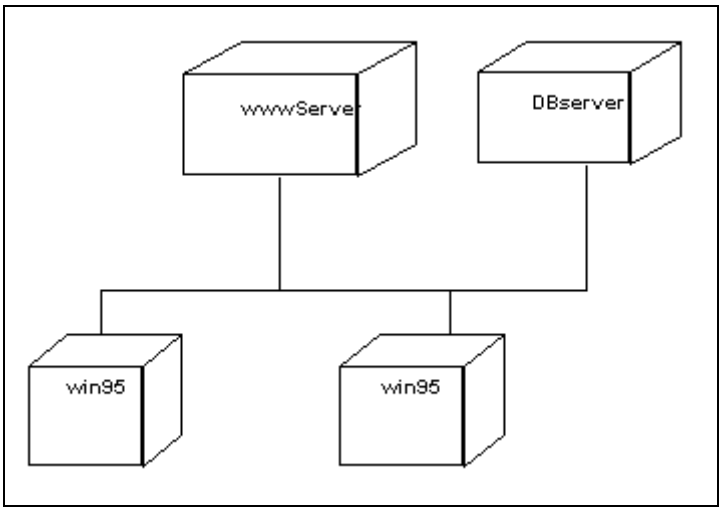


Figura 2.24. Exemplo de diagrama de distribuição

II.7 – JAVA SERVER PAGES (JSP)

Segundo (FIELDS, 2000) a JSP é uma tecnologia baseada em Java que simplifica o processo de desenvolvimento de *sites* dinâmicos. A JSP possibilita aos *designers* e programadores incorporar rapidamente elementos dinâmicos em páginas *web* usando Java embutido e algumas *tags* de marcação simples. Estas *tags* fornecem ao *designer* de HTML[5] uma forma de acessarem dados e lógicas de negócios armazenados em objetos Java, sem a necessidade do domínio das especificidades do desenvolvimento de aplicações.

Por outro lado, *Java Server Pages* são arquivos de texto, normalmente com a extensão .jsp, que substituem as páginas HTML tradicionais. Os arquivos .jsp contém código HTML tradicional junto com a programação, permitindo o acesso a dados do código Java rodando no servidor.

[5] HTML – é a sigla de *Hypertext Markup Language* (linguagem de marcação de hipertexto). Essa linguagem consiste em um conjunto de códigos (chamados de marcas ou *tags*) usados para definir a aparência e funcionalidade das páginas da *Web* vistas pelos navegadores (*browsers*). Ou seja, é com o HTML que se constroem as páginas pelas quais se navega na Internet.

Quando a página é solicitada por um usuário e processada pelo servidor *web* HTTP[6], a parte HTML da página é transmitida. No entanto, as partes de código da página são executadas no momento em que a solicitação é recebida e o conteúdo dinâmico gerado por este código é unido na página antes de seu envio para o usuário, o que propicia uma separação dos aspectos de apresentação HTML da página da lógica de programação contida no código.

A JSP oferece diversos benefícios como um sistema para a geração de conteúdo dinâmico. Primeiramente, por ser uma tecnologia baseada em Java, esta se aproveita de todas as vantagens que a linguagem Java fornece em relação a desenvolvimento e acionamento. Como uma linguagem orientada a objetos com forte digitação, encapsulamento, tratamento de exceções e gerenciamento de memória automático, o uso de Java conduz a uma produtividade aumentada do programador e a um código mais robusto. Pelo fato de haver uma API[7] padrão publicada para JSP, e pelo fato do *bytecode* de Java compilado ser portável através de todas as plataformas que aceitam uma *Java Virtual Machine* (JVM), o uso de JSP não o restringe ao uso de uma plataforma de *hardware*, sistema operacional ou *software* de servidor específico. Se uma mudança em qualquer um destes componentes se tornar necessária, todas as páginas JSP e classes de Java associadas podem ser migradas no estado em que se encontram.

Além disso, a JSP pode se aproveitar de todas as outras APIs de Java padrão, incluindo aquelas para acesso a bancos de dados compatíveis com muitas plataformas, serviços de diretórios, processamento distribuído e criptografia. Esta habilidade em alavancar uma ampla gama de fontes de dados, recursos de sistema e serviços de rede significa que a JSP é uma solução altamente flexível para a criação de aplicações baseadas na *web* e repletas de recursos.

Por oportuno, para o funcionamento da JSP, na Internet, necessita-se que os servidores HTTP possuam um container JSP de terceiros (FIELDS, 2000). O Tomcat, desenvolvido em conjunto pela Sun Microsystems e pela Apache Software Foundation, no projeto Jakarta, é um pacote que serve como a implementação de referência tanto para *servlets* quanto para JSP. Por ser uma ferramenta gratuita, o Tomcat é uma excelente plataforma para fazer experiências com a tecnologia e desenvolver e validar aplicações.

Uma vez instalado e configurado, é relativamente fácil entender o funcionamento de um container JSP. Os arquivos JSP são adicionados à hierarquia de documentos normais do servidor HTTP e são diferenciados de outros documentos da *web* pelas suas extensões .jsp.

---

[6] HTTP – O protocolo HTTP é usado para transferência de documentos no formato HTML e outros encontrados na *Web* - Internet/Intranet e Extranet. O processo de transferência desses documentos ocorre quando o cliente conecta com o servidor HTTP, digitando um endereço de um *site* na Internet no campo endereço do navegador.

[7] API – A *Application Programming Interface* (ou Interface de Programação de Aplicativos) é um conjunto de rotinas e padrões estabelecidos por um *software* para utilização de suas funcionalidades por outros programas aplicativos.

A execução da JSP começa com uma solicitação por uma página JSP, o processamento é feito nas tag(s) JSP presentes na página a fim de gerar conteúdo dinamicamente e o output deste processamento combinado com a HTML estática da página, deve ser retornado para o navegador da *web*. Cada página JSP é compilada em um *servlet* específico de página cujo propósito é gerar o conteúdo dinâmico especificado pelo documento JSP original. Assim, sempre que o servidor HTTP receber uma solicitação correspondente a uma página JSP, aquela solicitação é enviada para o container JSP, o qual invoca o *servlet* compilador de página para tratar da solicitação, compilando o arquivo JSP em um *servlet*, caso necessário, isto é, se este ainda não foi compilado ou sofreu alterações.

II.8 – MYSQL

O MySQL é um sistema gerenciador de banco de dados (SGBD) SQL de código aberto (*open source)*. Foi desenvolvido e distribuído pela MySQL AB, empresa comercial fundada pelos desenvolvedores do MySQL, cuja função precípua é fornecer serviços relacionados ao sistema de gerenciamento de bancos de dados MySQL (MYSQL, 2005).

Com efeito, o MySQL é um SGBD relacional, pois armazena os dados em tabelas separadas em vez de colocá-los em um só local, o que proporciona velocidade e flexibilidade. Utiliza a linguagem de consultas *Structured Query Language* (SQL), padrão de todos os bancos de dados oferecidos no mercado. Além disso, funciona em diversas plataformas como, por exemplo, Windows e Linux, e é *open-source,* permitindo aos usuários usar e modificar seu código fonte.

O Servidor MySQL foi desenvolvido originalmente para lidar com bancos de dados muito grandes de maneira muito mais rápida que as soluções existentes e tem sido usado em ambientes de produção de alta demanda por diversos anos de maneira bem sucedida.

Apesar de estar em constante desenvolvimento, o servidor MySQL oferece hoje um rico e proveitoso conjunto de funções. A conectividade, velocidade, e segurança fazem com que o MySQL seja altamente adaptável para acessar bancos de dados na Internet.

II.9 – COMENTÁRIOS

Neste capítulo abordaram-se os principais conceitos necessários para o entendimento deste trabalho. No próximo capítulo será apresentado um o estudo sobre o projeto de SIGs, com a idealização de um modelo para sua criação utilizando a orientação a aspectos.

CAPÍTULO III – O PROJETO DE UM SIG

III.1 – INTRODUÇÃO

A ausência de um plano de implementação adequado é uma das principais razões do fracasso de um sistema de informação geográfica (KORTE, 1997). Por isso, neste capítulo apresentar-se-á um processo de implementação de SIGs que poderá variar conforme o tamanho da organização, a extensão das aplicações de SIG, o nível de detalhamento e a equipe comprometida com o projeto.

III.2 – O PROCESSO

O processo de implementação de um sistema de informação geográfica apresentado a seguir baseia-se no modelo de prototipagem de desenvolvimento de *software* (PRESSMAN, 2002), no qual o cliente define, inicialmente, um conjunto de objetivos gerais para o sistema. Um projeto piloto é então realizado, concentrando-se na representação daqueles aspectos do *software* que ficarão visíveis ao cliente/usuário.

Revela-se imprescindível a presença de um gerente de projetos, preferencialmente especializado em SIGs, para gerenciar e acompanhar o desenvolvimento do SIG.

Além disso, o processo deve ser revisado por um comitê, que poderá ser dirigido pelo próprio gerente de projetos, consistindo em representantes de diversos departamentos chaves da empresa cliente que utilizarão o sistema. O gerente de projetos deverá prover uma direção global para o processo e promover a aprovação do comitê das etapas do projeto, além de servir também como um coordenador para a distribuição dos relatórios e para a reunião de comentários de revisão.

Agregue-se que o processo de implementação desenvolvido neste trabalho também deriva do proposto por (KORTE, 1997). Contudo, distingue-se pelo fato de não se basear em aplicações de SIG já existentes como, por exemplo, as fornecidas pela ESRI.

Na verdade, o modelo criado neste trabalho representa um processo para a implementação de um sistema de informação geográfica desenvolvido especificamente para o cliente, sem a utilização de aplicações de SIG comercializadas no mercado.

A seguir visualiza-se o processo de implementação de SIGs:

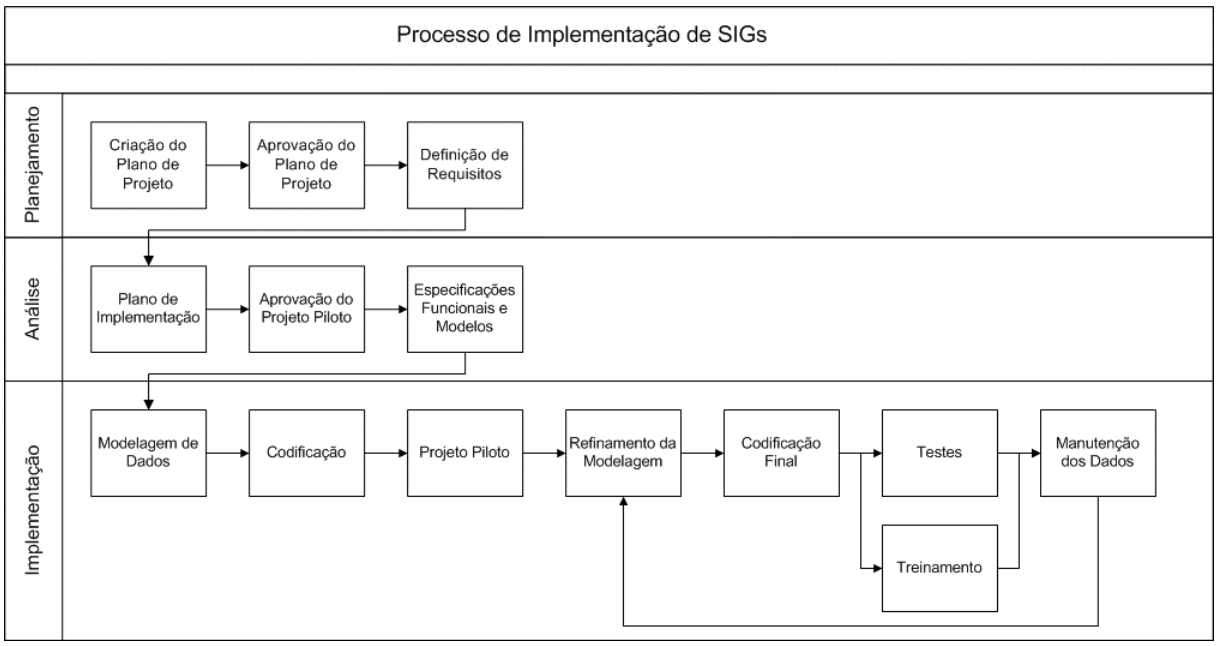


Figura 3.1. Processo de Implementação de SIGs

Cabe ressaltar que neste processo existem inúmeros pontos-chave de decisão, que permitem ao gerente do projeto adiar ou cancelar o projeto em diversos estágios.

III.2.1 – PLANEJAMENTO

III.2.1.1 – CRIAÇÃO DO PLANO DE PROJETO

O primeiro passo para a criação de um SIG é o planejamento de seu projeto, etapa que inclui a justificativa, os objetivos e seus benefícios, o cronograma e o orçamento do projeto.

III.2.1.2 – APROVAÇÃO DO PLANO DE PROJETO

Após a conclusão do plano de projeto, é imprescindível que este seja aprovado pelo cliente. Este é o momento de envolver os gerentes-de-topo da empresa-cliente, caso ainda não tenham sido envolvidos. O suporte dos mencionados gerentes é crítico para o sucesso de um SIG.

Cabe ressaltar, ainda, a indispensabilidade de que o responsável pela aprovação do plano de projeto do SIG entenda que a simples aprovação do aludido plano não autoriza a completa implementação do sistema, mas que o processo contém periódicas oportunidades de aprovação e desaprovação do conceito enquanto a avaliação prossegue.

III.2.1.3 – DEFINIÇÃO DE REQUISITOS

Uma vez aprovado o plano de projeto, inicia-se a definição de requisitos do SIG, por meio da qual será verificado como a empresa-cliente utiliza dados geográficos, incluindo sua análise, armazenamento, apresentação e distribuição. Esta análise deverá examinar as operações dos possíveis usuários do sistema, para enumerar todas as funcionalidades necessárias ao SIG.

Além disso, a análise de requisitos pode ser desenvolvida por meio de entrevistas com os pontos-focais de cada um dos departamentos da empresa-cliente que utilizará o *software*.

Portanto, esta fase do processo deve definir a missão, operação e organização de cada departamento da empresa-cliente, isto é, como a informação é coletada, usada, analisada e distribuída, bem assim os problemas da utilização destes dados e o custo das operações existentes, visando demonstrar o benefício da utilização do futuro SIG. Este dado do custo será utilizado para a justificação financeira do investimento. Os resultados desta análise devem ser registrados por escrito.

III.2.2 – ANÁLISE

III.2.2.1 – PLANO DE IMPLEMENTAÇÃO

Logo que terminada a fase da definição de requisitos, o próximo passo é analisar os dados coletados e desenvolver um plano de implementação, em conjunto com os pontos-focais dos diversos departamentos da empresa-cliente que usarão o sistema de informação geográfica.

Uma forma prática e eficaz de preparar um plano de implementação é por meio de um *workshop*, no qual os participantes, no mínimo todos os entrevistados quando da realização da definição de requisitos, validarão cada um dos itens da implementação, dentre os quais se encontram:

a) equipamentos existentes, *software,* dados e operações;
b) requisitos de dados do SIG;
c) requisitos de hardware do SIG;
d) comunicação da rede de dados;
e) manutenção dos dados do SIG;
f) equipe e organização;
g) treinamento;
h) custos e benefícios;

i) fases de implementação e responsabilidades;

Todas as decisões tomadas no *workshop* deverão ser documentadas em um relatório.

III.2.2.2 – APROVAÇÃO DO PROJETO PILOTO

Este é o primeiro ponto de decisão, quando o relatório da análise de requisitos e o plano de implementação são apresentados para a gerência de topo, para revisão e aprovação. Se a gerência aprovar o projeto piloto, estará se comprometendo com uma despesa significativa. Entretanto, a gerência deve ter ciência de que haverá outra oportunidade para decidir sobre a integral implementação do sistema, após o término do projeto piloto.

Quando completo, o projeto final do SIG é distribuído para a gerência de topo. Assim, o gerente do projeto, juntamente com o comitê (pontos focais dos diversos departamentos da empresa), realizam reuniões com a gerência para explicar o programa. Uma apresentação formal pode ser uma boa opção, priorizando a demonstração dos benefícios, custos e impacto organizacional do projeto. Exemplos de outras aplicações SIG podem ser utilizados para facilitar o entendimento de alguns conceitos do sistema.

Com a aprovação da mencionada gerência, o processo passa para a próxima etapa.

III.2.2.3 – ESPECIFICAÇÕES FUNCIONAIS E MODELOS

Nesta etapa os resultados das análises anteriores devem ser revisados, criando-se duas especificações distintas: uma descreverá os requisitos de *hardware* e de *software*, e outra, os requisitos para a conversão dos dados existentes na empresa para o formato da base de dados do SIG.

III.2.3 – IMPLEMENTAÇÃO

III.2.3.1 – MODELAGEM DE DADOS

Uma vez completas as fases de planejamento e análise do SIG, inicia-se a sua implementação.

Na modelagem de dados deve-se estruturar o banco de dados do sistema, criando suas tabelas, esquemas, campos, relacionamentos e restrições, isto é, definindo a forma como os gráficos serão simbolizados (cores, tamanhos, símbolos e outros); como os arquivos gráficos e dados não-gráficos serão estruturados; como os diretórios de arquivos serão organizados; a forma como os arquivos serão nomeados; e, a aplicação de restrições de segurança no acesso.

III.2.3.2 – CODIFICAÇÃO

A primeira fase da codificação consiste no desenvolvimento dos componentes (módulos) indispensáveis para o funcionamento básico do SIG, visando possibilitar, a execução do plano piloto.

III.2.3.3 – PROJETO PILOTO

Após a codificação inicial, inserem-se na base de dados informações de uma pequena e representativa parte da área do projeto, visando apresentar e testar o sistema de informação geográfica. Dessa forma, pode-se estimar os custos da conversão dos dados, bem assim apresentar alguns dos benefícios do SIG para a gerência de topo.

Por sua vez, a gerência deve revisar os resultados do plano piloto e atualizar a análise de custo e benefício para decidir sobre a implementação integral do sistema.

Cabe ressaltar que esta é a última oportunidade de aprovar, cancelar, ou atrasar o projeto do SIG antes das despesas principais serem feitas.

III.2.3.4 – REFINAMENTO DA MODELAGEM

A criação da base de dados digital normalmente representa a maior porção do investimento do SIG e, por isso, é importante aproveitar todas as oportunidades para ajustar a estrutura da base de dados antes da conversão dos dados de toda a área do projeto, uma vez que mudanças no banco, embora possam ser realizadas após a entrada dos dados, são muito mais caras do que quando executadas no início do projeto.

III.2.3.5 – CODIFICAÇÃO FINAL

Nesta fase serão implementados todos os demais componentes necessários para o completo funcionamento do sistema, levando-se em consideração as alterações porventura realizadas na etapa anterior.

III.2.3.6 – TESTES

Após a total codificação, conversão e inserção das informações na base de dados, inicia-se a fase de testes para verificar a exatidão dos dados convertidos, uma vez que o

processo de conversão geralmente é complicado. Além disso, verifica-se também o correto funcionamento do sistema, executando-se cada uma de suas funcionalidades.

III.2.3.7 – TREINAMENTO

Sistemas de informação geográfica são uma nova tecnologia e representam uma radical e diferente forma de conduzir o trabalho diário das empresas e funcionários que irão utilizá-lo. Assim, torna-se imprescindível seu treinamento para que saibam manusear o sistema, bem assim todas as suas funcionalidades. Além disso, o treinamento permitirá aos usuários manter os dados do SIG sempre atualizados.

III.2.3.8 – MANUTENÇÃO DE DADOS

A credibilidade de uma base de dados de um SIG é proporcional ao quanto a informação ali armazenada reflete a realidade do espaço representado.

Assim, como o mundo encontra-se em constante mudança, é necessário que a base de dados do SIG também seja atualizada, para refletir estas mudanças.

Por sua vez, é comum no desenvolvimento de SIGs o constante refinamento da modelagem do sistema, incluindo novas informações, funções ou corrigindo falhas encontradas durante o seu uso.

III.3 – COMENTÁRIOS

As quatorze fases descritas neste capítulo representam um processo de implementação de SIGs para a execução de projetos bem sucedidos.

Com efeito, muitos projetos de SIGs fracassam devido à ausência de um processo esquemático de planejamento, análise e implementação, como o descrito neste capítulo. Ao invés disso, estes projetos invariavelmente deixaram de executar uma dessas fases.

No próximo capítulo será apresentada a AspectUML, extensão da UML criada para a representação dos aspectos, bem assim os benefícios que a utilização da programação orientada a aspectos pode proporcionar no desenvolvimento de SIGs.

CAPÍTULO IV – PROGRAMAÇÃO ORIENTADA A ASPECTOS E ASPECTUML

IV.1 – INTRODUÇÃO

No capítulo anterior apresentou-se um processo de implementação de SIGs para auxiliar o desenvolvimento de projetos de sistemas de informação geográfica. Por sua vez, no mencionado processo constavam fases referentes à modelagem e à implementação, que abrangem a utilização da programação orientada a aspectos.

Este capítulo abordará a identificação de requisitos transversais nos SIGs, bem assim os benefícios da adoção do paradigma da programação orientada a aspectos (POA) no desenvolvimento de SIGs, além da AspectUML, a extensão da UML criada para a representação dos aspectos, que auxiliará na modelagem e no desenvolvimento do sistema.

IV.2 – REQUISITOS TRANSVERSAIS NOS SISTEMAS DE INFORMAÇÕES GEOGRÁFICAS

O paradigma da orientação a objetos tem sido amplamente empregado no desenvolvimento de *softwares*, incluindo a maioria das ferramentas disponíveis para o desenvolvimento de Sistemas de Informações Geográficas (SIGs).

Como visto, este paradigma é baseado no princípio do agrupamento de elementos similares do mundo em classes e por tornar explícitos comportamentos e propriedades comuns por meio do mecanismo de herança. Utilizando-se esse mecanismo, classes derivadas compartilham propriedades comuns de uma classe base, enquanto mantêm seus comportamentos específicos.

Com efeito, no desenvolvimento de SIGs, esse paradigma leva freqüentemente ao estabelecimento de classes diretamente relacionadas aos diferentes tipos de representações geométricas. Por exemplo, um típico SIG orientado a objetos pode ter classes básicas tais como imagem, polígono, grade regular, mapa de polígonos, TIN (grade irregular triangular), conjunto de pontos e ponto. As classes contêm tanto a estrutura de dados subjacente quanto o correspondente conjunto de algoritmos.

Por outro lado, requisitos do sistema como a autorização, *logging*, persistência de dados e gerenciamento de armazenamento estão distribuídos em vários módulos do sistema.

Entretanto, existem desvantagens na aplicação direta da orientação a objetos no

desenvolvimento de SIGs, isto é, o acoplamento dos algoritmos às estruturas de dados e a dispersão do código de um dado requisito em diversos módulos do sistema.

Quanto à primeira desvantagem, um grande número de algoritmos não depende da implementação particular de uma estrutura de dados, mas apenas de algumas propriedades semânticas fundamentais destas, podendo ser abstraídos e descritos apenas em termos de suas propriedades. Apesar disso, em muitas bibliotecas de SIGs, o uso dos princípios de orientação a objeto tem resultado em classes que contêm tanto a estrutura de dados subjacente quanto o conjunto de algoritmos correspondente.

Logo, os algoritmos estão desnecessariamente acoplados às estruturas de dados, isto é, foram implementados várias vezes, em cada uma delas.

Como visto, os requisitos de autorização, *logging*, persistência de dados e gerenciamento de armazenamento estão distribuídos em vários módulos do sistema. Por exemplo, sempre que um módulo necessitar gravar alguma informação referente ao *log* será afetado por esse requisito. Portanto, os códigos de *logging* e de persistência estão dispersos por inúmeros módulos do sistema, o que caracteriza a dispersão de código ou, no inglês, *scattering code*.

Além disso, cabe ressaltar que os requisitos de *logging*, persistência e da própria lógica do sistema encontram-se entrelaçados no mesmo componente, o qual, a princípio, deveria cuidar apenas da lógica do sistema. Este entrelaçamento gera, portanto, um acoplamento indesejado, também conhecido como entrelaçamento de código ou, no inglês, *tangled code*.

IV.3 – PROGRAMAÇÃO ORIENTADA A ASPECTOS E SEUS BENEFÍCIOS

A programação orientada a aspectos (POA) é um novo paradigma de programação que objetiva separar totalmente os requisitos transversais dos funcionais no sistema, propondo que aqueles sejam implementados em uma unidade de programação nova, chamada aspecto. Esta nova unidade seria incorporada, quando necessária, aos objetos do sistema que implementam os requisitos funcionais.

A implementação dos aspectos separada dos objetos possibilita maior reuso do código, facilita a manutenção e provê melhor performance, evitando os problemas na programação do *software* acima mencionados, isto é, o entrelaçamento (*tangled code*) e a dispersão (*code scattering*) do código.

No entanto, para facilitar o desenvolvimento dos SIGs orientados a aspectos e auxiliar a sua documentação, é indispensável a utilização de uma extensão da UML para a

representação dos requisitos transversais (aspectos), ainda inexistente. Assim, vislumbrou-se a necessidade da criação da AspectUML, além de um sistema *web* para acompanhar e documentar o desenvolvimento dos SIGs orientados a aspectos.

IV.4 – A ASPECTUML

A representação dos aspectos nos diagramas da UML revela-se fundamental para possibilitar aos programadores, analistas e engenheiros de sistemas, a modelagem dos *softwares* com a utilização desse novo paradigma de programação.

Com efeito, (BEIER, 2002) e (GROHER, 2003) sugerem a utilização de pacotes para a representação dos aspectos na UML, estabelecendo que o *software* a ser modelado deva ter apenas 3 (três) pacotes, quais sejam: a) o pacote base (que contém a lógica do negócio); b) o de aspectos; e, c) o do conector (responsável pela composição dos requisitos funcionais com os aspectos).

Contudo, suas abordagens não especificam, detalhadamente, as estruturas do paradigma da programação orientada a aspectos, utilizando estereótipos, valores com *tag* e restrições, impossibilitando a especificação e modelagem de suas interações, além da documentação minuciosa dos aspectos do sistema com a UML.

Dessa forma, revela-se imprescindível a criação de uma extensão para a UML que possibilite a representação dos aspectos e a completa documentação de suas interações com as classes do núcleo do sistema, a AspectUML.

A UML provê mecanismos que propiciam sua adaptação para atender os requisitos de um determinado domínio específico, ou seja, possibilita a extensão de seu meta-modelo padrão, permitindo a customização e a extensão de seus elementos como, por exemplo, do elemento classe base (*base class*) <<classe>>.

Os mecanismos de extensão da UML são os estereótipos (*stereotypes*), os valores com *tag* (*tagged values*), e as restrições (*constraints*).

Os estereótipos são utilizados para classificar novos elementos derivados dos já existentes no modelo UML, que possuem suas próprias propriedades, mas específicos para o novo domínio. Geralmente são representados como uma seqüência de caracteres entre os símbolos << >>.

Os valores com *tag* permitem a extensão das propriedades de um estereótipo, isto é, a criação de novas informações associadas ao novo elemento da UML. Um valor com *tag* é representado em um diagrama como uma seqüência de caracteres delimitada por colchetes [ ].

As restrições estendem a semântica de um elemento da UML, ou seja, são restrições colocadas nos estereótipos, permitindo a adição ou modificação de regras.

Como visto, os aspectos são interesses gerais do sistema que normalmente estão interligados em todo o modelo; pequenas partes dos aspectos podem ser encontradas em diferentes partes do *software*.

Utilizando o paradigma da orientação a aspectos no desenvolvimento de um sistema, os aspectos são analisados e desenvolvidos separadamente das outras funcionalidades principais. Logo, durante a fase de implementação, os componentes funcionais e os aspectos precisam ser integrados ou, no inglês, *weaved*, para o perfeito funcionamento do sistema.

Com efeito, nota-se que os principais elementos da linguagem orientada a aspectos são os aspectos (*aspects*), os pontos de junção (*joinpoints*), os pontos de corte (*pointcuts*) e os comportamentos (*advices*).

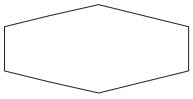
Portanto, estes são os principais elementos a serem representados pela AspectUML, incluindo-se, ainda, os relacionamentos, pois, por exemplo, para se criar um aspecto de sincronização é imprescindível um estereótipo de controle, desde que o aludido aspecto controle o comportamento de componentes funcionais do sistema, como se verá adiante.

IV.4.1 – ESTEREÓTIPO ASPECTO (*ASPECT*)

Um aspecto é um estereótipo do elemento <<classe>> da UML.

Na tabela a seguir visualizam-se as propriedades do estereótipo <<aspecto>>.

Tabela 4.1 – Estereótipo aspecto

Elemento da UML	Estereótipo	Diagrama	Descrição
Classe	<<aspecto>>		O estereótipo <<aspecto>> é utilizado para a representação de um aspecto, ao invés de uma classe do núcleo do sistema.

Assim como as classes, um aspecto pode ser abstrato. Um aspecto abstrato é aquele que tem um ou mais pontos de corte abstratos. Por sua vez, um ponto de corte abstrato é aquele que tem somente uma assinatura (*signature*), cuja definição encontra-se na classe que o implementa.

A *tag* booleana {abstrato} é usada para indicar um aspecto abstrato.

A relação aspecto-classe varia de acordo com o sistema modelado. Alguns aspectos controlam o comportamento de classes do núcleo do sistema, outros somente realizam

funções de *trace* e *log*. A *tag* booleana {ativo} é usada para representar um aspecto que afete o comportamento de classes do núcleo do sistema. Por outro lado, a *tag* booleana {passivo} indica aspectos que não afetam as funcionalidades de classes do núcleo do sistema.

A tabela 4.2 adiante resume os valores com *tag* booleana propostos para os aspectos:

Tabela 4.2 – Valores com *tag* booleana dos aspectos

Elemento da UML	Valor com tag	Descrição
Classe	{abstrato}	Usado para aspectos que tem pelo menos um ponto de corte abstrato
Classe	{aspecto ativo}	Usado para aspectos que afetam a funcionalidade de uma classe do núcleo do sistema, ou alteram as condições iniciais de funções. Exemplos são aspectos de sincronização e de distribuição.
Classe	{aspecto passivo}	Usado para aspectos que não afetam o comportamento de classes do núcleo do sistema. Exemplos são aspectos de *tracing, logging* e *reporting*.

IV.4.2 – RELACIONAMENTO ASPECTO-CLASSE

O relacionamento entre aspectos e classes é representado por meio da relação associação da UML. Para afetar o sistema, cada aspecto precisa ter ao menos uma associação com uma classe do núcleo do *software*.

Por oportuno, os aspectos podem ser classificados de acordo com a forma como afetam as classes com que se relacionam. Assim, apresenta-se a seguir algumas classificações para os relacionamentos aspecto-classe.

Tabela 4.3 – Valores com tag da relação aspecto-classe

Elemento da UML	Valor com tag	Descrição
Associação	{Ajuste}	A relação "Ajuste" pode ser utilizada quando os aspectos são responsáveis por ajustar características da classe.
Associação	{Controle}	Esta relação é utilizada quando os aspectos controlam o comportamento de classes do núcleo do sistema.
Associação	{Tratamento de Erros}	Esta relação pode ser usada quando o aspecto é responsável pelo tratamento de erros que eventualmente podem ocorrer no sistema.
Associação	{Tratamento de Exceções}	Esta relação pode ser usada quando o aspecto for o responsável pelo tratamento de exceções que ocorrerem no sistema.
Associação	{Validação}	A relação "Validação" pode ser utilizada por um aspecto responsável por validar as condições de diferentes funcionalidades para execução de uma determinada funcionalidade.

Associação	{Tracking}	Esta relação é utilizada quando os aspectos não afetam as funcionalidades do sistema, mas são responsáveis pela simples impressão de mensagens de informações do sistema.
Associação	{Relatório}	As relações "Relatório" são próximas das relações "Tracking". Podem ser utilizadas quando um aspecto guarda informações em um arquivo de *log*, ou quando este é responsável por realizar alguma estatística.
Associação	{Salvar}	Esta relação pode ser utilizada quando um aspecto é persistente, responsável por salvar classes.

A classificação anterior não representa exaustivamente todos os possíveis tipos de relacionamentos aspecto-classe, pois os aspectos variam de acordo com o sistema. Com efeito, mais *tags* podem ser definidas estendendo a AspectUML com novos valores.

Além disso, ressalte-se que uma associação aspecto-classe pode ter mais de uma *tag*, dependendo da semântica do sistema modelado.

IV.4.3 – ESTEREÓTIPO PONTO DE CORTE (*POINTCUT*)

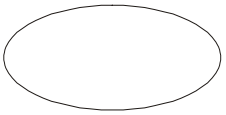
Pontos de junção (*join points*) são pontos bem definidos na execução do programa onde um requisito vai atravessar (*crosscut*) a aplicação (GRADECKI, 2003). É o ponto no qual um objeto recebe uma chamada a um método e pontos nos quais um campo de um objeto é referenciado. Os pontos de junção podem ser chamadas aos métodos, ao construtor, tratamento de exceções, ou outros pontos na execução de um programa. Um ponto de corte (*pointcut*) combina pontos de junção por meio dos operadores "&&", "||" e "!". Um ponto de corte estabelece onde o código de comportamento (*advice*) do aspecto começa a ser executado.

A AspectUML não diferencia pontos de junção de pontos de corte, pois, por definição, um ponto de junção é um ponto de corte cuja expressão é um nome de função ou um atributo, modelando um ponto de corte como um estereótipo do elemento da UML "classe".

Além disso, o estereótipo ponto de corte se relaciona com o aspecto por meio da associação <<tem ponto de corte>>. O ponto de corte não tem funções, contudo possui a definição de onde o aspecto se relaciona com a classe do núcleo do sistema, o que é a definição do ponto de junção.

Na tabela a seguir visualizam-se as propriedades do estereótipo <<ponto de corte>>:

Tabela 4.4 – Estereótipo ponto de corte

Elemento da UML	Valor com tag	Diagrama	Descrição
Classe	<<ponto de corte>>		O estereótipo <<ponto de corte>> indica pontos na classe nos quais um código de comportamento (*advice*) do aspecto começa a ser executado.

Um aspecto pode ter nenhum ou mais pontos de corte. Quando não possui nenhum ponto de corte significa que o aspecto não afeta diretamente uma classe do núcleo do sistema.

A figura adiante reproduz um aspecto de sincronização que controla a classe "Ponto" por meio do ponto de corte (*pointcut*) <<Movimento>>.

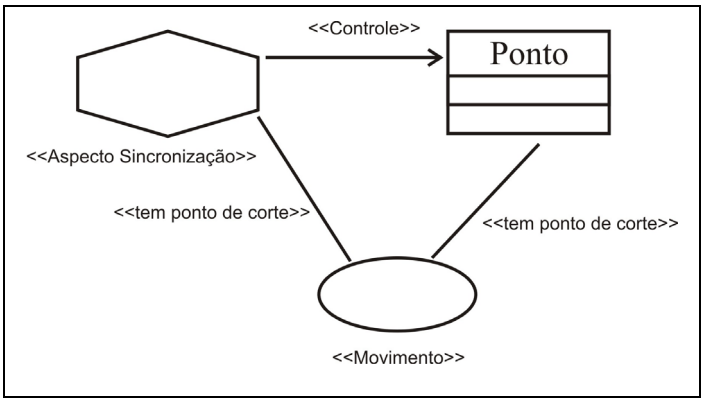


Figura 4.1 – Relacionamento Aspecto-Classe

Por sua vez, o ponto de corte é um elemento muito importante para os aspectos, pois especifica como o aspecto se relaciona com uma classe do núcleo do sistema, funcionando como uma interface exposta aos programadores para determinar os pontos da execução nos quais o código de comportamento do aspecto (*advice*) será ativado.

Um ponto de corte pode ser abstrato. Se um aspecto tem pelo menos um ponto de corte abstrato, então este aspecto é abstrato. Os aspectos que estendem os aspectos abstratos devem possuir a definição de um ponto de corte abstrato. A *tag* booleana {abstrato} é usada para identificar um ponto de corte abstrato. A tabela a seguir apresenta a *tag* booleana {abstrato}:

Tabela 4.5 – Tag booleana {abstrato}

Elemento da UML	Valor com tag	Descrição
Classe	{abstrato}	Usado para pontos de corte que são registrados em um aspecto, mas não são especificados. Os pontos de corte abstratos são definidos em outros aspectos, isto é, nos aspectos que estendem os aspectos abstratos.

IV.4.4 – RELACIONAMENTO ASPECTO-ASPECTO

Durante o estudo do relacionamento aspecto-aspecto, vislumbrou-se dois tipos principais de relacionamentos. O primeiro relativo aos aspectos que estendem outros aspectos, representado pela relação padrão de "generalização" [8] da UML, que se visualiza na figura 4.2.

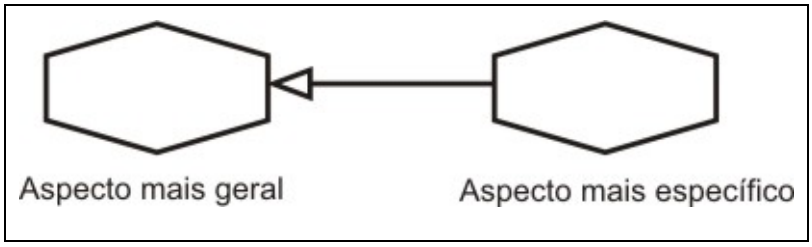


Figura 4.2 – Relacionamento de Generalização

O segundo referente à precedência de um aspecto sobre outro. Neste caso, dependendo do sistema, se não houver a especificação de precedência dos aspectos, a ordem deles não é importante.

Para especificar quando um aspecto tem prioridade maior que a de outro, criou-se um estereótipo (<<precede>>), baseado na "associação" do modelo UML.

Na tabela a seguir visualizam-se as propriedades do estereótipo <<precede>>:

Tabela 4.6 – Estereótipo de precedência

Elemento da UML	Valor com tag	Descrição
Associação	<<precede>>	Relação entre dois aspectos que especifica qual deles tem maior prioridade que o outro. A seta da associação aponta do com a maior prioridade para o de menor prioridade.

IV.4.5 – ESTEREÓTIPO COMPORTAMENTO (*ADVICE*)

Como visto, na maioria das especificações da programação orientada a aspectos, o código de comportamento (*advice)* é executado em diferentes momentos quando um *join point* é acionado.

A AspectJ prevê um conjunto de alternativas para o momento da execução do *advice*, são eles: *before, after, around, after returning* e *after throwing.*

Assim, é patente que a AspectUML deveria prevê-los também, como elementos do tipo "operação" do modelo UML.

[8] Generalização é um relacionamento de um elemento mais geral com outro mais específico. O elemento mais específico pode conter apenas informações adicionais. Uma instância (um objeto é uma instância de uma classe) do elemento mais específico pode ser usada onde o elemento mais geral seja permitido.

Adiante visualiza-se um resumo dos estereótipos dos comportamentos (*advice*):

Tabela 4.7 – Estereótipo comportamento

Elemento da UML	Estereótipo	Descrição
Operação	<<antes>>	O comportamento (*advice*) é executado após o ponto de junção ser alcançado, mas imediatamente antes de seu processamento.
Operação	<<depois>>	O comportamento (*advice*) é executado após o ponto de junção ser alcançado, tendo ocorrido exceção ou não.
Operação	<<durante>>	O comportamento (*advice*) é executado durante a execução de um ponto de junção (*join point*), controlando quando e se o programa funcionou.
Operação	<<após o retorno>>	O comportamento (*advice*) é executado após o método interceptado rodar com sucesso, ou seja, sem a ocorrência de uma exceção.
Operação	<<após exceção>>	O comportamento (*advice*) é executado após o método interceptado rodar sem sucesso, ou seja, após a ocorrência de uma exceção.

IV.4.6 – OUTROS DIAGRAMAS COM EXEMPLO

Para modelar um sistema usando a UML com a extensão AspectUML concluiu-se que a modelagem do *software* deve ser dividida em três principais partes, quais sejam: a identificação e especificação dos requisitos funcionais[9], não-funcionais[10], e a composição dos requisitos transversais.

A identificação e especificação dos requisitos funcionais representam a modelagem tradicional, utilizando a UML, na qual o modelo dos casos de uso é a principal técnica de especificação.

Por outro lado, com relação aos requisitos não-funcionais, deve-se identificar quais destes são transversais, isto é, aqueles que estão dispersos no código da aplicação. Os requisitos transversais são os candidatos a serem aspectos e devem ser especificados da seguinte forma:

Tabela 4.8 – Especificação de um requisito transversal

Requisito Transversal	Segurança
Descrição	Segurança no acesso do sistema, verificada durante o seu funcionamento.
Prioridade	Alta
Requisitos	R1, R2, (...)

[9] Requisitos funcionais são as funcionalidades que o sistema deve oferecer ao usuário.
[10] Requisitos não-funcionais são propriedades que expressam condições de comportamento e restrições acerca dessas funcionalidades.

Após identificados os requisitos funcionais, não-funcionais, e os aspectos, incluídos neste último grupo, inicia-se a composição dos requisitos funcionais com os aspectos, identificando e solucionando os conflitos eventualmente criados.

A seguir visualiza-se o processo de modelagem mencionado.

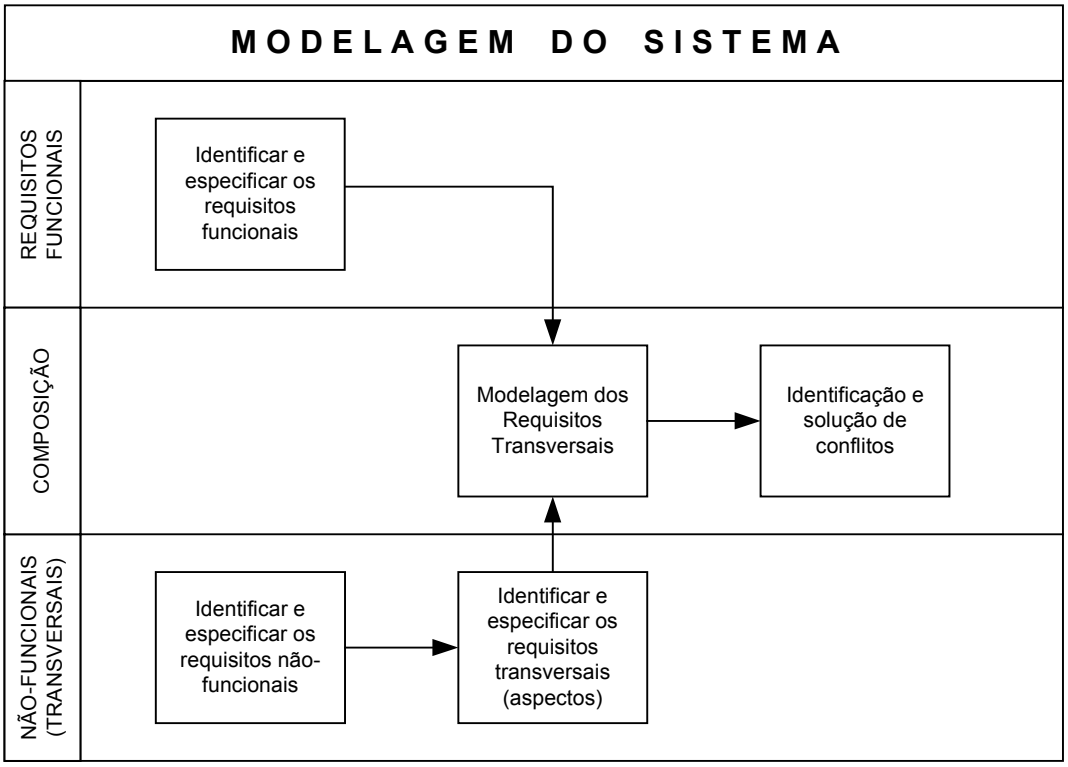


Figura 4.3 – Processo de Modelagem do Sistema

O exemplo escolhido é um projeto hipotético de um sistema de informações geográficas, de uma empresa prestadora de serviços essenciais à comunidade.

A CEDAE – Companhia de Águas e Esgotos do Rio de Janeiro vem digitalizando toda sua base cartográfica disponível, referente às tubulações de água e esgoto da cidade do Rio de Janeiro, para inserir esses dados em um sistema de informações geográficas, a ser implementado, objetivando facilitar o gerenciamento e a tomada de decisões na mencionada companhia.

Para este sistema, foram estabelecidos alguns requisitos, a saber:

a) permitir a visualização de áreas da cidade do Rio de Janeiro, bem assim das informações associadas a cada um dos objetos constantes nos mapas;

b) a inserção de informações no banco de dados deve permitir associá-la a sua posição exata, isto é, georreferenciada;

c) permitir a edição dos mapas, além de possuir a funcionalidade de zoom e visualização por camadas/temas;

d) ter diferentes níveis de acesso, de acordo com os cargos da companhia, através de contas de usuário distintas;

e) documentar, em seu *log*, todas as ações executadas pelos usuários.

IV.4.6.1 – IDENTIFICAÇÃO E ESPECIFICAÇÃO DOS REQUISITOS NÃO-FUNCIONAIS

O sistema descrito na seção anterior contém requisitos funcionais e não-funcionais. Com efeito, para se obter uma descrição completa de cada um dos requisitos não-funcionais, deve-se discutir com o gerente do projeto da empresa, ou o responsável pela sua implementação, quais as restrições que o sistema deve satisfazer.

No exemplo escolhido, este será o momento em que se deve decidir sobre como será feito o *log* das ações dos usuários e as demais políticas de acesso e segurança do sistema, por exemplo. Em uma análise mais detalhada, pode-se identificar que os acessos não-permitidos ao sistema, ou a funcionalidades deste, devem ser registradas no *log*.

O requisito de *logging* pode ser descrito por um grupo de exigências, senão veja-se:

E1. Quando um usuário buscar informações sobre um determinado ponto georreferenciado, o sistema registrará a mencionada busca, bem assim o momento de sua ocorrência e a identificação do respectivo usuário.

E2. Quando um gerente (com permissão para incluir, alterar ou excluir dados) efetuar a inclusão, alteração ou exclusão de informações, o sistema registrará a aludida ação, além do momento de sua ocorrência e a identificação do respectivo usuário.

E3. Quando um administrador do sistema (com permissão para incluir, alterar e excluir usuários), efetuar a inclusão, alteração ou exclusão de um usuário, o sistema registrará a citada ação, o momento de sua ocorrência e a identificação do administrador.

E4. Quando um usuário efetuar uma ação (inclusão, alteração ou exclusão) sem permissão para tanto, o sistema registrará sua tentativa, identificando o tipo da ação, o momento e a sua autoria.

E5. Quando um usuário tentar efetuar o *login* no sistema usando nome de usuário ou senha inválidos, o sistema registrará a ocorrência, bem assim o nome de usuário e senha informados.

A natureza dessas exigências pode ser melhor analisada após o estudo detalhado dos requisitos funcionais.

Os outros requisitos não-funcionais do sistema são, dentre outros, a segurança e o sistema multi-usuário.

IV.4.6.2 – IDENTIFICAÇÃO E ESPECIFICAÇÃO DOS REQUISITOS FUNCIONAIS

Para especificar os requisitos funcionais do sistema propõe-se a utilização do modelo de casos de uso e diagramas de interação da UML.

Analisando-se o conjunto de requisitos, identifica-se os seguintes atores:

- Funcionário: responsável por buscar informações no SIG;
- Gerente: responsável por inserir, alterar e excluir dados no SIG. Também pode buscar informações no SIG;
- Administrador: responsável por cadastrar, alterar e excluir usuários visitantes e editores do sistema. Também detém os poderes do usuário editor;

A seguir encontram-se os casos de uso exigidos pelos atores listados acima após a sua autenticação no sistema:

- Autenticação: responsável por verificar se o usuário tem acesso ao SIG, bem assim suas prerrogativas no sistema.
- Buscar dados: responsável por buscar na base de dados do SIG as informações requisitadas pelo usuário e apresentá-las na tela do sistema;
- Inserir dados: permite a inserção de dados no SIG;
- Alterar dados: permite a alteração de dados já existentes no SIG;
- Excluir dados: responsável pela exclusão de dados do SIG;
- Cadastrar usuários: permite o cadastramento de usuários visitantes e editores ao sistema;
- Alterar usuários: permite a alteração do cadastro dos usuários do sistema;
- Excluir usuários: responsável pela exclusão dos usuários do sistema.

A figura 4.4, a seguir, representa o diagrama de casos de uso para o sistema proposto.

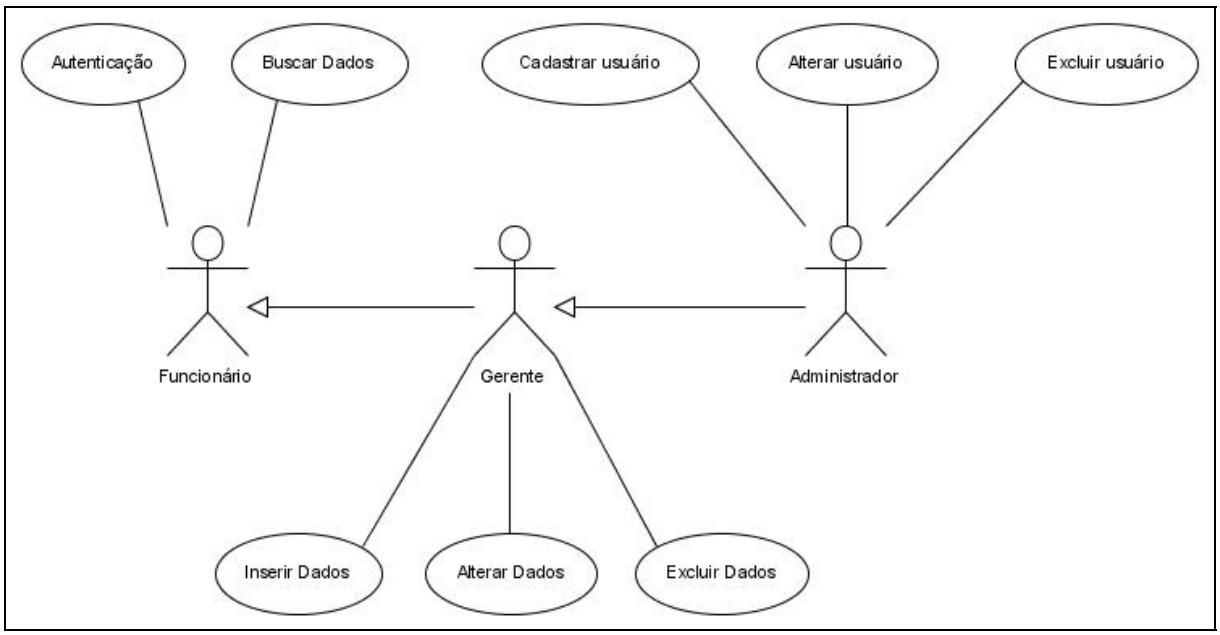


Figura 4.4 – Diagrama de casos de uso para o sistema

Para todos os casos de uso acima, exceto o *Buscar Dados,* identifica-se, no mínimo, dois cenários possíveis; um para lidar com os usuários que possuem o direito de executar a atividade pretendida, e, outro para tratar daqueles que não tem a prerrogativa necessária. Cada um destes cenários pode ser descrito usando diagramas de seqüência.

A figura 4.5, adiante, representa um diagrama de seqüência do cenário primário acima mencionado para o caso de uso *Alterar Dados*.

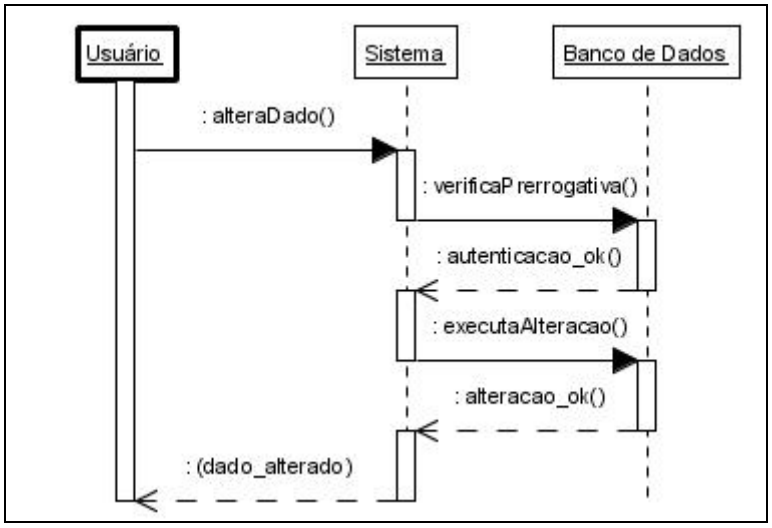


Figura 4.5 – Diagrama de Seqüência para o caso de uso *Alterar Dados*

A idéia é que o SIG seja representado pelos objetos Sistema e Banco de Dados, em conjunto, ou seja, a troca de mensagens entre os objetos Sistema e Banco de Dados acima foi representada apenas para facilitar o entendimento do funcionamento do sistema.. O usuário interage somente com o SIG, requisitando, por exemplo, a alteração de um dado. O sistema

verifica se este usuário tem o direito de executar a tarefa e, caso possua, executa a alteração solicitada na base de dados, enviando, ao final, uma mensagem para o usuário informando-lhe que a ação foi executada com sucesso.

IV.4.6.3 – IDENTIFICAÇÃO E ESPECIFICAÇÃO DOS REQUISITOS TRANSVERSAIS

Um requisito não-funcional é transversal se ele afetar mais de um caso de uso. Por exemplo, considerando-se o *log* das ações dos usuários no sistema como um requisito não-funcional, conclui-se facilmente que o mencionado requisito é transversal, pois afeta todos os casos de uso do sistema.

O requisito transversal deve ser especificado conforme a Tabela 4.9, a seguir.

Assim, para o requisito de *logging*, ter-se-ia:

Tabela 4.9 – Especificação do requisito transversal de *logging*

Requisito Transversal	*Logging*
Descrição	Registra todas as ações dos usuários no sistema, além das tentativas de autenticação frustradas.
Prioridade	Alta
Requisitos	E1, E2, E3, E4, E5

IV.4.6.4 – COMPONDO REQUISITOS TRANSVERSAIS EM MODELOS UML COM A ASPECTUML

Os critérios para integrar os requisitos funcionais e transversais são a integralidade e a suficiência. Com a integralidade garante-se que todos os requisitos necessários para suportar a composição estão incluídos no aspecto. Com a suficiência garantimos que cada requisito no aspecto deve estar associado a um impacto no processo de composição.

Veja-se na figura 4.6, a seguir, a composição do aspecto *logging* com o diagrama de casos de uso utilizado anteriormente para a identificação e especificação dos requisitos funcionais:

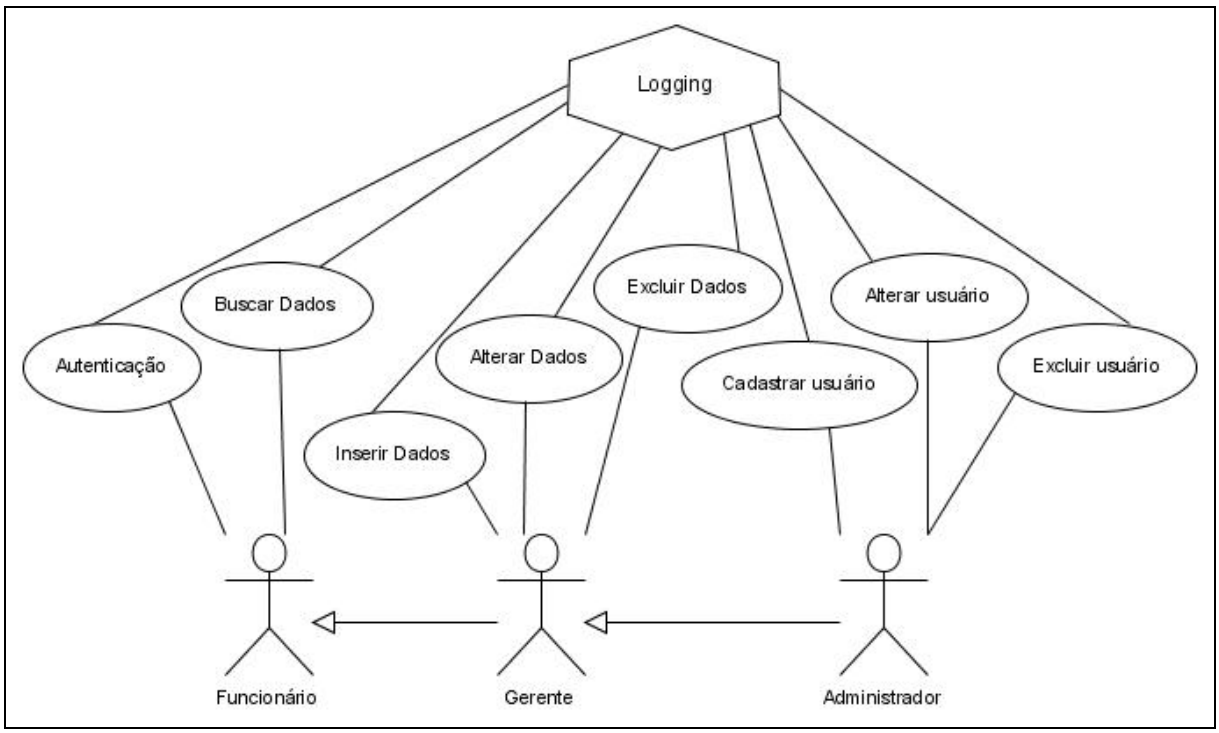


Figura 4.6 – Diagrama de casos de uso composto com o aspecto *Logging*

O aspecto *Logging* monitora os casos de uso *Autenticação*, *Buscar Dados, Inserir Dados, Alterar Dados, Excluir Dados, Cadastrar usuário, Alterar usuário* e *Excluir usuário*, isto é, as funcionalidades referentes a cada um dos casos de uso citados é monitorada pelo comportamento descrito nos requisitos do aspecto *Logging*. O aludido aspecto registra informações relativas a cada uma das ações (casos de uso) observadas, para todos os usuários ativos no sistema.

IV.4.6.5 – IDENTIFICANDO E SOLUCIONANDO CONFLITOS

A composição dos interesses multidimensionais em um modelo de requisitos pode revelar a existência de conflitos que precisam ser solucionados, isto é, um requisito transversal pode causar situações contraditórias no sistema como, por exemplo, a interação do aspecto *logging* com o aspecto responsável pela segurança do sistema.

Quando realizada uma tentativa de autenticação com um usuário desconhecido no sistema, dependo das prioridades de cada um dos aspectos acima mencionados, duas hipóteses podem ocorrer: o acesso pode ser impedido sem o registro da tentativa frustrada ou a ocorrência é registrada e, simultaneamente, a entrada do usuário no sistema é impossibilitada.

Para evitar este conflito, especifica-se qual dos aspectos tem maior prioridade, estabelecendo que o *logging* prevalece sobre a segurança, o que propicia o funcionamento do sistema conforme a segunda hipótese acima citada.

Logo, diante disso, sugere-se a realização de um estudo prévio das contribuições de cada um dos requisitos transversais para os demais. Esta contribuição pode ser positiva ou negativa. Se dois (ou mais) interesses multidimensionais contribuem negativamente para cada um deles, tem-se um comportamento conflitante se, e somente se, estes afetarem os mesmos requisitos funcionais do sistema.

IV.4.6.6 – COMPARANDO OS CÓDIGOS DE *LOGGING*: OO X POA

Geralmente, nos sistemas que utilizam a programação orientada a objetos, as chamadas ao *log* ficam espalhadas nos métodos, como a seguir:

```
public class ControlaMapaSHP {

Logger logger = Logger.getLogger("global");
private void AbreMapa(Arquivo nomedomapa){
//Utiliza o objeto logger, registrando a entrada no método
//Código referente a abertura e o carregamento de um mapa no sistema
//Utiliza o objeto logger, registrando a saída do método
}
private void FechaMapa (Arquivo nomedomapa){
//Utiliza o objeto logger, registrando a entrada no método
//Código referente ao fechamento do arquivo de um mapa no sistema
//Utiliza o objeto logger, registrando a saída do método
}
}
```

Figura 4.7 - Classe ControlaMapaSHP

O exemplo anterior mostra apenas a classe controladora dos mapas, entretanto o entrelaçamento do código repete-se também em outras classes do sistema, referentes à inserção, alteração e exclusão de dados nos mapas, bem assim nas de gerenciamento dos usuários. Esta repetição de código, conhecida como dispersão de código, deve ser evitada, pois quanto maior o número de métodos na aplicação e a necessidade de registrar suas atividades no *log*, maior será a quantidade de linhas espalhadas pelo código implementando esta funcionalidade no sistema. Isto pode acarretar valores extremamente altos, característica indesejável, pois dificulta a manutenção da aplicação e, diminui sua performance.

Realizando a separação do requisito de *logging* dos métodos da classe controladora dos mapas, originando um aspecto, o mesmo código ficará como adiante:

```
public class ControlaMapaSHP {

   private void AbreMapa(Arquivo nomedomapa){
   //Código referente a abertura e o carregamento de um mapa no sistema
   }
   private void FechaMapa (Arquivo nomedomapa){
   //Código referente ao fechamento do arquivo de um mapa no sistema
   }
}
```

Figura 4.8 - Classe ControlaMapaSHP

```
public aspect ALogging {

   Logger logger = Logger.getLogger("global")

   Pointcut logMethod() : execution (* Controla*.*(..));

   before():logMethod() {
     Signature sig = thisJoinPointStaticPart.getSignature();
     logger.logp(Level.INFO,sig.getDeclaringType().getName(),sig.getName(),
"Entrando");
   }

   after():logMethod() {
     Signature sig = thisJoinPointStaticPart.getSignature();
     logger.logp(Level.INFO,sig.getDeclaringType().getName(),sig.getName(),
"Saindo");
   }
}
```

Figura 4.9 - Aspecto ALogging

As figuras anteriores mostram a separação do código em unidades diferentes. Para implementar a funcionalidade de *logging* foi criado o aspecto *ALogging*, que utiliza os conceitos de pontos de corte e pontos de junção para determinar quando e qual código será executado em um dado momento. Assim, o aspecto *ALogging* define o ponto de corte *logMethod* que determina que este será ativado quando forem executados os métodos das classes controladoras, isto é, todas que comecem com o prefixo *Controla*. As regras *before* e *after* determinam os códigos que devem ser executados antes e depois de qualquer classe ou método capturado pelo ponto de corte *logMethod*. O resultado é exatamente o mesmo do apresentado na figura 4.7, que tem as chamadas ao *log* dentro de cada um dos métodos.

Portanto, pode-se concluir facilmente que a programação orientada a aspectos permite maior modularidade do sistema, eliminando os códigos espalhados dos requisitos transversais que são desenvolvidos como aspectos, o que não é possível com a programação orientada a objetos. Essa característica é muito útil na implementação de sistemas de informações geográficas, porque diminui o esforço de manutenção do referido sistema, diminuindo o custo de suas atualizações e aumentando sua performance.

IV.4.6.7 – DIAGRAMA DE CLASSES E ASPECTOS

O diagrama de classes e aspectos, extensão do diagrama de classes da UML para o suporte a aspectos, foi sugerido por (RESENDE, 2005), que, baseando-se nos elementos que compõem uma classe (atributos e métodos) e sua respectiva representação na UML, concluiu, por analogia, pela possibilidade da representação dos aspectos e de seus *pointcuts* e *advices* por meio da caixa de representação adiante disposta:

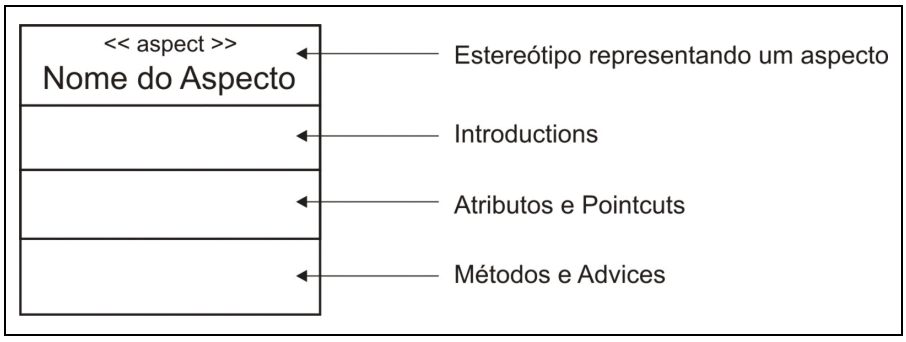


Figura 4.10 – Caixa de representação de um aspecto

Como se vê na figura 4.10, a caixa de representação de um aspecto assemelha-se à representação de uma classe da UML, entretanto contém alguns pontos adicionais, isto é, a possibilidade de inclusão dos *pointcuts* e *advices* nos espaços que eram reservados para atributos e métodos, respectivamente. Ademais, pode-se notar uma nova área, reservada para os *introductions*, além da inclusão do estereótipo <<aspect>>, já mencionado no item 4.4.1 deste trabalho.

Segundo (RESENDE, 2005) o *introduction* é utilizado para alterar a estrutura da classe, introduzindo novos atributos, métodos, construtores, *getters, setters,* herança e interface.

Com efeito, a sintaxe dos elementos da caixa de representação deve seguir os padrões da orientação a aspectos, escrevendo-se somente a assinatura no caso de *advices* e métodos inseridos por *introductions*. Os *pointcuts* e os outros elementos devem ser representados de forma idêntica a que se encontram declarados nos fontes.

Na figura 4.11, a seguir, visualiza-se a representação do aspecto *ALogging*, mencionado anteriormente, bem assim de seus elementos:

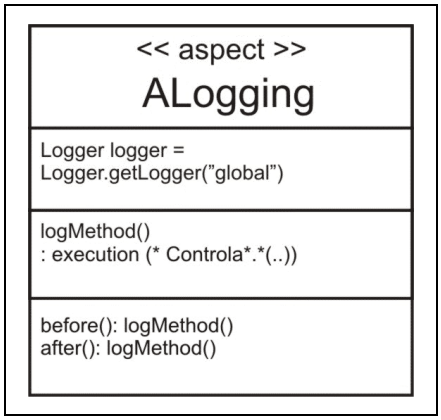


Figura 4.11 – Exemplo de representação do aspecto *ALogging*

No exemplo anterior, definiu-se o aspecto *ALogging* que contém um *pointcut logMethod()* que intercepta as execuções de métodos das classes controladoras, isto é, todas que comecem com a *string Controla*. Além disso, dois *advices* serão executados antes e depois do *pointcut* citado.

Para facilitar o entendimento da representação do aspecto, pode-se utilizar os estereótipos antes da lista de elementos, como a seguir:

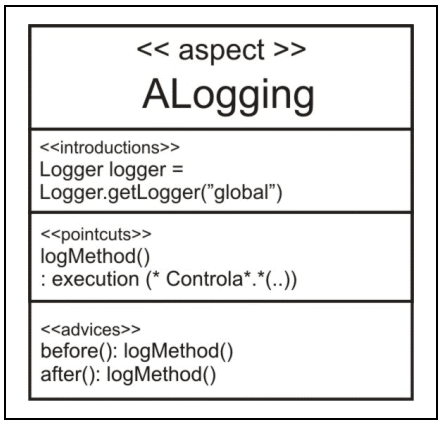


Figura 4.12 – Exemplo de representação do aspecto *ALogging* com estereótipos

A figura 4.13, adiante, traz um exemplo do diagrama de classes e aspectos utilizando a representação proposta, no qual o aspecto *ALogging* intercepta a execução da classe *ControlaMapaSHP*:

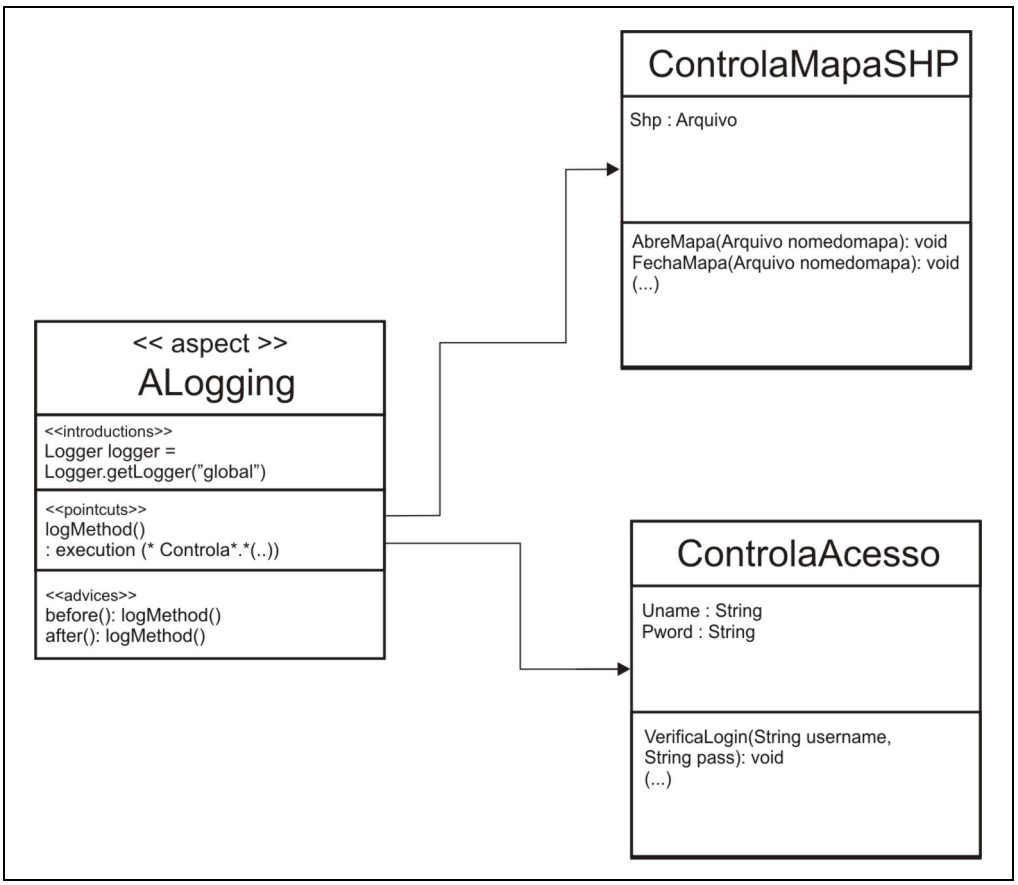


Figura 4.13 – Exemplo do diagrama de classes e aspectos

IV.4.6.8 – DIAGRAMA DE SEQÜÊNCIA

A interceptação de *pointcuts* e a execução dos *advices* devem ser enfatizadas na representação dos aspectos em diagramas de seqüência. Contudo, diante da dinâmica gerada pelos *pointcuts*, relacionada ao problema de *obliviousness*[11], é desnecessária à representação de aspectos em todos os diagramas de seqüência da aplicação.

Assim, sugere-se a criação de apenas alguns diagramas de seqüência com aspectos em um nível abstrato, para a verificação, validação e análise do comportamento da aplicação.

A notação proposta por (RESENDE, 2005) para representação de aspectos em diagramas de seqüência encontra-se descrita na tabela 4.10, a seguir.

Tabela 4.10 – Elementos para representação de aspectos em diagramas de seqüência

Elementos	Descrição
Pointcuts	*Pointcuts* devem ser representados como atores do sistema. No caso de um *pointcut* chamado "teste", o nome do ator deve ser definido como "*pointcut(*teste)"

[11]Conhece-se como *obliviousness* o problema gerado pela inserção de classes e métodos, ou até mesmo a alteração de um *pointcut*, que podem implicar em várias mudanças no sistema, tornando inviável a expressão de todas as ligações do *pointcut* manualmente em um diagrama.

Advices	Os *advices* de acordo com sua forma de execução, por exemplo, antes (*before*) ou depois (*after*) de um *pointcut*, devem estabelecer uma ligação com o mesmo, e deste ponto em diante definir normalmente a seqüência de operações executadas. O nome da representação de um *advice* executado antes do *pointcut* "teste" deve ser representado como "before: teste(Parâmetros)"
Introductions	Os *Introductions* podem ser invocados normalmente a partir da inclusão da classe estática no diagrama de seqüência.

Na figura adiante visualiza-se o uso das definições da tabela mencionada, veja-se:

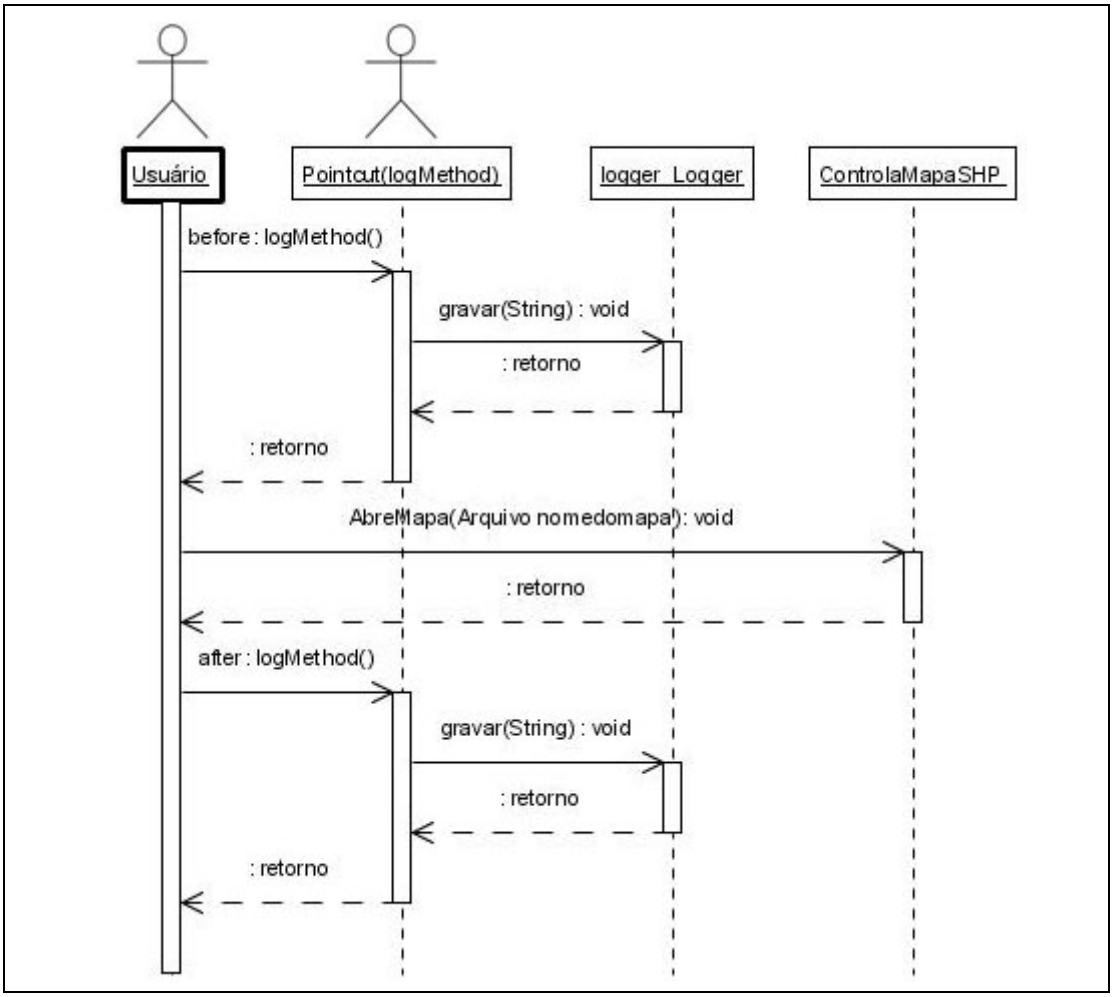


Figura 4.14 – Exemplo de diagrama de seqüência com aspectos

Ressalte-se que o diagrama anterior ilustra um objeto da classe *Logger* invocado pelos *advices* e a chamada pelo usuário ao método *AbreMapa(Arquivo nomedomapa)*, não se preocupando em representar o aspecto em si, mas sim as chamadas aos seus elementos, sendo o principal interesse demonstrar a seqüência com que as invocações são realizadas, aproveitando as notações padrão da UML.

Com efeito, em virtude da utilização das notações padrão da UML, notam-se limitações como, por exemplo, na representação de *pointcuts,* que, neste modelo, recebem invocações, ao invés de interceptações em tempo de execução, como na realidade.

Entretanto, embora a notação oficial da UML não preveja a representação dos aspectos em seus diagramas, isto é, o uso da extensão AspectUML ora proposta, pode-se criar os estereótipos aqui mencionados em aplicações de modelagem que permitam a aludida customização.

IV.5 – COMENTÁRIOS

Neste capítulo propôs-se a adoção de um processo de modelagem de sistemas utilizando a programação orientada a aspectos para a obtenção de maior modularidade do *software*, além da ausência de dispersão e entrelaçamento de código, características que dificultam e encarecem o desenvolvimento e a manutenção de qualquer sistema.

Com efeito, para auxiliar a modelagem e o desenvolvimento de sistemas utilizando a programação orientada a aspectos, criou-se uma extensão da UML, a AspectUML, para a representação dos aspectos nos diagramas da UML durante a fase de modelagem do sistema.

Por oportuno, exemplificou-se a utilização do processo mencionado acima e da AspectUML, permitindo a visualização dos diagramas de modelagem do sistema.

Além disso, concluiu-se que a programação orientada a aspectos traz inúmeros benefícios aos sistemas com ela desenvolvidos, o que evidentemente sugere sua adoção na implementação de sistemas de informação geográfica.

No próximo capítulo serão analisados os requisitos de um sistema *web* para auxiliar o desenvolvimento, a manutenção e a documentação de sistemas de informação geográfica usando a programação orientada a aspectos. Também serão abordados um exemplo e um estudo de caso visando à demonstração das vantagens da programação orientada a aspectos.

CAPÍTULO V – SISTEMA *WEB* DE ACOMPANHAMENTO DO DESENVOLVIMENTO DE SIGS USANDO PROGRAMAÇÃO ORIENTADA A ASPECTOS

V.1 – INTRODUÇÃO

No capítulo anterior abordou-se a identificação de requisitos transversais nos SIGs, bem como os benefícios da adoção do paradigma da programação orientada a aspectos (POA) no desenvolvimento de SIGs, além da AspectUML, uma extensão da UML para a representação dos aspectos, que auxilia na modelagem e no desenvolvimento de sistemas.

Como exemplo, utilizou-se um estudo de caso para representar os diagramas de casos de uso, de seqüência e de classes, todos da UML, com os aspectos.

Este capítulo versará sobre os requisitos do sistema *web* GD-SIGPOA, implementado neste trabalho, cujo objetivo precípuo é auxiliar o desenvolvimento, a manutenção e a documentação de SIGs usando a programação orientada a aspectos. Além disso, serão abordados um exemplo e um estudo de caso para se demonstrar a utilização da AspectUML e os benefícios da orientação a aspectos.

V.2 – ANÁLISE DE REQUISITOS DO SISTEMA

Os objetivos do sistema *web* proposto são acompanhar e auxiliar as fases do desenvolvimento de sistemas de informação geográfica utilizando a programação orientada a aspectos, bem como a sua manutenção e documentação. Além disso, será um repositório de conhecimento, pois estimulará a reutilização de código e de soluções já implementadas em outros SIGs.

Para tanto, a aplicação possuirá 4 (quatro) perfis de acesso, quais sejam:

- **Administrador do sistema**: responsável por registrar, alterar e excluir usuários do sistema.
- **Gerente de projetos**: responsável por cadastrar e alterar os projetos de SIGs sob sua coordenação, além das informações principais inerentes a cada um deles, que deverão ser mantidas atualizadas.
- **Analista**: responsável por inserir e alterar informações sobre os requisitos funcionais, não-funcionais e transversais de cada um dos projetos em que faz parte da equipe.
- **Visitante**: pode visualizar todas as informações constantes no sistema.

O administrador, os gerentes e os analistas devem se autenticar no sistema para ter acesso às funcionalidades restritas aos seus perfis.

Todos os usuários cadastrados no sistema (administrador, gerentes e analistas) podem alterar suas informações pessoais.

V.2.1 – NOVOS PROJETOS DE SIGS E ACOMPANHAMENTO

Cada projeto possui informações de seu cadastramento e andamento, que serão inseridas pelo gerente do projeto do SIG. Ao cadastrar novos projetos no sistema, o gerente pode informar os seguintes itens: nome do projeto, equipe responsável pelo projeto, usuários do sistema, seus objetivos e restrições, escopo, cronograma, custo, patrocinador, plataformas necessárias ao sistema (características), modelo de ciclo de vida.

Ao atualizar as informações de um determinado projeto, o gerente deste projeto pode alterar todos os campos acima descritos, bem como o campo de data de desativação e a sua justificativa.

A aplicação possui o histórico do cadastramento e de todas as alterações feitas posteriormente pelo gerente do projeto.

V.2.2 – CADASTRAMENTO DE REQUISITOS: FUNCIONAIS, NÃO-FUNCIONAIS E TRANSVERSAIS (ASPECTOS)

Ao cadastrar um requisito na aplicação, o analista deve categorizá-lo, indicando qual o tipo de requisito, além de informar os seguintes itens: nome do requisito (ou da funcionalidade), descrição, requisitos (ou exigências), palavras-chaves, componentes do sistema afetados e prioridade.

Nos casos de requisitos funcionais, deve-se registrar também qual(is) agente(s) externo(s) que o utiliza(m).

Os requisitos podem ser incrementados, ou alterados. Entretanto não haverá histórico das mudanças, somente a última versão será armazenada na base de dados.

V.2.3 – CADASTRAMENTO DE CLASSES, DE ASPECTOS E DE SEUS RELACIONAMENTOS

As classes e os aspectos de sistemas de informação geográfica implementam seus requisitos funcionais e não-funcionais.

Portanto, ao cadastrar uma classe, o analista deve relacioná-la a um determinado requisito, previamente cadastrado, além de informar os seguintes itens: nome da classe, descrição, atributos e métodos.

No caso do cadastramento de um aspecto, além do relacionamento com o requisito, devem ser informados os seguintes itens: nome do aspecto, descrição, *introductions*, *pointcuts* e *advices*.

Além disso, o analista poderá cadastrar relacionamentos *aspecto – aspecto*, *classe – classe* e *aspecto – classe*, utilizando as associações descritas no capítulo anterior.

V.2.4 – RELATÓRIOS

Vários relatórios poderão ser gerados no sistema, quais sejam:

a) **Requisitos do SIG** – visualização de todos os requisitos cadastrados no sistema, agrupados em requisitos funcionais e não-funcionais.

b) **Aspectos do SIG** – visualização de todos os aspectos cadastrados, agrupados de acordo com o requisito não-funcional que implementam. Neste relatório também se informam quais os relacionamentos de cada um dos aspectos.

c) **Classes do SIG** – visualização de todas as classes do sistema, agrupadas de acordo com o requisito que implementam. Além disso, também se informam quais os relacionamentos de cada uma das classes.

d) **Detalhamento do Projeto do SIG** – visualização das informações detalhadas do projeto atualizadas e de suas versões anteriores, utilizando o histórico de mudanças.

V.2.5 – BUSCAS (PESQUISAS)

O sistema permite a realização de buscas em seu conteúdo para, por meio dessa forma de gestão de conhecimento, propiciar a reutilização de códigos e informações de projetos anteriores, e até mesmo daqueles ainda em andamento.

Espera-se que a documentação fornecida pelo sistema, por meio de seus relatórios, seja útil a ponto de ser reutilizada, economizando-se tempo e dinheiro em novos projetos e nas suas manutenções, tornando os SIGs mais dinâmicos com os aspectos.

Os usuários do sistema poderão buscar informações das seguintes formas:

a) **Busca por categoria de requisito** – nesta opção visualizam-se todos os requisitos cadastrados na categoria escolhida pelo usuário.

b) **Busca por aspectos** – por meio de uma palavra-chave o usuário pode buscar aspectos cadastrados no sistema, bem como os requisitos não-funcionais por eles implementados e seus relacionamentos.

C) **Busca por classes** – por meio de uma palavra-chave o usuário pode buscar classes cadastradas no sistema, bem como os requisitos por elas implementados e seus relacionamentos.

V.2.6 – PREMISSAS DO SISTEMA

O sistema contém documentação técnica, modelo de dados e tutorial simplificado de utilização, constantes nos apêndices deste trabalho.

Além disso, foi desenvolvido para ser utilizado no servidor *web Apache* juntamente com o *Tomcat*, ambos de código aberto, com a base de dados *MySQL*, também *open-source*, funcionando de forma adequada nos navegadores *Internet Explorer 5.0* ou superiores e seus semelhantes (*Mozilla*).

As linguagens de programação utilizadas foram: *Java Server Pages* (JSP), *JavaScript*, *Hypertext Markup Language* (HTML) e *Structured Query Language* (SQL).

V.2.7 – TESTES

Para validar o bom funcionamento do sistema de gerenciamento do desenvolvimento de sistemas de informação geográfica usando a programação orientada a objetos (GD-SIGPOA), analisaram-se, durante este trabalho, um exemplo e um estudo de caso, sendo o primeiro mais simples - projeto de SIG para a CEDAE - e o segundo mais complexo - projeto do SIGAIAPI de (NASCIMENTO, 2004) - utilizando a orientação a aspectos.

Cabe ressaltar, que no exemplo e no estudo de caso adiante mencionados serão abordados requisitos de cada um dos projetos acima citados que poderiam ser implementados como um aspecto, visando facilitar a manutenção do código e o seu reuso, além de evitar seu entrelaçamento e a sua dispersão.

V.2.7.1 – SIG CEDAE

Como visto anteriormente, no item 4.4.6, a CEDAE – Companhia de Águas e Esgotos do Rio de Janeiro vem digitalizando toda sua base cartográfica disponível, referente às

tubulações de água e esgoto da cidade do Rio de Janeiro, para inserir esses dados em um sistema de informação geográfica, a ser implementado, para facilitar o gerenciamento e a tomada de decisões na mencionada empresa.

Dentre os inúmeros requisitos funcionais e não-funcionais do sistema, encontra-se o de *logging* das ações de seus usuários – requisito que afeta mais de um caso de uso – , podendo-se concluir que se trata de um requisito transversal. Portanto, o aludido requisito pode ser implementado como um aspecto.

Na seção 4.4.6.6 mostrou-se que, geralmente, as chamadas ao *log* ficam espalhadas nos métodos do sistema, repetindo seu código em diversas classes, de forma que quanto maior for o número de métodos na aplicação e a necessidade de registrar suas atividades no *log*, maior será a quantidade de linhas espalhadas pelo código implementando essa funcionalidade no sistema. Posteriormente, realizou-se a separação do requisito de *logging* dos métodos da classe controladora dos mapas, originando um aspecto. Como exemplo, utilizou-se a classe *ControlaMapaSHP* e o aspecto *ALogging*.

Por sua vez, no item 4.4.6.7, apresentou-se o diagrama de classes e aspectos sugerido por (RESENDE, 2005) para o caso analisado, informando os *introductions*, *pointcuts* e *advices*.

V.2.7.2 – SIGAIAPI

O sistema de informação geográfica para auxiliar a implantação e avaliação de provedores de internet (SIGAIAPI) foi concebido e integralmente implementado por (NASCIMENTO, 2004) para armazenar informações sobre os municípios e provedores de acesso à Internet do Estado do Rio de Janeiro, relacionadas com a população, qualidade de linhas de comunicação de dados, renda, urbanização e concorrência em uma região composta por determinado grupo de municípios, permitindo consultas para a obtenção da melhor opção para se implantar um novo provedor de acesso.

Para o funcionamento da mencionada aplicação foi criada uma DLL (*Dynamic Link Library*, biblioteca de vínculo dinâmico) dividida em duas classes, ambas com acessos à base de dados do sistema, *clsAcesso* – responsável apenas pela validação do usuário, retornando se o mesmo é válido, inválido ou não possui permissão – e *clsFuzzy*, responsável por todo o processamento do sistema em si, conforme a figura a seguir:

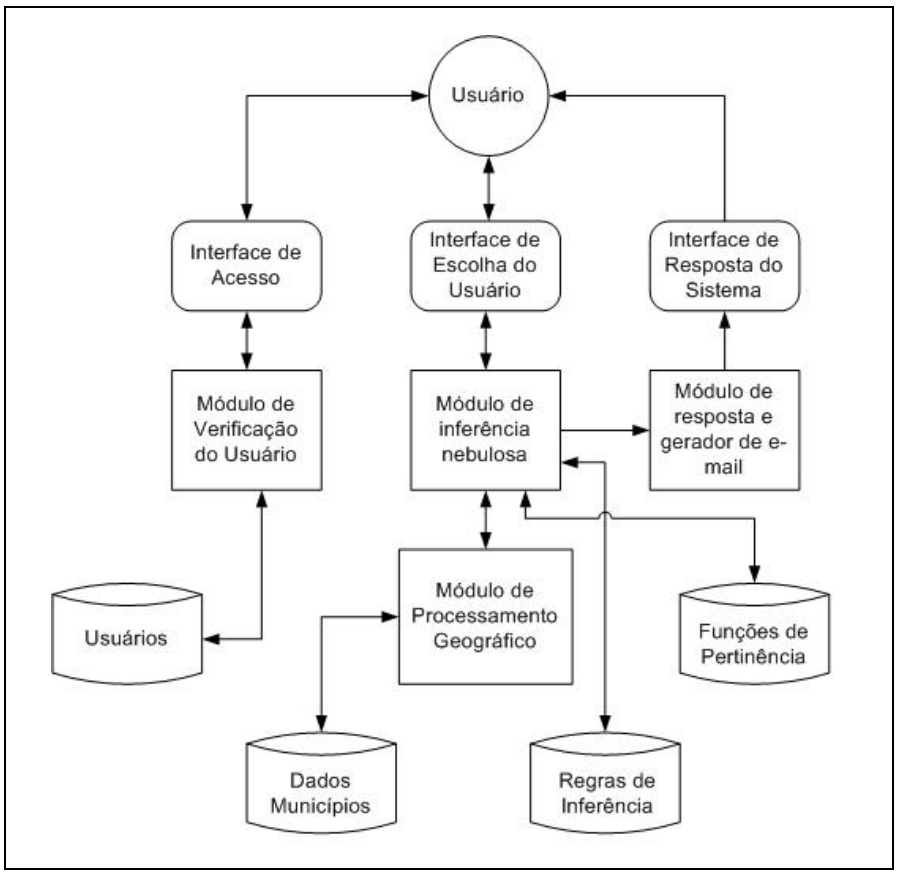


Figura 5.1 – Arquitetura do sistema (NASCIMENTO, 2004)

Como se vê na figura 5.1, cada módulo do sistema possui um acesso à base de dados distinto, permitindo-se concluir pela existência de dispersão no código da aplicação referente à conexão ao banco de dados, o que diminui a sua performance e dificulta a sua manutenção.

Cabe ressaltar que esse problema poderia ser solucionado com a criação de um aspecto para acessar a base de dados, uma vez que concentraria todo o código necessário à conexão ao banco de dados em um só ponto do sistema.

O aspecto *ABanco* monitoraria os casos de uso de *Verificação do Usuário*, *Inferência Nebulosa* e *Processamento Geográfico*, isto é, as funcionalidades referentes a cada um dos casos de uso citados seria monitorada pelo comportamento descrito nos requisitos do aspecto *ABanco*.

Portanto, pode-se concluir, facilmente, que a programação orientada a aspectos permite a maior modularidade do sistema, eliminando os códigos espalhados dos requisitos transversais que são desenvolvidos como aspectos, o que não é possível com a programação orientada a objetos. Essa característica é muito útil na implementação de sistemas de informações geográficas, como visto neste estudo de caso, pois diminui o esforço de manutenção do sistema, minimizando o custo de suas atualizações e aumentando sua performance.

Adiante se encontra a arquitetura do sistema SIGAIAPI com o aspecto *ABanco*:

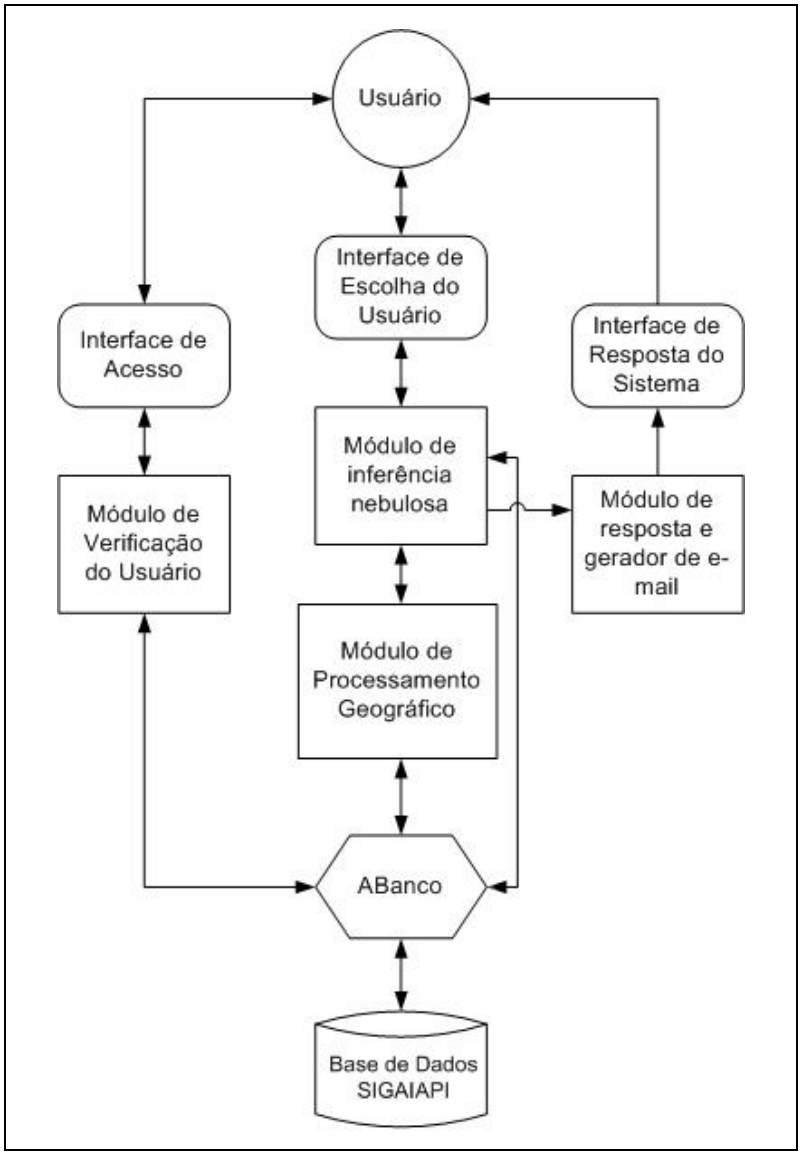


Figura 5.2 – Arquitetura do sistema SIGAIAPI com POA

Por fim, cabe ressaltar que todas as informações descritas no exemplo e no estudo de caso, referentes ao requisito de *logging* e de acesso à base de dados, respectivamente, foram incluídas no sistema GD-SIGPOA.

V.3 – COMENTÁRIOS

Neste capítulo analisaram-se os requisitos do sistema *web* idealizado (GD-SIGPOA) para o auxílio do desenvolvimento, da manutenção e da documentação de sistemas de informação geográfica que utilizem o paradigma da orientação a aspectos.

Além disso, abordaram-se um exemplo e um estudo de caso visando à demonstração da utilização da programação orientada a aspectos e seus benefícios em casos concretos.

CAPÍTULO VI – CONCLUSÃO

Os *softwares*, ao longo dos anos, vêm adquirindo maior complexidade em razão do avanço tecnológico inerente as mais diversas áreas do conhecimento.

Por sua vez, os sistemas de informação geográfica – SIGs – desenvolvidos atualmente contam com inúmeras funcionalidades, para automatizar e auxiliar o processo de tomada de decisão das empresas, dentre elas aquelas que ficam dispersas pela aplicação, responsáveis pelo controle, ou gerenciamento, de seu funcionamento.

A programação orientada a objetos tem sido a mais utilizada nos dias atuais para o desenvolvimento de SIGs. Contudo, não é hábil o suficiente para lidar com requisitos multidimensionais do sistema, isto é, requisitos que afetem diversas funcionalidades da aplicação como, por exemplo, políticas de segurança, *logging*, acessos a base de dados e outros códigos que ficam dispersos e entrelaçados pelos componentes da aplicação.

Para atender essa crescente demanda, em razão da perda de performance e do aumento do custo de manutenção dos *softwares* com os problemas acima descritos, surgiu um novo paradigma, a programação orientada a aspectos, linguagem que foi objeto de estudo ao longo deste trabalho para a aplicação no desenvolvimento de SIGs.

Este trabalho teve por finalidade apresentar um novo processo de desenvolvimento de sistemas de informação geográfica usando a orientação a aspectos, como também criar uma extensão para a UML (linguagem de modelagem unificada) voltada para a representação dos aspectos – a AspectUML, além do sistema *web* GD-SIGPOA para o gerenciamento do desenvolvimento de sistemas de informação geográfica usando a programação orientada a aspectos, todos objetivando auxiliar a modelagem e a implementação dos SIGs com os benefícios da POA.

Realizou-se um exemplo e um estudo de caso, o primeiro mais simples, no qual se demonstrou os benefícios da utilização da orientação a aspectos na implementação do requisito de *logging* para um SIG com políticas de segurança rígidas, onde todos os acessos e ações dos usuários deveriam ser registrados. O segundo baseou-se no SIGAIAPI, desenvolvido integralmente por (NASCIMENTO, 2004), que contava com as rotinas de acesso a base de dados dispersas pelos diversos componentes da aplicação.

As principais contribuições deste trabalho foram a criação da AspectUML e do sistema GD-SIGPOA, ambos para auxiliar o desenvolvimento de SIGs mais complexos, com diversos requisitos multidimensionais que, necessariamente, devem ser implementados como aspectos, quando se objetiva alcançar maior performance e menores custos de manutenção do código.

A partir desta dissertação são apresentadas algumas sugestões para trabalhos futuros:

- O aperfeiçoamento do sistema proposto GD-SIGPOA, aumentando suas funcionalidades de gerência de projetos de SIGs;
- A inclusão de mais valores com *tag* na AspectUML, objetivando representar todos os possíveis tipos de associações e relacionamentos entre aspectos e classes;
- O desenvolvimento de um aplicativo para a criação de diagramas UML com a extensão AspectUML;
- A abordagem de outros possíveis requisitos não funcionais que poderiam originar aspectos nos sistemas de informação geográfica.

REFERÊNCIAS

ANSELMO, F., 2002, *Tudo o que você queria saber sobre JSP quando utiliza o Servidor TomCat com o Banco MySQL*, 1ª edição, Santa Catarina : VisualBooks Editora, 192 p.

ARONOFF, S., 1989, *Geographic Information Systems*. Canadá : WDL Publications.

BARROS, P., 2000, *UML - Linguagem de Modelagem Unificada*. Disponível em <http://www.eribeiro.com.br/pablo/uml/>. Último acesso em 30/04/2005.

BEIER, G.; KERN, M., 2002, *Aspects in UML Models from a Code Generation Perspective in* Aspect-Oriented Modeling with UML as part of the AOSD, Second International Workshop on Aspect-Oriented Modeling with UML, Alemanha. Disponível em <http://lgl.epfl.ch/workshops/uml2002/papers/beier.pdf>. Último acesso em 30/06/2005.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I., 2000, *UML — Guia do Usuário*, Editora Campus.

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

BORGES, K. A. V.; LAENDER, A. H. F.; DAVIS JR, C. A., 1999, *Spatial data integrity constraints in object oriented geographic data modeling in* ACM GIS SYMPOSIUM ON ADVANCES IN GEOGRAPHIC INFORMATION SYSTEMS, Kansas City: ACM.

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

CÂMARA, G., 1995, *Modelos, Linguagens e Arquiteturas para Bancos de Dados Geográficos*, PhD thesis, INPE.

CÂMARA, G.; CASANOVA, M.A.; HEMERLY, A.; MEDEIROS, C.M.B.; MAGALHÃES, G., 1996, *Anatomia de Sistemas de Informação Geográfica.* SBC, X Escola de Computação, Campinas.

CÂMARA, G.; DAVIS, C.; MONTEIRO, A.M.V., 2002, *Introdução à Ciência da Geoinformação*. DPI INPE Disponível em: <http://www.dpi.inpe.br/gilberto/livro/introd/index.html>. Último acesso em 22/02/2005.

COWEN, D. J., 1990, *GIS versus CAD versus DBMS: what are the Differences? in* Introductory Readings in Geographical Information Systems, 1990, Taylor and Francis.

EGENHOFER, M., FRANK A., 1992, *Object-Oriented Modeling for GIS*. URISA Journal, v. 4, n. 2, pp. 3-19.

EGENHOFER, M., 1995, *Object-oriented GISs: The Principles,* Technical report, NCGIA.

ELMASRI, R.; NAVATHE, S. B., 2000, *Fundamentals of Database Systems*, 3ª Edição, Menlo Park : Addison-Wesley.

FIELDS, D. K.; KOLB, M. A., 2000, *Desenvolvendo na Web com JavaServer Pages*, Rio de Janeiro: Editora Ciência Moderna Ltda.

FRUTOSO, E., VALENTE, M., 2004, *Implementação de um sistema web usando programação orientada por aspectos in* I Simpósio Mineiro de Sistemas de Informação, MG, p.94-105.

GOODCHILD, M. *et al.*, 1992, *Integrating GIS and Spatial Data Analysis: Problems and Possibilities*. International Journal of Geographical Information Systems, pp. 407-424.

GROHER, I.; SCHULZE, S., 2003, *Generating Aspect Code from UML Models in* Aspect-Oriented Modeling with UML as part of the AOSD, Third International Workshop on Aspect-Oriented Modeling with UML, Boston, EUA. Disponível em <http://lgl.epfl.ch/workshops/aosd2003/papers/Groher-AspectCodeFromUML.pdf>. Último acesso em 30/06/2005.

HARMON, P.; KING, D., 1988, *Sistemas Especialistas*, Rio de Janeiro: Campus.

JACOBSON, I.; CHRISTERSON, M.; JONSSON, P.; OVERGAARD, G., 1992, *Object-Oriented Software Engineering - A Use Case Driven Approach*, Addison-Wesley Publishing Company, Wokingham, England.

KICZALES, G.; LAMPING, J.; MENHDHEKAR, A.; MAEDA, C.; LOPES, C.; LOINGTIER, J.; IRWIN, J., 1997, *Aspect Oriented-Programming,* ECOOP – European Conference on Object-Oriented Programming, Spring-Verlag.

KICZALES, G.; HILSDALE, E.; HUGUNIN, J.; KERSTEN, M.; PALM, J.; GRISWOLD, W.G., 2001, *An Overview of AspectJ,* ECOOP – European Conference on Object-Oriented Programming, Spring-Verlag.

KORTE, G. B., 1997, *The Gis Book: understanding the value and implementation of geographic information systems,* 4th. Edition, Santa Fe: OnWord Press, 411p.

KORTH, H. F.; SILBERCHATZ, A.; SUDARSHAN, S., 1999, *Sistemas de Bancos de Dados*, 3ª edição, São Paulo : Makron Books, 778p.

LADDAD, R., 2003, *AspectJ in Action,* Manning Publications, 1st ed.

LISBOA FILHO, J.; IOCHPE, C., 1999, *GeoFrame: um Framework Conceitual para Especificação de Padrões de Análise em Banco de Dados Geográficos*. Porto Alegre: PGCC da UFRGS.

LISBOA FILHO, J. *in* LADEIRA, M.; NASCIMENTO, M.E.M., 2000, III Escola Regional de Informática do Centro-Oeste. Brasília-DF: SBC - Sociedade Brasileira de Computação, pp.137-171.

MAGUIRE, D., GOODCHILD, M., RHIND, D., 1993, *Geographical Information Systems - volume I*, 2ª. edição, John Wiley and Sons.

MATOS, A.V., 2002, *UML: Prático e Descomplicado*, São Paulo: Érica.

MONTEIRO, E.; PIVETA, E., 2003, *Programação Orientada a Aspectos em AspectJ - Anais do V Encontro de Estudantes de Informática do Tocantins*, pp. 313-322. Palmas, TO.

MYSQL, 2005, *MySQL Reference Manual: Overview of the MySQL Database Management System*, Disponível em: <http://dev.mysql.com/doc/mysql/en/what-is.html>. Acesso em 27.03.2005.

NASCIMENTO, C., 2004, *Sistema de informação geográfica para auxiliar a implantação e avaliação de provedores de internet – SIGAIAPI,* Dissertação de Mestrado, UERJ.

PARENT, C.; SPACCAPIETRA, S.; ZIMANYI, E.; DONINI, P.; PLAZANET, C.; VANGENOT, C., 1998, *Modeling spatial data in the MADS conceptual model. In: INTERNATIONAL SYMPOSIUM ON SPATIAL DATA HANDLING,* Canadá.

PRESSMAN, R., 2002, *Engenharia de Sofware,* 5ª edição, Rio de Janeiro : McGraw-Hill, 843p.

RESENDE, A.; SILVA, C., 2005, *Programação orientada a aspectos em Java*, Rio de Janeiro : Brasport.

SILVA, R. P., PRICE, R. T., 1998, *A busca de generalidade, flexibilidade e extensibilidade no processo de desenvolvimento de frameworks orientados a objetos*. Porto Alegre: PPGC da UFRGS.

SMITH, T., MENON, S., STAR, J., ESTES, J., 1987, *Requirements and Principles for the Implementation and Construction of Large-scale Geographic Information Systems*. International Journal of Geographical Information Systems, 1(1):13-31.

UML, 2002. Disponível em: <http://www.uml.org/>. Último acesso em 24/09/2005.

VINHAS, L., QUEIROZ, G., FERREIRA, K., CÂMARA, G., PAIVA, J., 2002, Programação Genérica Aplicada a Algoritmos Geográficos in IV Simpósio Brasileiro de GeoInformática. Caxambu - MG, v.1, p.117-122.

APÊNDICES

APÊNDICE A – DOCUMENTAÇÃO TÉCNICA DO SISTEMA GD-SIGPOA

A.1 – CONFIGURAÇÃO

O sistema requer, para seu perfeito funcionamento, a instalação em um servidor com o *JAVA* e o *Apache Tomcat* presentes, permitindo a leitura e execução de arquivos .jsp. Além disso, também é necessária a presença do servidor de banco de dados *MySQL* para o armazenamento dos dados da aplicação.

A.2 – ESTRUTURA DE DIRETÓRIOS

O diretório *gd_sigpoa* é o diretório raiz da aplicação e possui os seguintes subdiretórios: as pastas *adm*, *images* e *inc* conforme pode ser visualizado na figura abaixo.

Figura A.1 – Diretórios do Sistema GD-SIGPOA

A.2.1 – DESCRIÇÃO DETALHADA DOS DIRETÓRIOS

- **adm**: contém todos os arquivos .jsp utilizados na aplicação, exceto a tela de login inicial.
- **images**: pasta contendo todas as imagens utilizadas no sistema.
- **inc**: este diretório contém todos os arquivos *JavaScripts* utilizados na aplicação para a verificação dos formulários, bem como a folha de estilo (CSS), a parte superior e o rodapé de todas as páginas e a conexão a base de dados.

A.3 – ARQUIVOS DA APLICAÇÃO

A.3.1 – DIRETÓRIO *ADM*

Arquivos: asp_sig.jsp e asp_sig2.jsp
Descrição: Estes dois arquivos implementam a funcionalidade do relatório de aspectos do projeto de SIG.

Arquivos: bu_asp.jsp e bu_asp2.jsp

Descrição: Estes dois arquivos implementam a funcionalidade de busca por aspectos no sistema.

Arquivos: bu_catr.jsp e bu_catr2.jsp
Descrição: Estes dois arquivos implementam a funcionalidade de busca por categoria de requisitos.

Arquivos: bu_cla.jsp e bu_cla2.jsp
Descrição: Estes dois arquivos implementam a funcionalidade de busca por classes no sistema.

Arquivos: cad_classes_aspectos.jsp, cad_classes_aspectos2.jsp e cad_classes_aspectos3.jsp
Descrição: Estes três arquivos implementam a funcionalidade de cadastro de classes e aspectos.

Arquivos: cad_projeto.jsp e cad_projeto2.jsp
Descrição: Estes dois arquivos implementam a funcionalidade de cadastro de projetos.

Arquivos: cad_relacionamentos.jsp, cad_relacionamentos2.jsp e cad_relacionamentos3.jsp
Descrição: Estes três arquivos implementam a funcionalidade de cadastro de relacionamentos.

Arquivos: cad_requisitos.jsp e cad_requisitos2.jsp
Descrição: Estes dois arquivos implementam a funcionalidade de cadastro de requisitos.

Arquivos: cad_usuario.jsp e cad_usuario2.jsp
Descrição: Estes dois arquivos implementam a funcionalidade de cadastro de usuários.

Arquivos: cla_sig.jsp e cla_sig2.jsp
Descrição: Estes dois arquivos implementam a funcionalidade do relatório de classes de um projeto de SIG.

Arquivos: del_usuario.asp
Descrição: Este arquivo implementa a funcionalidade de exclusão de usuários.

Arquivos: det_proj.jsp, det_proj2.jsp e det_proj3.jsp
Descrição: Estes três arquivos implementam a funcionalidades de visualização do detalhamento dos projetos e suas alterações.

Arquivos: edit_classes_aspectos.jsp, edit_classes_aspectos2.jsp, edit_classes_aspectos3.jsp e edit_classes_aspectos4.jsp
Descrição: Estes quatro arquivos implementam a funcionalidade de alteração de classes e aspectos do sistema.

Arquivos: edit_projeto.jsp, edit_projeto2.jsp e edit_projeto3.jsp

Descrição: Estes três arquivos implementam a funcionalidade de alteração dos projetos cadastrados. Nesta edição os dados alterados são gravados em uma tabela de histórico do acompanhamento (atualizações) do projeto.

Arquivos: edit_relacionamentos.jsp, edit_relacionamentos2.jsp, edit_relacionamentos3.jsp e edit_relacionamentos4.jsp
Descrição: Estes quatro arquivos implementam a funcionalidade de alteração de relacionamentos.

Arquivos: edit_requisitos.jsp, edit_requisitos2.jsp e edit_requisitos3.jsp
Descrição: Estes três arquivos implementam a funcionalidade de alteração dos requisitos de um projeto de SIG. Cabe ressaltar, que na edição de requisitos não há um histórico das alterações realizadas.

Arquivos: edit_usuario.jsp, edit_usuario2.jsp e edit_usuario3.jsp
Descrição: Estes três arquivos implementam a funcionalidade de alteração das informações pessoais dos usuários.

Arquivos: edit_usuario.asp, edit_usuario2.asp e edit_usuario3.asp
Descrição: Estes três arquivos implementam a funcionalidade de edição das informações pessoais dos usuários, permitindo ao usuário alterar seus dados e senha.

Arquivo: home.jsp
Descrição: Este arquivo é chamado logo após o *login* do usuário no sistema. Tem a funcionalidade de apresentar uma mensagem para o usuário.

Arquivos: req_sig.jsp e req_sig2.jsp
Descrição: Estes dois arquivos implementam a funcionalidade do relatório de requisitos do projeto de SIG.

A.3.2 – DIRETÓRIO *INC*

Arquivo: bottom.htm
Descrição: Este arquivo contém o rodapé das páginas do sistema.

Arquivo: buscas.js
Descrição: Este arquivo contém todos os scripts de verificação de formulário que são usados nas buscas do sistema.

Arquivo: cad_classes_aspectos.js
Descrição: Este arquivo contém todos os scripts de verificação de formulário que são usados no cadastro e alteração de classes e aspectos.

Arquivo: cad_projeto.js
Descrição: Este arquivo contém todos os scripts de verificação de formulário que são usados no cadastro e alteração de projetos.

Arquivo: cad_relacionamentos.js
Descrição: Este arquivo contém todos os scripts de verificação de formulário que são usados no cadastro e alteração de relacionamentos.

Arquivo: cad_requisitos.js
Descrição: Este arquivo contém todos os scripts de verificação de formulário que são usados no cadastro e alteração de requisitos.

Arquivo: cad_usuario.js
Descrição: Este arquivo contém todos os scripts de verificação de formulário que são usados no cadastro e edição de usuários.

Arquivo: gd_sigpoa.css
Descrição: Este arquivo é a folha de estilo (CSS) do sistema.

Arquivo: menu.jsp e menu_home.jsp
Descrição: Estes arquivos contêm os menus da aplicação, bem como a funcionalidade de segurança, isto é, só deve mostrar os menus de acordo com o perfil do usuário autenticado no sistema.

Arquivo: mm_menu.js
Descrição: Este arquivo contém o código fonte da funcionalidade dos pop-up menus.

Arquivo: db_conn.jsp
Descrição: Este arquivo contém a conexão com a base de dados MySQL do sistema.

Arquivo: top.jsp e top_home.jsp
Descrição: Estes arquivos contêm o design da barra superior do sistema.

APÊNDICE B – MODELO DE DADOS DO SISTEMA GD-SIGPOA

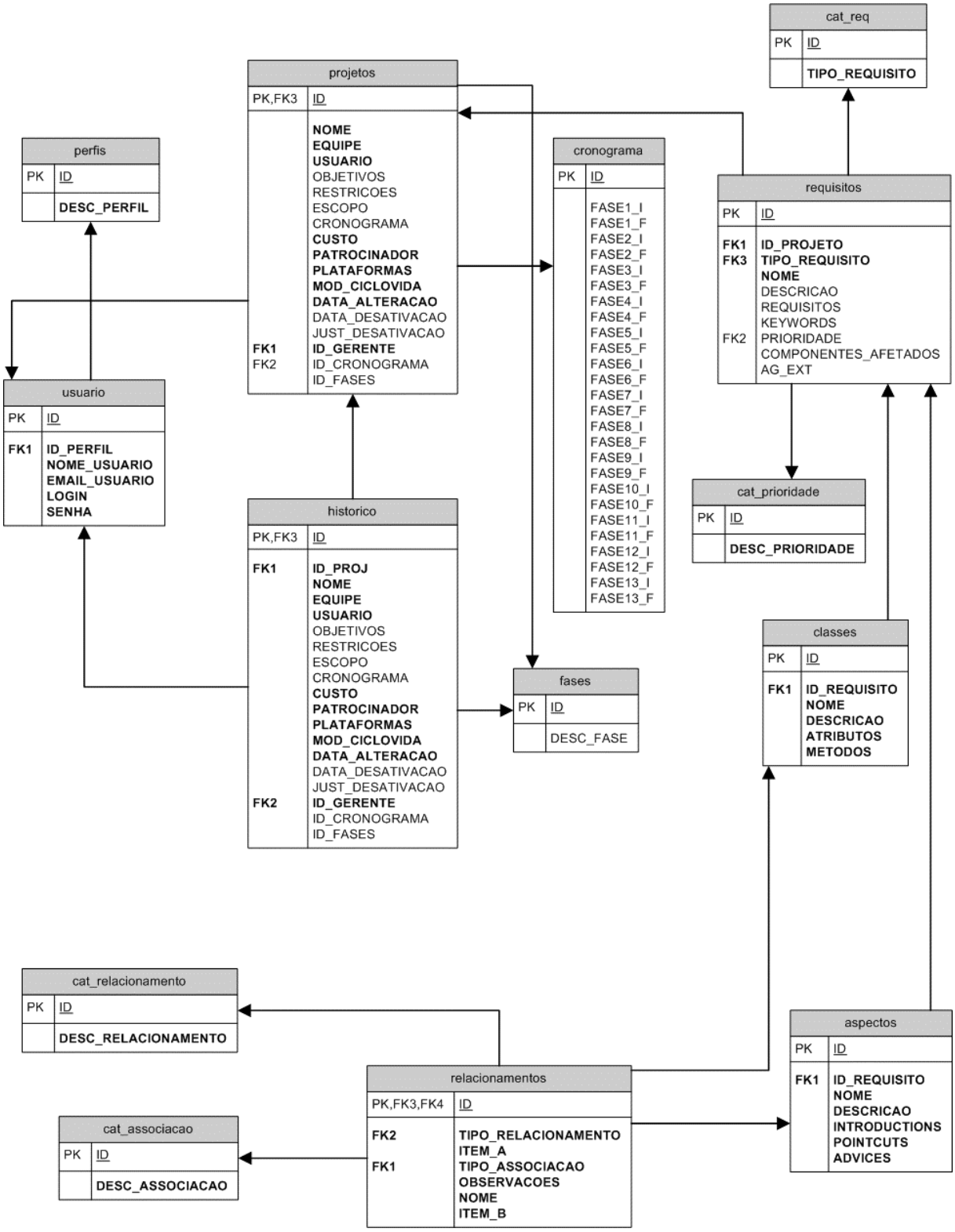


Figura B.1 – Modelo de dados do sistema GD-SIGPOA

APÊNDICE C – DOCUMENTAÇÃO DA BASE DE DADOS DO SISTEMA GD-SIGPOA

C.1 – BASE DE DADOS MYSQL

Servidor: localhost

Base de Dados: gd_sigpoa

Usuário: gd_sigpoa_user

Senha: xptovoid

C.2 – TABELAS

Tabela C.1 – Tabela Aspectos

ASPECTOS						
Coluna	Tipo de Dado	Chave Primária	Tamanho	Não Nulo	Auto Incremento	Observações
ID	INT	X	11	X	X	Identificador dos aspectos
ID_REQUISITO	INT		11	X		Relaciona o aspecto a um requisito
NOME	VARCHAR		100			O nome do aspecto
DESCRICAO	VARCHAR		2000			A descrição do aspecto
INTRODUCTIONS	VARCHAR		500			Os introductions do aspecto
POINTCUTS	VARCHAR		500			Os pointcuts do aspecto
ADVICES	VARCHAR		500			Os advices do aspecto

Tabela C.2 – Tabela CAT_ASSOCIACAO

CAT_ASSOCIACAO						
Coluna	Tipo de Dado	Chave Primária	Tamanho	Não Nulo	Auto Incremento	Observações
ID	INT	X	11	X	X	Identificador da categoria de associação
DESC_ASSOCIACAO	VARCHAR		100			Descrição da associação

Tabela C.3 – Tabela CAT_PRIORIDADE

CAT_PRIORIDADE						
Coluna	Tipo de Dado	Chave Primária	Tamanho	Não Nulo	Auto Incremento	Observações
ID	INT	X	11	X	X	Identificador da prioridade
DESC_PRIORIDADE	VARCHAR		10			Descrição da prioridade

Tabela C.4 – Tabela CAT_RELACIONAMENTO

CAT_RELACIONAMENTO						
Coluna	Tipo de Dado	Chave Primária	Tamanho	Não Nulo	Auto Incremento	Observações
ID	INT	X	11	X	X	Identificador da categoria de relacionamento
DESC_RELACIONAMENTO	VARCHAR		100			Descrição da categoria de relacionamento

Tabela C.5 – Tabela CAT_REQ

CAT_REQ						
Coluna	Tipo de Dado	Chave Primária	Tamanho	Não Nulo	Auto Incremento	Observações
ID	INT	X	11	X	X	Identificador do tipo de requisito
TIPO_REQUISITO	VARCHAR		20			Descrição do tipo de requisito

Tabela C.6 – Tabela CAT_PRIORIDADE

CAT_PRIORIDADE						
Coluna	Tipo de Dado	Chave Primária	Tamanho	Não Nulo	Auto Incremento	Observações
ID	INT	X	11	X	X	Identificador da prioridade
DESC_PRIORIDADE	VARCHAR		10			Descrição da prioridade

Tabela C.7 – Tabela CLASSES

CLASSES						
Coluna	Tipo de Dado	Chave Primária	Tamanho	Não Nulo	Auto Incremento	Observações
ID	INT	X	11	X	X	Identificador das classes
ID_REQUISITO	INT		11	X		Relaciona a classe a um requisito
NOME	VARCHAR		100			O nome da classe
DESCRICAO	VARCHAR		2000			A descrição da classe
ATRIBUTOS	VARCHAR		500			Os atributos da classe
METODOS	VARCHAR		500			Os metodos da classe

Tabela C.8 – Tabela HISTORICO

HISTORICO						
Coluna	Tipo de Dado	Chave Primária	Tamanho	Não Nulo	Auto Incremento	Observações
ID	INT	X	11	X	X	Identificador dos itens de historico
ID_PROJ	INT		11	X		Identificador do projeto relacionado
NOME	VARCHAR		100	X		Nome do projeto
EQUIPE	VARCHAR		500	X		Equipe do projeto
USUARIO	VARCHAR		500	X		Usuário do projeto
OBJETIVOS	VARCHAR		2000			Objetivos do projeto
RESTRICOES	VARCHAR		2000			Restrições do projeto
ESCOPO	VARCHAR		8000			Escopo do projeto
CUSTO	VARCHAR		100	X		Custo do projeto
PATROCINADOR	VARCHAR		200	X		Patrocinador do projeto
PLATAFORMAS	VARCHAR		200	X		Plataformas do projeto
MOD_CICLOVIDA	VARCHAR		200	X		Modelo do ciclo de vida do projeto
DATA_ALTERACAO	DATE			X		Data de alteração
DATA_DESATIVACAO	DATE					Data de desativação do projeto
JUST_DESATIVACAO	VARCHAR		2000			Justificativa da desativação
ID_GERENTE	INT		11			Identificador do gerente
ID_CRONOGRAMA	INT		11			Identificador do cronograma
ID_FASE	INT		11			Identificador da fase

Tabela C.9 – Tabela FASES

FASES						
Coluna	Tipo de Dado	Chave Primária	Tamanho	Não Nulo	Auto Incremento	Observações
ID	INT	X	11	X	X	Identificador da fase do projeto de SIG
DESC_FASE	VARCHAR		50			Descrição da fase do projeto de SIG

Tabela C.10 – Tabela CRONOGRAMA

CRONOGRAMA						
Coluna	Tipo de Dado	Chave Primária	Tamanho	Não Nulo	Auto Incremento	Observações
ID	INT	X	11	X	X	Identificador dos itens de historico
FASE1_I	DATE					Data de início da Fase 1
FASE1_F	DATE					Data de fim da Fase 1
FASE2_I	DATE					Data de início da Fase 2
FASE2_F	DATE					Data de fim da Fase 2
FASE3_I	DATE					Data de início da Fase 3
FASE3_F	DATE					Data de fim da Fase 3
FASE4_I	DATE					Data de início da Fase 4
FASE4_F	DATE					Data de fim da Fase 4
FASE5_I	DATE					Data de início da Fase 5
FASE5_F	DATE					Data de fim da Fase 5
FASE6_I	DATE					Data de início da Fase 6
FASE6_F	DATE					Data de fim da Fase 6
FASE7_I	DATE					Data de início da Fase 7
FASE7_F	DATE					Data de fim da Fase 7
FASE8_I	DATE					Data de início da Fase 8
FASE8_F	DATE					Data de fim da Fase 8
FASE9_I	DATE					Data de início da Fase 9
FASE9_F	DATE					Data de fim da Fase 9
FASE10_I	DATE					Data de início da Fase 10
FASE10_F	DATE					Data de fim da Fase 10
FASE11_I	DATE					Data de início da Fase 11
FASE11_F	DATE					Data de fim da Fase 11
FASE12_I	DATE					Data de início da Fase 12
FASE12_F	DATE					Data de fim da Fase 12
FASE13_I	DATE					Data de início da Fase 13
FASE13_F	DATE					Data de fim da Fase 13

Tabela C.11 – Tabela PERFIS

PERFIS						
Coluna	Tipo de Dado	Chave Primária	Tamanho	Não Nulo	Auto Incremento	Observações
ID	INT	X	11	X	X	Identificador do perfil do usuário
DESC_PERFIL	VARCHAR		20			Descrição do perfil do usuário

Tabela C.12 – Tabela RELACIONAMENTOS

RELACIONAMENTOS						
Coluna	ipo de Dad	Chave Primária	Tamanho	Não Nulo	Auto Incremento	Observações
ID	INT	X	11	X	X	Identificador do relacionamento
TIPO_RELACIONAMENTO	INT		11	X		Tipo do relacionamento
TIPO_ASSOCIACAO	INT		11	X		Tipo da associação
ITEM_A	INT		11	X		Item A
ITEM_B	INT		11	X		Item B
NOME	VARCHAR		100	X		Nome do relacionamento
OBSERVACOES	VARCHAR		500	X		Observações sobre o relacionamento

Tabela C.13 – Tabela PROJETOS

PROJETOS						
Coluna	Tipo de Dado	Chave Primária	Tamanho	Não Nulo	Auto Incremento	Observações
ID	INT	X	11	X	X	Identificador dos projetos
NOME	VARCHAR		100	X		Nome do projeto
EQUIPE	VARCHAR		500	X		Equipe do projeto
USUARIO	VARCHAR		500	X		Usuário do projeto
OBJETIVOS	VARCHAR		2000			Objetivos do projeto
RESTRICOES	VARCHAR		2000			Restrições do projeto
ESCOPO	VARCHAR		8000			Escopo do projeto
CUSTO	VARCHAR		100	X		Custo do projeto
PATROCINADOR	VARCHAR		200	X		Patrocinador do projeto
PLATAFORMAS	VARCHAR		200	X		Plataformas do projeto
MOD_CICLOVIDA	VARCHAR		200	X		Modelo do ciclo de vida do projeto
DATA_ALTERACAO	DATE			X		Data de alteração
DATA_DESATIVACAO	DATE					Data de desativação do projeto
JUST_DESATIVACAO	VARCHAR		2000			Justificativa da desativação
ID_GERENTE	INT		11			Identificador do gerente
ID_CRONOGRAMA	INT		11			Identificador do cronograma
ID_FASE	INT		11			Identificador da fase

Tabela C.14 – Tabela REQUISITOS

REQUISITOS						
Coluna	Tipo de Dado	Chave Primária	Tamanho	Não Nulo	Auto Incremento	Observações
ID	INT	X	11	X	X	Identificador do requisito
ID_PROJETO	INT		11	X		Identificador do projeto
TIPO_REQUISITO	INT		11	X		Tipo do requisito
NOME	VARCHAR		50	X		Nome do requisito
DESCRICAO	VARCHAR		2000			Descrição do requisito
REQUISITOS	VARCHAR		500			Requisitos/Exigências a serem cumpridas
KEYWORDS	VARCHAR		200			Palavras-chaves
PRIORIDADE	INT		11			Prioridade
COMPONENTES	VARCHAR		500			Componentes
AG_EXT	VARCHAR		200			Agentes externos

APÊNDICE D – TUTORIAL SIMPLIFICADO DO SISTEMA GD-SIGPOA

D.1 – OBJETIVO E FUNCIONALIDADES

O sistema de Gerenciamento do Desenvolvimento de Sistemas de Informação Geográfica usando Programação Orientada a Aspectos – GD-SIGPOA objetiva acompanhar e auxiliar as fases do desenvolvimento de SIGs utilizando a programação orientada a aspectos, bem assim a sua manutenção e documentação. Além disso, o mencionado sistema será um repositório de conhecimento, pois estimulará a reutilização de código e de soluções já implementadas em outros SIGs cadastrados na aplicação.

As principais funcionalidades do sistema são:

- Cadastrar e acompanhar projetos;
- Cadastrar e alterar requisitos;
- Cadastrar e alterar classes e aspectos;
- Cadastrar e alterar relacionamentos;
- Visualizar relatórios;
- Realizar buscas no sistema;

D.2 – TELA DE AUTENTICAÇÃO

Para acessar o sistema, o usuário já deve possuir uma conta previamente cadastrada pelo administrador. Caso ainda não a tenha, pode entrar em contato com este para solicitá-la.

Abaixo visualiza-se a tela de autenticação:

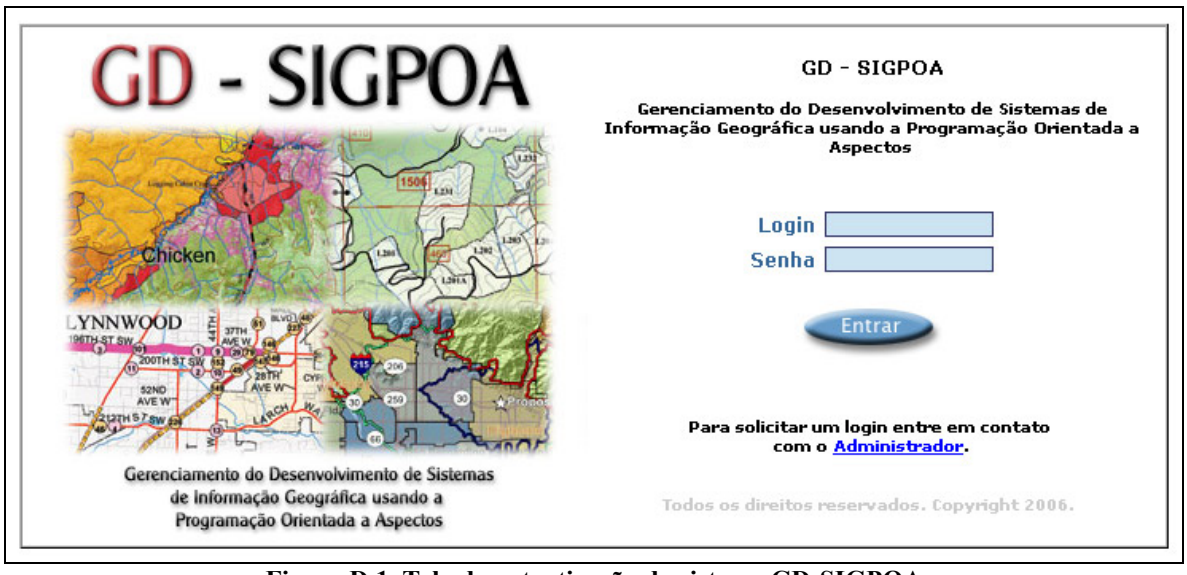


Figura D.1. Tela de autenticação do sistema GD-SIGPOA

D.3 – PERFIS DE ACESSO

O sistema possui quatro perfis de acesso diferentes:

- **Administrador:** responsável por cadastrar e excluir usuários;

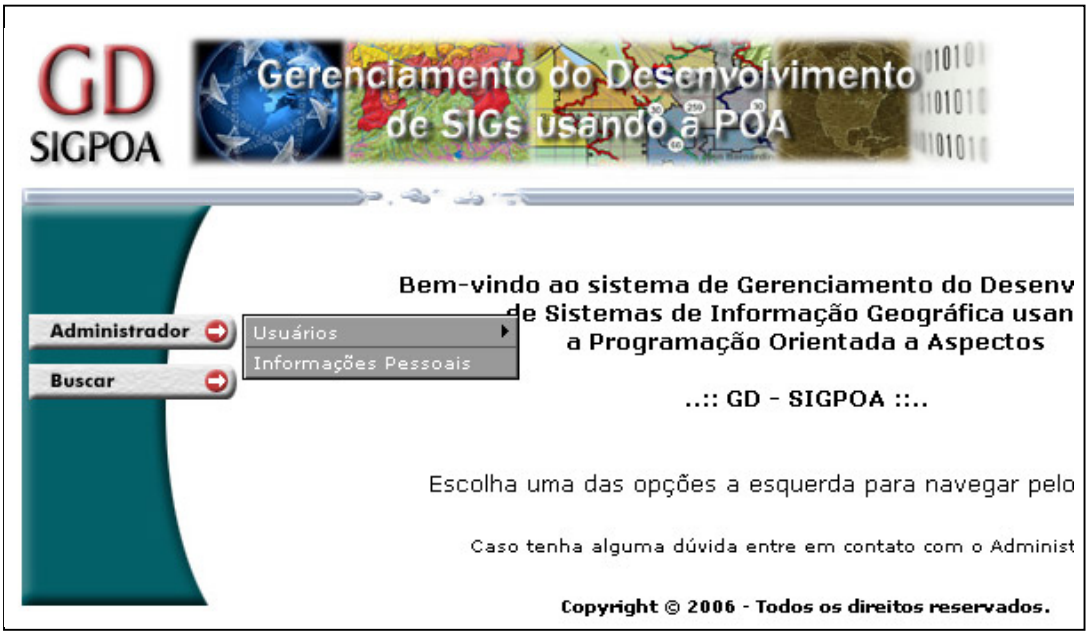


Figura D.2. Tela do sistema no perfil de acesso de Administrador

- **Gerente de projetos:** responsável por cadastrar e alterar os projetos de SIGs sob sua coordenação, além das informações principais inerentes a cada um deles, que deverão ser mantidas atualizadas;

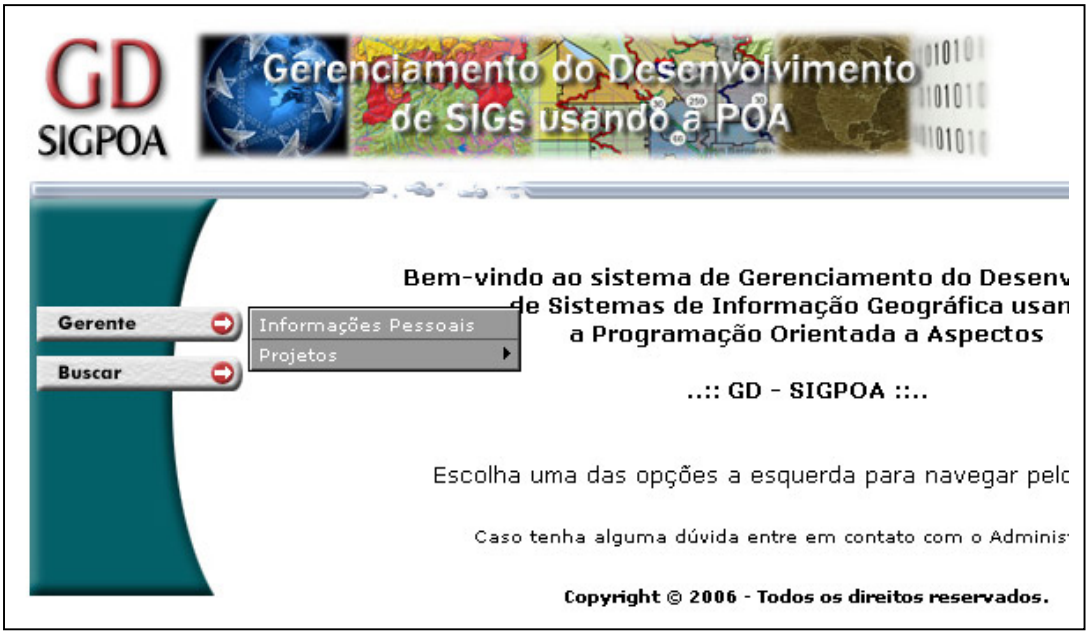


Figura D.3. Tela do sistema no perfil de acesso de Gerente

- **Analista:** responsável por inserir e alterar informações sobre os requisitos funcionais, não-funcionais e transversais de cada um dos projetos em que faz parte da equipe;

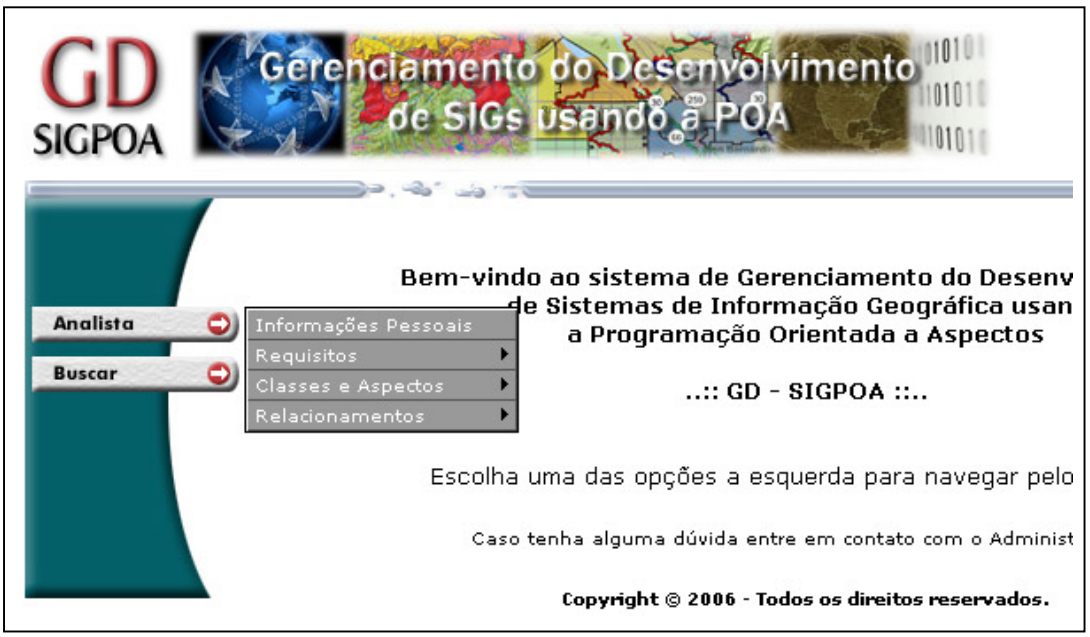


Figura D.4. Tela do sistema no perfil de acesso de Analista

- **Visitante:** pode visualizar todas as informações constantes no sistema.

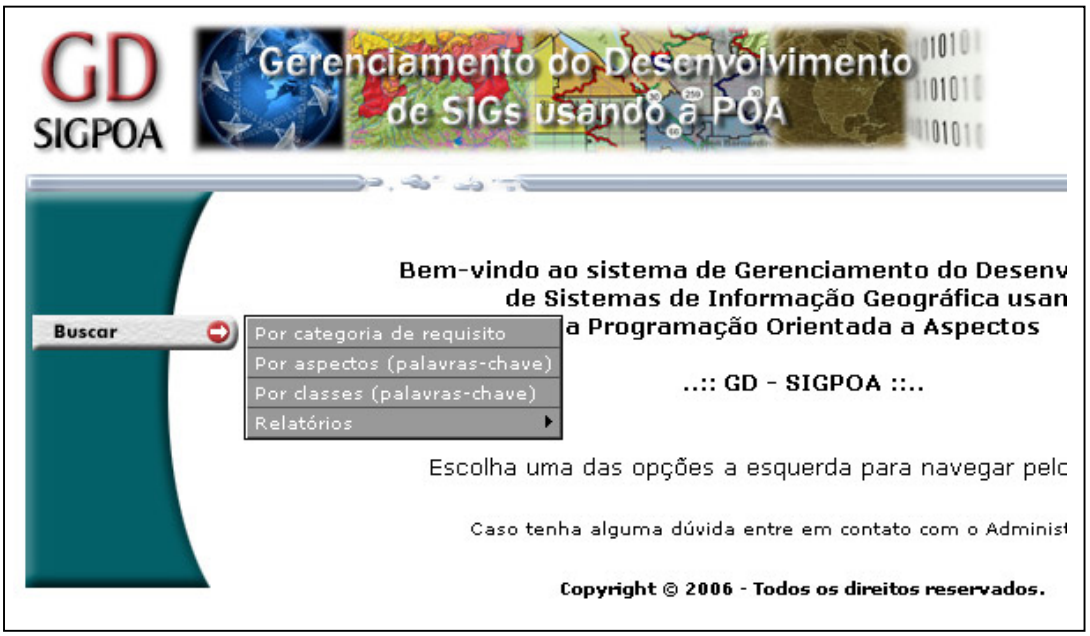


Figura D.5. Tela do sistema no perfil de acesso de Visitante

D.4 – RELATÓRIOS

O sistema possui quatro tipos diferentes de relatórios: detalhamento do projeto, requisitos, aspectos e classes, todos relativos a um dos projetos de SIG cadastrados na base de dados do GD-SIGPOA.

As figuras a seguir representam os relatórios de detalhamento do projeto e de aspectos, ambos referentes ao exemplo SIG CEDAE.

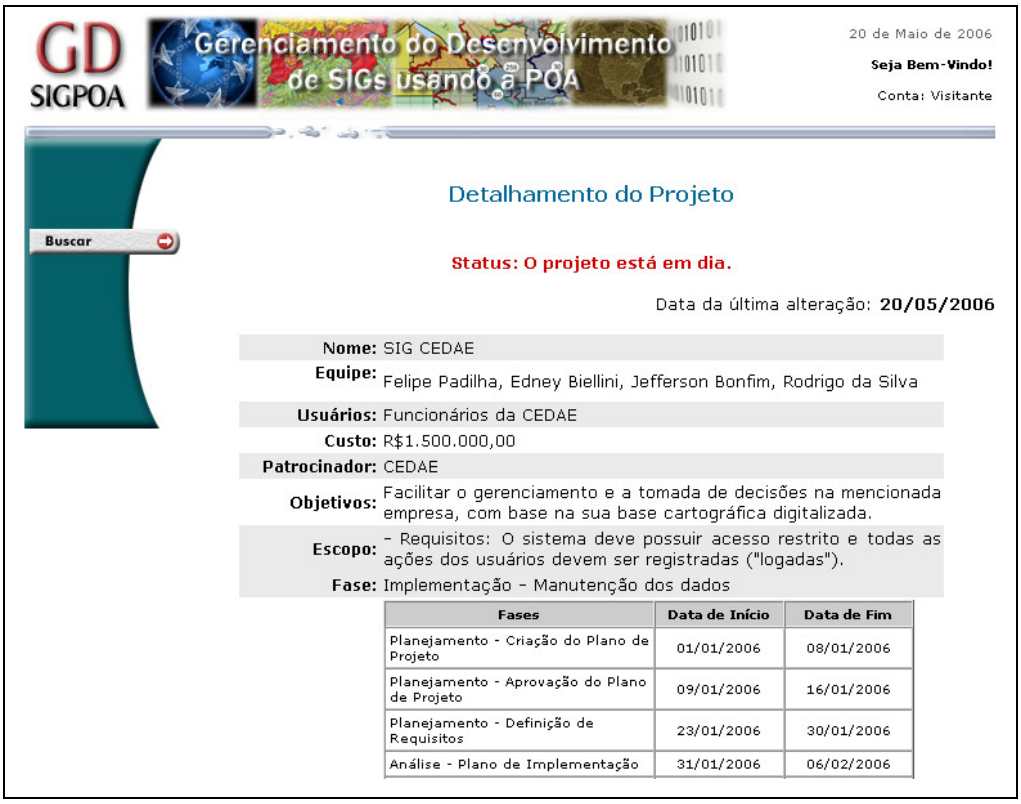


Figura D.6. Relatório de detalhamento de projeto

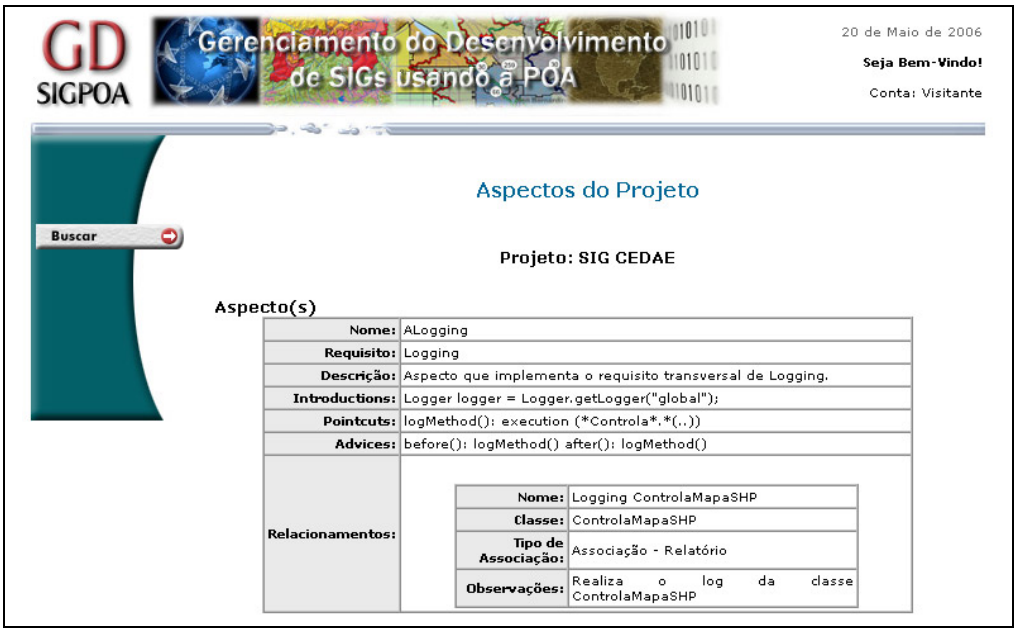


Figura D.7. Relatório de aspectos do SIG