Programação Orientada a Aspectos Aplicada à Criação de Sistemas de Informações Geográficas

Heitor Roméro Cajaty[1], Orlando Bernardo Filho[1], Neide Santos[1,2]

[1]UERJ — Pós-Graduação em Engenharia de Computação — Área de Concentração Geomática,
Departamento de Engª. de Sistemas e Computação
Rua São Francisco Xavier no. 524, bloco D, sala 5015
Maracanã, Rio de Janeiro, RJ, Cep. 20550-013

[1,2] UERJ — Departamento de Informática e Ciência da Computação

heitor.romero@cajaty.com, orlando@eng.uerj.br, neide@ime.uerj.br

Abstract. *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, introducing the aspects concept, which are properties that affects performance or components semantic, exceptions treatment, data consistency, security, etc. The present paper aims to provide the study of aspect oriented programming to develop geographical information systems. This new model of implementation will make easier to maintain and continue developing the software, minimizing the costs of its projects.*

1. Introdução

Normalmente, antes de iniciarmos o desenvolvimento de um sistema, na fase de análise de requisitos, definimos, 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 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 informações geográficas 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, tratamento de exceções, consistência de dados, segurança, etc.

Neste trabalho, de início, discutimos os sistemas de informação geográfica e comparamos a programação orientada a objetos e a programação orientada a aspectos. A seguir apresentamos uma proposta de sistema *web* voltado para o gerenciamento de SIGs, a ser desenvolvido seguindo o paradigma orientado a aspectos.

2. Revisão Bibliográfica

2.1 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. Dados geográficos são coletados a partir de diversas fontes e armazenados, via de regra, nos chamados bancos de dados geográficos.

Devido à sua ampla gama de aplicações, há diferentes formas de se caracterizar SIGs [MGR93, COW90, ARO89, SMSE87, BUR86]. 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.

Assim, numa 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 1, extraída de [CAM96], 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.

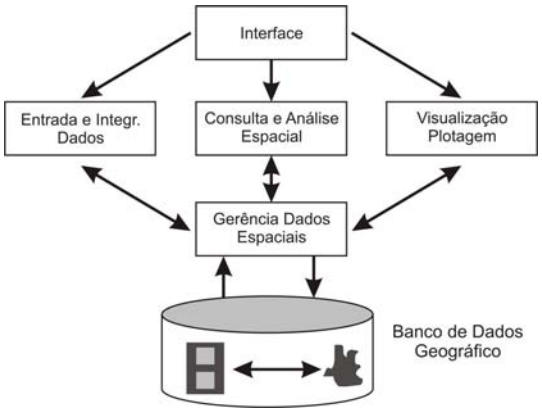

Figura 1. Arquitetura de SIGs

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.

Os dados de um SIG são geralmente organizados sob a forma de um banco 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.

2.1.1 Implatação 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.

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.

Banco de Dados Geográficos

Um banco de dados geográficos é um repositório da informação coletada empiricamente sobre os fenômenos do mundo real [EA92, EGE95]. 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.

Operação

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

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

2.2 - 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 existem duas categorias de classificação para os requisitos do sistema: requisito principal (*core concern*) e requisito transversal (*crosscutting concern*) [LAD03]. 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 informações geográficas de uma empresa de grande porte, por exemplo, deve se preocupar com diversos requisitos transversais, quais sejam a autenticação, *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 a *log* será afetado por esse requisito. A figura 2 demonstra os requisitos transversais de *logging* e persistência presentes em vários módulos do sistema.

Como se pode notar da figura 2, 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 *tangled code*.

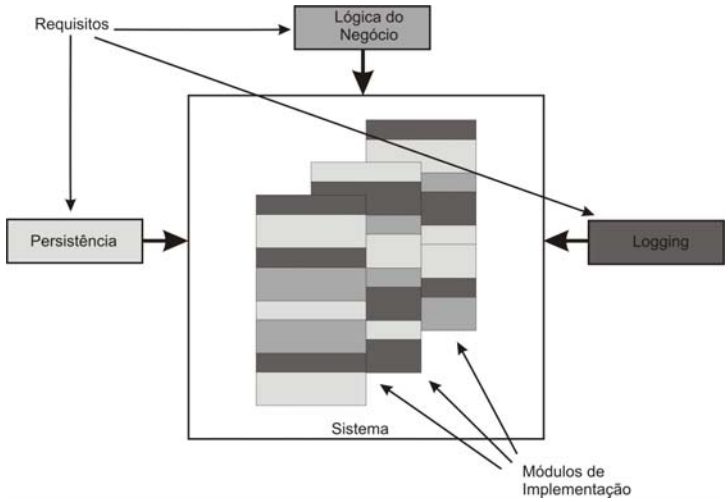

Figura 2. Requisitos Transversais no Sistema

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

2.3 Programação Orientada a Aspectos

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 (*crosscutting concerns*) [KIC97].

Os exemplos mais comuns de requisitos transversais estão ligados diretamente à segurança dos sistemas, como por exemplo, as funcionalidades de logs, autenticação, 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 de código (*tangled code*) e dispersão de código (*code scattering*). 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.

Assim, este artigo tem por objetivo propor a utilização da programação orientada a aspectos em um sistema de informações geográficas, fornecendo um modelo UML de uma aplicação *web* para acompanhar e documentar o desenvolvimento de um SIG.

3. Estudo de Caso

Visando demonstrar as diferenças e os benefícios da criação de SIGs utilizando a programação orientada a aspectos, desenvolvemos um exemplo, como estudo de caso hipotético, 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, visando inserir esses dados em um sistema de informações geográficas para facilitar o gerenciamento e a tomada de decisões na empresa.

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

1. permitir a visualização de áreas da cidade do Rio de Janeiro, bem como das informações associadas a cada um dos objetos constantes nos mapas;
2. a inserção de informações no banco de dados deve permitir associá-la a sua posição exata, isto é, georreferenciada;
3. permitir a edição dos mapas, além de possuir a funcionalidade de zoom e visualização por camadas/temas;
4. ter diferentes níveis de acesso, de acordo com os cargos da companhia, através de contas de usuário distintas;
5. documentar, em seu *log*, todas as ações executadas pelos usuários.

3.1. Programação orientada a aspectos aplicada à criação de SIGs

A aplicação das técnicas de orientação a objetos no domínio dos SIG foi inicialmente proposta por (Egenhofer e Frank, 1992) e subseqüentemente serviu como base para o desenvolvimento de estruturas de dados e modelos de representação. Essa abordagem, tradicionalmente, leva à criação de classes diretamente relacionadas aos diferentes tipos de geometrias. Constata-se, no entanto, uma desvantagem da aplicação direta da abordagem orientada a objetos no desenvolvimento de SIG: o forte acoplamento entre algoritmos e estruturas de dados.

Baseando-se nos requisitos acima especificados e naqueles inerentes a um sistema de informações geográficas, notamos, a princípio, a presença de três requisitos transversais: *logging*, segurança e o forte acoplamento entre os algoritmos e as estruturas de dados. Nesse artigo abordaremos somente o requisito de *logging*.

3.1.2 Implementando o *logging* com o uso de AspectJ

O AspectJ é uma extensão da linguagem de programação Java que permite a implementação dos aspectos, possuindo os seguintes conceitos: Pontos de junção (*Join points*), Pontos de corte (*Pointcuts*), Regras (*Advices*) e Aspectos.

O ponto de junção é um ponto bem definido no fluxo de execução de um programa orientado a objetos, como por exemplo, a chamada de métodos de tratamento de exceções, construtores, entre outros. Além disso, por ser o ponto de junção um lugar bem definido no programa, torna-se necessário que exista algum modo de agrupar determinados pontos ou de determinar quando o ponto de junção será considerado. Esse momento pode ser, por exemplo, em uma chamada na execução ou no lançamento de exceções. Tal mecanismo recebe o nome de ponto-de-corte (*pointcut*).

As regras determinam quando e quais códigos serão executados quando um ponto de corte for válido. Na programação orientada a aspectos esta regra pode ser executada em três momentos distintos: antes, durante e depois da expressão que ativou o ponto de corte.

O componente responsável pelo encapsulamento do código referente a determinado requisito é o próprio aspecto, que pode utilizar a própria linguagem base para codificar a implementação do requisito transversal da aplicação. Por fim, após codificar os requisitos transversais em aspectos, determinar os pontos de junção e de corte e codificar as regras, é necessário que uma ferramenta integre os aspectos (requisitos transversais) com os componentes da linguagem base (requisitos primários). Essa ferramenta é denominada *weaver* e o processo de integração é chamado *weaving*.

Seu funcionamento é simples: na entrada colocam-se os aspectos e os componentes da linguagem base e, seu produto é o sistema propriamente dito com todas as suas características funcionais e transversais devidamente implementadas. Geralmente, nos sistemas que utilizam

programação orientada por objetos, as chamadas ao *log* ficam espalhadas nos métodos, como abaixo:

```
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 3. Classe ControlaMapaSHP

O exemplo acima mostra apenas a classe controladora dos mapas, entretanto o entrelaçamento do código repete-se também nas classes de edição do mapa, bem como nas de identificação do usuário, por exemplo.

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 abaixo:

```
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. 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 5. Aspecto ALogging

As figuras acima mostram a separação do código em unidades diferentes. Para implementar a funcionalidade de *logging* foi criado o aspecto *ALogging*, o qual 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 capturado quando forem executados os métodos das

classes controladoras, isto é, todas que comecem com a *string Controla*. As regras *before* e *after* determinam os códigos que devem ser executados antes e depois de qualquer método capturado pelo ponto de corte *logMethod*. O resultado é exatamente o mesmo do apresentado na figura 3, 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 aumenta sua performance.

4. Sistema web para acompanhamento do desenvolvimento de SIGs utilizando a programação orientada a aspectos

Como visto, o desenvolvimento de sistemas de informações utilizando a programação orientada por aspectos traz inúmeros benefícios, tanto com relação à organização de seu código, como quanto à sua performance. Visando auxiliar o desenvolvimento e a posterior manutenção de SIGs, estamos desenvolvendo um sistema para a Web que acompanha e gerencia o desenvolvimento destes sistemas, documentando os aspectos criados durante a implementação do SIG.

4.1. Objetivo e requisitos do sistema web

O principal objetivo do sistema proposto é acompanhar e documentar o processo de desenvolvimento de um SIG utilizando a orientação a aspectos. Para tanto, cada projeto de SIG será cadastrado na aplicação, que contará com uma base de dados *MySQL*, e ficará hospedada em um servidor *web* que rode a linguagem *JSP – Java Server Pages*, utilizada no desenvolvimento do sistema.

Há três perfis de acesso no sistema: o administrador, o gerente de projetos e os analistas.

O administrador é o responsável por cadastrar, alterar e excluir os gerentes de projetos de SIGs. Além disso, pode visualizar e analisar os relatórios de utilização da aplicação.

Os gerentes de projetos devem inserir, atualizar e excluir, na aplicação, seus projetos, bem como todos os dados referentes a eles, entre eles o cronograma, a equipe responsável pelo desenvolvimento, o cliente, o custo, seus requisitos e suas restrições. São também os responsáveis por cadastrar, alterar e excluir os analistas que trabalharão em seus projetos, podendo visualizar todas as informações incluídas por estes.

Os analistas são os responsáveis por incluir e atualizar toda a documentação relacionada aos aspectos e componentes do sistema, e bem assim, os pontos de corte e pontos de junção, além de poderem visualizar todas as informações relacionadas aos projetos em que estão trabalhando.

Modelamos, utilizando UML, alguns casos de uso – a interação dos três tipos de usuários com o sistema.

4.2 Modelo UML de Casos de Uso

Um caso de uso descreve um objetivo que um ator externo ao sistema tem com o sistema. Um ator pode ser um elemento humano ou não que interage com o sistema. O ator se encontra fora do escopo de atuação do sistema, enquanto que o conjunto de casos de uso forma o referido escopo.

A linha que separa os atores dos casos de uso é a fronteira do sistema. O diagrama de casos de uso representa graficamente esta interação, definindo o contexto do sistema. Os atores se comunicam com os casos de uso, o que é representado por uma linha unindo os dois elementos. Uma seta pode, opcionalmente, representar o fluxo principal de informação nesta interação e ajudar a leitura do caso de uso.

Conforme visto acima, o sistema proposto contará com três atores distintos: o administrador, o gerente e o analista. Na figura 6, se encontra um diagrama de casos de uso através do qual se descreve os objetivos de cada um dos atores externos ao sistema.

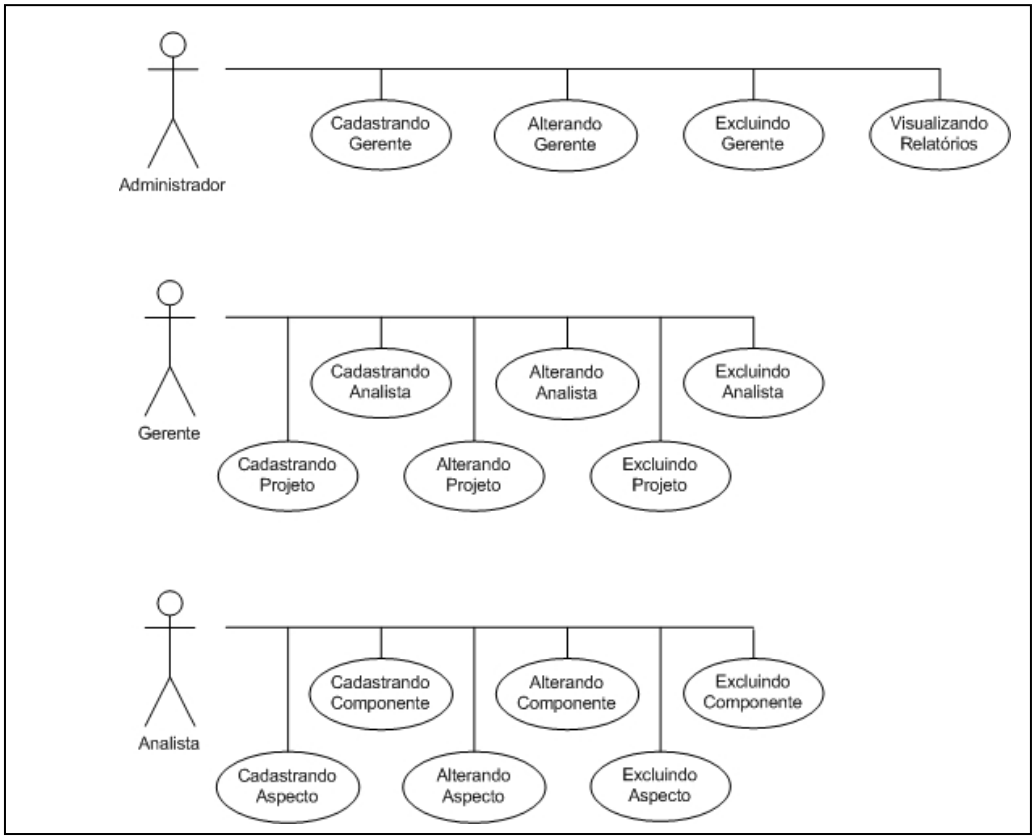

Figura 6. Diagrama UML de Casos de Uso

4.3. Próximos Passos

Além dos perfis de acesso ao sistema mencionados acima, desenvolveremos mecanismos que irão auxiliar cada um destes atores externos, lembrando-lhes de cada uma das tarefas inerentes a documentação do sistema de informações geográficas utilizando linguagem de programação orientada a aspectos.

5. Conclusões

A utilização da programação orientada a objetos no desenvolvimento de sistemas de informações geográficas tem sido dominante desde 1992. Entretanto, os SIGs possuem diversos requisitos transversais, trazendo problemas de entrelaçamento e dispersão do código. Logo, é necessária a adoção de novas técnicas de programação que permitam maior modularização do sistema, na qual cada uma das unidades englobe somente os requisitos para os quais foram realmente projetadas.

A programação orientada por aspectos melhora a modularização do sistema, facilitando sua manutenção, aprimoramento e desempenho.

Neste artigo abordamos um dos requisitos transversais de um SIG, o *logging*, apresentando como exemplos, a programação orientada a objeto e a orientada a aspectos. As figuras 3, 4 e 5 demonstram claramente as diferenças de implementação.

Idealizamos um sistema *web* para o acompanhamento e documentação do desenvolvimento de SIGs utilizando a programação orientada a aspectos.

O sistema encontra-se em estágio inicial de desenvolvimento, mas é de extrema importância, pois irá auxiliar a documentação do processo de desenvolvimento de SIGs, além de ser um repositório de conhecimento e código, que podem ser utilizados posteriormente.

Por fim, podemos concluir que a programação orientada a aspectos aplicada a SIGs pode levar, realmente, a um sistema mais modularizado, proporcionando maior facilidade na manutenção e no entendimento do código. Nossos próximos passos direcionam-se para a conclusão da modelagem do sistema, sua implementação e testes.

Referências Bibliográficas

[ARO89] S. Aronoff. Geographic Information Systems. WDL Publications, Canada, 1989.

[BUR86] P. Burrough. Principles of Geographical Information Systems for Land Resources Assessment. Clarendon Press, 1986.

[CAM95] G. Camara. Modelos, Linguagens e Arquiteturas para Bancos de Dados Geográficos. PhD thesis, INPE, dezembro 1995.

[CAM96] G. Camara., M. Casanova, A. Hemerly, G. Magalhães, C. Medeiros, Anatomia de Sistemas de Informação Geográfica, 1996.

[COW90] D. Cowen. GIS versus CAD versus DBMS: what are the Differences? In Introductory Readings in Geographical Information Systems, pages 52-61. Taylor and Francis, 1990.

[EA92] M. Goodchild et al. Integrating GIS and Spatial Data Analysis: Problems and Possibilities. International Journal of Geographical Information Systems, (5):407-424, 1992.

[EGE92] M. Egenhofer, A. Frank. Object-Oriented Modeling for GIS. URISA Journal, v.4 n. 2 p. 3-19, 1992.

[EGE95] M. Egenhofer. Object-oriented GISs: The Principles. Technical report, NCGIA, fevereiro 1995.

[FRU04] E. Frutoso; M. Valente. Implementação de um sistema web usando programação orientada por aspectos In: I Simpósio Mineiro de Sistemas de Informação, MG, 2004, p.94-105

[KIC97] G. Kiczales. Aspect Oriented-Programming In ECOOP, Spring-Verlag 1997

[KIC01] G. Kiczales. An Overview of AspectJ. In ECOOP, Spring-Verlag 2001

[LAD03] R. Laddad. AspectJ in Action. Manning Publications, 1ed. 2003.

[MGR93] D. Maguire, M. Goodchild, and D. Rhind, editors. Geographical Information Systems - volume I. John Wiley and Sons, 2 edition, 1993.

[SMSE87] T. Smith, S. Menon, J. Star, and J. Estes. Requirements and Principles for the Implementation and Construction of Large-scale Geographic Information Systems. International Journal of Geographical Information Systems, 1(1):13-31, 1987.

[VIN02] L. Vinhas; G. Queiroz; K. Ferreira; G. Câmara; J. Paiva. Programação Genérica Aplicada a Algoritmos Geográficos. In: IV Simpósio Brasileiro de GeoInformática, Anais. Caxambu, MG, 2002. v.1, p.117-122.