

Universidade Federal do Pampa

Ricardo Burg Machado

# **Uma Arquitetura Web para Sistemas de Informações Geográficas aplicada a Geotecnia**

Alegrete

2015



Ricardo Burg Machado

## **Uma Arquitetura Web para Sistemas de Informações Geográficas aplicada a Geotecnia**

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Engenharia de Software.

Orientador: Prof<sup>o</sup>. Me. João Pablo Silva da Silva

Coorientador: Prof<sup>o</sup>. Me. Magnos Baroni

Alegrete

2015



Ricardo Burg Machado

## Uma Arquitetura Web para Sistemas de Informações Geográficas aplicada a Geotecnia

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Engenharia de Software.


Trabalho de Conclusão de Curso defendido e aprovado em 03 de Dezembro de 2015.

Banca examinadora:



---

Prof<sup>o</sup>. Me. João Pablo Silva da Silva  
Orientador



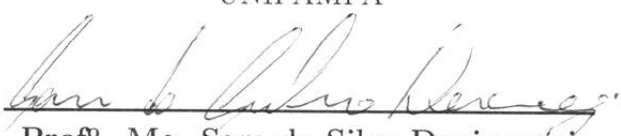
---

Prof<sup>o</sup>. Me. Magnos Baroni  
Co-orientador  
UFSM



---

Prof<sup>o</sup>. Me. Jean Felipe Patikowski  
Cheiran  
UNIPAMPA



---

Prof<sup>o</sup>. Me. Sam da Silva Devincenzi  
UNIPAMPA



# Agradecimentos

Agradeço primeiramente a minha família, que me apoia em todos os momentos da vida, me trazendo conforto e força.

Ao meu orientador João Pablo Silva da Silva e co-orientador Magnos Baroni, pela parceria durante esse trabalho.

Agradeço a dedicação e profissionalismo dos professores que me acompanharam ao decorrer de minha graduação, compartilhando seu tempo e conhecimento.

Por fim, agradeço a Universidade Federal do Pampa (UNIPAMPA) pela oportunidade de um ensino de qualidade de forma gratuita.

Obrigado à todos!





# Resumo

Os Sistemas de Informações Geográficas Web Geotécnicas constituem uma classe de aplicação e têm como seus principais aspectos a manipulação de dados de georeferenciamento, o gerenciamento de registros e correlações de informações geotécnicas. Pelo fato de não ser usada corriqueiramente uma arquitetura padrão para o desenvolvimento dessa classe de aplicação, este trabalho tem como motivação explorar a oportunidade de definir um modelo arquitetural que consiga viabilizar estruturalmente a construção dessas aplicações. O objetivo desse trabalho é apresentar um modelo de arquitetura para a construção de Sistemas de Informações Geográficas Web Geotécnicas. A metodologia utilizada neste trabalho consiste em um estudo aprofundado sobre Sistemas de Informações Geográficas e Arquitetura de Software. Através da identificação de responsabilidades foram definidos os elementos da arquitetura e seus relacionamentos. Para a validação, foi realizada a instanciação da arquitetura através de um estudo de caso. O resultado final obtido foi uma arquitetura composta por cinco módulos e criada a partir da modelagem de domínio e a integração de dois *frameworks*. Conclui-se que a partir da utilização desse modelo arquitetural, foi possível obter vários benefícios como o encapsulamento de informações feito para entrada de dados no sistema, a conversão automática de dados e a persistência automatizada que trabalha com a conversão reversível de dados.

**Palavras-chave:** Arquitetura de Software. Sistemas de Informações Geográficas. Aplicação Web. Engenharia Geotécnica. Mapeamento Objeto/Relacional.



# Abstract

The Geographic Information Systems Geotechnical Web are an application class and have as their main aspects the manipulation of georeferencing data and the management of records and correlations of geotechnical information. Because not be used routinely a standard architecture for the development of this class of application, this work has as motivation to explore the opportunity of defining an architectural model that is able structurally enable the construction of these applications. The objective of the present work is to propose a model of architecture which permits the building of Geographic Information Systems Geotechnical Web. The methodology used in this work consists on a deep study about Geographic Information Systems and Software Architecture. By identifying responsibilities the architectural elements and their relationships were defined. For validation, the instantiation of architecture was carried out through a case study. The final result was an architecture comprised of five modules and created from the domain modeling and the integration of two frameworks. It's concluded that from the use of this architectural model, lots of benefits were gotten such as the information encapsulation done to the data entrance in the system, the automatic conversion of data and the automated persistence that works with the data reversible conversion.

**Key-words:** Software Architecture. Geographic Information Systems. Web Application. Geotechnical Engineering. Object/Relational Mapping.



# Lista de ilustrações

Figura 1 – Estrutura de um Sistema de Informações Geográficas (SIG). . . . .	32
Figura 2 – Contexto de uma Aplicação <i>web</i> . . . . .	32
Figura 3 – Módulos do <i>Spring Framework</i> . . . . .	34
Figura 4 – Padrão <i>Front Controller</i> no contexto do <i>Spring MVC</i> . . . . .	35
Figura 5 – Estrutura do <i>Spring MVC</i> . . . . .	36
Figura 6 – Arquitetura do <i>Hibernate</i> de forma resumida. . . . .	38
Figura 7 – Elementos da arquitetura do <i>Hibernate</i> . . . . .	38
Figura 8 – Arquitetura proposta para um SIG móvel sensível ao contexto. . . . .	42
Figura 9 – Proposta da arquitetura em camadas feita por Souto (2012). . . . .	43
Figura 10 – Proposta da arquitetura feita por Silva (2012). . . . .	44
Figura 11 – Arquitetura Arquitetura Orientada a Serviços (SOA) aplicada a dados geográficos. . . . .	44
Figura 12 – Arquitetura utilizada por Cunha et al. (2009). . . . .	46
Figura 13 – Arquitetura utilizada por Veiga (2012). . . . .	47
Figura 14 – Arquitetura em camadas da ferramenta WITS. . . . .	47
Figura 15 – Visão geral da arquitetura de software para a infra estrutura de serviços para a modelagem de nicho ecológico. . . . .	48
Figura 16 – Modelo da estrutura física feita por Chen et al. (2012). . . . .	48
Figura 17 – Arquitetura utilizada por Shen et al. (2010). . . . .	49
Figura 18 – Arquitetura utilizada por Graettinger, Ryals e Smith (2011). . . . .	50
Figura 19 – Arquitetura utilizada por Sun (2012). . . . .	51
Figura 20 – Arquitetura utilizada por Wang et al. (2014). . . . .	51
Figura 21 – Arquitetura organizada em módulos. . . . .	54
Figura 22 – Módulo Core Container. . . . .	55
Figura 23 – Módulo Modelo. . . . .	56
Figura 24 – Modelo - Gerenciamento de Arquivos. . . . .	57
Figura 25 – Modelo - Gerenciamento de Expressões. . . . .	57
Figura 26 – Modelo - Gerenciamento de Dados Geotécnicos. . . . .	58
Figura 27 – Modelo - Mapas. . . . .	59
Figura 28 – Modelo - Exceções . . . . .	59
Figura 29 – Modelo - Gerenciamento de Usuários. . . . .	59
Figura 30 – Módulo Controle. . . . .	60
Figura 31 – Módulo Persistência. . . . .	61
Figura 32 – Principais atividades dentro do contexto do sistema. . . . .	65
Figura 33 – Diagrama de Casos de Uso. . . . .	67
Figura 34 – Diagrama de classes - Módulo Gerenciamento de Arquivos. . . . .	69

Figura 35 – Diagrama de classes - Módulo Gerenciamento de Dados Geotécnicos. . . . .	70
Figura 36 – Diagrama de classes - Módulo Gerenciamento de Expressões (Parte 01). . . . .	70
Figura 37 – Diagrama de classes - Módulo Gerenciamento de Expressões (Parte 02). . . . .	71
Figura 38 – Diagrama de classes - Módulo Gerenciamento de Usuários. . . . .	71
Figura 39 – Diagrama Entidade Relacionamento (DER) utilizado no projeto do sistema. . . . .	72
Figura 40 – Diagrama de classes - Módulo Mapas. . . . .	73
Figura 41 – Diagrama de classes - Módulo Controle (Parte 01). . . . .	73
Figura 42 – Diagrama de classes - Módulo Controle (Parte 02). . . . .	74
Figura 43 – Diagrama de classes - Módulo Persistência (Parte 01). . . . .	75
Figura 44 – Diagrama de classes - Módulo Persistência (Parte 02). . . . .	76
Figura 45 – Tela de visualização do mapa no sistema. . . . .	78
Figura 46 – Tela de cadastro de Parâmetro de Plotagem no sistema. . . . .	78
Figura 47 – Tela de seleção de parâmetros para plotar o gráfico no sistema. . . . .	79
Figura 48 – Tela de visualização de gráfico no sistema. . . . .	79
Figura 49 – Circuito de atividades do protocolo de testes. . . . .	82
Figura 50 – Diagrama de Sequência - Visualizar Mapa. . . . .	85
Figura 51 – Diagrama de Sequência - Cadastrar Ilha. . . . .	106
Figura 52 – Diagrama de Sequência - Importar Arquivo. . . . .	107
Figura 53 – Diagrama de Sequência - Manter Fator de Ajuste. . . . .	108
Figura 54 – Diagrama de Sequência - Manter Parâmetro de Plotagem. . . . .	109
Figura 55 – Diagrama de Sequência - Template Padrão. . . . .	110
Figura 56 – Ensaio organizado em uma planilha a ser carregada e interpretada pelo sistema. . . . .	113

# Lista de tabelas

Tabela 1 – Caso de Uso Visualizar Mapa . . . . .	67
Tabela 2 – Caso de Uso Cadastrar Ilha . . . . .	68
Tabela 3 – Particionamento por Equivalência . . . . .	80
Tabela 4 – Resultados obtidos através do experimento . . . . .	83





# Lista de siglas

- ADL** *Architecture Description Language*
- AJAX** *Asynchronous Javascript and XML*
- AOP** *Aspect Oriented Programming*
- API** *Application Programming Interface*
- BD** Banco de Dados
- CAD** *Computer Aided design*
- CORBA** *Common Object Request Broker Architecture*
- DER** Diagrama Entidade Relacionamento
- DESO** Companhia de Saneamento do Sergipe
- DSSA** Arquitetura de Software para um Domínio Específico
- EL** *Expression Language*
- GPS** *Global Positioning System*
- HQL** *Hibernate Query Language*
- HTML** Linguagem de Marcação de Hipertexto
- HTTP** *Hypertext Transfer Protocol*
- IoC** *Inversion of Control*
- JDBC** *Java Database Connectivity*
- JDO** *Java Data Objects*
- JMS** *Java Message Service*
- JMX** *Java Management Extensions*
- JPA** *Java Persistence API*
- JSF** *Java Server Faces*
- JSON** *JavaScript Object Notation*
- JSP** *Java Server Pages*

**JSTL** *JavaServer Pages Standard Tag Library*

**MVC** *Model View Control*

**OGC** *Open Geospatial Consortium*

**OO** *Orientação a Objetos*

**ORM** *Object-relational Mapping*

**OXM** *Object XML Mapping*

**POJO** *Plain Old Java Object*

**REST** *REpresentational State Transfer*

**RUP** *Rational Unified Process*

**SAAM** *Software Architecture Analysis Method*

**SADE** *Atendimento de Despacho de Emergências em Santa Catarina*

**SAPPGAM** *Sistema para Análise e Processamento de Parâmetros Geotécnicos das Argilas Moles Brasileiras*

**SGBD** *Sistema de Gerenciamento de Banco de Dados*

**SIG** *Sistema de Informações Geográficas*

**SOA** *Arquitetura Orientada a Serviços*

**SQL** *Structured Query Language*

**UML** *Unified Modeling Language*

**URL** *Uniform Resource Locator*

**WFS** *Web Feature Service*

**WMS** *Web Map Services*

**XML** *eXtensible Markup Language*

# Sumário

<b>1</b>	<b>INTRODUÇÃO</b>	<b>19</b>
1.1	Motivação	20
1.2	Objetivos	21
1.3	Metodologia	21
1.4	Contribuições	21
1.5	Organização do Texto	22
<b>2</b>	<b>FUNDAMENTAÇÃO TEÓRICA E TECNOLÓGICA</b>	<b>23</b>
2.1	Arquiteturas de Software	23
2.1.1	Visão Arquitetural	24
2.1.2	Estilos e Padrões Arquiteturais	26
2.1.2.1	Padrões	26
2.1.2.2	Estilos	28
2.1.3	Arquitetura de Software para um Domínio Específico	29
2.2	Sistemas de Informações Geográficas	29
2.2.1	Aplicações de SIGs	30
2.2.2	Tecnologias para SIG	31
2.2.3	Estrutura Geral de um SIG	31
2.3	Desenvolvimento de Sistemas Web	32
2.3.1	Spring Framework	33
2.3.2	Hibernate	36
2.4	Engenharia Geotécnica	39
2.5	Lições do Capítulo	39
<b>3</b>	<b>TRABALHOS RELACIONADOS</b>	<b>41</b>
3.1	Protocolo de Busca	41
3.2	Trabalhos Selecionados	41
3.2.1	SIGs para Diversas Áreas de Aplicação	41
3.2.2	SIGs para Engenharia Geotécnica	50
3.3	Lições do capítulo	52
<b>4</b>	<b>ARQUITETURA WEB PARA SIG</b>	<b>53</b>
4.1	Visão Geral	53
4.2	Modelo Arquitetural	54
4.2.1	Módulo <i>Core Container</i>	55
4.2.2	Módulo Visualização	55

4.2.3	Módulo Modelo . . . . .	55
4.2.4	Controle . . . . .	60
4.2.5	Persistência . . . . .	61
<b>4.3</b>	<b>Implementação do Modelo . . . . .</b>	<b>62</b>
<b>4.4</b>	<b>Lições do Capítulo . . . . .</b>	<b>64</b>
<b>5</b>	<b>ESTUDO DE CASO . . . . .</b>	<b>65</b>
<b>5.1</b>	<b>Visão Geral . . . . .</b>	<b>65</b>
<b>5.2</b>	<b>Análise e Projeto . . . . .</b>	<b>66</b>
<b>5.3</b>	<b>Implementação . . . . .</b>	<b>75</b>
<b>5.4</b>	<b>Verificação e Validação . . . . .</b>	<b>79</b>
5.4.1	Testes . . . . .	79
5.4.2	Experimento . . . . .	81
5.4.2.1	Protocolo de Testes . . . . .	81
5.4.2.2	Análise dos Resultados . . . . .	82
<b>5.5</b>	<b>Lições do Capítulo . . . . .</b>	<b>83</b>
<b>6</b>	<b>CONCLUSÕES . . . . .</b>	<b>87</b>
<b>6.1</b>	<b>Trabalhos Futuros . . . . .</b>	<b>88</b>
	<b>REFERÊNCIAS . . . . .</b>	<b>89</b>
	<b>Índice . . . . .</b>	<b>95</b>
	<b>APÊNDICES . . . . .</b>	<b>97</b>
	<b>APÊNDICE A – CASOS DE USO EXPANDIDOS . . . . .</b>	<b>99</b>
	<b>APÊNDICE B – DIAGRAMAS DE SEQUÊNCIA . . . . .</b>	<b>105</b>
	<b>ANEXOS . . . . .</b>	<b>111</b>
	<b>ANEXO A – TEMPLATE PADRÃO DE RESULTADOS . . . . .</b>	<b>113</b>

# 1 Introdução

Uma arquitetura de software é a estrutura base para a construção de produtos de software. Bass, Clements e Kazman (2003) definem arquitetura de software como:

A estrutura de um sistema de software ou sistema de computação que é a estrutura ou estruturas de um sistema, que incluem elementos de software, as propriedades externamente visíveis desses elementos, e as relações entre eles.

A partir dessa definição, se entende que essas estruturas estão contidas dentro de outras estruturas que juntas formam o sistema. Para que ocorra a inserção de comportamentos e serviços nos elementos que compõem uma arquitetura, são impostas restrições, regras e diretrizes que vem a definir a forma de como ocorre a comunicação entre os elementos. Isso é um dos pontos chave para caracterização da arquitetura. Uma arquitetura tem sua concepção através de um conjunto de decisões que definem sua estrutura e seu comportamento, que impacta em suas colaborações (LARMAN, 2007).

Essas decisões são realizadas de diversas maneiras, isso porque uma decisão pode envolver diversas questões neste contexto. Em primeiro lugar, no ponto de vista de Engenharia de Software, uma arquitetura deve atender indiretamente requisitos funcionais, mas de uma maneira mais direta, devem ser identificados e atendidos os requisitos não funcionais. Para isso, existem os padrões, estilos, princípios e técnicas, que dão suporte para a concepção de uma arquitetura.

Com uma definição que consegue ressaltar a importância da arquitetura para um sistema, o padrão ISO/IEEE.1471-2000 (2007) define a arquitetura como uma “organização fundamental de um sistema, representada por seus componentes, seus relacionamentos com o ambiente, e pelos princípios que conduzem seu *design* e evolução”. Esse padrão além de evidenciar a importância da arquitetura e ser consistente com as demais definições, também enfatiza que a arquitetura proporciona vantagens que vão além do estágio de implantação ou entrega do produto, mas também em sua evolução.

Os SIGs são aplicações utilizadas nos mais diversos segmentos da sociedade, como por exemplo, empresas, instituições de pesquisas e órgãos do governo. Para esses segmentos essas aplicações conseguem prover soluções para análise de recursos ambientais, planejamento de infraestrutura para o estado, planejamento do uso solo (uso da terra) entre outras. Mas também atacar problemas como controle de queimadas, gerenciamento de desmatamento, planejamento e gerenciamento urbano, transporte etc. Um Sistema de Informações Geo-referenciadas ou SIG, é definido por Aida e Osumi (1988) como:

Um sistema que contém dados espacialmente referenciados que podem ser analisados e convertidos em informações para uso em um conjunto específico de finalidades.

De um modo geral, SIGs podem ser consideradas em muitos casos ferramentas, que possibilitam seus usuários ou operadores realizar análises diagnósticas ou preventivas dentro dos mais variados contextos alinhados a manipulação de informações georeferenciadas. Dessa maneira, essas soluções através de seus mecanismos, dispõe de diferentes modos de apresentar informações georeferenciadas através de mapas com demarcação de áreas, rotas, identificação de lugares e eventos, comparação e armazenamento de informações. Além do mais, os SIGs conseguem trabalhar de forma distribuída, tanto para receber como para prover recursos de forma integrada. Os recursos podem ser vistos como infinitas possibilidades, pois podem representar uma funcionalidade completa, como por exemplo a construção de um mapa; como podem representar uma parte, através do envio de informação georeferenciada (latitude/longitude) para promover a renderização de mapas em uma segunda aplicação SIG.

## 1.1 Motivação

O desenvolvimento de SIG Web Geotécnico requer uma arquitetura que consiga trabalhar e se adequar a todas as possibilidades, que não são poucas dessa classe de aplicação. Primeiramente a forma de comunicação como ocorre a transição de dados internamente ou externamente à aplicação. Internamente, pois deve haver uma homogeneidade dos dados ao circular dentro da arquitetura entre suas partes. A partir disso, externamente esses dados podem entrar ou sair da aplicação, e por isso devem seguir padrões de tipos para possibilitar as devidas integrações. Essa demanda existe pois artefatos como gráficos, arquivos, imagens, mapas são construídos em diversas perspectivas, como: bibliotecas, API, extensões de banco de dados, e isso implica que esses dados devem seguir padrões universais.

Além do mais, esta classe de aplicação deve ter um potencial de armazenamento que consiga se diferenciar de uma aplicação normal, pois precisa estar preparada para responder a transações de persistência de grandes massas de dados de forma eficiente. Caso essa aplicação não consiga responder a essas transações, deve possuir meios de configurar algum mecanismo que amenize essas questões. Num panorama geral, essa classe de aplicação requer uma estrutura que comporte adaptar, flexibilizar, desacoplar, armazenar e diminuir a complexidade de uma maneira estratégica.

A motivação desse trabalho está no estudo e criação de um modelo arquitetural que esteja alinhado aos desafios que se tem no desenvolvimento dessa classe de aplicação. Com a obtenção de ganhos como flexibilidade, adaptabilidade, baixa complexidade, persistência eficiente, o desenvolvimento da aplicação tende a ter impactos ainda mais positivos como a economia de recursos (humanos e financeiros), ganho de tempo de entrega do produto final e qualidade.

## 1.2 Objetivos

Este trabalho tem como objetivo geral a construção de um modelo de arquitetura de software web capaz de atender as necessidades para a construção de SIGs Web Geotécnicos. De modo complementar, são definidos os seguintes objetivos específicos:

- Identificação das características comuns que impactam na definição de um modelo de arquitetura para SIG Web Geotécnico.
- Identificação dos padrões arquiteturais aplicáveis na definição de um modelo de arquitetura para SIGs Web Geotécnicos.

## 1.3 Metodologia

A metodologia empregada nesse trabalho se iniciou a partir de um estudo em detalhes sobre tudo que está envolvido na definição de uma arquitetura. Desde o entendimento dos principais padrões utilizados até o comportamento aplicado por esses padrões. Para a aplicabilidade de SIGs e a Engenharia Geotécnica também foi feito um estudo aprofundado para entender as suas principais características e o seu escopo. Na sequência, foram estudadas as tecnologias *Spring MVC* e *Hibernate* para entender as potencialidades e viabilidade técnica de integração entre ambas. Principalmente na forma como os módulos são definidos e sua comunicação (*Spring MVC*) e, como acontece todo o ciclo que envolve uma transação de persistência, neste caso o *Hibernate*.

Até esse ponto foram definidos os grandes módulos *Visualizacao* para interagir com o usuário, *Controle* para centralizar a comunicação e tratar requisições, *Modelo* para encapsular a camada de negócio, o *Core Container* que viabiliza o uso do *Spring MVC* e o módulo *Persistencia* para o armazenamento de dados. No entanto, foi feita uma compilação de trabalhos que utilizaram alguma arquitetura para construção de SIGs no intuito de identificar as principais responsabilidades para a definição de interfaces no modelo da arquitetura. Essas interfaces, representam responsabilidades que complementam as demais fornecidas pelos *frameworks Spring MVC* e *Hibernate*. Para finalizar, foi realizada a validação da arquitetura por meio da implementação de um estudo de caso, o qual consiste na construção de uma ferramenta SIG para análise e armazenamento de informações geotécnicas. Por meio desse estudo de caso, foi possível ter as primeiras impressões da arquitetura quanto a sua estrutura e seu comportamento, impactando em oportunidades de refatoração do modelo.

## 1.4 Contribuições

As principais contribuições desse trabalho são:

- a apresentação de dois *frameworks* maduros que dispõe um estrutura bastante rica para o desenvolvimento de aplicações e persistência de dados;
- a modelagem de domínio do modelo de arquitetura, a qual pode ser implementada para a construção de uma classe de aplicações SIG;
- uma estrutura arquitetural que possui entre suas principais vantagens o encapsulamento de dados entre todos os módulos, a conversão automática e a persistência automatizada de dados;
- o projeto e implementação de um sistema SIG que funciona como uma ferramenta de análise de dados geotécnicos para profissionais da Engenharia Civil.

## 1.5 Organização do Texto

O restante deste trabalho está organizado de acordo com a sequência de capítulos apresentada a seguir:

- **Capítulo 2:** aborda a fundamentação teórica sobre os conceitos que necessários para a compreensão da solução proposta.
- **Capítulo 3:** apresenta os trabalhos relacionados selecionados para servir como base de conhecimento para a construção da solução.
- **Capítulo 4:** realiza a apresentação do modelo arquitetural com seus detalhes estruturais, comportamentais e de relacionamento.
- **Capítulo 5:** expõe a implementação da arquitetura proposta diante de um estudo de caso.
- **Capítulo 6:** apresenta as conclusões finais do trabalho.



## 2 Fundamentação Teórica e Tecnológica

O presente capítulo propõe uma fundamentação teórica a fim de introduzir de forma aprofundada conceitos sobre SIGs e Arquitetura de Software. A [seção 2.1](#) apresenta uma fundamentação sobre Arquitetura de Software. A [seção 2.2](#) apresenta a fundamentação sobre Sistemas de Informações Geográficas. A [seção 2.3](#) expõe conceitos sobre o Desenvolvimento de Sistemas Web. A Engenharia Geotécnica é apresentada na [seção 2.4](#). E por último, a [seção 2.5](#) apresenta as lições.

### 2.1 Arquiteturas de Software

A arquitetura de software teve sua concepção ao final de 1960 devido a uma crise de software que chamou a atenção da comunidade. Naquela época o software começou a fazer sucesso, pois em comparação ao *hardware*, projetistas de software tinham mais liberdade na escolha ou na organização de estruturas de software. Mas também ficou claro que, o processo de desenvolvimento de software é muito diferente do processo de outros produtos, como por exemplo carros, edifícios, etc ([GARLAN; SHAW, 1994](#)).

O primeiro registro do conceito de Arquitetura de Software pode ser encontrado em “*The Structure of de The Multiprogramming System*” com autoria de [Dijkstra \(1968\)](#). Nesse trabalho, [Dijkstra \(1968\)](#) discutiu como usar camadas na representação de um sistema de grande escala e, em seguida, criou uma representação com estruturas mais claras e de fácil manutenção. Além disso, David Parnas também contriubui nos fundamentos da arquitetura de software. Sua visão foi esconder informações por meio do uso de interfaces ([PARNAS, 1972](#)), a separação de estrutura ([PARNAS, 1974](#)) e a relação entre a estrutura e a qualidade de software ([PARNAS, 1976](#)), de forma que tudo isso veio a se tornar regras de ouro para arquitetos de software e programadores.

No momento em que a arquitetura de software se tornava algo deslumbrante, no período de 1994 à 2000 foram realizadas diversas conferências. Então, no ano de 1995 um *workshop* internacional sobre especificação e projeto de software promoveu um espaço para pesquisadores da área de arquitetura de software. Ainda nesse ano, ocorreu a primeira oficina de arquitetura de software. A partir de 1995 foram realizadas diversas conferências voltadas especificamente à área de arquitetura de software. As contribuições da indústria e das universidades se tornaram mais ricas e começaram a revolucionar os processos de desenvolvimento de software e metodologias, incluindo métodos de avaliação de arquitetura como, por exemplo, o método *Software Architecture Analysis Method* ([SAAM](#)).

Os autores [Shaw e Garlan \(1996\)](#), afirmam que uma arquitetura de software envolve a descrição de elementos dos quais os sistemas são construídos, interações entre esses elementos, padrões que guiam suas composições e restrições sobre estes padrões.

[D'Souza e Wills \(1999\)](#) afirmam que:

arquitetura de um sistema consiste da(s) estrutura(s) de suas partes (incluindo as partes de software e hardware envolvidas no tempo de execução, projeto e teste), a natureza e as propriedades relevantes que são externamente visíveis destas partes (módulos com interfaces, unidades de hardware, objetos), e os relacionamentos e restrições entre elas.

[Gacek \(1995\)](#) apresenta uma definição de arquitetura de software que preza pelos interesses de quem a utiliza. Assim, diferentes interesses geram diferentes pontos de vista sobre a mesma arquitetura. Os autores sintetizam bem o termo quando descrevem a arquitetura do software como:

- uma coleção de componentes de software e de sistemas, e restrições;
- uma coleção de declarações sobre os interesses envolvidos;
- uma coleção de declarações que demonstra o contexto que motivou determinada definição do sistema pela utilização de componentes, conexões, e restrições particulares.

### 2.1.1 Visão Arquitetural

Uma arquitetura de software diante de suas características, possui em sua grande maioria uma certa complexidade, dessa forma podem existir diversas maneiras de entendimento dessas estruturas ([GORTON, 2006](#)). Desse modo, uma arquitetura de software possui diferentes características, como as suas propriedades, elementos, e relacionamentos que por vez, fornecem um único e claro entendimento. As diferentes perspectivas que apresentam uma arquitetura pode ser chamado de visão. Neste sentido, [Kummel \(2007\)](#) define uma visão arquitetural:

Uma visão consiste em um conjunto de modelos focados em um determinado aspecto da arquitetura, omitindo outros detalhes que são menos importantes neste contexto e serve como insumo para cada passo do processo de construção do software.

No entanto, [Sommerville \(2011\)](#) enfatiza que é impossível representar todas as informações relevantes sobre a arquitetura de um sistema em um único modelo de arquitetura, pois cada modelo mostra apenas uma visão ou perspectiva do sistema.

Nesse sentido [Clements \(1994\)](#) afirma:

Para o completo entendimento da arquitetura de um software, é necessário considerar não apenas a perspectiva estrutural, mas também aspectos como a distribuição física, processo de comunicação e sincronização, entre outros.

Na literatura é possível encontrar diversas maneiras de contornar essa situação por meio de métodos apropriados. Paula (2003) apresenta duas maneiras de representar os componentes ou elementos de um sistema por meio do Desenho Interno e Desenho Externo. Quanto ao nível de abstração dos elementos que compõe essas formas de representação, pode-se evidenciar o nível estratégico. Esse nível é chamado de desenho arquitetônico, pois é voltado para arquitetura do sistema e é fundamental para determinar a capacidade do quanto o sistema suporta os seus requisitos, e o custo para se atingir essa capacidade.

O modelo de Kruchten (2000), também chamado como modelo 4+1 (*The 4+1 View Model*), consiste em organizar o modelo de visões em cinco visões concorrentes. Assim, cada visão apresenta um conjunto específico de informações de acordo com o interesse dos diferentes *stakeholders*. A primeira das cinco visões, é a visão lógica, essa descreve aspectos estáticos, como componentes, conectores e interfaces. A segunda é a visão de processo, encarregada de descrever aspectos de processo, incluindo a concorrência e sincronização. A visão física descreve o mapeamento dos componentes e processos para *hardware* e sua distribuição. A visão de desenvolvimento descreve a organização estática do software no ambiente de implementação. E por último, a visão de cenários descreve os casos de uso do sistema em sua arquitetura.

Uma proposta a partir de quatro visões é feita por (HOFMEISTER; NORD; SONI, 1999). A primeira é a Visão Conceitual, a qual mapeia características funcionais para a arquitetura e também descreve os componentes e conectores. A Visão de Execução descreve os processos mapeados por meio de módulos, a taxa de uso e sua distribuição. A Visão de Módulo consiste em decompor o software em subsistemas (módulos). A última é a visão de Código, a qual descreve como os módulos da visão de módulos são mapeados para arquivos de código-fonte e como os processos da visão de processos são mapeados para arquivos executáveis.

De acordo com as visões de Hofmeister, Nord e Soni (1999) e Kruchten (2000), existe uma ênfase em separar aspectos dinâmicos e aspectos estáticos em visões diferentes da arquitetura. As perspectivas dinâmicas tratam de aspectos como sincronização, concorrência e comportamento, já as estáticas descrevem os componentes e suas inter-relações.

Outra maneira de representar a arquitetura de software pode ser feita por meio do uso das Linguagens de Descrição de Arquiteturas, ou *Architecture Description Languages* (ADLs). Uma variedade de linguagens para projeto arquitetural vem sendo proposta a fim de prover para os arquitetos de software notações para a especificação e análise das arquiteturas (MONROE et al., 1997). As ADLs visam formalizar ainda mais a representação de arquiteturas de software em relação aos diagramas informais.

Basicamente, uma ADL deve descrever os elementos arquiteturais e os conectores em uma arquitetura. Além disso, uma ADL deve descrever o sistema como uma compo-

sição de conectores e conexões independentes. Esses são descritos em um alto nível de abstração, sem se deter a detalhes de implementação, permitindo a reutilização de padrões de relacionamento componente-conector (como uma estrutura cliente-servidor) em descrições arquiteturais diferentes daquela para a qual foram originalmente especificados (SHAW; GARLAN, 1996).

De modo a apresentar um menor formalismo na especificação de uma arquitetura, a *Unified Modeling Language* (UML) apresenta vantagens. Entre elas, o fato de apresentar um padrão bastante difundido, tanto na academia, quanto na indústria para a modelagem de software com o paradigma de Orientação a Objetos (OO).

Diante das diversas abordagens para se representar arquiteturas de software, existem questões que podem gerar dúvidas na hora de documentar uma arquitetura. Como exemplo, Gorton (2006) enfatiza que ainda não existe uma forma universal para documentar arquiteturas, uma vez que artefatos dessa natureza muitas vezes são analisados e modificados por diferentes *stakeholders*.

## 2.1.2 Estilos e Padrões Arquiteturais

Os estilos e padrões arquiteturais traduzem organizações estruturais conhecidas para sistemas de software, estabelecendo uma topologia para a arquitetura e restrições sobre os componentes e conectores (VASCONCELOS, 2007). No entanto, de acordo com Pressman (2010), o padrão difere do estilo em um certo número de modos fundamentais. Por exemplo: 1) o escopo de um padrão é menos amplo, enfoca um aspecto da arquitetura em vez de toda a arquitetura; 2) um padrão impõe uma regra na arquitetura, descrevendo como o software vai manipular em algum tópico de sua funcionalidade em termos de infraestrutura; 3) padrões arquiteturais tendem a atender tópicos comportamentais específicos no contexto arquitetural. Mesmo diante dessas diferenças, os mesmos podem ser usados em conjunto para compor a estrutura de um sistema.

Uma questão importante a ser ressaltada em relação aos estilos e padrões arquiteturais, é que normalmente os estilos e padrões são utilizados por combinações, de modo a formar arquiteturas heterogêneas (SHAW; GARLAN, 1996). Assim, se consegue construir arquiteturas bastante diversificadas para atender as mais diversas necessidades.

### 2.1.2.1 Padrões

Quanto aos padrões arquiteturais, Buschmann, Meunier e Rohnert (1996) afirmam que:

Padrões arquiteturais, expressam um esquema de organização estrutural de sistema. Através dessa organização, são definidos os subsistemas, seus relacionamentos, bem como regras e orientações para um bom uso do padrão arquitetural.

Esses mesmos autores também apresentam um catálogo de padrões arquiteturais, sendo os seguintes:

- **Microkernel:** esse padrão consegue realizar o encapsulamento dos serviços considerados fundamentais da aplicação dentro do componente *microkernel*, esse é o núcleo do sistema, o qual tem um serviço de registro e configuração de componentes em tempo de execução. Esse núcleo consiste em prover serviços de infraestrutura para compor as outras funcionalidades. As demais funcionalidades que são consideradas estendidas e específicas devem ser encapsuladas nos demais componentes da arquitetura.
- **MVC:** através de uma organização baseada em responsabilidades, esse padrão é composto por três componentes: modelo, visão e controlador. A visão é responsável por apresentar as informações ao usuário, tendo também como responsabilidade seus formatos de saída específicos e obtendo seus dados a partir do modelo. O controlador faz o gerenciamento dos eventos de entrada dos usuários que são disparados na visão, e então são mapeados e traduzidos para requisições de serviços ao modelo ou à visão. Por último, o componente modelo encapsula os dados e as funcionalidades do negócio, de forma a tornar-se independente de representações de saída e tratamentos das interações dos usuários com a aplicação.
- **Camadas (*Layers*):** é um dos padrões mais conhecidos, esse padrão propõe a decomposição de aplicações em grupos de subtarefas, onde cada grupo se apresenta em um nível particular de abstração.
- ***Pipes & Filters*:** esse padrão arquitetural suporta a divisão de uma função ou processo em várias etapas de processamento sequenciais. Cada etapa recebe, processa e disponibiliza uma cadeia de dados e o fluxo de saída de uma etapa corresponde ao fluxo de entrada da etapa seguinte.

De acordo com [Appleton \(1997\)](#), um padrão arquitetural é descrito por um esquema de três partes gerais, sendo guiado por uma orientação similar à documentação de padrões em geral. O esquema é descrito da seguinte forma:

- **Contexto:** descreve as situações em que um problema ocorre. A escolha do contexto correto de um padrão é uma tarefa difícil. Uma questão a ser analisada nesse item, é que essa abordagem não garante que são cobertas todas as situações relevantes, visto a larga amplitude de estados que um contexto pode assumir. Porém, ao menos fornece uma orientação valiosa das possibilidades de utilização do padrão.
- **Problema:** trata do problema recorrente que aparece em um determinado contexto. Descreve uma especificação geral do problema, capturando sua essência. A

declaração geral do problema é composta também pelo conjunto de forças. O termo força é descrito por [Appleton \(1997\)](#) como sendo “todo aspecto do problema que deve ser considerado durante sua solução”. Em geral, os itens das forças discutem o problema de pontos de vista variados e permitem a sua compreensão mais detalhadamente.

- **Solução:** cada padrão especifica uma certa estrutura, uma configuração espacial dos elementos. Aqui é encontrada a descrição de como solucionar o problema recorrente. Essa estrutura foca nos aspectos estáticos da solução, um padrão arquitetural deve também especificar um comportamento dinâmico da solução, pela descrição da colaboração e comunicação de seus componentes.

Cada padrão é destinado a resolver algum problema recorrente que pode surgir durante o projeto de software. Como por exemplo, o gerenciamento do processamento paralelo e a distribuição de partes que compõe o software. A partir desse tipo de classificação, se tem os padrões organizados em categorias, cada uma representando uma característica central do projeto ([BUSCHMANN; MEUNIER, 1994](#)).

#### 2.1.2.2 Estilos

Um estilo arquitetural define um conjunto de regras de projeto que identificam tipos de componentes, seus conectores e restrições existentes para sua composição, os quais podem ser usados para compor uma família de sistemas e subsistemas. Conectores são o meio de comunicação entre componentes de um estilo arquitetural. Cada conector possui a especificação de um protocolo que define suas propriedades, dentre elas: os tipos de interfaces que ele pode intermediar e a ordem em que os eventos acontecem ([SHAW; GARLAN, 1996](#)).

A vantagem de trabalhar com estilos arquiteturais é que estes encapsulam decisões importantes sobre os elementos da arquitetura, seus relacionamentos e suas restrições. Sua utilidade também se aplica à coordenação entre os desenvolvedores responsáveis pela arquitetura do software ([XAVIER, 2001](#)).

A seguir são apresentados alguns dos estilos arquiteturais mais conhecidos:

- **Cliente-Servidor:** serviços compartilhados, fornecidos através da solicitação dos clientes distribuídos.
- **Processos de comunicação:** sistema é composto de processos independentes e concorrentes
- **Interpretadores:** ligações entre programas abstratos e as máquinas sobre as quais devem rodar.

- **Baseado em eventos e invocação implícita:** a interação entre os componentes é realizada através do anúncio de um ou mais eventos.
- **Repositório:** sistemas que possuem um repositório de informações compartilhadas.
- **Organização em programa principal e subrotinas:** estabelecimento de hierarquias de dados e procedimentos.
- **Orientados a um domínio específico:** customização de uma estrutura para uma família de aplicações.

### 2.1.3 Arquitetura de Software para um Domínio Específico

De acordo com (HAYES, 1994) uma Arquitetura de Software para um Domínio Específico (DSSA) é entendida da seguinte maneira:

Representa mais do que uma arquitetura para um determinado domínio específico. Um DSSA é uma coleção de componentes de software, especializados para um determinado tipo de tarefa (domínio), generalizados para que seja possível seu uso efetivo através do domínio, e compostos em uma estrutura padronizada (topologia) efetiva para a construção de aplicações.

Refinamentos no modelo de domínio permitem a criação de componentes reutilizáveis, sendo que esses podem fazer parte de uma infraestrutura de reutilização. Todas essas atividades fazem parte de um processo denominado Engenharia de Domínio (WERNER; BRAGA, 2005). De forma geral existe um certo consenso para a definição das etapas da Engenharia de Domínio: análise de domínio, projeto de domínio e implementação do domínio. De modo a enfatizar o projeto de domínio, o mesmo tem como objetivo principal a construção de arquiteturas de software capazes de descrever os requisitos identificados durante a engenharia de domínio e representados através do modelo de domínio.

A importância dessa arquitetura de referência é constatada a partir da base para instanciação de aplicações em um domínio de aplicação específico. Um de seus principais objetivos, é propor uma solução estruturada para a construção de uma família de aplicações capazes de atender aos requisitos de um determinado domínio (KUMMEL, 2007).

## 2.2 Sistemas de Informações Geográficas

Históricamente a tecnologia SIG pode ser relacionada a evolução dos modos de fazer e usar mapas desde o surgimento das sociedades humanas. Atualmente, existe a necessidade de organizar informações do espaço geográfico considerando objetivos militares, de registro de propriedades de terra, de gestão do uso e ocupação do solo e de circulação

e transportes. Para o conhecimento das feições da superfície terrestre, se usam os mapas (NOGUEIRA, 2008).

O início do desenvolvimento da tecnologia SIG se deu por volta da década de 1960, nos Estados Unidos e no Canadá, onde no Laboratório de Computação Gráfica da Universidade de *Harvard*, ocorreu o marco de sua criação (MONTEIRO; D'ALGE, 2001). Um SIG é habilitado para armazenar grandes quantidades de dados com facilidade de acesso, compartilhamento e gerenciamento. De modo a ser desenvolvido numa plataforma para análises, visualização e cruzamento de dados alfanuméricos e gráficos.

O termo SIG é aplicado para sistemas que realizam o tratamento computacional de dados geográficos e recuperam informações não apenas com base nas suas características alfanuméricas, mas também através de sua localização espacial, podendo ser referenciado por coordenadas geográficas ou espaciais, ou seja, localizados na superfície terrestre e representados numa projeção cartográfica (MONTEIRO; D'ALGE, 2001).

### 2.2.1 Aplicações de SIGs

Os SIGs vêm sendo progressivamente usados por empresas, órgãos do governo, em negócios, pesquisas e num amplo leque de aplicações, que incluem análise de recursos ambientais, planejamento de uso do solo, análise locacional, avaliação de impostos, planejamento de infraestrutura, análise de bens imóveis, de marketing, demográfica e arqueológica (BADIN et al., 2002).

Com uma visão um pouco mais detalhada, Ramirez (1994) lista algumas áreas de aplicação classificadas em cinco grandes grupos: Ocupação Humana, Uso da Terra, Uso de Recursos Naturais, Meio Ambiente e Atividades Econômicas.

A área Ocupação Humana está alinhada as demandas que o estado tem em relação ao planejamento e gerenciamento urbano, como redes de infraestrutura como água, luz, telecomunicações, gás e esgoto. Nessa área também estão envolvidas os setores como Saúde e Educação, Transporte e Segurança, que trata por exemplo, a supervisão do espaço aéreo, marítimo e terrestre.

O Uso da Terra é voltada a ideias de como fazer um planejamento agropecuário; como criar uma estratégia para permitir um melhor escoamento agrícola; modos de classificação de solos e vegetação; estratégias de como planejar barragens e mapeamento do uso da terra.

Em uma linha um pouco mais como diagnóstica a área de Recursos Naturais, permite realizar o controle de extrativismo vegetal e mineral. Também, para planejamento de gasodutos e oleodutos, gerenciamento costeiro e marítimo. A área de Meio Ambiente se resume em atividade como o controle de queimadas, estudos de modificações climáticas, gerenciamento florestal de desmatamento e reflorestamento.



Para soluções voltadas tanto para empresas como para o estado, a área de Atividades Econômicas trata de questões como o planejamento de marketing; pesquisas sócio-econômicas; distribuição de produtos e serviços e transporte de matéria-prima e insumos.

### 2.2.2 Tecnologias para SIG

Além das áreas de aplicação vistas anteriormente, de acordo com [Câmara e Queiroz \(2000\)](#), os SIGs também são classificados em modos em que são disponibilizados do ponto de vista tecnológico. Assim são divididos em quatro tecnologias: Desktop, Gerenciadores de Dados Geográficos, Componentes e Servidores *Web* de Dados Geográficos.

Os SIGs Desktop são sistemas herdeiros da tradição de Cartografia, com suporte de banco de dados limitado e cujo paradigma típico de trabalho é o mapa (chamado de cobertura ou de plano de informação). Essa classe de sistemas é utilizado principalmente em sistemas isolados, sem a preocupação de gerar arquivos digitais de dados.

Gerenciadores de Dados Geográficos consistem em armazenar os dados espaciais em um ambiente multi-usuário. Essa classe se caracteriza por ser concebida para uso em ambientes cliente-servidor, acoplado a gerenciadores de banco de dados relacionais e com pacotes adicionais para processamento de imagens.

Os Componentes SIG são ambientes de programação que fornecem insumos para que o usuário crie seu próprio aplicativo geográfico. Essa classe se caracteriza pelo gerenciamento de grandes bases de dados geográficos, com acesso através de redes locais e remotas. Esses sistemas devem seguir os requisitos de interoperabilidade, de maneira a permitir o acesso de informações espaciais por SIGs distintos.

Os Servidores *Web* de Dados Geográficos são utilizados para publicação e acesso a dados geográficos via internet.

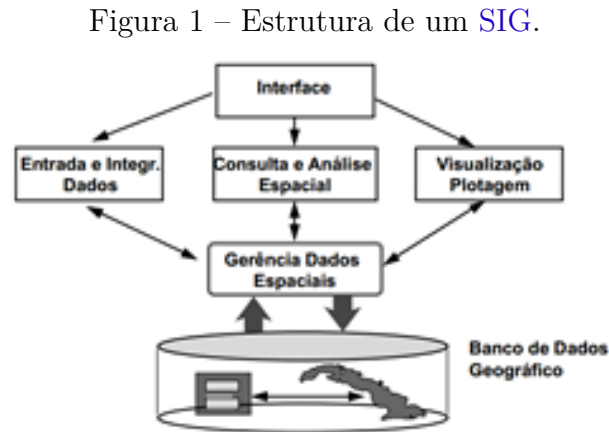
### 2.2.3 Estrutura Geral de um SIG

De acordo com [Schenatto \(2012\)](#), a arquitetura de um SIG é composta dos seguintes componentes: interface com o usuário; entrada e integração dos dados; funções de consulta de análises espaciais; visualização e plotagem; armazenamento e recuperação dos dados. Estes componentes se relacionam de forma hierárquica, em que no nível mais próximo ao usuário se encontra a interface que define como o sistema é operado e controlado. Em um nível intermediário estão todos os mecanismos de processamento dos dados espaciais, incluindo a entrada, edição, análise, visualização e saída dos dados. E no nível mais profundo é feito o armazenamento dos dados.

Em geral as funções de processamento de um SIG operam sobre dados em uma área de trabalho em memória principal. Através de mecanismos de seleção e consultas

são feitas as ligações entre os dados geográficos e as funções de processamento (KUBO, 2004).

A Figura 1 apresenta o relacionamento entre os principais componentes ou subsistemas que compõem um SIG.



Fonte: Câmara e Queiroz (2000).

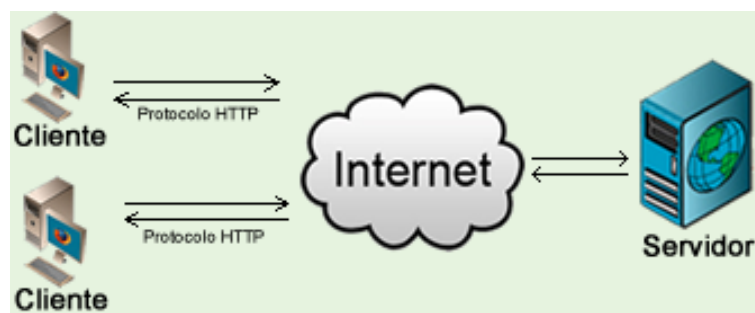
Um sistema SIG pelo fato de ter peculiaridades entre seus objetivos e necessidades, acaba implementando esses subsistemas de forma distinta, porém todos eles devem compor um SIG.

## 2.3 Desenvolvimento de Sistemas Web

Nos últimos anos o escopo de atuação das aplicações *web* vêm crescendo rapidamente e, atualmente, essa categoria de software tem apoiado o relacionamento entre empresas e indivíduos em atividades do dia-a-dia relacionadas ao trabalho ou lazer (SILVA, 2011).

O contexto de uma aplicação *web* típica (Figura 2), basicamente acontece através da interação das partes Cliente (*browser* do usuário) e Servidor, onde os dados são processados e disponibilizados para o Cliente.

Figura 2 – Contexto de uma Aplicação *web*.



A participação do cliente em uma aplicação Cliente/Servidor acontece quando o mesmo a fim de enviar e receber recursos através das mensagens, inicia as chamadas, denominadas requisições (*Request*) e repostas (*Response*). Essas chamadas, são feitas por meio do protocolo *Hypertext Transfer Protocol* (**HTTP**), que traduzido para português significa: “Protocolo de Transferência de Hipertexto”.

Já o Servidor pode ser entendido como um ambiente que armazena, processa e envia serviços e recursos como resposta para o Cliente. O autor [Sommerville \(2011\)](#) enfatiza que a vantagem mais importante de um modelo Cliente-Servidor é justamente por funcionar de uma maneira distribuída. Assim, é possível realizar a adição de novos recursos no sistema de forma transparente, de modo a não afetar o funcionamento do sistema como um todo.

Como pode ser visto no contexto apresentado pela [Figura 2](#), uma aplicação *web* possui em sua estrutura o estilo Cliente-Servidor o qual proporciona organizar essa de uma maneira adequada, ou seja, de acordo com a aplicação que está sendo desenvolvida.

### 2.3.1 Spring Framework

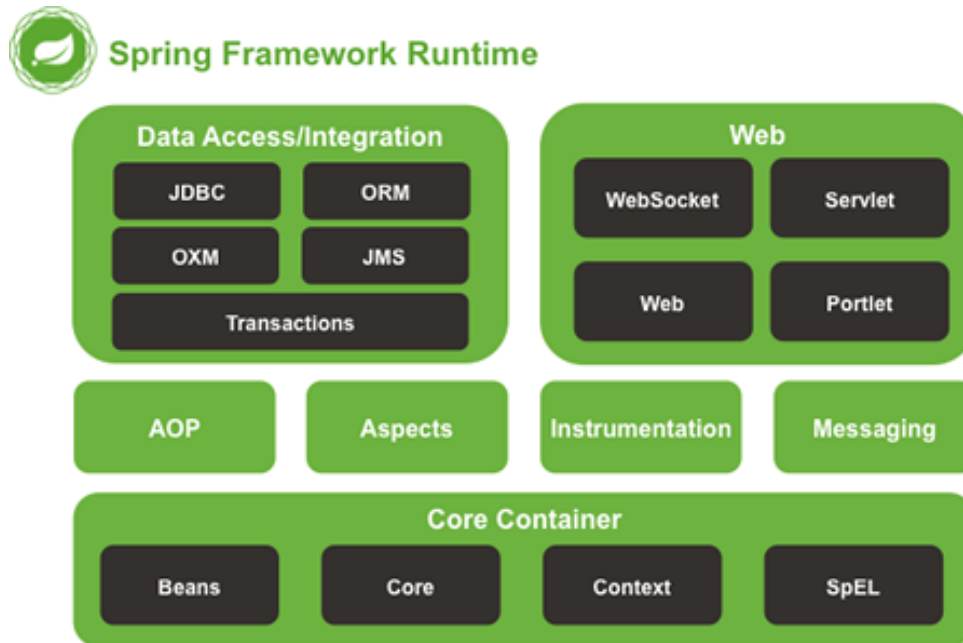
O *Spring Framework*<sup>1</sup> pode ser visto como uma solução madura para o desenvolvimento de aplicações, de modo a ser considerado um modelo abrangente na configuração e programação de aplicações corporativas baseadas na linguagem Java ([SPRING, 2013](#)). A organização desse *framework*<sup>2</sup> se dá a partir de módulos como mostra a [Figura 3](#).

O módulo **Spring-AOP** é compatível com o paradigma *Aspect Oriented Programming* (**AOP**), o qual permite por exemplo, definir métodos interceptadores e *pointcuts*. Já o **Spring-Aspects** permite a integração com a linguagem de programação *AspectJ*. O módulo **Spring-Instrumentation** permite a instrumentação de classes e implementação de *ClassLoader* em determinados servidores de aplicação. Dessa forma, esse módulo oferece facilidades para o suporte para a aplicação através da implementação de *Java Management Extensions* (**JMX**). Essa tecnologia permite acompanhar em tempo de execução o desempenho da aplicação, gerando várias estatísticas das operações *Java Database Connectivity* (**JDBC**) e gerenciamento de transações com *Framework Hibernate* ou *Java Persistence API* (**JPA**).

A partir de quatro abstrações, o **Spring-Messaging** serve como base para aplicações baseadas em mensagens. Uma das possibilidades de implementação é a partir da integração do *Spring* com a *API Java Message Service* (**JMS**). Para isso, o *Spring* fornece a classe *JMS Template*, sendo que essa simplifica o uso do **JMS**, pois realiza o gerenci-

<sup>1</sup> <http://projects.spring.io/spring-framework/>

<sup>2</sup> Um *framework* é um projeto reutilizável de um programa ou parte de um programa expresso como um conjunto de classes ([JOHNSON; FOOTE, 1988](#)).

Figura 3 – Módulos do *Spring Framework*.

Fonte: [Spring \(2013\)](#).

amento (criação de liberação) de recursos durante o envio e recepção de mensagens de forma síncrona.

As aplicações precisam registrar e recuperar dados em alguma fonte de dados. O módulo **Acesso a Dados e Integração**, oferece suporte às principais tecnologias de persistência adotadas no desenvolvimento Java. Esse módulo provê suporte para [JDBC](#), *Object-relational Mapping* ([ORMs](#)), como *Hibernate*, *iBatis*, [JPA](#), *Java Data Objects* ([JDO](#)) e *Object XML Mapping* ([OXM](#)). Além de suportar essas tecnologias, esse módulo oferece também uma nova hierarquia de exceções. Ainda dentro desse módulo, o *Spring* dispõem por meio das *Transactions* o controle transacional, sendo uma implementação de transações programáticas, de modo a ser considerado uma evolução baseado no que é oferecido hoje pelos servidores de aplicação ([WEISSMANN, 2012](#)).

O módulo **Core Container** é o único módulo obrigatório para o uso do *Spring*, pois os submódulos **Spring-Core** e **Spring-Beans** são considerados o núcleo do *framework* onde são implementados o suporte a Inversão de Controle e Injeção de Dependências. Assim, esse *Container*<sup>3</sup> pelo fato de trabalhar com *Plains Old Java Object* ([POJOs](#)) são minimamente intrusivos<sup>4</sup>, possuem inicialização rápida e consomem poucos recursos computacionais, são independentes do ambiente de execução e definem requisitos míni-

<sup>3</sup> É o elemento arquitetural de uma aplicação, responsável por gerenciar o ciclo de vida dos componentes do nosso sistema ([WEISSMANN, 2012](#)).

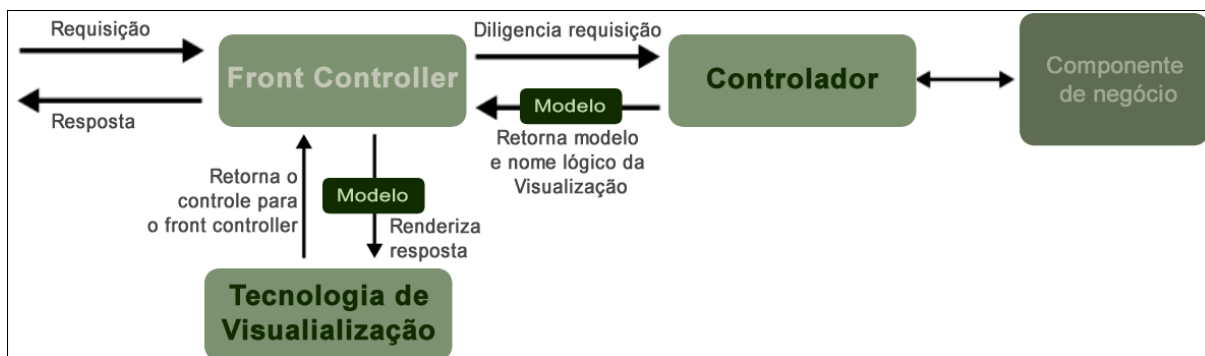
<sup>4</sup> Um container é intrusivo quando requer que seus objetos gerenciados possuam auto dependência de alguma maneira, como por exemplo, obrigando-os a implementar alguma interface ou possuírem alguma anotação específica.

mos para que um componente possa ser gerenciado (JOHNSON, 2004). Já no submódulo **Spring-Context** existe a implementação mais avançada e comumente usada no *Container*, o *ApplicationContext*. Por vez, o *ApplicationContext* é baseado nos módulos *Core* e *Spring-Beans*, e oferece recursos de internacionalização, propagação de eventos, AOP e gerenciamento de recursos.

O módulo *Web* é formado pelos submódulos **Spring-Web**, **Spring-WebMVC**, **Spring-WebSocket** e **Spring-WebMVCPortlet**. O **Spring-Web** basicamente provê recursos de integração voltados à *web*, tais como a funcionalidade de *upload* de arquivo *Multipart file* e a inicialização do *container Inversion of Control (IoC)* utilizando *listeners Servlet* e aplicação orientada ao contexto *web*. O submódulo **Spring-MVC** também conhecido como *WebServlet*, utiliza o padrão de projeto *Model View Control (MVC)* e o estilo arquitetural *REpresentational State Transfer (REST)* para aplicações *web*.

O funcionamento do **Spring-MVC** possui como seu principal “protagonista” o *DispatcherServlet*; esse, aplica o padrão de projeto *Front Controller* (Figura 4), sendo muito utilizado para a construção de *frameworks* voltados para aplicações *web*. Além disso, consiste em fornecer um ponto central de entrada para todas as requisições direcionadas à aplicação. Dessa maneira, sua principal função é interpretar essas requisições e decidir qual o componente responsável por seu processamento e eventual retorno para o usuário.

Figura 4 – Padrão *Front Controller* no contexto do *Spring MVC*.



Adaptação baseada na obra de Weissmann (2012).

Para melhor entender o comportamento do Spring MVC apresentado pela Figura 5 se pode assumir o seguinte cenário: **1)** O usuário envia uma requisição; essa requisição é “ouvida” pelo *DispatcherServlet*. **2)** Logo após é atendida pelo mapeador de requisições *HandlerMapping*, o qual é responsável pela descoberta de qual o controlador deve ser acionado juntamente com o *HandlerAdapter* **(3)** que auxilia na leitura das *anotações*<sup>5</sup> *@RequestMapping* da aplicação. **4)** Em seguida, com o controlador alvo já en-

<sup>5</sup> Uma anotação ou *Annotation* possibilita configurar uma classe através da declaração de metadados, ao invés destas informações serem expostas diretamente na própria classe.

contrado, é obtido o nome do componente de visualização(5); 6) Feito isso, é acionado o gerenciador de visualização (*ViewResolver*) que realiza a construção do *template* de visualização (*View*). A partir disso, é injetado na *View* um conjunto de variáveis (*Model*) e então é encaminhada ao *DispatcherServlet*(7) que retorna a resposta ao usuário para ser renderizada ou simplesmente recebida.

Figura 5 – Estrutura do *Spring MVC*.

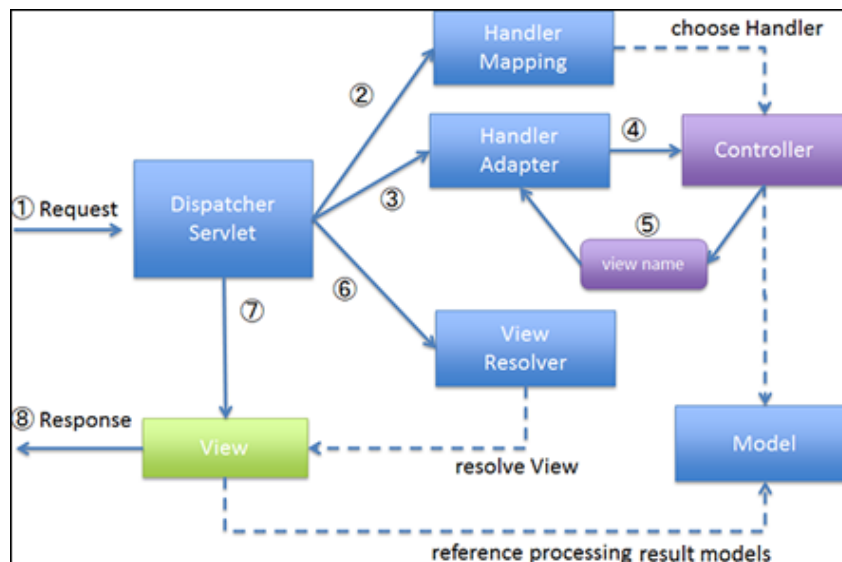


Figura adaptada de [Guideline \(2012\)](#).

### 2.3.2 Hibernate

Atualmente a maioria das aplicações precisam que seus dados sejam armazenados de alguma maneira. Para isso, existe a persistência de dados, cujo processo permite armazenar dados de forma permanente e então recuperá-los a qualquer momento, mesmo após a aplicação ter sido finalizada. Uma das soluções disponíveis de persistência para aplicações é o *Hibernate*. O *Framework Hibernate* é uma poderosa ferramenta [ORM](#) para persistir dados em aplicações Java que permite de forma automática e transparente ser imbutido no código. De acordo com [Bauer e King \(2007\)](#) definem [ORM](#) da seguinte maneira:

É a persistência automatizada dos objetos em uma aplicação Java para as tabelas de um banco de dados relacional, utilizando metadados que descrevem o mapeamento entre os objetos e o banco de dados. Assim, [ORM](#), em sua essência, trabalha com transformação (reversível) dos dados de uma representação para outra.

Com a implementação desse *framework* de uma maneira natural se pode obter uma gama de benefícios como produtividade no desenvolvimento da aplicação; manutibilidade através da obtenção de um código mais limpo; performance entre outras. No

entanto, esses ganhos podem depender do quanto e de que forma se utiliza o *Hibernate* em um sistema. De acordo com Fussel (1997), são definidos quatro níveis de uso que o *Hibernate* pode prover: o Relacional Puro, o Mapeamento Leve, o Mapeamento Médio e o Mapeamento Completo.

O nível Relacional Puro toda a aplicação, inclusive a interface com o usuário, é desenhada com base no modelo relacional e operações relacionais baseadas em *Structured Query Language (SQL)*. A partir dessa abordagem, sistemas de grande porte podem não conseguir extrair os benefícios desse tipo de solução. No entanto, para aplicações simples isto é viável, onde a baixa reutilização de código é tolerada.

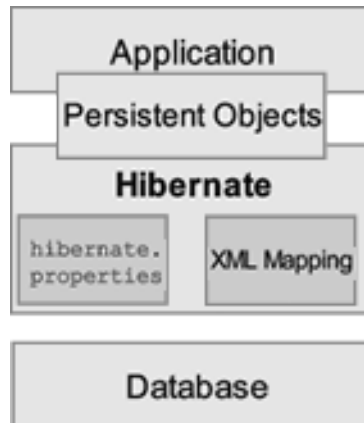
Para o Mapeamento Leve, entidades são representadas como classes que são mapeadas manualmente para as tabelas relacionais. O *SQL/JDBC* codificado à mão é escondido da lógica de negócio.

O nível Mapeamento Médio, a aplicação é desenhada em torno do modelo de objetos. O *SQL* é gerado no momento da construção utilizando ferramentas de geração de código, ou em tempo de execução pelo código do *framework*. As associações entre os objetos são suportadas pelo mecanismo de persistência, e consultas podem ser especificadas utilizando uma linguagem de expressão orientada para objetos.

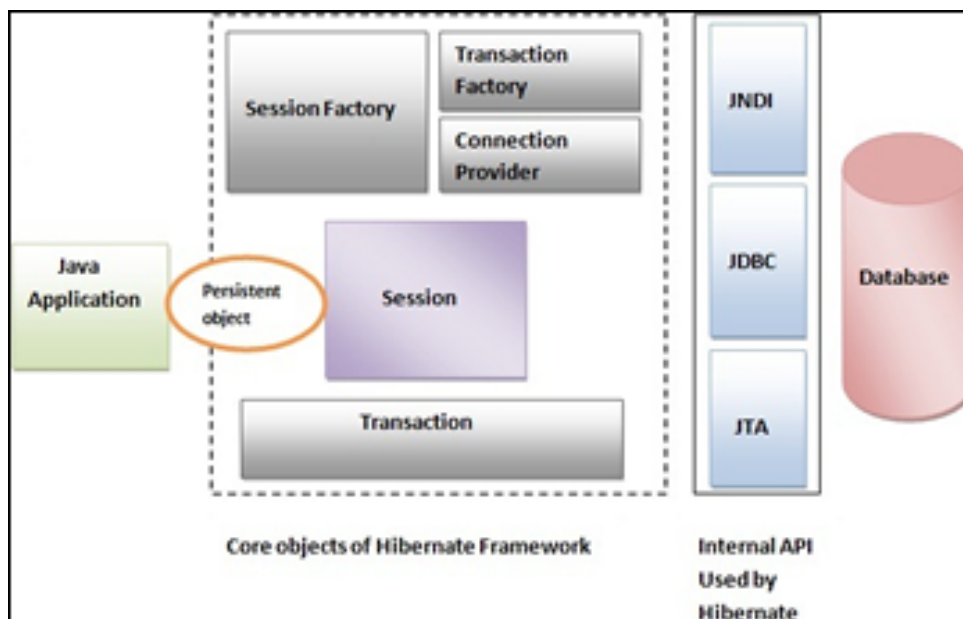
O Mapeamento Completo suporta a modelagem de objetos sofisticada: composição, herança, polimorfismo e persistência por transitividade. A camada de persistência implementa uma persistência transparente, classes persistentes não herdam de uma classe básica especial ou têm que implementar uma interface em especial.

O *Hibernate* possui como base para a sua persistência o módulo *Hibernate Core*, com sua *API* nativa e seus metadados de mapeamento guardados em arquivo *eXtensible Markup Language (XML)*. Esse módulo possui uma linguagem de consulta chamada *Hibernate Query Language (HQL)* muito parecida com o *SQL*, assim como interfaces de consulta programática para consultas do tipo *Criteria*. O *Hibernate* cria uma camada entre o Banco de Dados (*BD*) e a aplicação. Ele carrega os detalhes de configuração, como mostra a [Figura 6](#) da arquitetura em uma visão mais compacta (BAUER; KING, 2007).

O *Hibernate* cria objetos persistentes que servem para sincronizar dados entre a aplicação e o banco de dados. A [Figura 7](#) apresenta uma visão mais abrangente do *framework*. Com a finalidade de manter os dados em um *BD*, o *Hibernate* cria uma instância da classe de entidade (classe Java mapeada com a tabela do *BD*). Esses objetos são chamados de *Transients Objects*, pois estes ainda não estão associados a uma sessão ou ainda não foram persistidos no *BD*. Para então persistir o objeto no *BD*, a instância da interface *SessionFactory* é criada. *SessionFactory* é uma instância *Singleton* que implementa o padrão de projeto *Factory*. A *SessionFactory* carrega o arquivo de confi-

Figura 6 – Arquitetura do *Hibernate* de forma resumida.

Fonte: Patel (2011).

Figura 7 – Elementos da arquitetura do *Hibernate*.

Fonte: JavaTpoint (2010).

guração “hibernate.cfg.xml”, e com a ajuda do *TransactionFactory* e *ConnectionProvider* implementa todas as definições e configurações com o banco de dados. Cada conexão do BD no *Hibernate* é criado através de uma instância da interface *Session*.

Cada *Session* representa uma única conexão com o banco de dados. Assim, os objetos da sessão são criados a partir da *SessionFactory*.



## 2.4 Engenharia Geotécnica

A Engenharia Geotécnica é uma área da Engenharia Civil, voltada para a análise, concepção e construção de estruturas que utilizam o solo como suporte, dentre elas é possível citar: fundações superficiais e profundas, estruturas de contenções, estabilização de encostas, realização de aterros ou cortes, etc. (BRAJA, 2011; MASSAD, 2010).

Para a elaboração de um projeto geotécnico que atenda as normativas existentes e também a boa prática da engenharia, é necessária a obtenção de parâmetros geotécnicos representativos do subsolo (MILITITSKY; CONSOLI; SCHNAID, 2005). Segundo Almeida e Marques (2010), Schnaid e Odebrecht (2012) os ensaios de campo e laboratório apresentam reconhecida utilidade na determinação das propriedades de resistência, deformabilidade e condutividade hidráulica dos solos.

Dentre as várias técnicas de ensaios *in situ* o piezocone<sup>6</sup> permite uma excelente definição da estratigrafia do solo, além da estimativa prévia da história de tensões e dos parâmetros de resistência e de adensamento do solo (ROBERTSON, 1990). O ensaio de palheta é o ensaio mais empregado para a determinação *in situ* da resistência ao cisalhamento não-drenada nos depósitos de argilas moles (COUTINHO; OLIVEIRA; OLIVEIRA, 2000). O ensaio de adensamento oedométrico<sup>7</sup> realizado em laboratório fornece parâmetros utilizados para a estimativa da velocidade e magnitude de recalques de estruturas assentes sobre solos moles Almeida e Marques (2010).

O estudo dos resultados e correlações entre estes três ensaios, juntamente com os ensaios de caracterização do solo, constituem os ensaios consagrados na prática de Engenharia Geotécnica de solos moles no Brasil Almeida e Marques (2010).

## 2.5 Lições do Capítulo

A definição de uma arquitetura para um domínio específico envolve sobretudo a tomada de decisões entre prioridades, utilização de requisitos, limitação de escopo e viabilidade. Essas decisões permitem guiar a construção do modelo arquitetural através da utilização de padrões e estilos arquiteturais para possibilitar um modelo com estrutura e comportamento esperado. As tecnologias para SIG definidas por Câmara e Queiroz (2000), quase que em sua totalidade convergem para a implementação na plataforma *web*, a qual tem como seu principal padrão o Cliente-servidor que permite as aplicações trabalhar de forma distribuída.

<sup>6</sup> O ensaio piezocone consiste basicamente na cravação no terreno de uma ponteira padronizada. A partir deste se pode caracterizar a estratigrafia do perfil do solo através de diferentes sistemas de classificação

<sup>7</sup> Seus principais objetivos são a obtenção dos parâmetros de compressibilidade: tensão de sobreadensamento, índice de recompressão, índice de compressão e índice de expansão.

Para a definição de uma aplicação a ser executada na *web*, o padrão arquitetural mais utilizado é o *MVC*, pois o mesmo possui a sua estrutura adequada para trabalhar no lado cliente através do seu módulo *View*, e do lado servidor por meio dos módulos *Model* e *Controller*, responsável por fazer o “meio de campo” entre os demais módulos.

As tecnologias *Spring MVC* e *Hibernate* em uma perspectiva arquitetural, possuem qualidades relevantes para uma estrutura que consiga servir como base para *SIGs web* geotécnicos, pois o *Spring MVC* permite que sejam expandidos seus módulos(*back-end*) e ao mesmo tempo possibilita conectar-se aos diversos meios de persistência de dados, neste caso o *Hibernate*, como uma estratégia de persistência automatizada. Essa necessidade de expansão para aplicações *SIGs* é vista como um fator estratégico, pois essa classe de aplicação evolui constantemente com novas tecnologias, principalmente de reúso software, muitas vezes tendo impacto significativo na arquitetura da aplicação.

## 3 Trabalhos relacionados

Com o propósito de obter conhecimento dos trabalhos voltados para definição de modelos de arquitetura SIGs, ou seja, saber o estado da arte em modelos de arquitetura para SIGs, foi realizada uma busca por trabalhos que norteiam este tema. Este capítulo apresenta na [seção 3.1](#) a metodologia de trabalho que foi adotada para a revisão de literatura. Na [seção 3.2](#) são apresentados como resultado os trabalhos encontrados na revisão de literatura. Para finalizar, na [seção 3.3](#) são apresentadas as lições do capítulo.

### 3.1 Protocolo de Busca

Foi elaborado um protocolo de busca, no qual foram estabelecidos alguns critérios. Primeiro foi definida qual a questão a ser respondida para a definição das buscas, sendo definido a seguinte: “Quais são as soluções existentes que apresentam uma proposta de modelo de arquitetura de software específico para SIGs”.

Após isso, foi definida a fonte ou mecanismo para serem feitas as buscas. Como resultado das buscas, se obteve um total de 62 trabalhos retornados. A partir disso, foram aplicados os seguintes critérios de seleção: O trabalho deve ter sido publicado no período de 2009 até 2014 e deve ter no mínimo seis páginas. Também deve apresentar uma proposta de arquitetura de software para SIGs sendo que esta deve ter sido experimentada. A partir desse refinamento final, foram selecionados por fim um total de 15 trabalhos.

### 3.2 Trabalhos Selecionados

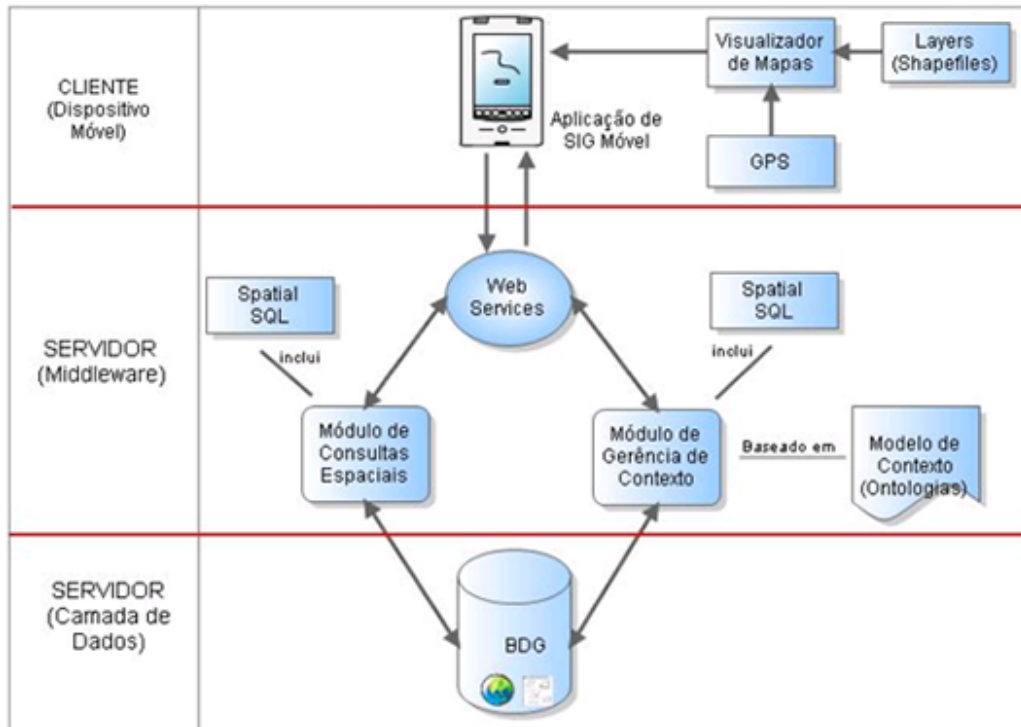
A partir da obtenção desses trabalhos, os mesmos foram divididos em dois grupos; soluções SIG voltadas exclusivamente a Geotecnia e outro grupo com soluções SIG para as mais diversas áreas de aplicação.

#### 3.2.1 SIGs para Diversas Áreas de Aplicação

O trabalho feito por [Lamas \(2009\)](#) propõe uma arquitetura ([Figura 8](#)) para o desenvolvimento de aplicações SIGs móveis capazes de gerenciar informações de contexto. Essa proposta se baseia na especificação de um modelo de contexto baseado em ontologias e em um conjunto de *web services* para acesso a informações armazenadas remotamente em um banco de dados geográfico. Para a validação da arquitetura foi elaborado um estudo de caso. Esse trabalho conseguiu realizar a definição de uma arquitetura para

SIGs móveis sensíveis ao contexto. Também foi feita uma proposta da reutilização de um modelo baseado em ontologias (SeCOM).

Figura 8 – Arquitetura proposta para um SIG móvel sensível ao contexto.

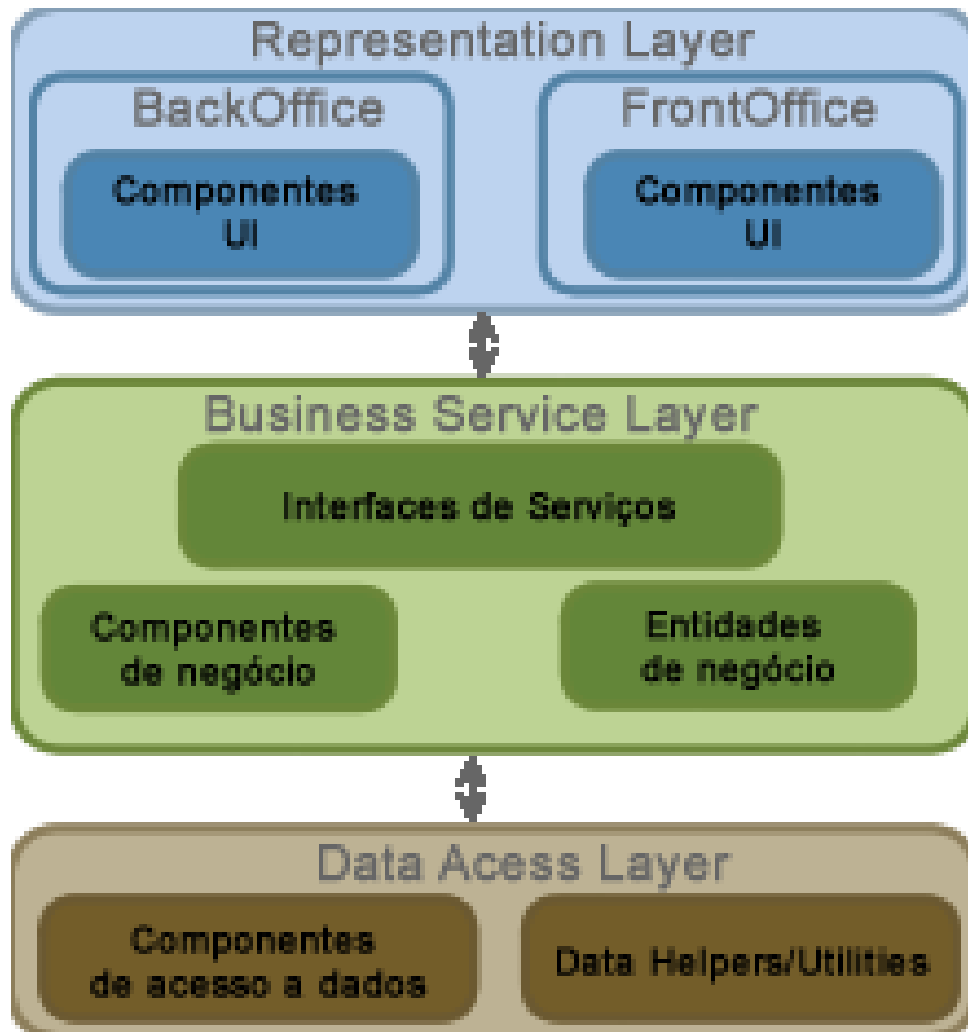


Fonte: Lamas (2009).

Com o objetivo de apresentar uma solução SIG que ofereça uma maior amplitude de opções de gestão de pontos de interesse e respectiva georeferenciação do que algumas aplicações existentes no mercado, Souto (2012) apresentou um projeto de um SIG, sendo composto por um portal de administração onde se possa categorizar pontos de interesse. A arquitetura utilizada (Figura 9) foi baseada no modelo multicamadas, orientado a serviços. A abordagem arquitetural e tecnológica seguida nesse projeto teve como pressupostos a modularidade, funcionalidade, reutilização, garantindo assim uma solução orientada a serviços através de camadas independentes e projetando soluções futuras ou extensões desta em que seja possível a reutilização de serviços e funcionalidades sem redundância de códigos.

Silva (2012) apresenta a proposta de uma arquitetura de software para sistemas de informação no domínio de gestão das águas ou recursos hídricos. Esse trabalho foi baseado na linha superior do *framework* Zachman, assim, foi criado um modelo de arquitetura formado por níveis arquiteturais: Escopo; modelo de negócio e modelo de sistema. Para a implementação desse modelo, foi implementado um estudo de caso, o Sistema Integrado para Gestão de Recursos Hídricos (SINGERHnet). Com a utilização desse *framework* para auxiliar a construção da arquitetura, o mesmo se mostrou benéfico no sentido de

Figura 9 – Proposta da arquitetura em camadas feita por Souto (2012).



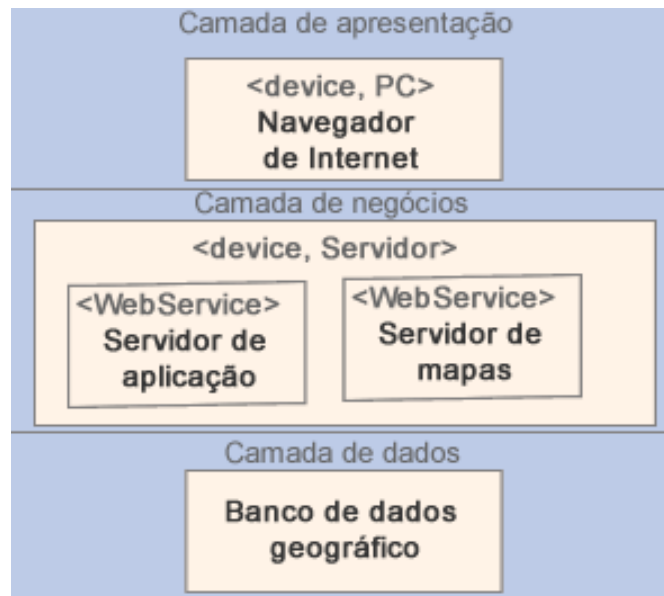
Fonte: adaptada da obra de Souto (2012).

proporcionar todas as partes que compõe a arquitetura em um único lugar, facilitando a leitura e utilização da arquitetura. No entanto, esse *framework* demanda por melhorias para viabilizar a sua expansão. A arquitetura pode ser vista na Figura 10.

Com um trabalho voltado para soluções na agricultura, Schenatto (2012) propõe a elaboração de um software para determinação dos níveis de infestação de doenças foliares em plantações de laranjeiras por meio de técnicas de processamento digital de imagens, geoprocessamento e algoritmos de interpolação de dados. Para facilitar a manipulação dos dados espaciais, integrou-se a *API Geotools*. Para a arquitetura do sistema foi utilizado o padrão MVC. Assim, a *API Geotools* conseguiu suprir a necessidade na representação e melhor compreensão das informações geográficas. Já para o armazenamento e gerenciamento dos dados foi utilizado o PostgreSQL e sua extensão espacial PostGIS.

Silva e Araujo (2007) apresenta um trabalho que consiste em rever as normas do serviço OpenGIS Consortium e estudar a implementação de uma arquitetura (Figura 11)

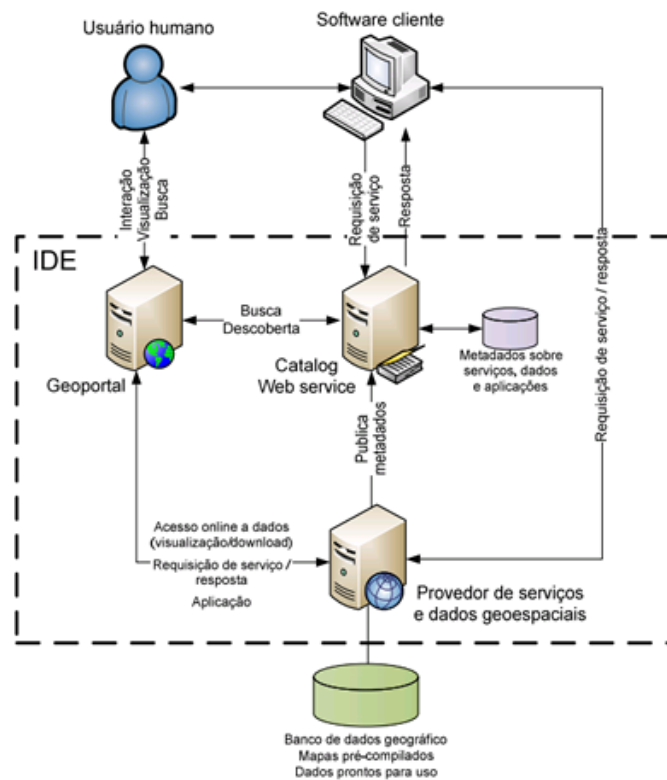
Figura 10 – Proposta da arquitetura feita por [Silva \(2012\)](#).



Fonte: adaptada da obra de [Silva \(2012\)](#).

que permita a fácil utilização de dados espaciais em sistemas distribuídos. A implemen-

Figura 11 – Arquitetura [SOA](#) aplicada a dados geográficos.



Fonte: [Silva e Araujo \(2007\)](#).

tação da arquitetura é basicamente da configuração de três fases: **BD** (corresponde aos

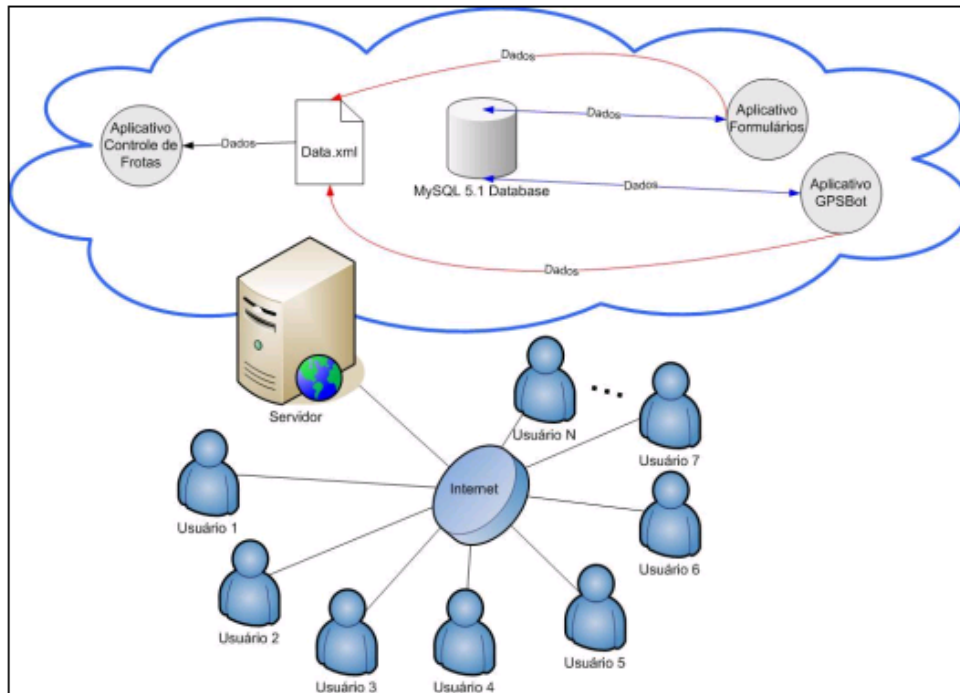
dados geográficos), serviço web (corresponde a um provedor de conexão de serviços web *Open Geospatial Consortium* (OGC) com o BD) e o software do cliente (corresponde a visualização e acesso do cliente aos dados). De forma a comparar o trabalho realizado com o serviço do *Google Maps*, esse sistema possui a vantagem de existir a possibilidade de combinação das imagens de modo a dar maior liberdade ao usuário na construção de uma visualização. Esse serviço também permite que o usuário configure e combine os dados de acordo com suas necessidades.

Com o objetivo da construção de uma aplicação SIG para Companhia de Saneamento do Sergipe (DESO), Melo (2012) utilizou para o desenvolvimento desse sistema o *TimeMap* que permite a visualização de mapas através de um *applet*. Também foram utilizadas as tecnologias *Java Server Faces* (JSF) e *Asynchronous Javascript and XML* (AJAX). O processo de desenvolvimento foi baseado na metodologia *Rational Unified Process* (RUP). A solução construída possui características inovadoras que permite agilizar a tomada de decisão. A arquitetura utilizada foi a Cliente Servidor, sendo que nesse trabalho ela foi utilizada baseando-se no princípio de que é mais eficiente para manipular grandes massas de dados.

Perillo (2011) propõe um Atendimento de Despacho de Emergências em Santa Catarina (SADE). Inicialmente foi criado um projeto lógico, o qual contempla alguns módulos básicos. De posse desse projeto lógico foi feito o desenvolvimento do sistema. O sistema foi baseado no conceito SIG, e também nos paradigmas de aspectos e anotações. A arquitetura SADE incorporou técnicas pouco utilizadas na indústria, para obter flexibilidade para a adição e modificação de componentes transversais e a produtividade da equipe. O modelo de integração com outros sistemas e com uma interface georreferenciada pode vir a ser utilizado como base para arquiteturas de sistemas similares. Em termos de *design*, o SADE foi construído em quatro camadas, mas ao mesmo tempo atendendo ao padrão MVC. As quatro camadas definidas foram apresentação, negócios, persistência e BD. Além dessas camadas existem outros componentes satélites, que auxiliam e envolvem as camadas, como aspectos, *web services*, interceptadores, etc.

A partir da utilização da API Google Maps, Cunha et al. (2009) fez a construção de um sistema de gerenciamento de frotas, com a possibilidade de cadastrar pessoas, viaturas e chamados. Todas as modificações realizadas no aplicativo são armazenadas no banco de dados e, logo após replicado em um XML. Foi implementado o componente de *GPSBot*, que tem o propósito de alterar o posicionamento das viaturas existentes, mediante a alteração de suas coordenadas. O *Mysql* é o Sistema de Gerenciamento de Banco de Dados (SGBD) utilizado. Quanto a arquitetura de software (Figura 12) foi utilizado o padrão MVC juntamente com as tecnologias *Java Server Pages* (JSP) e *Servlets*. A arquitetura *web* tornou o projeto escalável, isto é, sem quaisquer alterações no software desenvolvido podemos ampliar o número de usuários atendidos pelo sistema.

Figura 12 – Arquitetura utilizada por Cunha et al. (2009).



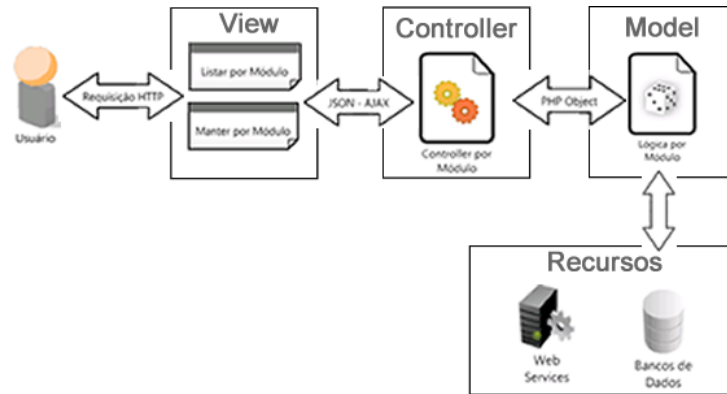
Fonte: Cunha et al. (2009).

Veiga (2012) apresenta em seu trabalho um estudo sobre a qualidade de dados no contexto de informática para biodiversidade que permita identificar causas que afetam a qualidade de dados de ocorrências de espécies em seus domínios de dados mais relevantes e identificar técnicas e recursos computacionais que permitam melhorar aspectos importante na qualidade desses dados. A arquitetura utilizada foi o MVC (Figura 13), e como estudo de caso foi implementado um sistema de informação a fim de melhorar a qualidade dos dados por meio da prevenção a erros durante a digitalização de dados de ocorrências de espécies. Como destaque se tem a metodologia utilizada para o estudo, pois ela pode ser usada em outras pesquisas sobre Qualidade de Dados aplicada em outros domínios de aplicação.

Com a intenção de propor um mecanismo de avaliação da informação com relação a sua utilidade a cada momento, Ferreira (2011) utilizou uma abordagem de análise cognitiva orientada a objetivos, sendo que esta abordagem tem seu processo constituído por quatro passos. Foram selecionados dois casos atípicos para avaliação e entrevistados alguns comandantes participantes. Para a construção do aplicativo que atenda aos requisitos, e ao mesmo tempo seja flexível e expansível para acomodar novas funcionalidades e integrações com outros sistemas foi projetada uma arquitetura dividida em quatro camadas, mostrada na Figura 14. A principal contribuição desse trabalho é a proposta de filtragem da informação por meio de critérios de utilidade com o intuito de se fornecer a informação correta para a pessoa certa no momento adequado.

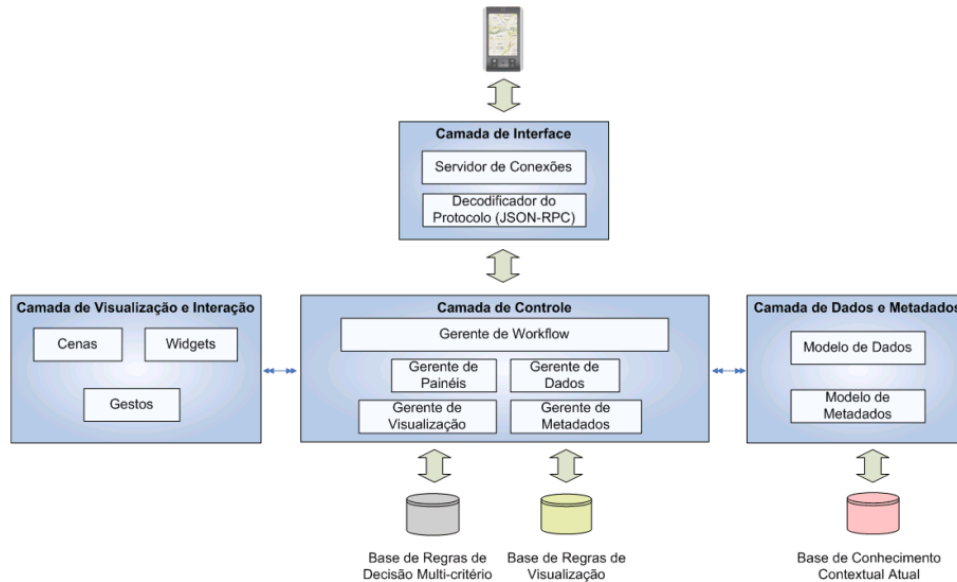


Figura 13 – Arquitetura utilizada por Veiga (2012).



Fonte: adaptada da obra de Veiga (2012).

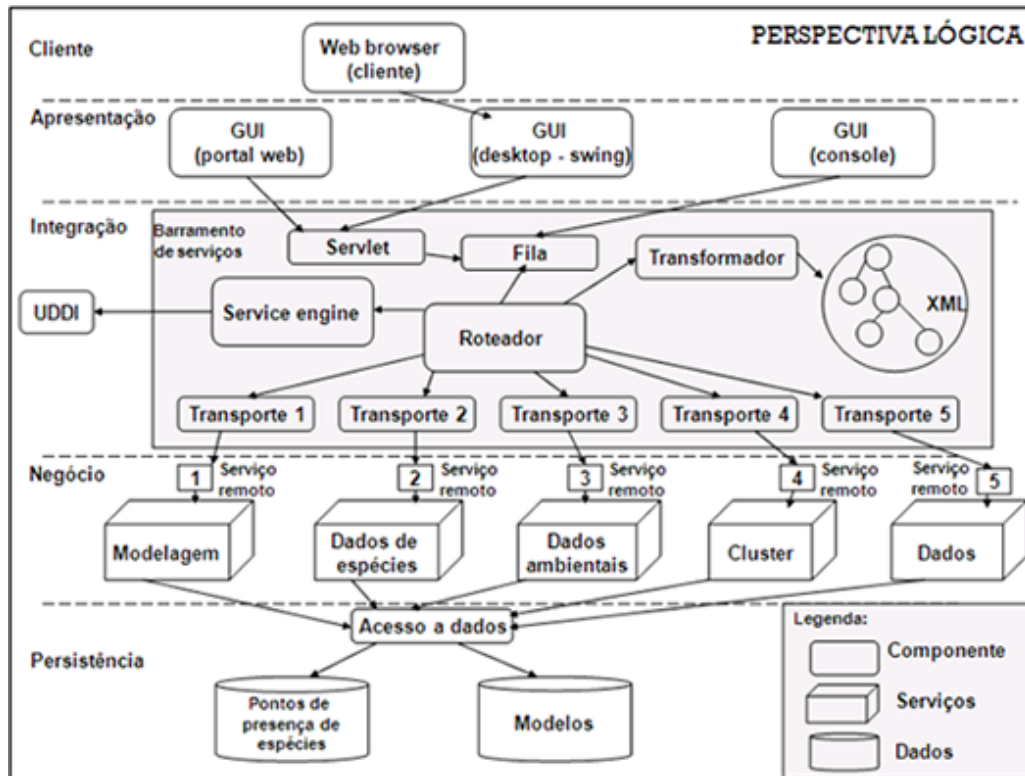
Figura 14 – Arquitetura em camadas da ferramenta WITS.



Fonte: Ferreira (2011).

Santana (2009) em seu trabalho tem como objetivo definir uma infraestrutura orientada a serviços (Figura 15) para a construção de sistemas computacionais para a modelagem de nicho ecológico. Foi feita uma análise para identificar problemas em aberto para modelagem de nicho ecológico. A abordagem arquitetural foi baseada em diversos trabalhos, que foram utilizados os modelos de referência para processamento de objetos distribuídos, a partir dos quais foi possível definir uma arquitetura de referência. Os algoritmos construídos nesse trabalho representam contribuições para a modelagem de nicho ecológico, na medida em que oferecem melhorias reais no desempenho dos algoritmos de modelagem. Como contribuição adicional, está o fato de que esta infraestrutura proposta pode ser aplicada também para endereçar outros problemas, não relacionados ao de modelagem de nicho ecológico.

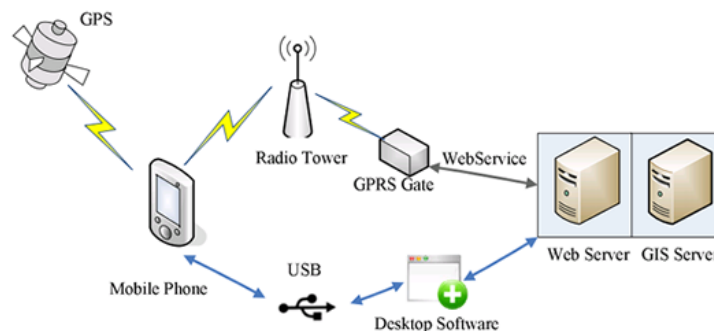
Figura 15 – Visão geral da arquitetura de software para a infra estrutura de serviços para a modelagem de nicho ecológico.



Fonte: Santana (2009).

Para a construção de um SIG para agricultura com processamento em tempo real, Chen et al. (2012) elaborou um modelo de arquitetura (Figura 16) que é a integração entre sistema móvel SIG, sistema *Global Positioning System* (GPS), comunicação *wireless* e outras tecnologias. Esse sistema tem como diferencial gerenciar diversas áreas ao mesmo tempo. A partir das informações obtidas das imagens ocorre a detecção de doenças das plantas como também a fertilidade do solo.

Figura 16 – Modelo da estrutura física feita por Chen et al. (2012).



Fonte: Chen et al. (2012).

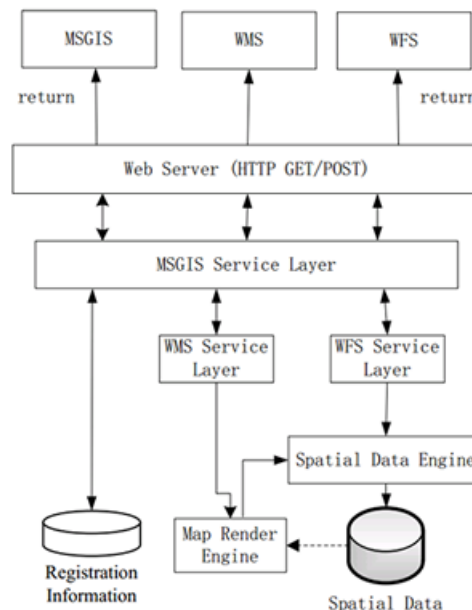
O trabalho feito por Ji (2012) projeta um esquema para uma plataforma integrada

de dutos subterrâneos de gerenciamento de informações baseado em **SOA**. Para a construção da plataforma foi utilizada a tecnologia *web services*, sendo que arquitetura geral é baseada em **SOA**. Assim, diferentes módulos funcionais do sistema são encapsulados em *web services* para a troca de dados. As funções de negócio podem ser configuradas de acordo com os diferentes fluxos de operação. Assim, este resolve conflitos entre compartilhamento de dados e gestão de segurança.

**Katakis (2009)** apresenta um trabalho que pretende criar uma **SOA** para o gerenciamento de recursos computacionais como também atender solicitações mais exigentes de ambiente multi-usuário. A implementação é baseada na arquitetura *Common Object Request Broker Architecture (CORBA)*. Isso por que essa suporta o paradigma de **OO** e a interoperabilidade entre aplicações feitas em diferentes tecnologias/linguagens de programação. Os mecanismos de controle e recursos e armazenamento adicionam escalabilidade e permitem a integração de novas funcionalidades SIG ao sistema.

Com o objetivo de criar uma especificação para gerenciar serviços baseado em **SOA**, **Shen et al. (2010)** propôs uma estrutura (**Figura 17**) interna com duas camadas de serviços para os usuários. Os serviços *Web Map Services (WMS)* e *Web Feature Service (WFS)* esses implementam o compartilhamento de dados e promovem a interoperabilidade entre sistemas heterogêneos. A partir da implementação desta especificação pode ser constatado

Figura 17 – Arquitetura utilizada por **Shen et al. (2010)**.



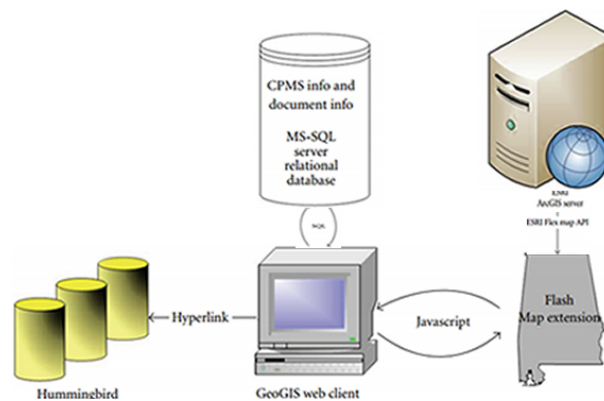
Fonte: **Shen et al. (2010)**.

que a mesma consegue resolver deficiências da complexidade no gerenciamento de serviços, a dificuldade de integração e do gerenciamento dinâmico.

### 3.2.2 SIGs para Engenharia Geotécnica

Os autores [Graettinger, Ryals e Smith \(2011\)](#) realizaram o desenvolvimento de um sistema para facilitar o acesso e armazenamento das informações geotécnicas do estado do Alabama nos Estados Unidos. Esse sistema *web* armazena informações geotécnicas sobre projetos de transporte, tais como dados de superfície, desenhos de construção e informações de projeto. A arquitetura ([Figura 18](#)) é formada por um banco de dados relacional do lado *Back-end*; já o lado cliente é atribuído à gestão de documentos e

Figura 18 – Arquitetura utilizada por [Graettinger, Ryals e Smith \(2011\)](#).



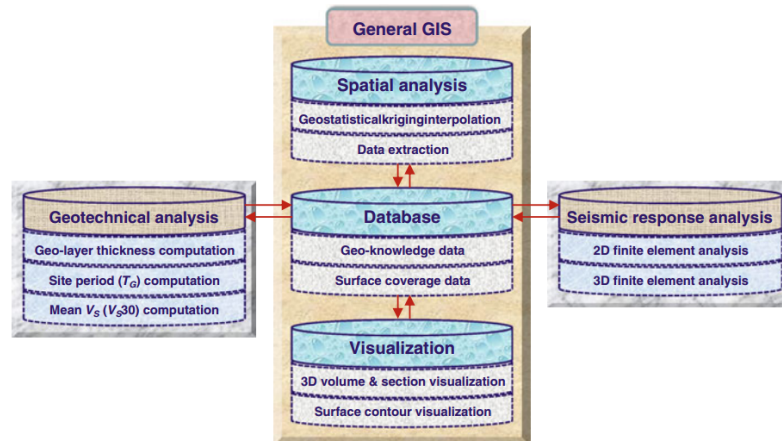
Fonte: [Graettinger, Ryals e Smith \(2011\)](#).

visualização de mapas através do ESRI Flex. O ESRI Flex é uma [API](#) que se comunica com ArcGIS Server para obter as informações geotécnicas. Assim, a incorporação do mapa construído com a ESRI Flex é feita através do JavaScript (*Front-end*).

Com a proposta da construção de um sistema para prevenção de terremotos em uma área urbana na Coreia do Sul, o autor [Sun \(2012\)](#) desenvolveu um sistema capaz de monitorar o comportamento sísmico do solo em tempo real. A arquitetura ([Figura 19](#)) é formada por cinco componentes: [BD](#), Análise Espacial, Análise de Resposta Sísmica, Análise Geotécnica e Visualização. O [BD](#): contém informações sobre as camadas geotécnicas, bem como a cobertura espacial dos cursos de água, edifícios e estradas; Análise Espacial: através dos dados fornecidos pelo [BD](#) é feita a interpolação de dados sobre a área de interesse através da geoestatística; Análise de Resposta Sísmica: para avaliar informações geotécnicas adicionais; Análise Geotécnica: contém módulos para analisar a espessura das camadas geotécnicas e profundidade do leito rochoso; Visualização: apresenta os dados interpolados em duas ou três dimensões.

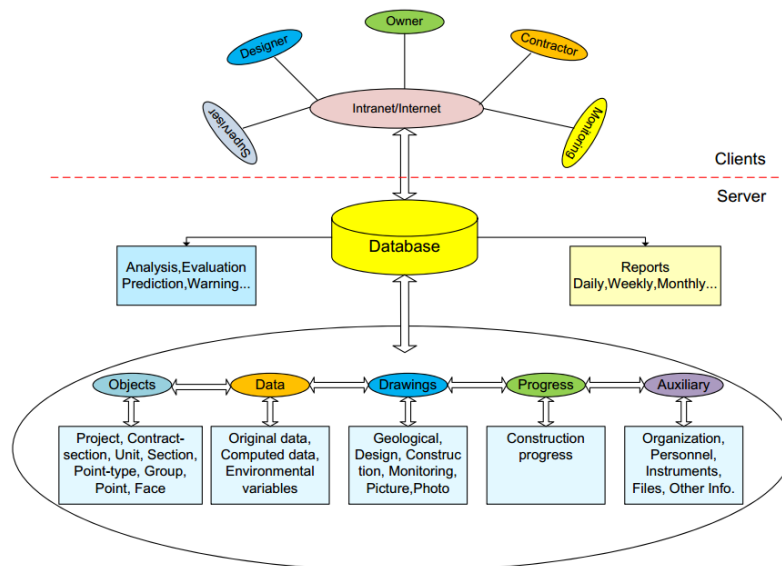
O trabalho de [Wang et al. \(2014\)](#) consiste no desenvolvimento de um sistema de gestão, previsão e alerta para monitoramento geotécnico. O software integra vários tipos de informações, gestão de documentos (através de arquivos) e seu devido processamento, desenhos *Computer Aided design (CAD)*, modelagem de dados e previsão, bem como uma função de alerta precoce. A arquitetura desse sistema é apresentada na [Figura 20](#). A par-

Figura 19 – Arquitetura utilizada por Sun (2012).



Fonte: Sun (2012).

Figura 20 – Arquitetura utilizada por Wang et al. (2014).



Fonte: Wang et al. (2014).

tir da comunicação com o banco de dados estão conectadas diversos módulos destinados a processar as regras de negócio do sistema da seguinte forma: *Objects*: implementação de uma biblioteca para monitoramento de objetos; *Data*: uma biblioteca de dados associada aos objetos de monitoramento; *Drawings*: uma biblioteca de gráficos em que vários desenhos CAD e dados de imagem são armazenados; *Progress*: uma biblioteca que gerencia o progresso de construção de camadas; *Auxiliary*: uma biblioteca voltada as informações auxiliares, relacionadas a informações dos envolvidos na construção, documentos entre outros. De acordo com o autor o sistema possui uma boa integração entre suas funcionalidades. No entanto, o apoio à tomada de decisão pode ser melhorada. Além disso, as funcionalidades do sistema podem ter uma melhor eficácia caso seja incorporado um

cálculo estrutural e módulo de análise numérica.

### 3.3 Lições do capítulo

A partir da análise feita nos trabalhos, se pode constatar que para a construção das aplicações foram utilizadas principalmente o modelo MVC, Camadas e em muitos casos não foi utilizado um padrão de arquitetura. Sendo que essas aplicações foram feitas nas mais diferentes plataformas, como para a Web, Desktop, plataformas móveis entre outras de acordo com a necessidade da solução.

Em alguns casos a comunicação entre diferentes plataformas foi necessária, e foi resolvida por meio da implantação de *web services* que promovem a interoperabilidade necessária. A comunicação distribuída acontece em SIGs por diversos motivos, sendo que um deles pode ser a necessidade do uso de provedor de mapas. Esses serviços são feitos por comunicação via *web services* e organizados em diferentes camadas. Na arquitetura feita por Ferreira (2011), se utilizou uma camada chamada “Camada de Interface” para realizar a comunicação entre um dispositivo móvel e a camada responsável por tratar requisições.

Para o armazenamento dos dados de modo geral nos trabalhos, existe uma camada específica para a persistência dos mesmos. Porém, em alguns casos a construção de mapas é feita internamente pela aplicação, não utilizando um servidor de mapas. Assim são utilizadas extensões de SGBD que são feitas para gerenciar o armazenamento de dados espaciais.

As estruturas apresentadas na subseção 3.2.2 permitem visualizar a importância de um módulo específico para a análise geotécnica. No caso do autor Wang et al. (2014), mesmo tendo sucesso na implantação do sistema, foi destacado a possibilidade de refatoração para incorporar alguma estrutura que dê suporte a análise e processamento geotécnico. Ainda nesse caso, é importante frisar que a adoção de uma arquitetura *web* para esses sistemas não é regra, visto que o acesso aos mesmos foi feito via Intranet (rede local de computadores). No entanto, caso haja a necessidade de disseminar as informações por meio do sistema para outros lugares, não será possível.

De modo geral, partindo do princípio que SIGs possuem uma grande gama de soluções, é visualizado o reuso de soluções por meio de componentes, bibliotecas, *frameworks*, APIs e serviços externos (por exemplo serviços web). No entanto, a organização destas soluções (APIs etc) varia bastante entre as aplicações, ou seja, são inseridas diretamente na camada de apresentação ou na camada de dados, não existindo uma camada específica para agrupar os mesmos.

## 4 Arquitetura Web para SIG

O presente capítulo apresenta o modelo de arquitetura proposto neste trabalho. Na [seção 4.1](#) é apresentada uma visão geral de toda a estrutura do modelo. Em seguida, na [seção 4.2](#) é apresentada a estrutura completa do modelo; e na sequência, na [seção 4.3](#) são descritas as restrições, limitações, integrações e procedimentos para utilização da arquitetura. Para finalizar, na [seção 4.4](#) são apresentadas as lições do capítulo.

### 4.1 Visão Geral

As aplicações *SIGs Web* Geotécnicas constituem uma classe de aplicação, e de forma natural possuem suas peculiaridades. Com uma visão mais ampla de suas principais funcionalidades típicas se tem: 1) a manipulação de dados de georeferenciamento, que promovem a construção de mapas, e 2) o gerenciamento de registros e correlação de informações geotécnicas, característicos da Engenharia Geotécnica.

O construção de mapas permite agregar informações de análise nas mais diversas perspectivas no que se refere ao georeferenciamento. Através dos mapas é possível realizar mapeamentos, identificar lugares, traçar rotas, entre outras coisas; tudo isso, com informações sempre atualizadas ou mesmo em tempo real. Pois esses sistemas estrategicamente trabalham com compartilhamento de informações de forma distribuída.

A visualização dos dados cadastrados em forma de mapas aliado a opção de traçar perfis representativos das propriedades dos solos em análise constitui uma ferramenta de alto valor agregado. Assim, é possível comparar as características geotécnicas de terrenos próximos e prever qual será a melhor solução em termos de engenharia civil para o tratamento do terreno em estudo, ou seja, utilizar a ferramenta na realização de pré-projetos de aterros, edifícios, estradas, etc.

As correlações de dados geotécnicos são feitas através de uma hierarquia de dados. Essa hierarquia consiste em ilhas formadas por ensaios que contém parâmetros geotécnicos. Esses parâmetros constituem uma grande massa de dados, e para que esses sejam organizados, analisados, interpretados, mapeados e guardados é inviável realizar essas tarefas por meios convencionais, e sim de forma automatizada devido aos fatores quantidade e precisão dos resultados. Caso não exista a corretude das informações geradas, essas podem implicar na elaboração de projetos falhos.

Diante desse contexto, o presente trabalho explora a possibilidade de criar um modelo arquitetural para *SIGs web* geotécnicos com uma arquitetura “desenhada” a partir de padrões de projeto como *MVC* e Injeção de Dependências, os quais permitem ter uma

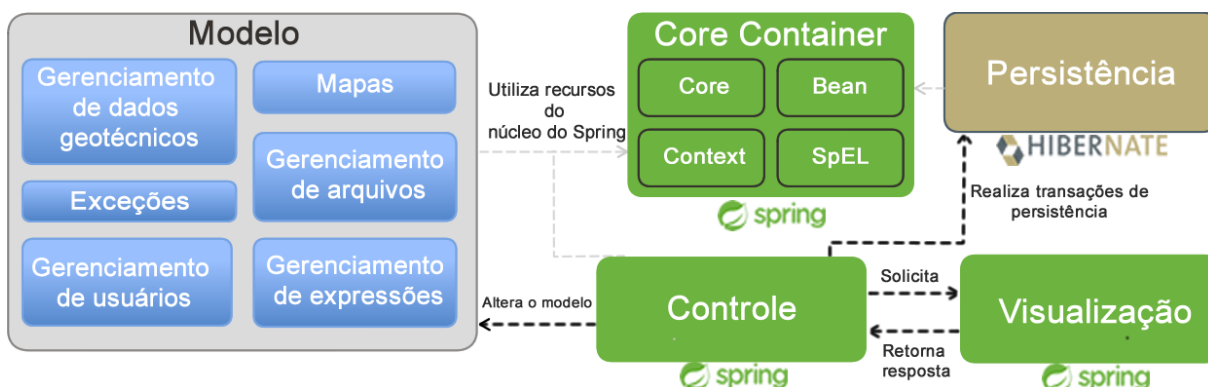
melhor organização e comunicação entre seus módulos. Além disso, a arquitetura permite realizar a conversão de tipos de forma automatizada, tanto no lado cliente (*Front-end*), como no lado servidor (*Back-end*). A partir dessas características, são esperados benefícios plausíveis para uma arquitetura, como o baixo acoplamento (injeção de dependências), alta coesão (definição de módulos e submódulos) e a diminuição da complexidade que implica na diminuição de custos de desenvolvimento e manutenção, reutilização e evolução do sistema.

## 4.2 Modelo Arquitetural

Para a concepção do modelo arquitetural deste trabalho, teve-se como base a utilização e fusão de dois *frameworks*: o *Hibernate* para persistência de dados e o *Spring MVC*, o qual como o próprio nome diz, dispõem de três módulos relacionados (Visualização, Controle e Modelo). Além disso, com o objetivo de identificar responsabilidades e organizar melhor a arquitetura foi realizada a modelagem de domínio, a qual resultou na definição de novas interfaces fornecidas pela arquitetura. Dessa forma, o *framework* apresenta uma série de características que podem apoiar a utilização da infraestrutura proposta através das interfaces disponíveis no modelo.

A arquitetura proposta dispõe de uma infraestrutura formada por interfaces e classes que ao serem implementadas e estendidas permitem a utilização de recursos e a adequação desses a partir do domínio a ser implantado. Como pode ser visto na [Figura 21](#), a arquitetura é formada por cinco módulos, dos quais três (Modelo, Controle e Visualização) representam o padrão MVC, o qual é muito utilizado por aplicações *web*. O módulo Persistência possui o papel de realizar através da comunicação com o módulo Controle as transações de persistência em uma base de dados. Já o módulo *Core Container*, funciona como um núcleo e é responsável pelo suporte aos demais módulos da arquitetura.

Figura 21 – Arquitetura organizada em módulos.

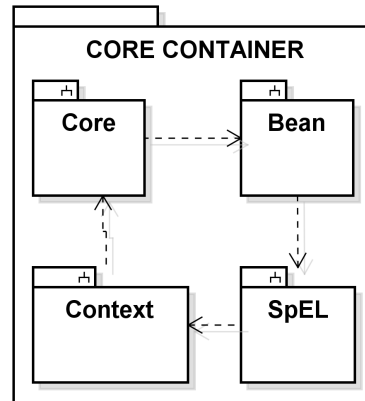




### 4.2.1 Módulo *Core Container*

Este módulo (Figura 22) é uma extensão obrigatória mesmo ao utilizar apenas extensões do *Spring Framework*, como é o caso do *Spring MVC*. Dessa forma, esse módulo

Figura 22 – Módulo Core Container.



é o núcleo do *Spring Framework* onde são implementados os conceitos fundamentais do *framework*: a inversão de controle e a injeção de dependências. A partir desses conceitos, é possível prover o baixo acoplamento entre toda a arquitetura, basicamente através da anotação `@Autowired`. Esse recurso possibilita ao *container* descobrir em tempo de execução quais as dependências que devem ser injetadas de maneira automática.

### 4.2.2 Módulo Visualização

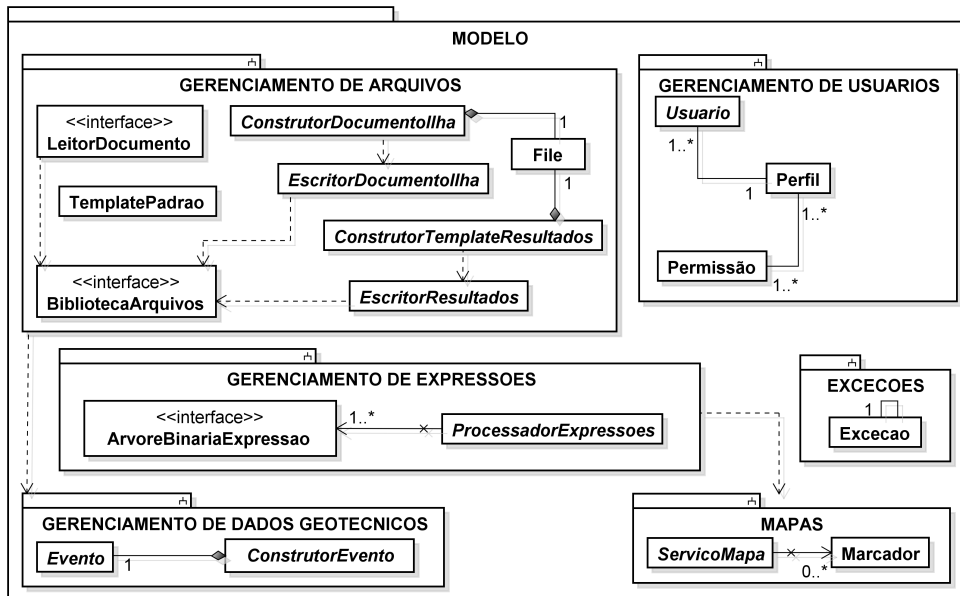
O módulo de **Visualização** é responsável por renderizar e manipular o conteúdo retornado pelo Controle. Por exemplo, uma página Linguagem de Marcação de Hipertexto (**HTML**), pode ser vista como o resultado final desse módulo que é apresentado ao usuário. Esse comportamento acontece via resposta do mapeamento feito pela anotação `@RequestMapping`; sendo que essa, ao ser aplicada a uma função vinculada ao controlador o transforma em uma *action*, isto é, em um método responsável pelo processamento da *Uniform Resource Locator* (**URL**) mapeada. Para recuperar e manipular os recursos disponíveis para a Visualização (**JSP**), podem ser utilizadas além do **HTML**, bibliotecas como *JavaServer Pages Standard Tag Library* (**JSTL**) e **Spring Form**. Ambas, possibilitam manusear objetos de forma simples e transparente, contribuindo para a manutenibilidade da aplicação.

### 4.2.3 Módulo Modelo

O **Modelo** (Figura 23) diz respeito a toda parte do sistema responsável pela lógica de negócio. Ao utilizar o *Spring MVC* de maneira convencional, esse módulo acaba assumindo também transações de persistência com a base de dados. No entanto, essa arquitetura proposta deixa essa responsabilidade isolada em único módulo,

o de Persistência. Visando promover maior coesão entre os módulos, foi criada uma estrutura interna passando a ter as seguintes partes (submódulos): Gerenciamento de Arquivos, Gerenciamento de Dados Geotécnicos, Mapas, Exceções, Gerenciamento de Expressões e Gerenciamento de Usuários.

Figura 23 – Módulo Modelo.



O submódulo Gerenciamento de Arquivos (Figura 24) contém classes que contemplam toda a funcionalidade relacionada a interpretação e construção de arquivos. A interface requerida BibliotecaArquivos, consiste em receber e então promover a utilização de uma biblioteca que possa acessar, modificar ou criar conteúdo através da manipulação de arquivos. Para realizar e organizar as ações de leitura, foi definida a interface LeitorDocumento que visa fornecer dados de um Evento para o ConstrutorEvento realizar a construção. A organização das ações de escrita são definidas pelas classes EscritorDocumentoIlha e EscritorResultados. Com a utilização dessas classes, a escrita é feita respectivamente a partir de dois cenários: 1) construir um evento com dados recuperados do BD, com a classe ConstrutorDocumentoIlha; 2) construir os resultados da plotagem feita em tempo de execução através da classe ConstrutorTemplateResultados. Ambas as classes, são utilizadas diretamente por um controlador.

O submódulo Gerenciamento de Expressões (Figura 25) foi idealizado com a intenção de propor uma estrutura para organizar e processar os elementos que formam uma expressão a fim de obter um resultado. Uma expressão é equivalente a um Parâmetro de Plotagem, o qual possui entre seus elementos operadores matemáticos e parâmetros geotécnicos. Através de uma expressão é possível aplicar fórmulas devidamente embasadas para evidenciar as propriedades dos solos. Para solucionar isso, existe o conceito de Árvore Binária, pois a partir da representação de uma expressão na árvore é possível obter a prioridade dos operadores de forma implícita. No contexto da construção de uma

Figura 24 – Modelo - Gerenciamento de Arquivos.

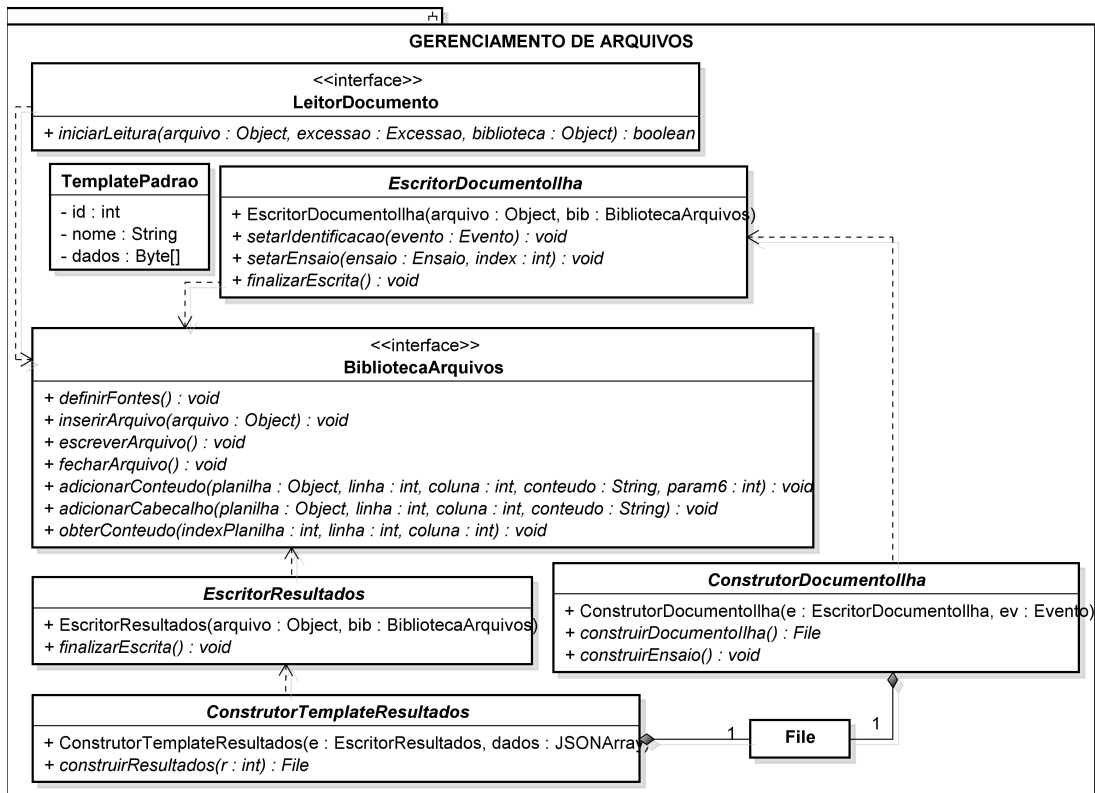
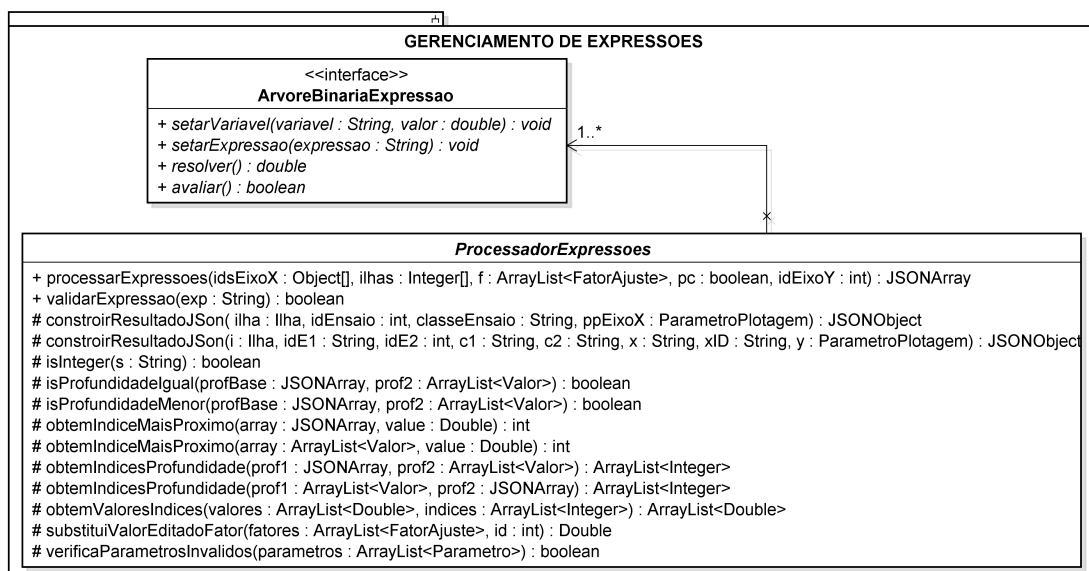


Figura 25 – Modelo - Gerenciamento de Expressões.



expressão aritmética, cada elemento (operando ou operador) vêm a ser um nó da árvore; o operador de menor prioridade fica na raiz e os operandos sempre aparecem nas folhas da árvore.

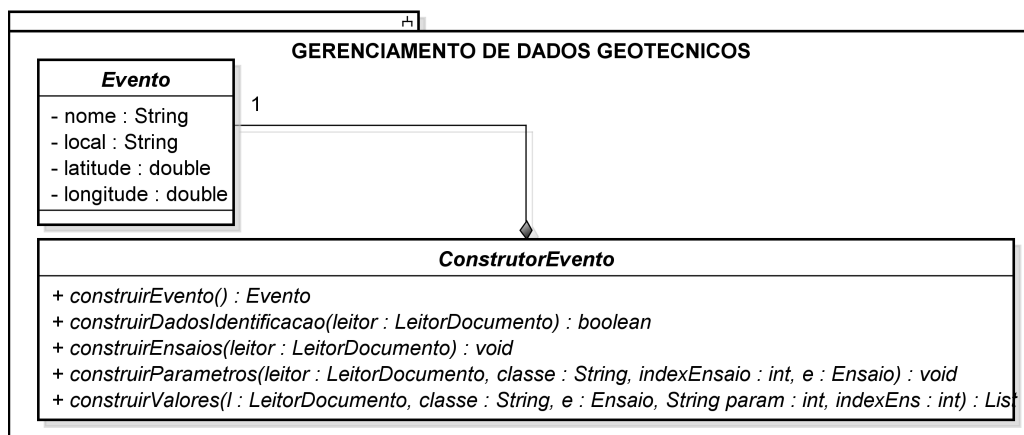
A interface requerida `ArvoreBinariaExpressao` realiza a inclusão de uma biblioteca que manipula expressões através do conceito de árvore binária. Entre suas assinaturas, se espera que essa biblioteca permita inserir elementos separadamente através do

`setarVariavel()`, ou a expressão completa com a função `setarExpressao()`; e então, realizar o processamento com a função `resolver()` ou `avaliar()`.

Para que se faça o uso dessa biblioteca dentro uma estrutura de ações para de fato receber parâmetros (operandos e operadores) e assim processar as expressões, foi definida a classe `ProcessadorExpressoes`. Essa classe, tem a responsabilidade de manipular através da biblioteca `ArvoreBinariaExpressao` os elementos criados em tempo de execução pelo usuário. Para isso, se tem o auxílio de um conjunto de funções que aplicam regras e algoritmos que permitem essa construção de resultados através do método `processarExpressoes()`.

O submódulo `Gerenciamento de Dados Geotécnicos` (Figura 26) foi definido visando dar suporte a definição de estruturas que possibilitam a análise de solo. Assim, a classe `ConstrutorEvento` a partir do método `ConstruirEvento` possibilita a construção de um `Evento` com todos os seus atributos (identificação, ensaios e parâmetros geotécnicos). No entanto, para que essa funcionalidade seja provida, existe uma dependência com o `LeitorDocumento`, que viabiliza a obtenção de dados a partir de arquivos para a construção do `Evento`.

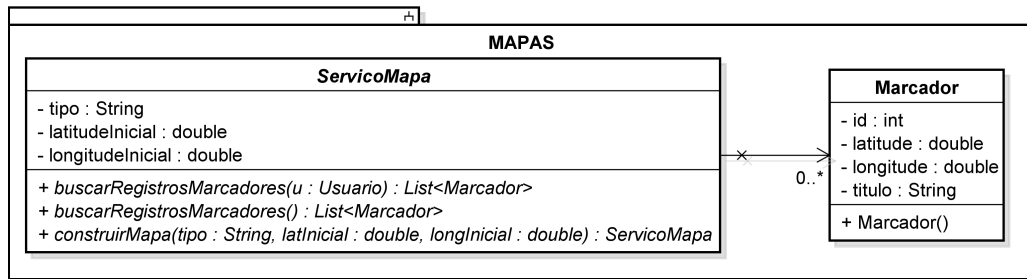
Figura 26 – Modelo - Gerenciamento de Dados Geotécnicos.



A manipulação de mapas em SIGs é extremamente fundamental, pois como visto nos trabalhos relacionados, essa funcionalidade faz parte de sua essência. Para proporcionar a agregação de mapas por meio do recebimento de serviços, foi definido o submódulo `Mapas` (Figura 27), o qual permite por meio da *interface* `ServicoMapa` implementar ações de construção do mapa cujo serviço já está configurado e também manipular ações de busca de marcadores. Um marcador permite realizar a representação de eventos no mapa. Por meio desses, é possível atribuir características de georeferência, como latitude e longitude.

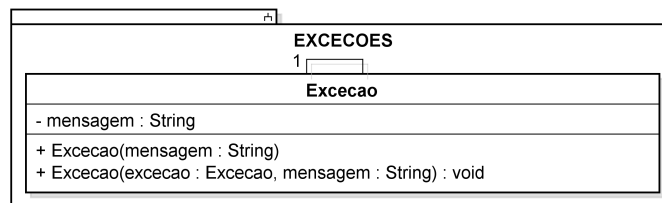
Visando promover uma organização para o disparo de exceções foi definido submódulo `Excecoes`. A classe `Excecao` tem a responsabilidade de encapsular tanto a exceção quanto a informação das causas na linguagem de usuário, também chamada de mensagem.

Figura 27 – Modelo - Mapas.



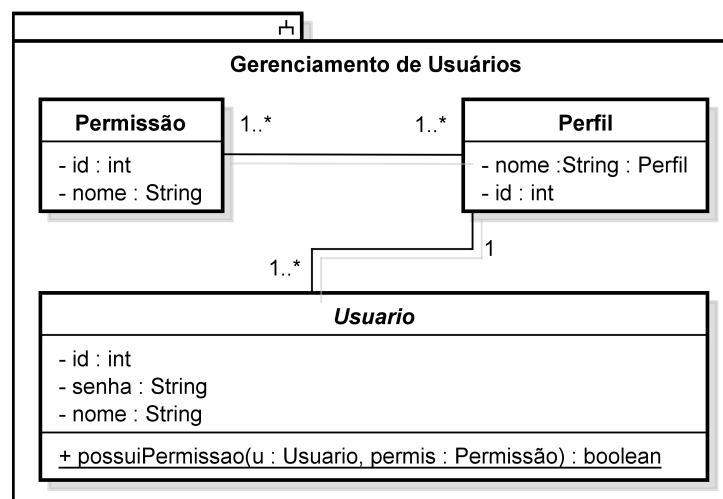
A mensagem é definida em módulos como **Modelo** e **Persistencia** podendo ser levantada no módulo **Controle**. Caso a exceção seja criada, o módulo **Controle** realiza a leitura da mensagem e repassa essa informação para o módulo **Visualização** para ser apresentada ao usuário.

Figura 28 – Modelo - Exceções



Mecanismos para controle de acesso em um sistema são utilizados em aplicações. Isso ocorre não apenas para restringir o acesso ao sistema propriamente dito, mas muitas vezes existe a necessidade de bloquear o acesso de uma ou várias funcionalidades para um determinado usuário. Visando implantar esta política dentro da arquitetura, foi definido o submódulo **Gerenciamento de Usuários** (Figura 29). Esse por sua vez, através da

Figura 29 – Modelo - Gerenciamento de Usuários.



classes **Permissao** e **Perfil** permite que seja definida a hierarquia de níveis de acesso ao

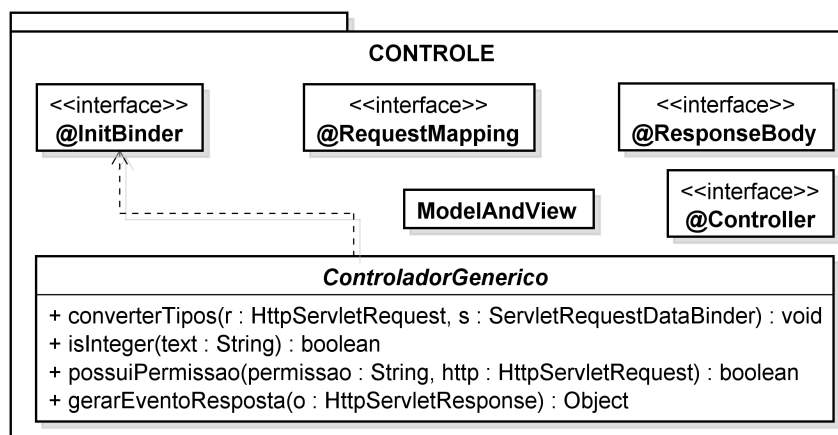
sistema. Por meio dessa estratégia, é possível flexibilizar a atribuição de permissões para a classe `Usuário` que também encapsula atributos e comportamentos.

#### 4.2.4 Controle

O **Controle** possui como responsabilidade principal promover a interação entre os demais módulos com a *Visualizacao*. Quando o usuário clica em um *link* no módulo *Visualizacao* o controlador é acionado. Este transforma os parâmetros de entrada para um formato que seja compatível com a interface disponibilizada pelo módulo *Modelo*, cujo resultado do processamento é recebido pelo controlador, o qual modifica caso seja necessário e em seguida o envia ao módulo de *Visualizacao* para que seja recebido pelo usuário.

A [Figura 30](#) apresenta a estrutura que deve ser implementada para criação de controladores e seus respectivos mapeamentos de requisições. Para definir um controlador deve ser estendida a classe `ControladorGenerico` e implementar a anotação `@Controller`. A classe `ControladorGenerico` disponibiliza ações importantes como a conversão de ti-

Figura 30 – Módulo Controle.



pos e a verificação de permissões de um usuário para acessar uma funcionalidade no módulo de *Visualização*. A interface `Controller` ao ser implementada permite informar ao `DispatcherServlet` do *SpringMVC* que essa classe passa a ser um controlador. Para o mapeamento de uma requisição dentro do controle, deve ser definido um método que implemente a interface `@RequestMapping`.

Após resolvidas questões de tratamento de requisições (mapeamento), também deve ser decidido como responder a essas requisições. O *Spring MVC* permite realizar isso de várias maneiras, a mais comum delas é retornar um objeto do tipo `ModelAndView`, o qual encapsula informações como a página a ser renderizada e objetos do Modelo que são injetados no módulo *Visualizacao*. Além dessa possibilidade de retorno também podem ser feitos redirecionamentos através do prefixo `"redirect:"` que indica ao *Spring MVC*

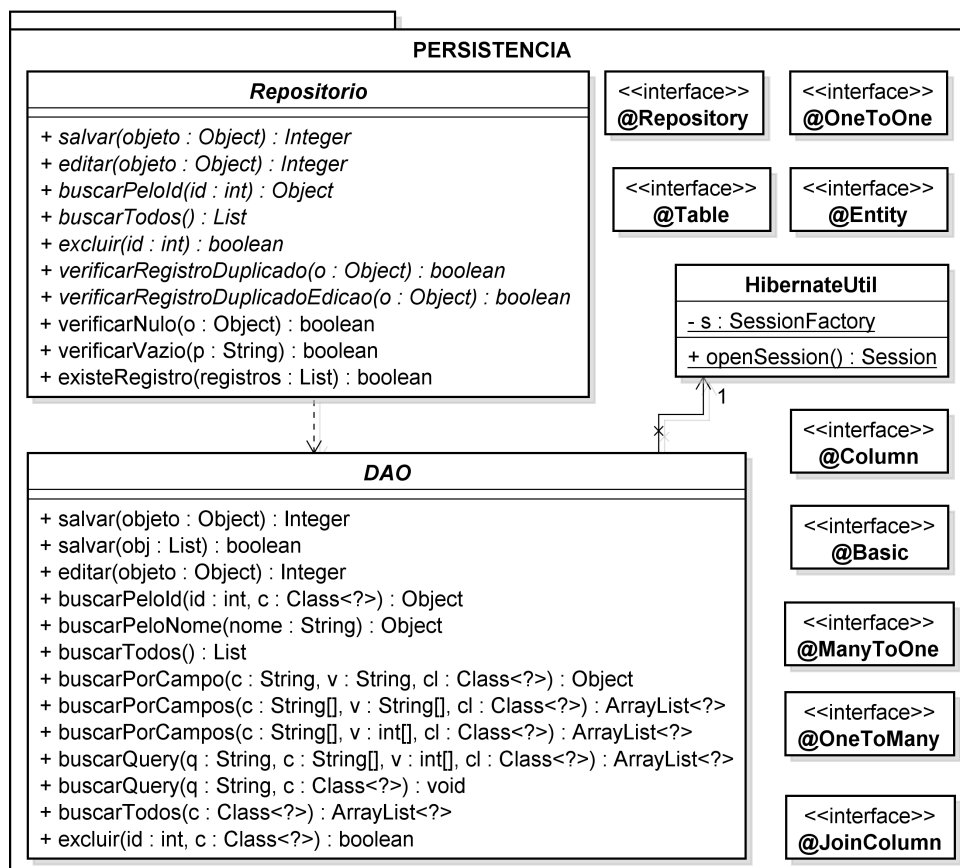
fazer uma nova requisição (*request*) para um novo endereço. A terceira possibilidade de retorno é feita pela interface `@ResponseBody`; essa, permite transformar qualquer objeto (ou conjunto de objetos) em um retorno do tipo *JavaScript Object Notation* (**JSON**).

#### 4.2.5 Persistência

A grande maioria das aplicações tem a necessidade de persistir seus dados, e a “Persistência” é um dos conceitos fundamentais no desenvolvimento de aplicações (BAUER; KING, 2007). A persistência consiste em possibilitar uma aplicação guardar seus dados em forma de registros em um banco de dados relacional. Para isso acontecer, a maneira convencional é utilizar apenas a linguagem **SQL** por meio de transações. Porém, existe outra forma de implementar uma estratégia de persistência em um sistema, através da persistência automatizada, como visto na subseção 2.3.2, o **ORM Hibernate** traz vários benefícios como produtividade no desenvolvimento, conversão automática de tipos e manutenibilidade.

A estrutura do módulo Persistência pode ser visualizada na Figura 31, a qual possui como destaque as classes a serem extendidas `Repositorio` e `DAO`, e as interfaces fornecidas pelo *Hibernate*. Para implementação dessa estrutura, devem ser organizadas

Figura 31 – Módulo Persistência.



duas frentes, o mapeamento de objetos e a organização de métodos responsáveis por

executar as transações entre a aplicação e o **BD**. Para fazer o mapeamento de uma classe a ser persistida, primeiramente devem ser implementadas as interfaces **Entity** e **Table**, as quais informam que a classe é reversível para uma entidade do banco de dados (ou vice-versa) e está sendo representada pela tabela configurada. Para finalizar o mapeamento, as interfaces **Column**, **Basic**, **ManyToOne**, **OneToMany**, **OneToOne**, **JoinColumn** servem para resolver questões de relacionamentos e equivalências entre atributo(s) da classe para com a(s) coluna(s) da tabela.

A organização dos métodos responsáveis por executar as transações é feita por meio das classes **Repositorio** e **DAO**. A primeira (**Repositorio**) tem contato direto com o resto do sistema, e por vez organiza os retornos de registros feitos pela **DAO**. Dessa forma, a **DAO** fica responsável pela implementação das consultas e transações do **Hibernate**. E por último, a classe **HibernateUtil** realiza a conexão com a base de dados e juntamente com a construção de sessões que encapsulam todo o processo da transação feita entre a aplicação e o **BD**.

### 4.3 Implementação do Modelo

Com o uso das *Expression Languages* (**ELs**) como **JSTL** e **Spring Form** no módulo de Visualização, é possível facilitar significativamente a manipulação de objetos vindos do módulo **Controle**. A grande vantagem da **EL** é o acesso direto aos objetos disponíveis nos vários escopos existentes sem a necessidade de instanciar os objetos que permitam tal acesso, muito menos usar *scriptlets*<sup>1</sup>. Porém, em alguns casos o **Spring Form** não permite mapear algum elemento **HTML** de um formulário, de modo a encapsular um objeto completamente para enviar para o **Controle**. Neste caso, o **JSTL** participa de forma complementar ajudando a mapear esses objetos para a construção de um formulário, mas não resolvendo o encapsulamento completo (todos os atributos) do objeto para envio.

O submódulo **Gerenciamento de Arquivos** possibilita a leitura de arquivos independente da sua extensão. Para isso, existe uma interface que permite a inclusão de uma biblioteca que seja adequada para realizar ações de leitura e escrita desses artefatos. No entanto, se faz necessário fazer um mapeamento dos principais pontos de possíveis exceções de modo a contribuir para a robustez do sistema.

O conceito de Árvore Binária é principal protagonista do submódulo **Gerenciamento de Expressões**, o qual a partir da utilização de uma biblioteca (*Binary Tree*) realiza a construção lógica dos elementos que compõe uma expressão aritmética. Para definir uma expressão é possível inserir elementos como operandos e operadores de forma dinâmica. A biblioteca por si só realiza uma validação desses elementos e faz o processa-

---

<sup>1</sup> Os *scriptlets* são códigos Java que podem ser colocados em qualquer parte de uma página **JSP**, mas que seguem a delimitação de caracteres especiais (**JUNIOR; FORTES, 2015**).



mento da expressão em tempo real ao usuário. Um ponto a ser melhorado nesse tipo de processamento é a necessidade de filtrar e eliminar alguns caracteres especiais que fazem parte dos nomes de operandos informados pelo usuário.

Como visto anteriormente, o núcleo do *Spring MVC* é o *Dispatcher Servlet*, que é o responsável por gerenciar todas as requisições recebidas pela aplicação. Sendo assim, para a utilização do *Spring MVC* é necessário configurar um arquivo `web.xml`<sup>2</sup> como é mostrado no código abaixo.

```

1 <servlet>
2   <servlet-name>DispatcherServlet</servlet-name>
3   <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
4   <init-param>
5     <param-name>contextConfigLocation</param-name>
6     <param-value>/WEB-INF/spring/spring-servlet.xml</param-value>
7   </init-param>
8   <load-on-startup>1</load-on-startup>
9 </servlet>

```

A tag `<servlet>` é usada para declarar um *servlet* disponibilizado pela aplicação. Assim que iniciado o *servlet*, o mesmo irá carregar um *container* do *Spring* configurado via [XML](#) por um arquivo chamado `WEB-INF/[nome do servlet]-servlet.xml`. A configuração desse arquivo é apresentada no código abaixo:

```

1 <mvc:resources mapping="/recursos/**" location="/recursos/" cache-period="0"/>
2 <mvc:annotation-driven>
3   <mvc:message-converters>
4     <bean class="org.springframework.http.converter.json.
5       MappingJackson2HttpMessageConverter"></bean>
6   </mvc:message-converters>
7 </mvc:annotation-driven>
8 <context:annotation-config />
9   <context:component-scan base-package="Controle"/>
10  <context:component-scan base-package="Modelo.GerenciamentoArquivos"/>
11  <context:component-scan base-package="Modelo.GerenciamentoDadosGeotecnicos"/>
12  <context:component-scan base-package="Modelo.GerenciamentoUsuarios"/>
13  <context:component-scan base-package="Modelo.GerenciamentoExpressoes"/>
14  <context:component-scan base-package="Modelo.Excessoes"/>
15  <context:component-scan base-package="Modelo.Mapas"/>
16  <context:component-scan base-package="Persistencia"/>
17 <bean id="viewResolver" class="org.springframework.web.servlet.view.
18   InternalResourceViewResolver" >
19   <property name="prefix">
20     <value>/Visualizacao/</value>
21   </property>
22   <property name="suffix">
23     <value>.jsp</value>
24   </property>
25 </bean>
26 <bean id="multipartResolver" class="org.springframework.web.multipart.commons.
27   CommonsMultipartResolver">
28   <property name="maxUploadSize" value="5000000"/>
29 </bean>

```

<sup>2</sup> O `web.xml` é um arquivo de configuração onde são definidos parâmetros de inicialização, níveis de autenticação e autorização, mapeamento de *servlets*, entre outras coisas.

Como pode ser visto nesse arquivo, a partir de instruções são definidos os módulos da arquitetura. A instrução `<bean id="viewResolver">`, define a implementação do módulo Visualização basicamente através das propriedades *prefix* e *suffix*. O prefixo informa qual o início do caminho que leva o arquivo `JSP` a ser renderizado. Início esse, que corresponde ao módulo Visualização definido em `<value>/Visualizacao/</value>`. Já o sufixo, diz respeito à extensão do arquivo. A instrução `<mvc : resources ...>` permite que seja definido um pacote para acomodar todas arquivos de diversas linguagens *Front-end* e recursos como imagens a serem utilizadas no módulo Visualização.

Além da Visualização, para a definição do restante da arquitetura é utilizada a instrução `<context:component-scan ...>` que permite a definição dos módulos e submódulos através do parâmetro `base-package`. Nesse momento são definidos todos os módulos e submódulos que compõe a arquitetura: `Modelo.GerenciamentoArquivos`, `Modelo.GerenciamentoDadosGeotecnicos`, `Modelo.GerenciamentoExpressoes`, `Modelo.Mapas`, `Controle`, `Modelo.GerenciamentoUsuarios`, `Persistencia`, `Modelo.Excessoes`.

## 4.4 Lições do Capítulo

Neste capítulo foi apresentado o modelo arquitetural para ser implantado na construção de aplicações *SIGs web*, sendo caracterizada como uma arquitetura de domínio específico. Foram apresentadas todas as partes que compõe os módulos em detalhes, de forma a expor informações e procedimentos de utilização das interfaces definidas. Através das configurações feitas para a agregação do *Spring MVC* junto à arquitetura, sabe-se é possível realizar a expansão da arquitetura de maneira simples. Ainda na linha de facilidades, é possível realizar os mapeamentos do *Hibernate* de forma bastante produtiva e organizada, principais características desse *ORM*.

Do ponto de vista de trabalhar com diferentes tipos de dados; essa arquitetura permite que uma informação seja capturada pelo módulo `Visualizacao` e feita a construção de um Objeto ainda nesse módulo. Para então, ser processada pelo `Controle` e posteriormente ser persistida em um *BD Relacional*. Isso demonstra o grande potencial que essa arquitetura proporciona no manejo de conversão de tipos através de *ELs* na Visualização, `@InitBinder` e `@ResponseBody` no `Controle` e as interfaces fornecidas pelo *Hibernate* na *Persistência*.

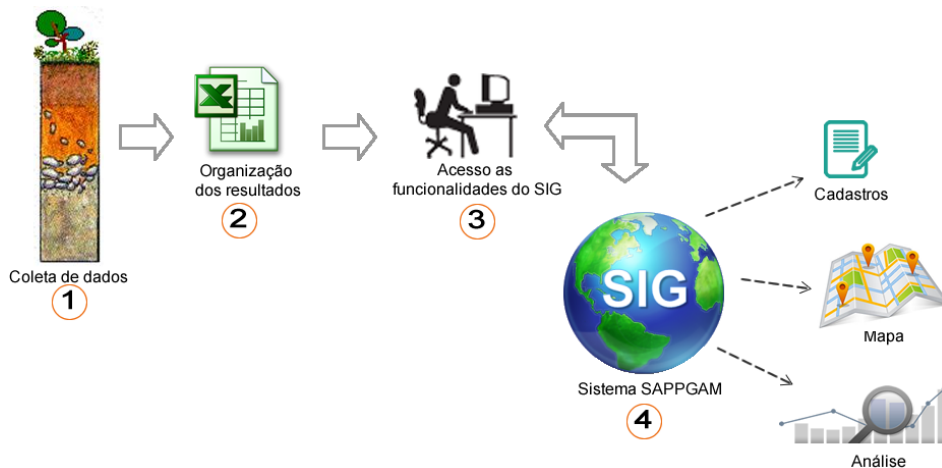
## 5 Estudo de Caso

O presente capítulo apresenta a o desenvolvimento de um estudo de caso que implementa a arquitetura proposta no presente trabalho. Inicialmente na [seção 5.1](#) é apresentada a contextualização do estudo de caso. Na [seção 5.2](#) são apresentadas a etapa de análise, onde foram definidos os requisitos e o escopo do sistema, e a etapa de projeto, onde são apresentados artefatos que possibilitam ter uma visão estática e comportamental da arquitetura. Na [seção 5.3](#) é apresentada a implementação do estudo de caso e algumas tecnologias que foram agregadas a aplicação. A Verificação e Validação é feita na [seção 5.4](#) onde é feito um panorâma geral dos testes e o experimento realizado. Para finalizar, na [seção 5.5](#) são apresentadas as lições do capítulo.

### 5.1 Visão Geral

Visando promover a implantação da arquitetura proposta no presente trabalho, foi realizado o desenvolvimento do Sistema para Análise e Processamento de Parâmetros Geotécnicos das Argilas Moles Brasileiras (SAPPGAM), apresentado na [Figura 32](#). Esta

Figura 32 – Principais atividades dentro do contexto do sistema.



ferramenta permite ao usuário organizar os parâmetros geotécnicos oriundos de ensaios de campo e laboratório no *template* padrão desenvolvido em arquivo com extensão xls. Após a inclusão dos resultados dos ensaios neste *template*, juntamente com os dados de localização do local em estudo e de suas coordenadas geográficas, é possível realizar o cadastro de todas as informações no sistema *web*. O sistema localiza o local em estudo através de um mapa, armazena as informações no BD e possibilita a manipulação dos dados, disponibilizando as opções de visualização dos parâmetros de maneira simples (parâmetro x

profundidade), ou então, através de correlações entre os diferentes parâmetros cadastrados pelo usuário.

Para a visualização dos parâmetros e correlações disponíveis, o usuário pode plotar de forma conjunta resultados de diferentes locais que são visualizados através de gráficos. Os gráficos gerados, podem ser exportados do sistema automaticamente através de figuras em formato JPEG ou semelhante. Caso o usuário queira aplicar outras ferramentas de análise nos dados gerados, é possível fazer a exportação dos parâmetros utilizados para gerar o gráfico requerido em formato xls.

Existe também a possibilidade de extração de todos os dados cadastrados no [BD](#) do sistema, isso possibilita que o usuário selecione um dos *templates* cadastrados e faça o *download* das informações com a mesma organização que ele foi cadastrado inicialmente.

## 5.2 Análise e Projeto

Para desenvolver o sistema [SAPPGAM](#), primeiramente foi realizado o levantamento de *requisitos*<sup>1</sup>. Durante essa etapa, técnicas como reuniões, entrevistas e prototipação foram feitas para a eliciação de requisitos junto aos *stakeholders*<sup>2</sup>. A partir dessa definição inicial de requisitos, foram feitas análise quanto a sua viabilidade e então um refinamento e melhor definição dos requisitos.

Com base nos requisitos definidos foi criado o Diagrama de Casos de Uso [UML](#) ([Figura 33](#)). A seguir, são apresentados alguns exemplos dos principais casos de uso de forma expandida ([Tabela 1](#) e [Tabela 2](#)), os demais podem ser visualizados no [Apêndice A](#).

Como pode ser visualizado no diagrama da [Figura 33](#), o ator “Gerenciador” reúne todos os aspectos que o ator “Visitante” possui. Em um nível mais alto na hierarquia, o ator “Administrador” acumula todas as propriedades de “Gerenciador” e ainda de “Visitante”. Desse modo, o “Administrador” além de seus casos de uso que estão relacionados diretamente, também irá poder “Visualizar Mapa”, “Plotar Gráfico”, “Exportar Arquivo Padrão”, “Cadastrar Ilha”, “Exportar Ilha” e “Exportar Parâmetros de Plotagem”.

A atividade de projeto tem por objetivo propor a solução para o problema investigado na fase de análise. Os artefatos de projeto devem ser compatíveis com arquitetura utilizada, assim, devem estar alinhados quanto as restrições e limitações impostas.

Para se ter uma melhor visualização e entendimento do diagrama de classes de

---

<sup>1</sup> Um requisito é uma condição ou capacidade que deve ser alcançada ou estar presente em um sistema ou componente de sistema para satisfazer um contrato, norma, especificação ou outro documento formalmente imposto [IEE Standard 610.12-1990].

<sup>2</sup> Um *stakeholder* de um sistema é uma pessoa ou uma organização que tem uma influência (direta ou indireta) nos requisitos de um sistema. ([POHL; RUPP, 2012](#)).

Figura 33 – Diagrama de Casos de Uso.

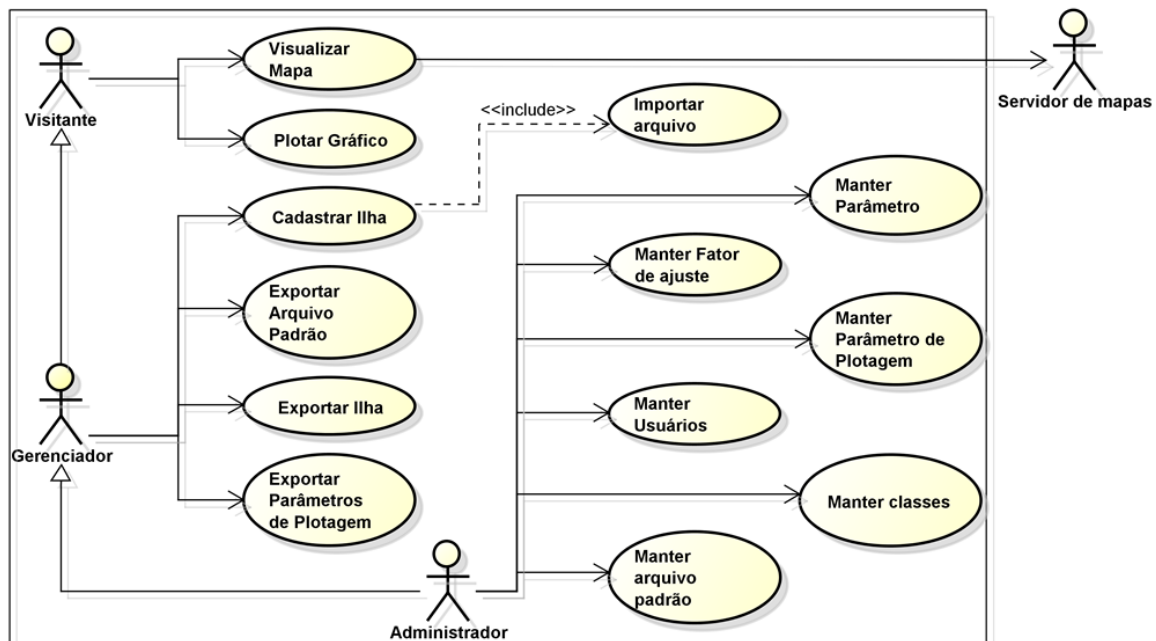


Tabela 1 – Caso de Uso Visualizar Mapa

---

**UC01 Visualizar Mapa**


---

**Ator Principal:** Visitante

**Pré-condição:**

1. O sistema obtém o serviço externo para construção do mapa fornecido por um servidor de mapas.
2. O sistema tem armazenado um ou vários registros de ilhas.

**Fluxo Principal:**

1. O sistema obtém as ilhas que estão armazenadas.
2. Para cada ilha, o sistema obtém as informações básicas para apresentação no mapa (latitude, longitude, nome, endereço, código, referência, data de execução, executor).
3. O sistema inicia a renderização do mapa na tela principal juntamente com as representações das ilhas encontradas anteriormente.
4. O sistema finaliza a renderização do mapa.

**Fluxo Alternativo:**

- 1.a. O sistema realiza a busca por ilhas e não obtém registros.
  - 1.a.1. O sistema apresenta a mensagem: “Não foram encontradas ilhas”.
- 4.a. O usuário seleciona uma ou mais ilhas.
  - 4.a.1. Para cada ilha selecionada o sistema recupera o identificador.
- 4.b. O usuário realiza a navegação pelo mapa.
  - 4.b.1. O servidor de mapas atualiza o mapa em tempo real.

**Exceções:**

- a. A sessão do usuário expira. O sistema apresenta a mensagem: “Você não tem permissão para acessar esta funcionalidade”.
    - a.1. O usuário é redirecionado para a tela de login.
  - b. O usuário tenta acessar outra página alterando o endereço da URL do navegador.
    - b. O sistema apresenta a mensagem: “Você não tem permissão para acessar esta funcionalidade.”
- 

implementação, o mesmo foi fragmentado e apresentado a partir de seus módulos e submódulos. Visando a distinção de classes e interfaces, as coloridas de amarelo correspondem a implementação, as classes em branco são da arquitetura.

Tabela 2 – Caso de Uso Cadastrar Ilha

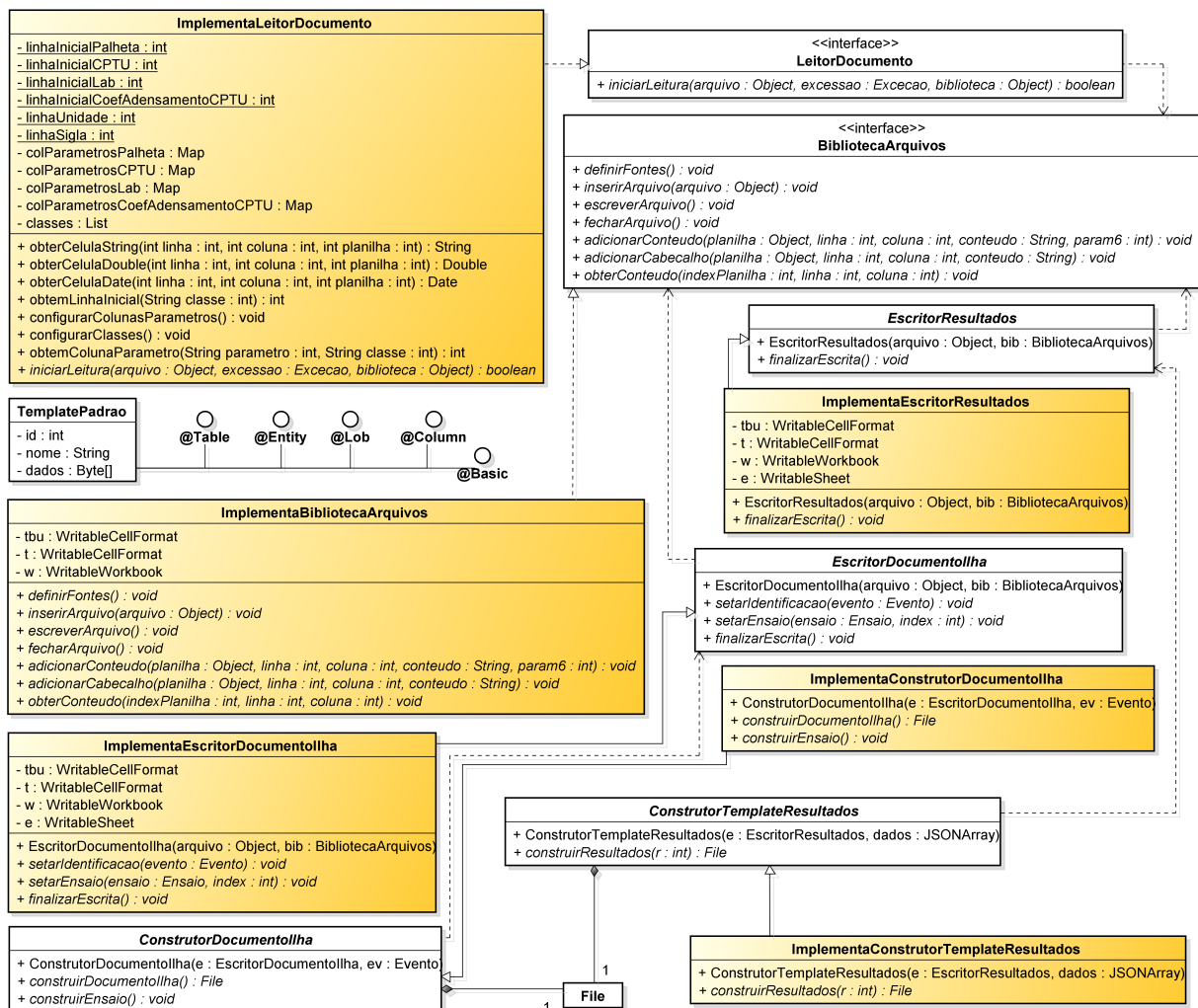
---

UC02 Cadastrar Ilha
<p><b>Ator Principal:</b> Administrador</p> <p><b>Pré-condição:</b></p> <ol style="list-style-type: none"> <li>1. O sistema tem armazenado um ou vários registros de parâmetros correspondentes a ilha a ser cadastrada.</li> <li>2. O sistema possui um mapeamento das informações contidas no arquivo importado.</li> </ol> <p><b>Fluxo Principal:</b></p> <ol style="list-style-type: none"> <li>1. O administrador seleciona a opção de cadastro de ilha.</li> <li>2. O sistema apresenta a tela de seleção de arquivo.</li> <li>3. O administrador seleciona um arquivo e o envia para o sistema.</li> <li>4. O sistema carrega o arquivo e o caso de uso Importar Arquivo é iniciado.</li> <li>5. É realizado o registro dos básicos da ilha (id, nome, executor, data de execução, endereço, latitude, longitude, cidade e referência).</li> <li>6. É realizado o registro dos ensaios encontrados, sendo que para cada ensaio são registrados os parâmetros e os valores correspondentes.</li> <li>7. O sistema finaliza o registro da ilha.</li> <li>8. O sistema apresenta a seguinte mensagem: “Registro efetuado com sucesso!”.</li> </ol> <p><b>Fluxo Alternativo:</b></p> <ol style="list-style-type: none"> <li>1.a. O administrador acessa a tela de cadastro da ilha e clica em cancelar. <ol style="list-style-type: none"> <li>1.a.1. O sistema apresenta a tela principal.</li> </ol> </li> <li>2.a. Na tela de seleção de arquivo, o administrador não seleciona arquivo e clica em cancelar. <ol style="list-style-type: none"> <li>2.a.1. O sistema apresenta a tela de cadastro de ilha.</li> </ol> </li> <li>5.a. O administrador clica em cancelar. <ol style="list-style-type: none"> <li>5.a.1. O sistema apresenta a tela principal.</li> </ol> </li> </ol> <p><b>Exceções:</b></p> <ol style="list-style-type: none"> <li>3.a. O administrador não seleciona um arquivo. <ol style="list-style-type: none"> <li>3.a.1. O administrador clica em abrir.</li> <li>3.a.2. O sistema apresenta a mensagem: “Arquivo não selecionado”.</li> </ol> </li> </ol>

---

Inicialmente a [Figura 34](#) apresenta a implementação de classes que envolvem o módulo Gerenciamento de Arquivos. A classe `ImplementaBibliotecaArquivos` é a raiz desse módulo; ela tem como principal parâmetro de entrada o arquivo recebido de uma classe `Controle`, a partir disso, são implementadas as assinaturas e feitas as configurações para manipular o arquivo. Por meio da utilização dessa classe, outras três classes se beneficiam de seus serviços: a) a classe `ImplementaLeitorDocumento` por meio de mapeamento e informações sincronizadas com registros do `BD` recupera os dados do *template*, apresentado no [Apêndice A](#). Esses dados coletados, são utilizados na sequência pela classe `ImplementaConstrutorEvento` para construir uma ilha a ser persistida na base de dados; b) a classe `ImplementaEscritorResultados` recebe a biblioteca e uma estrutura de dados para realizar a escrita e construção de um arquivo para retornar ao usuário via *download*, para satisfazer o caso de uso `Exportar Parâmetros de Plotagem`; c) a classe `ImplementaEscritorDocumentoIlha` tem o papel de realizar a construção de uma Ilha (conjunto de ensaios geotécnicos) em arquivo para disponibilizar o mesmo na funcionalidade `Exportar Ilha`. A classe `TemplatePadrao` implementa algumas *annotations* de mapeamento do `Hibernate`, com destaque para a anotação `@Lob`, a qual permite que seja persistido um arquivo no `BD`, pois ela guarda uma sequência de bytes ou caracteres de

Figura 34 – Diagrama de classes - Módulo Gerenciamento de Arquivos.

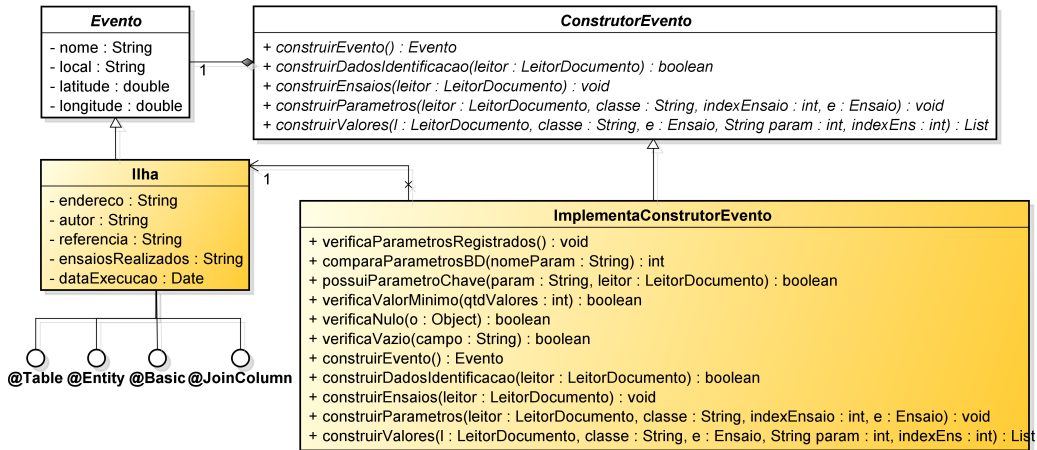


grande dimensão.

A classe `ImplementaConstrutorEvento` tem como responsabilidade de organizar todas as informações recebidas do leitor de arquivos e então construir uma ilha. Este processo é realizado no momento em que o usuário for cadastrar uma ilha no sistema, no caso de uso `CadastrarIlha`. Na [Figura 35](#) são apresentadas as classes relacionadas a essa funcionalidade.

Para implementar o caso de uso `Plotar Gráfico` se fez necessário a utilização do módulo `GerenciamentoDeExpressoes`. A classe `ImplementaProcessadorExpressoes` ([Figura 37](#)) através do método `ProcessarExpressoes` realiza a comparação e busca de valores relacionados aos resultados de ensaios geotécnicos, para então montar e calcular os resultados por meio da classe `ImplementaArvoreExpressao`. De posse de todos esses resultados, é organizada uma estrutura de dados que é retornada para o Controle responsável por repassar para a renderização do gráfico. A classe `ParametroPlotagem` possui entre seus atributos uma expressão formada por elementos que podem ser tanto

Figura 35 – Diagrama de classes - Módulo Gerenciamento de Dados Geotécnicos.



operadores matemáticos como também parâmetros geotécnicos, todos junto formam uma expressão a ser calculada.

Figura 36 – Diagrama de classes - Módulo Gerenciamento de Expressões (Parte 01).

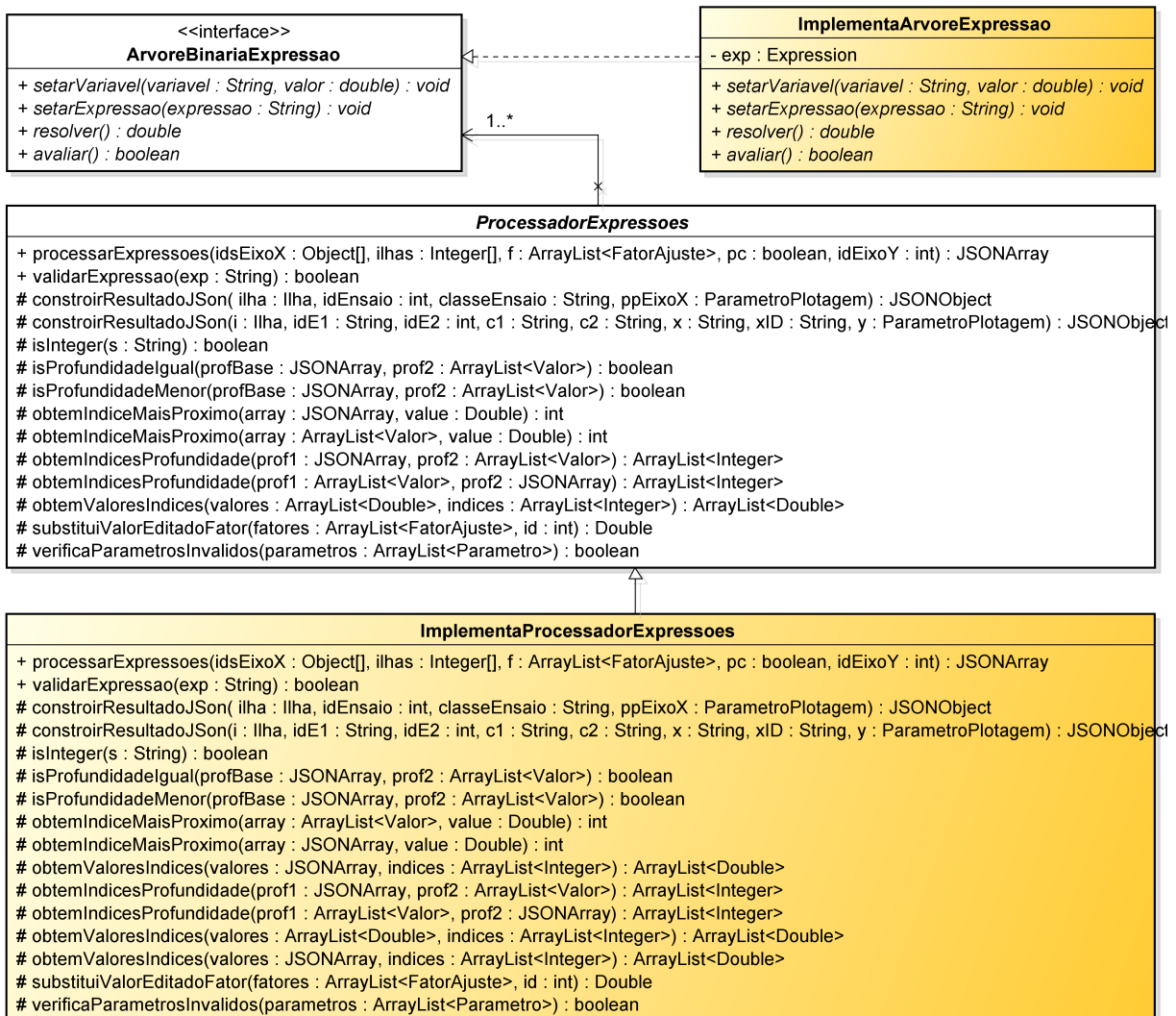
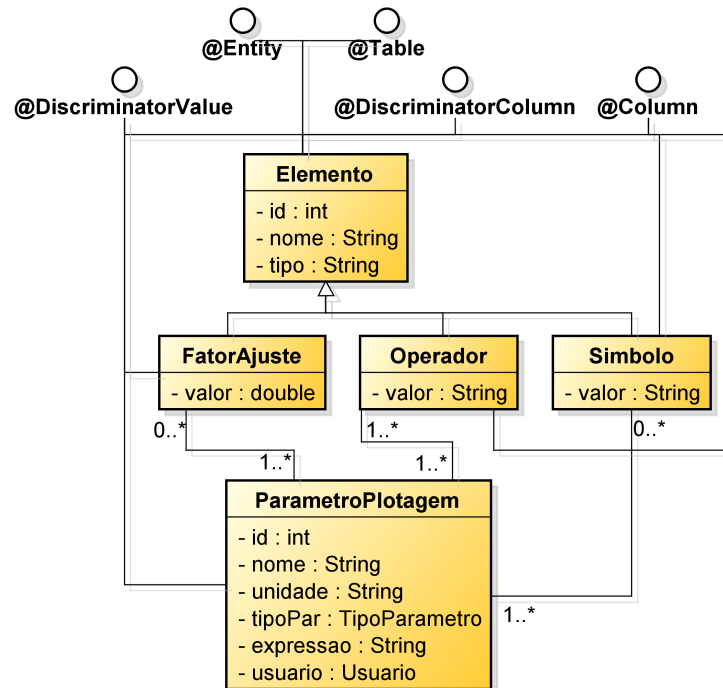


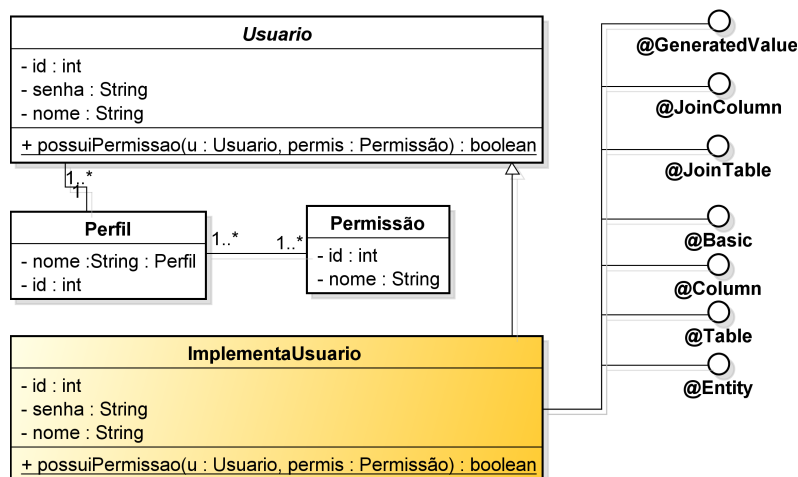


Figura 37 – Diagrama de classes - Módulo Gerenciamento de Expressões (Parte 02).



Como foi apresentado no diagrama de casos de uso (Figura 33) os usuários possuem um direcionamento específico para diversas funcionalidades; isso é decidido baseado no perfil em que esse usuário se encaixa. Conforme visto na Figura 38 são necessárias algumas *annotations* do *Hibernate* que possibilitam unir tabelas do banco de dados. Neste caso,

Figura 38 – Diagrama de classes - Módulo Gerenciamento de Usuários.



para a classe **Perfil** foi utilizado a seguinte configuração ORM para mapear uma lista de permissões:

```

1 @Entity
2 @Table(name="perfil")
3 public class Perfil implements Serializable {
4 {...}
5 @ManyToMany
6 @JoinTable(name="perfil_permissao", joinColumns=

```

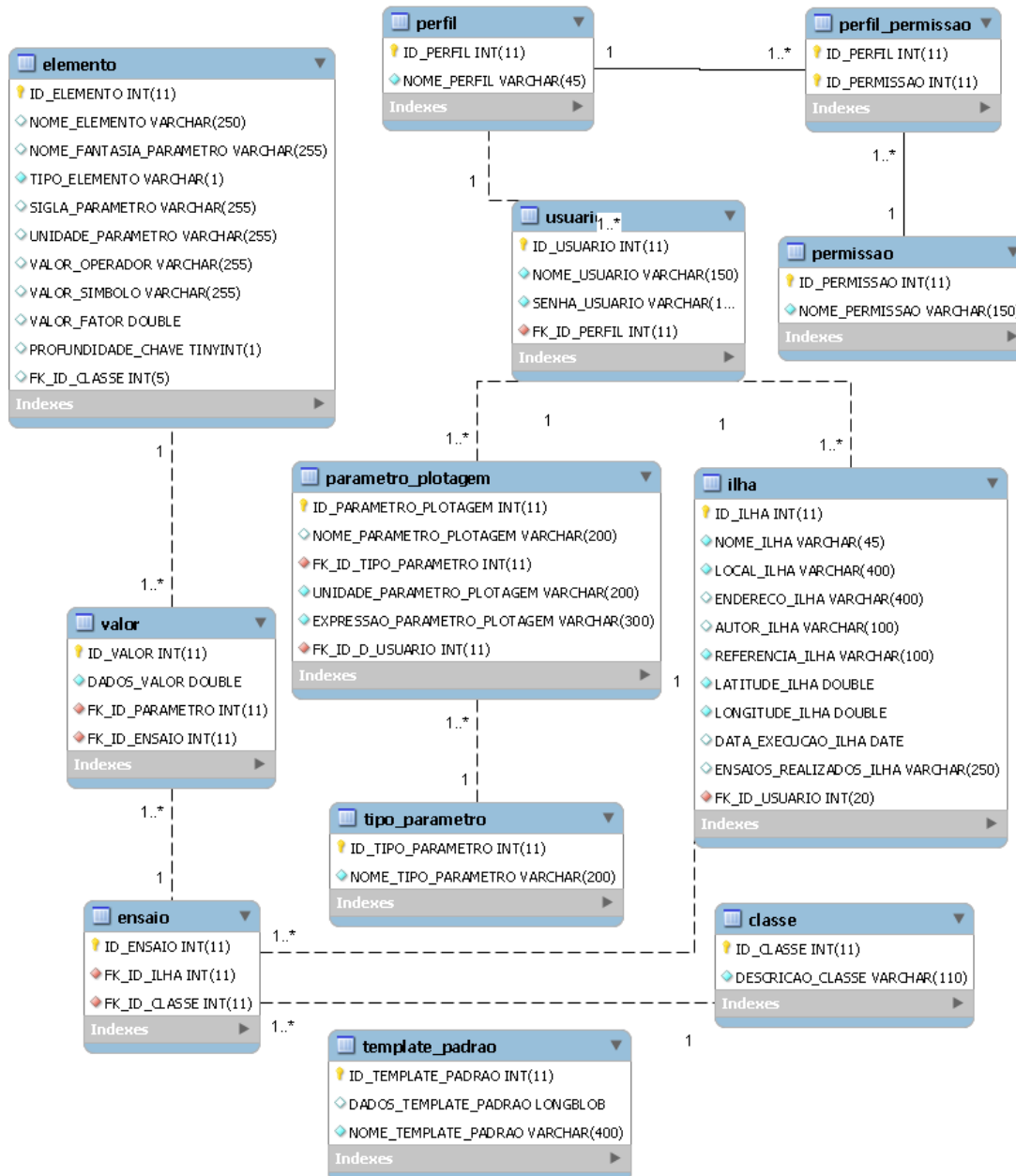
```

7  {@JoinColumn(name="ID_PERFIL")}, inverseJoinColumn=
8  {@JoinColumn(name="ID_PERMISSAO")})
9  private List<Permissao> permissoes; {...}

```

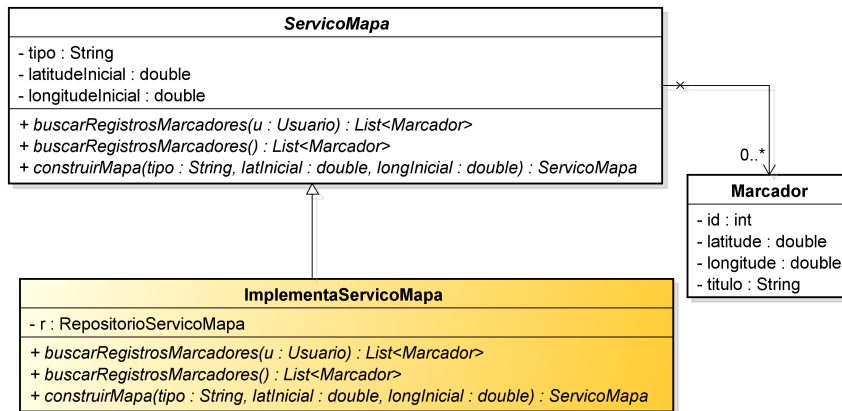
Com este mapeamento, é possível persistir uma lista de permissões na tabela `perfil_permissao` a partir dos identificadores das tabelas `perfil` e `permissao`, apresentados na Figura 39.

Figura 39 – DER utilizado no projeto do sistema.



A funcionalidade de apresentar o mapa no sistema para poder mostrar exatamente a localização das ilhas cadastradas no sistema, foi projetada através do submódulo MAPAS (Figura 40). Esse por sua vez, permite representar as ilhas dentro de um mapa por meio da classe `Marcador`. Essa possui entre outros, os atributos de georeferenciamento de uma ilha. Para a construção desses marcadores, é utilizada a classe `ImplementaServicoMapa` que

Figura 40 – Diagrama de classes - Módulo Mapas.



através da classe `RepositorioServicoMapa` recupera as informações de georeferência das ilhas e faz a devida construção. Além disso, por meio da classe `ImplementaServicoMapa` é responsável pela construção da resposta que implica na renderização do mapa no módulo `Visualizacao`.

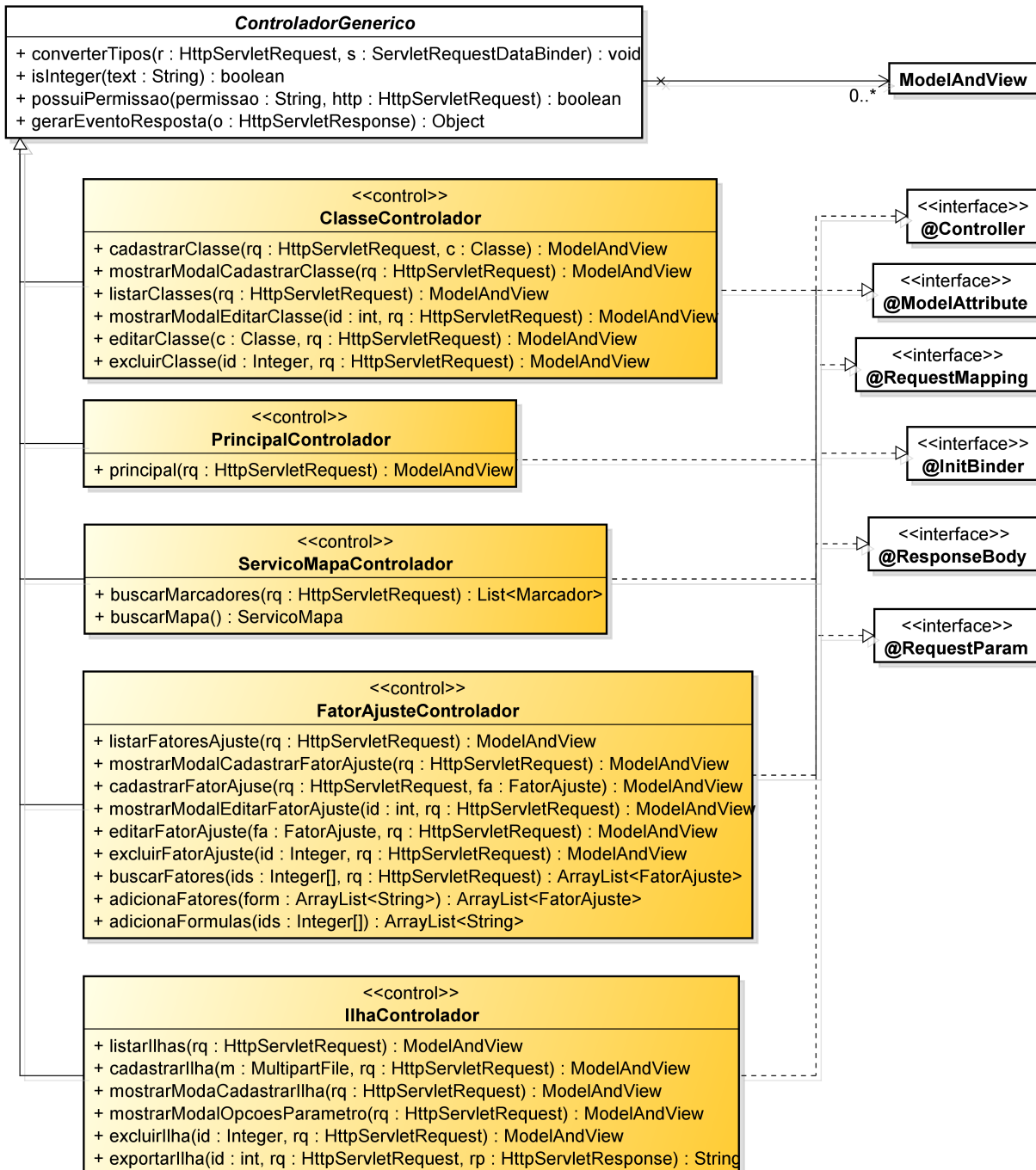
O diagrama de classe do módulo `Controle` é apresentado na [Figura 41](#) e [Figura 42](#).

Figura 41 – Diagrama de classes - Módulo Controle (Parte 01).



As classes `IlhaControlador`, `ParametroControlador`, `UsuarioControlador`, `ClasseControlador` e `FatorAjuste` possuem funcionalidades parecidas dentro de seu con-

Figura 42 – Diagrama de classes - Módulo Controle (Parte 02).

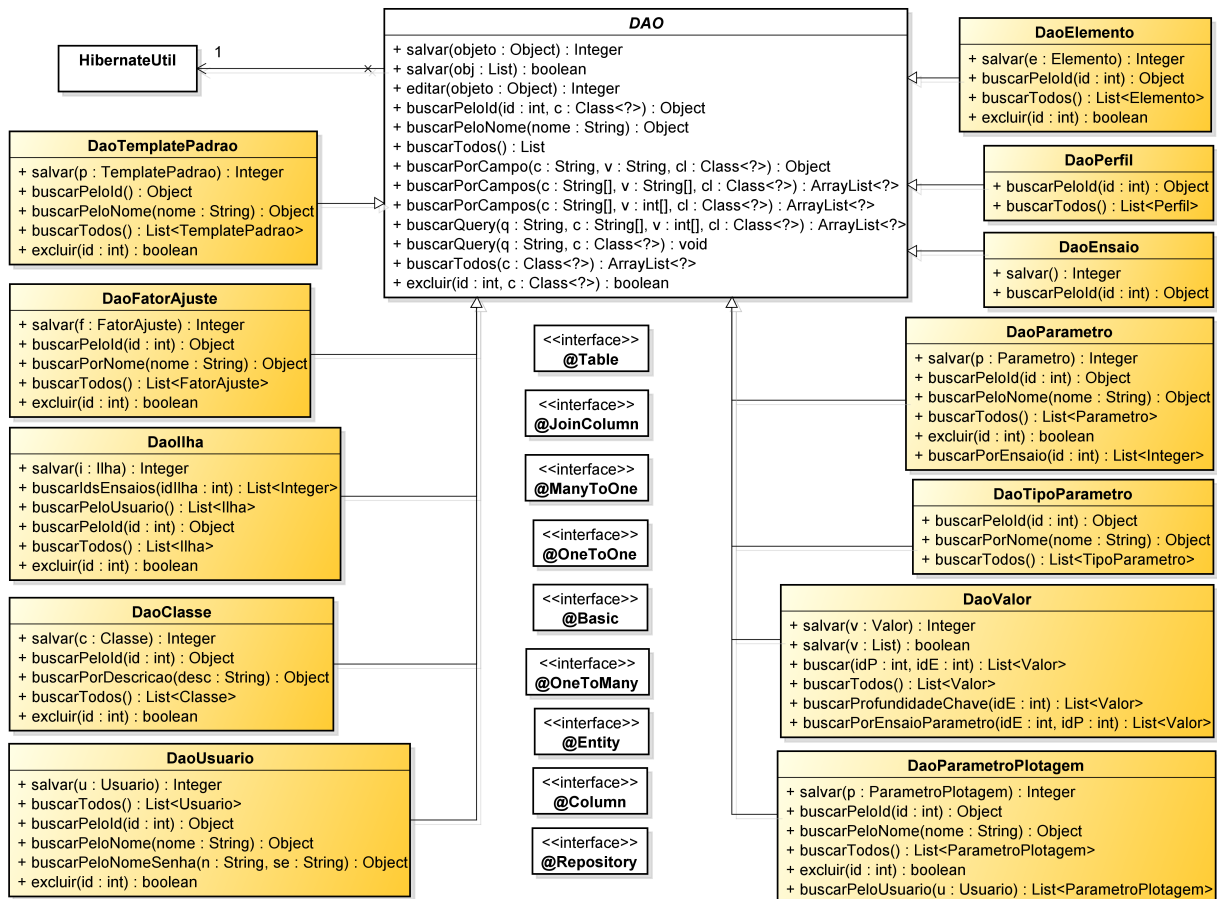


texto. No caso, apresentar formulários de cadastros e listagem, e encaminhar solicitações de persistência. Sempre verificando as permissões do usuário logado por meio do método `possuiPermissao()` que através do parâmetro `HttpServletRequest` recupera a sessão do navegador e obtém suas permissões. Já as demais classes, possuem outras possibilidades de retorno, como por exemplo, a classe `ServicoMapaControlador` ao receber a solicitação de registros de marcadores, consegue retornar via método um `List<Marcador>` que é convertido para `JSON` pelo `@ResponseBody` que faz isso sem nenhuma complicação.

ção. O `@ResponseBody` também pode retornar respostas do tipo `boolean`, neste caso isso foi aplicado no retorno de uma validação de expressão matemática feito pelo método `validarExpressao()` na classe `ParametroPlotagemControlador`.

O projeto das classes responsáveis pela persistência de dados no sistema é apresentado na [Figura 43](#) e [Figura 44](#).

Figura 43 – Diagrama de classes - Módulo Persistência (Parte 01).

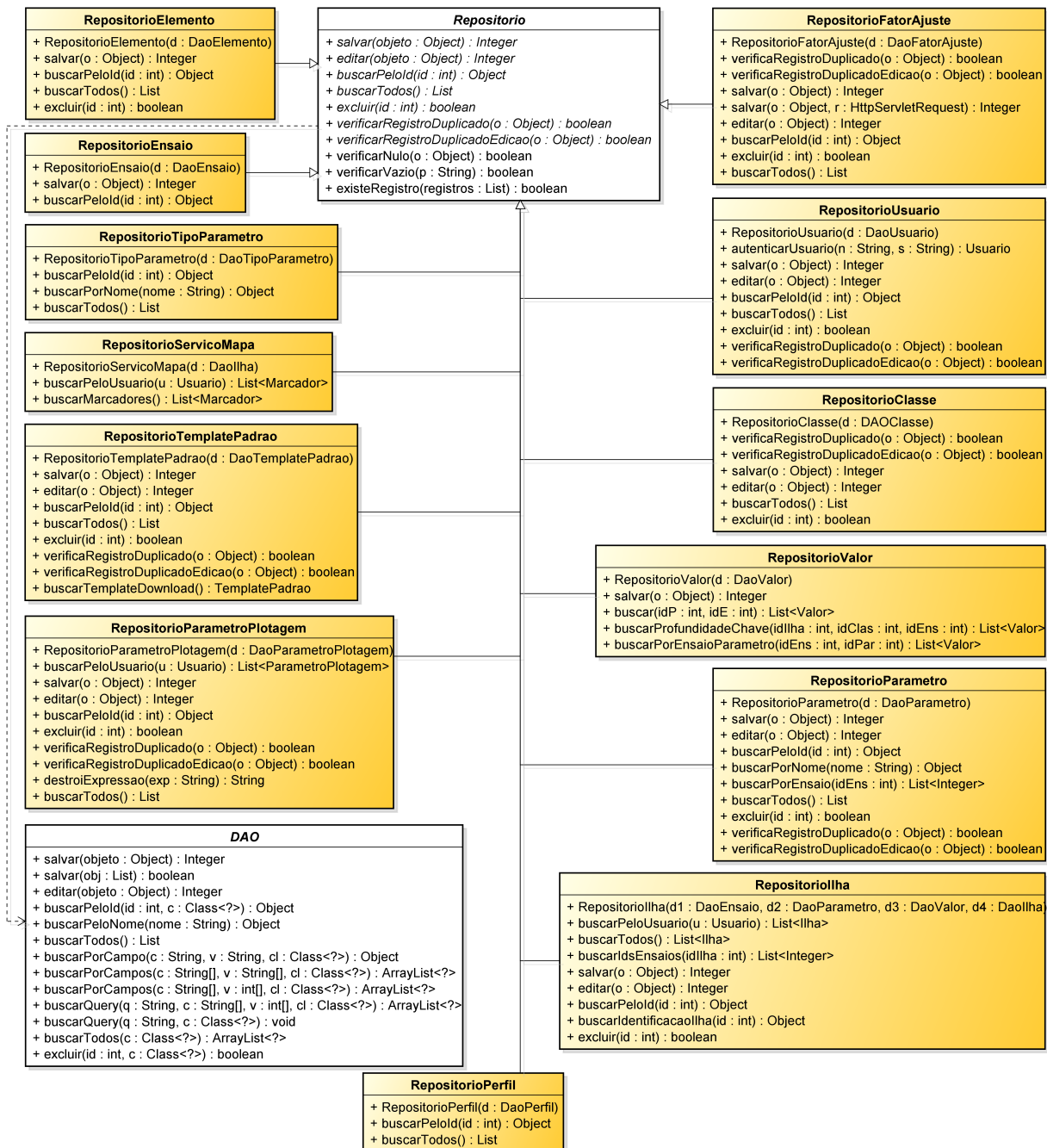


Baseado nas classes `Repository` e `DAO` que fornecem seus métodos e assinaturas, foi possível projetar de maneira muito eficiente e realizar o registro de dados e recuperá-los; pois, através do `Repository` é necessário apenas informar como parâmetro um objeto (`Object`), ou conjunto de objetos a serem persistidos, sendo que o tipo dos mesmo já era especificado diretamente na respectiva `DAO`.

## 5.3 Implementação

Para a implementação do sistema `SAPPGAM`, além da arquitetura proposta neste trabalho também foram utilizadas soluções como a técnica `AJAX`, a biblioteca `JQuery`, a `API Google Maps` e o `SGBD MySQL` visando implantar estruturas e funcionalidades

Figura 44 – Diagrama de classes - Módulo Persistência (Parte 02).



específicas para o domínio. A forma como foi definida a apresentação do mapa no sistema, impacta na idealização de uma estrutura de telas que sejam sobrepostas ao mapa, ou seja, o comportamento do sistema é afetado visando manter o mapa sempre carregado e o carregamento e navegação de telas sobre o mesmo. Para permitir esse comportamento, o sistema utilizou o **AJAX**. O **AJAX** trata-se de uma técnica de carregamento de conteúdos em uma página web com uso de **XML**, **HTML**, **JSON**, **TXT** ou qualquer outra linguagem de marcação ou programação que seja capaz de ser recuperada de um servidor. Assim, essa técnica permite que as telas do sistema sejam operadas sem a necessidade de recarregar

a página, sendo alterados apenas os conteúdos de elementos gráficos específicos de uma página. Porém para utilizar o [AJAX](#) de uma forma “crua” são necessárias várias linhas de código *JavaScript*. No entanto para simplificar o uso do [AJAX](#), existem bibliotecas *Front-end* que conseguem encapsular funcionalidades e simplificar a sua aplicabilidade. Dessa maneira, foi utilizada a biblioteca *jQuery* que possui entre outras, as funções para requisições [AJAX](#), como: `$.get{}`, `$.post{}`, `$.getJSON{}`, `$.getScript{}`, `$.load{}` e a `$.ajax{}`; cada uma delas permite fazer uma requisição ao servidor aplicando regras bem específicas. A seguir é mostrado um exemplo de requisição em que também é utilizado o método `submit()` que permite escutar a submissão de formulários.

```
1 $('form_cad_ilha').submit(function(event) {
2     mostrarCarregando();
3     $.ajax({
4         url: "<c:url value="/cadatrarilha"/>",
5         data: new FormData(this),
6         processData: false,
7         contentType: false,
8         type: "POST",
9         success: function(data) {
10             $('#conteudoModal').html(data);
11             esconderCarregando();
12             });event.preventDefault();
13 });
```

Para representar as ilhas em uma mapa foi utilizado o serviço provido pelo *GoogleMaps*, entre suas funcionalidades estão a possibilidade de operar o mapa em várias perspectivas, incluir rotas, marcadores (*markers*), calcular distâncias, entre outras coisas. Para agregar este serviço ao sistema foi feita a seguinte configuração que permite informar a versão a ser utilizada a uma chave para recebimento do serviço:

```
1 <script type="text/javascript" src="https://maps.googleapis.com/maps/api/js?v=3.exp&key=
    A1zaSyAqFpSbY2aUXW9ngG3E9h2JwprbzLpjHT8" ></script >
```

Com o uso do *GoogleMaps* neste sistema, foram utilizadas figuras(ícones) em três cores para representar os diferentes estados de cada ilha no mapa (selecionada, não selecionada e mostrar identificação). Para viabilizar isso, foram utilizados os *markers*, os quais ao serem criados recebem do servidor a localização (latitude, longitude) e o identificador da ilha. A tela de visualização do mapa no sistema é apresentada na [Figura 45](#).

Outras funcionalidades que não envolvem a visualização de mapa também podem ter destaque, como criação de Parâmetros de Plotagem e a apresentação dos gráficos que permitem ao usuário fazer uma análise técnica sobre as correlações desses dados. A [Figura 46](#) apresenta a tela para cadastro de parâmetros de plotagem, a qual tem em sua organização elementos gráficos que permitem que o usuário faça a construção de uma expressão envolvendo parâmetros geotécnicos que posteriormente serão calculados pelo sistema. Essa funcionalidade envolveu uma estrutura significativa para tratamento de eventos e requisições ao servidor, devido a quantidade de possibilidades disponíveis ao

Figura 45 – Tela de visualização do mapa no sistema.



Figura 46 – Tela de cadastro de Parâmetro de Plotagem no sistema.

usuário. Dentre elas, a busca pelos valores atribuídos a cada parâmetro e a função para avaliar a expressão antes de registrá-la.

Para realizar a plotagem de gráficos com as correlações de parâmetros, são necessários dois passos. 1) a seleção de parâmetros de plotagem (Figura 47): permite que o usuário escolha correlacionar parâmetros do eixo x com parâmetros do eixo y de um gráfico, ou apenas correlacionar vários parâmetros do eixo x tendo como a profundidade como eixo y; 2) após aplicar suas correlações, o sistema apresenta as informações em um gráfico (Figura 48), que por sua vez, também permite que esses dados sejam exportados para o usuário.



Figura 47 – Tela de seleção de parâmetros para plotar o gráfico no sistema.

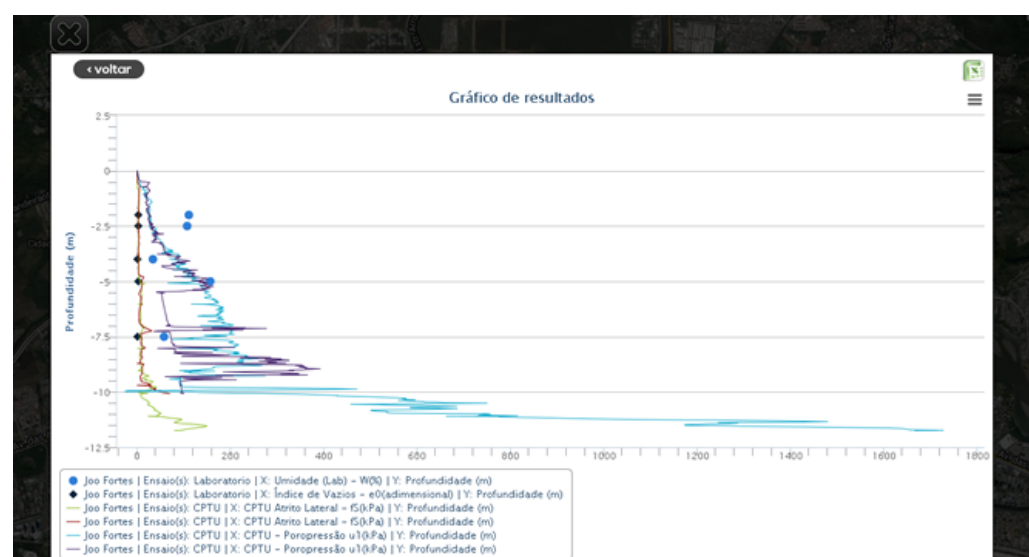
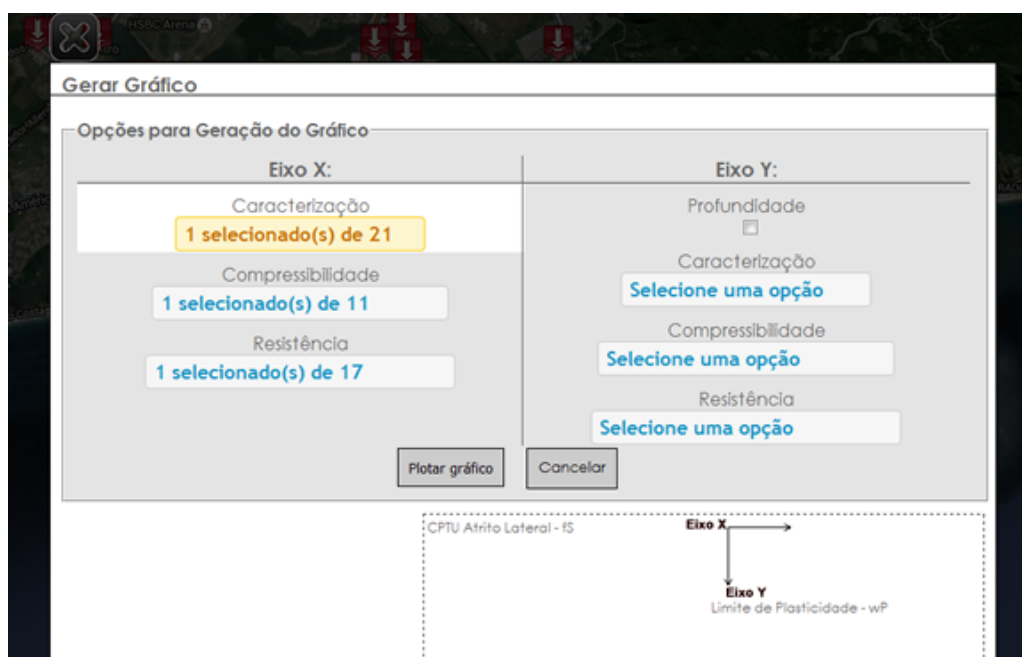


Figura 48 – Tela de visualização de gráfico no sistema.

## 5.4 Verificação e Validação

### 5.4.1 Testes

Na etapa de testes foram desenvolvidos testes unitários e estruturais. Nos casos que era possível a integração de diversos módulos foram explorados testes de integração, e ainda para promover uma maior abrangência de possibilidades quanto as entradas dos testes, foi utilizado o particionamento por equivalência. Essa técnica possibilita que sejam determinados os prováveis limites das entradas dos testes, vindo a proporcionar uma maior e melhor cobertura do sistema. A [Tabela 3](#) apresenta as classes de equivalência definidas.

Tabela 3 – Particionamento por Equivalência

Classe de Equivalência	Testes Escritos
Registro de objetos nulos	13
Obtenção de Lista vazia	12
Expressões sem operadores	10
Registros de objetos duplicados	13
Expressões sem hierarquia de parênteses	8
Expressões com hierarquia de parênteses	4
Exclusão de registro não permitido	12
Registro de lista de objetos encadeados	9
Exclusão de lista de objetos	13
Expressões com parênteses	10

Para analisar a cobertura dos testes diante do escopo do sistema, foi realizada a cobertura de testes baseada em requisitos. Para isso, foi feita uma comparação para descobrir o quanto os testes cobrem os requisitos definidos. Levando em consideração os requisitos classificados como funcionais e não funcionais. Os resultados indicam a totalidade da cobertura dos testes em relação aos requisitos. A trecho de código abaixo apresenta um fragmento de implementação dos testes.

```

1 public class RepositorioClasseTeste {
2     public RepositorioClasseTeste(){
3         repositorioClasse = new RepositorioClasse(new DAOClasse());
4     }
5     @AfterClass
6     public static void tearDownClass() {
7         excluirRegistrosProvisorios();
8     }
9     @Test(expected = java.lang.NullPointerException.class)
10    public void testeSalvarClasseNull() throws Excessao {
11        repositorioClasse.salvar(null, null);
12    }
13    @Test(expected = Excecoes.Excecao.class)
14    public void classeDuplicadoEdicao(){
15        Classe classe = new Classe("ClasseDuplicadaEdicao1");
16        idClassDuplicEdic = repositorioClasse.salvar(classe, null);
17        Classe classe2 = new Classe("ClasseDuplicadaEdicao2");
18        idClassDuplicEdic2 = repositorioClasse.salvar(classe2, null);
19        classe2.setId(idClassDuplicEdic2);
20        classe2.setDescricao("ClasseDuplicadaEdicao1");
21        repositorioClasse.editar(classe2);
22    }
23    private static void excluirRegistrosProvisorios(){
24        repositorioClasse.excluir(idClassDuplic);
25        repositorioClasse.excluir(idClassDuplicEdic);
26        repositorioClasse.excluir(idClassDuplicEdic2);
27        repositorioClasse.excluir(idClassBuscTodosSuces);
28        repositorioClasse.excluir(idBuscClassIDSucesso);
29    }

```

## 5.4.2 Experimento

De modo a realizar uma validação do sistema foi elaborado um experimento prático que teve por objetivo avaliar o nível de aceitação dos usuários em relação ao sistema. A fim de promover a avaliação com “usuários em potencial” foram selecionados os alunos do curso de Engenharia Civil da Unipampa, os quais totalizaram um grupo de 24 alunos. Assim, esses usuários puderam operar o sistema mediante um protocolo de testes.

Este protocolo foi elaborado de modo a possibilitar ao usuário executar as principais funcionalidades do sistema mediante um cenário próximo ao real. Para conduzir o usuário foram descritas tarefas dentro de uma sequência lógica a fim de possibilitar ao mesmo iniciar e finalizar a execução do protocolo com sucesso. O tempo médio para cada usuário realizar o experimento foi de 20 minutos. Juntamente com o documento do protocolo foi fornecido para cada usuário uma Ilha (arquivo com extensão .xls) contendo resultados de ensaios geotécnicos.

### 5.4.2.1 Protocolo de Testes

O protocolo de testes (Figura 49) possui uma organização sequencial a fim de proporcionar ao usuário o entendimento global do experimento. Antes de iniciar as etapas do protocolo são apresentados os pré-requisitos e a contextualização do sistema e suas soluções. As etapas do protocolo cobrem as funcionalidades essenciais do sistema, como por exemplo a importação da ilha, a validação de expressões, o armazenamento e plotagem de gráficos.

Para começar a realizar o protocolo, o usuário inicia o navegador (preestabelecido) e realiza o acesso ao sistema, sendo feita pelo sistema a validação de suas credenciais. Após isso, o usuário é instruído a realizar a importação da ilha anteriormente fornecida. Na sequência o usuário realiza o cadastro de um Parâmetro de Plotagem; esta etapa pode ser considerada uma das mais complexas para o usuário, pois são necessárias várias interações além da validação que é feita da expressão pelo sistema. Com estas etapas superadas, o usuário seleciona a ilha (cadastrada anteriormente) e avança para a etapa “Selecionar opção para gerar gráfico”. Nesta etapa o usuário deve selecionar o Parâmetro de Plotagem cadastrado pelo mesmo anteriormente para então avançar para a etapa “Visualizar gráfico de resultados” que possibilita ao usuário entender e analisar os resultados geotécnicos. As últimas etapas que cobrem a operação do sistema consistem em exportar os resultados apresentados e então sair do sistema.

Para finalizar o usuário é convidado a preencher um questionário, desta maneira foi possível registrar as percepções que o usuário teve durante a utilização do sistema. Para avaliar produtos de software se faz necessário selecionar características de qualidade relevantes para avaliação; Dessa forma, foi adotada a ISO/IEC 9126-1: 2003 que

Figura 49 – Circuito de atividades do protocolo de testes.



abrange as seguintes características de qualidade: usabilidade, eficiência, confiabilidade e funcionalidade. Esses atributos podem ser fracionados da seguinte maneira:

- **Usabilidade**

**Inteligibilidade** Clareza de *links*, imagens e títulos.

**Aprensibilidade** Visibilidade. Facilidade de uso.

**Operacionalidade** Praticidade de operação. Controle de operação.

**Atratividade** Conforto. Clareza de conteúdo.

- **Eficiência**

**Comportamento em relação ao tempo** Agilidade.

- **Confiabilidade**

**Maturidade** Falha/Erros. Segurança

**Tolerância a falhas** Proteção contra erros.

**Recuperabilidade** Falhas do usuário ou do *website*

- **Funcionalidade**

**Adequação** Informações. Serviços. Canais de comunicação

**Acurácia** Precisão nas informações e serviços.

#### 5.4.2.2 Análise dos Resultados

Através da obtenção do *feedback* dos usuários foi possível realizar uma análise a fim de resolver problemas e agregar valor ao software.

A [Tabela 4](#) apresenta de forma organizada os resultados obtidos.

Tabela 4 – Resultados obtidos através do experimento

Atributo de qualidade	Moda	Média	Mínima	Máxima
Operacionalidade	5	4,6	3	5
Apreensibilidade	5	4,6	4	5
Tolerância a falhas	3	3,7	1	5
Comportamento em relação ao tempo	5	4,2	3	5
Inteligibilidade	5	4,5	2	5
Adequação	5	4,5	3	5
Acurácia	4	4,5	3	5

Os valores dos resultados variam de 1 a 5.

Nos resultados, todos os atributos de qualidade tiveram atribuídos a nota máxima (5). No entanto, os atributos **Tolerância a Falhas** e **Inteligibilidade** obtiveram suas notas mínimas respectivamente 1 e 2. Assim, o primeiro atributo (Tolerância a falhas) visto como ponto negativo, pode estar relacionado a alguns casos ocorridos com usuários que não obtiveram êxito ao buscar/recuperar o Parâmetro de Plotagem cadastrado pelo mesmo durante o experimento. Já o atributo **Inteligibilidade** pode ser atribuído também ao *feedback* de alguns usuários que reclamaram da apresentação de alguns componentes gráficos, como por exemplo, *links* do menu principal e botões utilizados na criação da expressão relacionada ao Parâmetro de Plotagem.

A partir da análise da média dos atributos, é possível visualizar que a grande maioria dos atributos, exceto “Tolerância a Falhas”, possuem como média valores igual ou maior a 4.2, ou seja, resultados positivos muito próximos ao valor limite(5). Dessa forma, pode-se concluir que o sistema é considerado satisfatório a ponto de permitir que o usuário opere suas principais funcionalidade de maneira

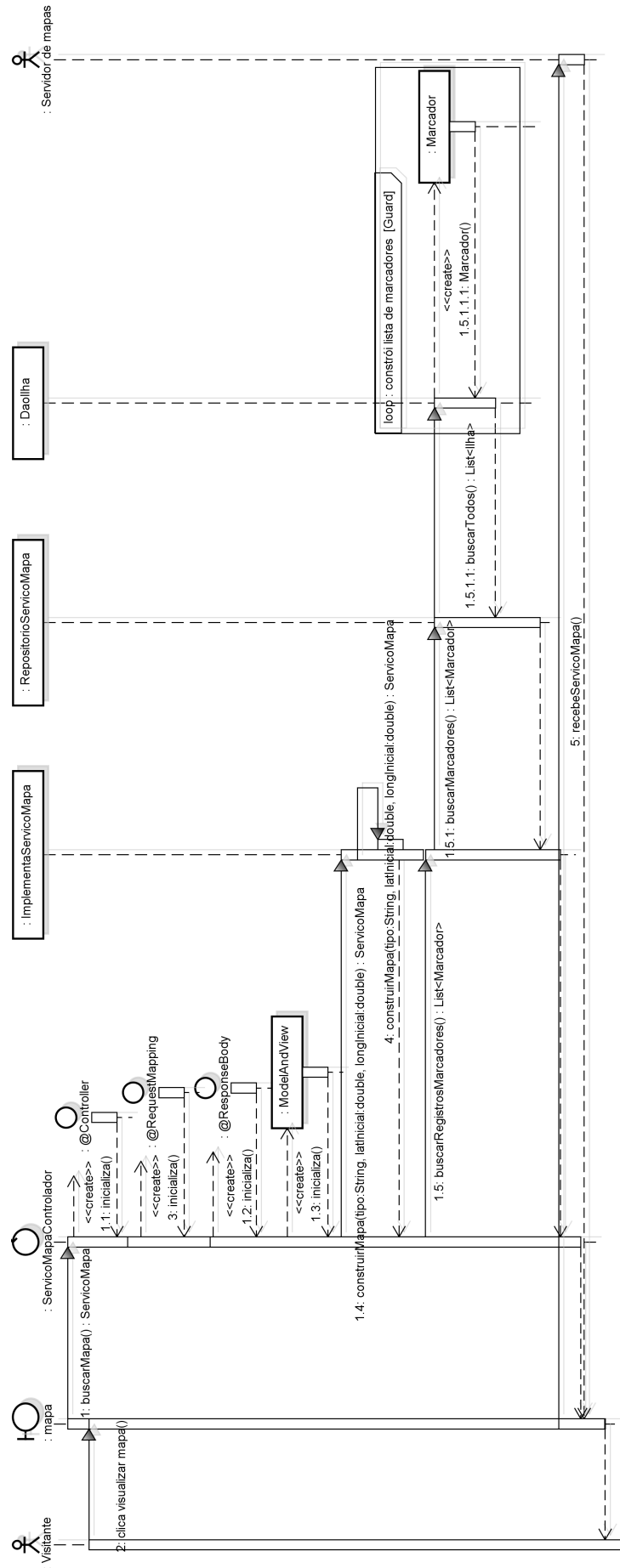
## 5.5 Lições do Capítulo

Neste capítulo foi apresentada a primeira experiência de implementação da arquitetura proposta neste trabalho mediante o estudo de caso do sistema **SAPPGAM**. Para a implantação desse sistema, foram executadas as fases de análise, projeto, implementação e testes. Para conseguir implementar todas as funcionalidades definidas no escopo, se fez necessário a utilização de outras tecnologias complementares, como a **API GoogleMaps** para agregar um mapa ao sistema e a biblioteca **jQuery**, para tratamento de eventos. Mediante esta experiência foi possível ter um *feedback* proveitoso, visto que a arquitetura utilizada está em sua primeira versão.

Questões de incompatibilidade de uma tela do sistema em relação ao navegador (*browser*) foram detectadas, visto que isso ocorreu mediante a programação de uma estrutura feita em *JavaScript*, muito utilizada na programação *Front-end* para tratamento

de eventos, sendo que esta não faz parte da arquitetura utilizada.

Figura 50 – Diagrama de Sequência - Visualizar Mapa.







## 6 Conclusões

Foi verificado durante o presente trabalho que existe a carência de um modelo de arquitetura que consiga dar suporte para a construção de aplicações SIGs Web Geotécnicos. O objetivo principal do presente trabalho é apresentar uma proposta de um modelo arquitetural específico que consiga dar suporte para a construção dessa classe de aplicações.

Como resultado apresentado nesse trabalho, foi desenvolvida uma arquitetura específica que provê suporte estrutural para o desenvolvimento de SIGs Web Geotécnicos. Essa estrutura possibilita através de uma modelagem de domínio definida, trabalhar com dados geotécnicos que são manipulados e armazenados nos mais variados contextos (análise, extração, interpretação, armazenamento). Diante desses contextos, a arquitetura dispõe de mecanismos para a geração e manipulação de artefatos como gráficos, cálculos de expressões matemáticas, leitura e escrita de arquivos, e a possibilidade de armazenar os registros de forma simplificada. Além de dados geotécnicos, o modelo também permite a utilização de serviços de mapas e o controle de acesso as funcionalidades por meio permissões de usuários. Através da apresentação das propriedades do solo juntamente com a perspectiva de seu georeferenciamento, essa arquitetura possibilita desenvolver uma poderosa ferramenta analítica para a engenharia civil.

A partir dessa primeira experiência de utilização da arquitetura, foi possível identificar algumas inconformidades. Por exemplo, num primeiro momento não foi possível implementar a funcionalidade que permite realizar o *download* de um arquivo persistido no BD através de uma resposta AJAX. Além dos vários tipos de retornos oferecidos como resposta pelo *Spring MVC*, não foi possível finalizar o evento de *download* no módulo *Visualizacao*. Para resolver esse caso isolado, foi utilizada uma Sessão, recurso nativo de uma aplicação *web*, visando armazenar os dados (arquivo) e permitir um novo percurso para inicializar o *download*.

De acordo com a documentação do *Hibernate*, para sua utilização em aplicações que persistem grandes quantidades de dados é recomendável a configuração de um *pool* de conexões, que consiste numa estratégia que permite reparar problemas de performance. Além disso, para utilizar o *Hibernate* de uma maneira geral, é necessário um equilíbrio conceitual e sincronismo entre o projeto do sistema e o projeto da base de dados.

É possível concluir que o modelo de arquitetura apresentado neste trabalho se demonstra promissor para a construção dessa classe de aplicações. Primeiro, porque possibilitou toda a implementação do escopo do estudo de caso realizado neste trabalho de forma adequada. Do mais, a sua estrutura em si, é adequada uma vez que permite a

fácil comunicação entre os módulos e impacta no baixo acoplamento e alta coesão, por meio da modelagem de domínio e a injeção de dependências. Em uma perspectiva de evolução tecnológica da arquitetura, pode se ter uma visão otimista pelo fato de se ter como base dois *frameworks* consolidados no desenvolvimento de software que conseguem “conversar” naturalmente, possuem uma documentação bastante rica e ainda estão sempre em constante evolução e alinhados as tendências de arquitetura de software e persistência de dados automatizada.

Com o desenvolvimento desse trabalho foi possível se aprofundar em padrões, técnicas e conceitos vistos de uma forma bastante prática e intensa que são utilizados para a definição de uma arquitetura e para o desenvolvimento de um SIG. A curva de aprendizagem dos conceitos que envolvem ORM é considerada significativa, em função da grande complexidade envolvida em suas estruturas. No entanto, com o tempo é possível agregar esses conhecimentos vindo a formar diversas habilidades que auxiliam muito no desenvolvimento de software.

## 6.1 Trabalhos Futuros

Com o modelo de arquitetura desenvolvido neste trabalho é possível ter suas primeiras impressões, neste caso, uma expectativa otimista devido ao seu experimento e suas respectivas validações. No entanto, para promover um melhor amadurecimento do modelo serão feitas novas implementações através de estudos de caso visando adequar o mesmo por meio de refatorações. Para agregar ainda mais qualidade a sua estrutura, serão pesquisados mecanismos que permitam a realização de testes de integração principalmente no módulo `Visualizacao` e `Controle`.

# Referências

- AIDA, K.; OSUMI, T. Geographic information systems: Evolution academic involvement and issues arising from proliferation of information. In: *Master's Thesis University of California*. Santa Barbara, USA: Master's Thesis University of California, 1988. Citado na página 19.
- ALMEIDA, M.; MARQUES, M. Aterros sobre solos moles: Projeto e desempenho. *Oficina de Textos*, v. 1, p. 31–127, 2010. Citado na página 39.
- APPLETON, B. Patterns and software: Essencial concepts and terminology. In: *Patterns and Software: Essencial Concepts and Terminology*. [S.l.]: Disponível em: <http://www.enteract.com/bradapp/docs/patterns-intro.html>, 1997. Citado 2 vezes nas páginas 27 e 28.
- BADIN, N. T. et al. Utilização de um sistema de informação geográfica para planejamento e gerenciamento de placas de sinalização viária: estudo de caso em Joinville. In: *Utilização de um sistema de informação geográfica para planejamento e gerenciamento de placas de sinalização viária: estudo de caso em Joinville*. [S.l.]: Anais do XXII Encontro Nacional de Engenharia de Produção, Curitiba, PR, 2002. Citado na página 30.
- BASS, L.; CLEMENTS, P.; KAZMAN, R. *Software Architecture in Practice*. 2nd. ed. [S.l.]: Addison Wesley, 2003. Citado na página 19.
- BAUER, C.; KING, G. *Java Persistence com Hibernate*. 1nd. ed. [S.l.]: Editora Ciência Moderna, 2007. Citado 3 vezes nas páginas 36, 37 e 61.
- BRAJA, M. Das fundamentos de engenharia geotécnica. *Tradução All Tasks: revisão técnico Pêrsio Leister de Almeida Barros*. São Paulo: Cengage Learning, 2011. Citado na página 39.
- BUSCHMANN, F.; MEUNIER, R. A systems of patterns. In: *A Systems of Patterns*. [S.l.]: Proceedings of the First International Conference on Pattern Languages of Programming, pp. 325–343, Illinois, EUA, 1994. Citado na página 28.
- BUSCHMANN, F.; MEUNIER, R.; ROHNERT, H. *Pattern-Oriented Software Architecture, A System of Patterns*. 1nd. ed. [S.l.]: John Wiley Sons, 1996. Citado na página 26.
- CHEN, X. et al. Research of real-time agriculture information collection system base on mobile GIS. In: *Research of real-time agriculture information collection system base on mobile GIS*. [S.l.]: Agro-Geoinformatics (Agro-Geoinformatics), 2012 First International Conference, 2012. Citado 2 vezes nas páginas 11 e 48.
- CLEMENTS, P. From domain models to architectures. In: *From Domain Models to Architectures*. [S.l.]: Workshop on Software Architecture, USC Center for Software Engineering, Los Angeles, CA, USA, 1994. Citado na página 24.
- CÂMARA, G.; QUEIROZ, G. R. de. Arquitetura de sistemas de informação geográfica. In: *Arquitetura de sistemas de informação geográfica*. [S.l.]: Introdução à Ciência da

Geoinformação, <http://www.dpi.inpe.br/gilberto/livro/introd> (2000), 2000. Citado 3 vezes nas páginas 31, 32 e 39.

COUTINHO, R.; OLIVEIRA, A.; OLIVEIRA, J. Palheta: Experiência, tradição e inovação. *SEFE IV-Seminário de Engenharia de Fundações Especiais e Geotecnia, São Paulo*, p. 53, 2000. Citado na página 39.

CUNHA, D. V. et al. Sistema de gerenciamento de frotas empregando informações georeferenciadas. In: *Sistema de Gerenciamento de Frotas Empregando Informações Georeferenciadas*. [S.l.]: Trabalho de conclusão, Escola Politécnica - Departamento de Eletrônica e de Computação - UFRJ, 2009. Citado 3 vezes nas páginas 11, 45 e 46.

DIJKSTRA, E. W. The structure of the "the" multiprogramming system. In: *The Structure of the "THE" Multiprogramming System*. [S.l.]: Communications of the ACM 1968, 1968. Citado na página 23.

D'SOUZA, D. F.; WILLS, A. C. *Objects, components, and frameworks with UML: the catalysis approach*. 1nd. ed. [S.l.]: Addison-Wesley Longman Publishing Co., Inc, 1999. Citado na página 24.

FERREIRA, A. F. E. Um modelo de apoio a percepção situacional na resposta a emergências. In: *Um Modelo de Apoio a Percepção Situacional na Resposta a Emergências*. [S.l.]: Diss. Dissertação de Mestrado. PPGI, UFRJ, 2011. Citado 3 vezes nas páginas 46, 47 e 52.

FUSSEL, M. *Foundations of object-relational mapping*. 1nd. ed. [S.l.: s.n.], 1997. Citado na página 37.

GACEK, C. Exploiting domain architectures in software reuse. In: *Exploiting domain architectures in software reuse*. [S.l.]: ACM SIGSOFT Software Engineering Notes. Vol. 20. No. SI. ACM, 1995, 1995. Citado na página 24.

GARLAN, D.; SHAW, M. *An introduction to software architecture*. [S.l.]: World Scientific, 1994. v. 1. Citado na página 23.

GORTON, I. *Essential Software Architecture*. 1nd. ed. [S.l.]: New York, Springer, 2006. Citado 2 vezes nas páginas 24 e 26.

GRAETTINGER, A. J.; RYALS, Z. T.; SMITH, R. K. A web-based geotechnical gis. *ISRN Civil Engineering*, Hindawi Publishing Corporation, v. 2011, 2011. Citado 2 vezes nas páginas 11 e 50.

GUIDELINE, T. F. D. Terasoluna. In: *TERASO-LUNA Global Framework Development Guideline*. [S.l.]: <http://terasolunaorg.github.io/guideline/1.0.1.RELEASE/en/Overview/SpringMVCOverview>, 2012. Citado na página 36.

HAYES, R. Architecture-based acquisition and development of software guidelines and recommendations from the arpa domain-specific software architecture (dssa). In: *Architecture-Based Acquisition and Development of Software Guidelines and Recommendations from the ARPA Domain-Specific Software Architecture (DSSA)*. [S.l.]: Technical Report, Tecknowledge Federal System, 1994. Citado na página 29.

HOFMEISTER, C.; NORD, R. L.; SONI, D. Describing software architecture with uml. In: *Software Architecture*. [S.l.]: Springer, 1999. p. 145–159. Citado na página 25.

ISO/IEEE.1471-2000. Systems and software engineering - recommended practice for architectural description of software-intensive systems. In: *Systems and Software Engineering - Recommended Practice for Architectural Description of Software-Intensive Systems*. [S.l.]: ISO/IEC 42010 IEEE Std 1471-2000 Primeira Edição 2007-07-15, páginas c1–24, Julho 2007, 2007. Citado na página 19.

JAVATPOINT. Hibernate architecture tutorial. In: . [S.l.]: <http://www.javatpoint.com/hibernate-architecture>, 2010. Citado na página 38.

JL, J. Y. . Y. T. . W. J. . C. Soa-based platform design for integrated urban underground pipelines. In: *SOA-based platform design for integrated urban underground pipelines*. [S.l.]: Geomatics for Integrated Water Resources Management (GIWRM), 2012 International Symposium, 2012. Citado na página 48.

JOHNSON, R. *Expert One-To-One J2EE Development Without EJB*. 1nd. ed. [S.l.]: Wrox Press, 2004. Citado na página 35.

JOHNSON, R. E.; FOOTE, B. Designing reusable classes. *Journal of object-oriented programming*, v. 1, n. 2, p. 22–35, 1988. Citado na página 33.

JUNIOR, E. A. de O.; FORTES, R. P. de M. Arquitetura de software na web atual: Processamento no servidor. 2015. Citado na página 62.

KATAKIS, K. Design and implementation of a service oriented architecture for webgis systems. In: *Design and implementation of a service oriented architecture for WebGIS systems*. [S.l.]: Diss. Technical University of Crete, 2009. Citado na página 49.

KRUCHTEN, P. *The Rational Unified Process: An Introduction*. 2nd. ed. [S.l.]: Addison-Wesley, 2000. Citado na página 25.

KUBO, V. T. V. Proposta de implantação de um sistema de informações geográficas para o 31 batalhão de polícia militar de conselheiro lafaiete. In: *PROPOSTA DE IMPLANTAÇÃO DE UM SISTEMA DE INFORMAÇÕES GEOGRÁFICAS PARA O 31 BATALHÃO DE POLÍCIA MILITAR DE CONSELHEIRO LAFAIETE*. [S.l.]: Trabalho de conclusão de curso, Universidade Presidente Antônio Carlos, Barbacena, 2004. Citado na página 32.

KUMMEL, G. Z. Uma abordagem para a criação de arquiteturas de referência de domínio a partir da comparação de modelos arquiteturais de aplicações. In: *Uma Abordagem para a Criação de Arquiteturas de Referência de Domínio a partir da Comparação de Modelos Arquiteturais de Aplicações*. [S.l.]: Tese de Doutorado. UNIVERSIDADE FEDERAL DO RIO DE JANEIRO, 2007. Citado 2 vezes nas páginas 24 e 29.

LAMAS, A. R. Uma arquitetura para o desenvolvimento de sistemas de informação geográfica móveis sensíveis ao contexto. In: *Uma arquitetura para o desenvolvimento de sistemas de informação geográfica móveis sensíveis ao contexto*. [S.l.]: Diss. Universidade Federal de Viçosa, Minas Gerais, 2009. Citado 2 vezes nas páginas 41 e 42.

- LARMAN, C. *Utilizando UML e padrões*. 3nd. ed. [S.l.]: Bookman, 2007. Citado na página 19.
- MASSAD, F. *Obras de terra: curso básico de geotecnia*. [S.l.]: Oficina de Textos, 2010. Citado na página 39.
- MELO, M. de. Aplicação sig na gestão de cadastro na companhia de saneamento de sergipe-deso. In: *APLICAÇÃO SIG NA GESTÃO DE CADASTRO NA COMPANHIA DE SANEAMENTO DE SERGIPE-DESO*. [S.l.]: Trabalho de conclusão de curso, 2012. Citado na página 45.
- MILITITSKY, J.; CONSOLI, N. C.; SCHNAID, F. *Patologia das fundações*. [S.l.]: Oficina de Textos, 2005. Citado na página 39.
- MONROE, R. T. et al. Architectural styles, design patterns and objects. In: *Architectural Styles, Design Patterns and Objects*. [S.l.]: IEEE Software, v. 14, n. 1 (January/February), pp. 43-52, 1997. Citado na página 25.
- MONTEIRO, A.; D'ALGE, J. Introdução à ciência da geoinformação. *São José dos Campos, INPE, São José dos Campos, SP. INPE – Instituto Nacional de Pesquisas Espaciais*. 2001. Disponível em: <http://www.dpi.inpe.br/gilberto/livros.html>. Acesso em Nov/2008, 2001. Citado na página 30.
- NOGUEIRA, R. E. Cartografia: Representação, comunicação e visualização de dados espaciais. In: *Cartografia: Representação, comunicação e visualização de dados espaciais*. [S.l.]: Florianópolis, SC. Editora da UFSC. 2a Edição Revista. 2008, 2008. Citado na página 30.
- PARNAS, D. L. On the criteria for decomposing systems into modules. In: *On the Criteria for Decomposing Systems into Modules*. [S.l.]: Communications of the ACM 1972 15 1053 1058, 1972. Citado na página 23.
- PARNAS, D. L. On a buzzword: Hierarchical structure. In: *On a "Buzzword": Hierarchical Structure*. [S.l.]: Proceeding of IFIP Congress 1974, Amsterdam, North Holland.1974:336 339, 1974. Citado na página 23.
- PARNAS, D. L. On the design and development of program families. In: *On the Design and Development of Program Families*. [S.l.]: Software Engineering, IEEE Transactions on 1976 SE-2: 1 9, 1976. Citado na página 23.
- PATEL, V. Introduction to hibernate framework. In: . [S.l.]: <http://viralpatel.net/blogs/introduction-to-hibernate-framework-architecture/>, 2011. Citado na página 38.
- PAULA, W. de P. *Engenharia de software*. 2nd. ed. [S.l.]: LTC, 2003. Citado na página 25.
- PERILLO, J. R. C. Sade–sistema de atendimento de despacho de emergências em santa catarina. In: *SADÉ–Sistema de Atendimento de Despacho de Emergências em Santa Catarina*. [S.l.]: XIII SIGE - Simpósio de aplicações operacionais em áreas de defesa, 2011. Citado na página 45.
- POHL, K.; RUPP, C. *Fundamentos da Engenharia de Requisitos*. 1nd. ed. [S.l.]: T&M Teste de Software Ltda., 2012. Citado na página 66.

PRESSMAN, R. S. *Engenharia de Software*. 6nd. ed. [S.l.]: McGraw-Hill, 2010. Citado na página 26.

RAMIREZ, R. M. Sistemas gerenciadores de bancos de dados para geoprocessamento. In: *Sistemas Gerenciadores de Bancos de Dados para Geoprocessamento*. [S.l.]: Dissertação de Mestrado, Rio de Janeiro: COPPE/UFRJ, 1994. Citado na página 30.

ROBERTSON, P. Soil classification using the cone penetration test. *Canadian Geotechnical Journal*, NRC Research Press, v. 27, n. 1, p. 151–158, 1990. Citado na página 39.

SANTANA, F. S. Uma infraestrutura orientada a serviços para a modelagem de nicho ecológico. In: *Uma infraestrutura orientada a serviços para a modelagem de nicho ecológico*. [S.l.]: Diss. Universidade de São Paulo - USP, 2009. Citado 2 vezes nas páginas 47 e 48.

SCHENATTO, K. Geoprocessamento aplicado a agricultura visando o mapeamento de doenças em de laranjeiras por meio de técnicas de processamento digital de imagens. In: *Geoprocessamento aplicado a agricultura visando o mapeamento de doenças em de laranjeiras por meio de técnicas de processamento digital de imagens*. [S.l.]: Universidade Tecnológica Federal do Paraná, Medianeira, 2012. Citado 2 vezes nas páginas 31 e 43.

SCHNAID, F.; ODEBRECHT, E. *Ensaio de Campo e suas aplicações à Engenharia de Fundações: 2ª edição*. [S.l.]: Oficina de Textos, 2012. Citado na página 39.

SHAW, M.; GARLAN, D. *Software Architecture: perspectives on an emerging discipline*. 1nd. ed. [S.l.]: Nova Jersey. Prentice - Hall, 1996. Citado 3 vezes nas páginas 24, 26 e 28.

SHEN, L. et al. Management service on wms/wfs services. In: *Management Service on WMS/WFS services*. [S.l.]: Geoinformatics, 2010 18th International Conference on (pp. 1-5). IEEE, 2010. Citado 2 vezes nas páginas 11 e 49.

SILVA, A. de J. In: *ARQUITETURA DE SOFTWARE PARA SISTEMAS DE INFORMAÇÃO PARA GESTÃO DE RECURSOS HÍDRICOS*. [S.l.]: Diss. Universidade Federal de Viçosa, Minas Gerais, 2012. Citado 3 vezes nas páginas 11, 42 e 44.

SILVA, B.; ARAUJO, C. A. D. J. Aplicação de serviços web ogc em infraestruturas de dados espaciais. In: *Aplicação de Serviços Web OGC em Infraestruturas de Dados Espaciais*. [S.l.]: Instituto de Educação Continuada - Pontifícia Universidade Católica de Minas Gerais Belo Horizonte – MG – Brasil, Departamento de Ciência da Computação - Universidade Federal de Minas Gerais Belo Horizonte – MG – Brasil, 2007. Citado 2 vezes nas páginas 43 e 44.

SILVA, J. L. M. da. *Uma Abordagem para Especificação de Requisitos Dirigida por Modelos Integrada ao Controle da Qualidade de Aplicações Web*. Tese (Doutorado) — Universidade Federal do Rio de Janeiro, 2011. Citado na página 32.

SOMMERVILLE, I. *Engenharia de Software*. 9nd. ed. [S.l.]: Pearson Prentice Hall, 2011. Citado 2 vezes nas páginas 24 e 33.

SOUTO, J. H. E. G. do. Aplicação sig (gestão de pontos de interesse de entidades). In: *Aplicação SIG (gestão de pontos de interesse de entidades)*. [S.l.]: Bragança: Escola Superior de Tecnologia e Gestão. Dissertação de Mestrado em Sistemas de Informação, 2012. Citado 3 vezes nas páginas 11, 42 e 43.

- SPRING. Spring framework. In: *Spring Framework*. [S.l.]: <http://docs.spring.io/spring/docs/current/spring-framework-reference/html/overview.html>, 2013. Citado 2 vezes nas páginas 33 e 34.
- SUN, C.-G. Applications of a gis-based geotechnical tool to assess spatial earthquake hazards in an urban area. *Environmental Earth Sciences*, Springer, v. 65, n. 7, p. 1987–2001, 2012. Citado 3 vezes nas páginas 11, 50 e 51.
- VASCONCELOS, A. P. V. de. Uma abordagem de apoio à criação de arquiteturas de referência de domínio baseada na análise de sistemas legados. In: *Uma abordagem de apoio à criação de arquiteturas de referência de domínio baseada na análise de Sistemas Legados*. [S.l.]: Diss. UNIVERSIDADE FEDERAL DO RIO DE JANEIRO, 2007. Citado na página 26.
- VEIGA, A. K. Um estudo sobre qualidade de dados em biodiversidade: aplicação a um sistema de digitalização de ocorrências de espécies. In: *Um estudo sobre qualidade de dados em biodiversidade: aplicação a um sistema de digitalização de ocorrências de espécies*. [S.l.]: Diss. Universidade de São Paulo, 2012. Citado 3 vezes nas páginas 11, 46 e 47.
- WANG, H. et al. A relationship-based and object-oriented software for monitoring management during geotechnical excavation. *Advances in Engineering Software*, Elsevier, v. 71, p. 34–45, 2014. Citado 4 vezes nas páginas 11, 50, 51 e 52.
- WEISSMANN, H. L. *Vire o Jogo com Spring Framework*. 1nd. ed. [S.l.]: Casa do Código, 2012. Citado 2 vezes nas páginas 34 e 35.
- WERNER, C. M. L.; BRAGA, R. M. M. A engenharia de domínio e o desenvolvimento baseado em componentes. In: *A Engenharia de Domínio e o Desenvolvimento Baseado em Componentes*. [S.l.]: GIMENES, I.M.S., HUZITA, E.H.M. (eds), Desenvolvimento Baseado em Componentes: Conceitos e Técnicas, Rio de Janeiro, Ciência Moderna, 2005. Citado na página 29.
- XAVIER, J. R. Criação e instanciação de arquiteturas de software específicas de domínio no contexto de uma infra-estrutura de reutilização. In: *Criação e Instanciação de Arquiteturas de Software Específicas de Domínio no contexto de uma Infra-estrutura de Reutilização*. [S.l.]: Diss. UNIVERSIDADE FEDERAL DO RIO DE JANEIRO, 2001, 2001. Citado na página 28.



# Índice

ADE, 5, 18, 19  
ADL, 24  
AJAX, 39  
ALP, 28  
API, 38, 40, 45, 48, 49  
  
BD, 38–40  
  
CORBA, 44  
  
DESO, 39  
DSSA, 28  
  
ERP, 27  
  
GPS, 42  
  
JSF, 39  
JSP, 40  
  
LPS, 28, 29  
  
MVC, 19, 38, 40, 44, 47, 53  
  
OGC, 39  
OO, 24, 44  
  
RUP, 39  
  
SAAM, 29, 30  
SADE, 40  
SEI, 28  
SGBD, 40, 44  
SIG, 5, 18, 19, 30–33, 35, 36, 39, 40, 42,  
44, 45, 47–50, 53  
SOA, 43, 44  
  
UML, 24  
  
WFS, 44  
WMS, 44  
  
XML, 40



# Apêndices



# APÊNDICE A – Casos de Uso Expandidos

## Casos de Uso Expandidos

---

### UC03 Importar Arquivo

---

**Ator Principal:** Não existe

**Pré-condição:**

O sistema realizou o carregamento do arquivo em memória.

**Fluxo Principal:**

1. O sistema obtém as informações dos dados (mapeamento) contidos no arquivo.
2. O sistema inicia a leitura das informações básicas de uma ilha (nome, executor, data de execução, endereço, latitude, longitude, cidade e referência).
3. O sistema obtém a quantidade total de ensaios.
4. Para cada ensaio o sistema realiza a leitura das seguintes informações: classe do ensaio, parâmetros, valores dos parâmetros.

**Fluxo Alternativo:**

- a. O administrador clica em cancelar.
  - a.1. O sistema apresenta a mensagem: “A operação de cadastro de ilha foi cancelada”.

**Excessões:**

- a. Falha na rede (problema na comunicação dos dados). O sistema apresenta a mensagem: “Problemas na rede, a operação de cadastro da ilha foi cancelada”.
    - 1.a. O arquivo recebido está corrompido. O sistema apresenta a mensagem: “Problemas na leitura do arquivo”.
    - 2.a. O sistema não encontra uma ou mais informações básicas da ilha. O sistema apresenta a mensagem: “A ilha não possui as informações básicas obrigatórias, esta ilha não poderá ser cadastrada”.
- 

### UC04 Plotar Gráfico

---

**Ator Principal:** Visitante

**Pré-condição:**

1. O usuário realizou a seleção da correlação de Parâmetros de Plotagem.
2. O sistema obtém os resultados atribuídos aos Parâmetros de Plotagem selecionados.

**Fluxo Principal:**

1. O sistema inicia o processamento de cada fórmula e gera os respectivos resultados.
2. O sistema realiza a renderização do gráfico dos resultados e apresenta ao usuário.

**Fluxo Alternativo:**

- a. A qualquer momento o usuário clica em “Selecionar outros parâmetros”.
    - a.1. O sistema apresenta a tela de seleção de opções.
    - b. A qualquer momento o usuário clica em “Exportar parâmetros”.
      - b.1. O sistema obtém as informações de mapeamento do arquivo a ser gerado.
      - b.2. O sistema inicia o registro dos resultados dos parâmetros. Para cada resultado serão também indexadas as seguintes informações: parâmetro de plotagem, id do ensaio, ilha, local.
    - c. A qualquer momento o usuário desmarca em um ou mais parâmetros na área 01.
      - c.1. O sistema mostra os pontos correspondentes ao parâmetro selecionado.
-

---

Casos de Uso Expandidos

---

**UC05 Exportar Arquivo Padrão**

---

**Ator Principal:** Gerenciador

**Pré-condição:**

1. O sistema tem armazenado o registro do arquivo

**Fluxo Principal:**

1. O usuário seleciona a opção para download do arquivo padrão.
2. O sistema busca o registro do arquivo padrão através do seu identificador.
3. O sistema constrói o arquivo e inicializa o download.
4. O sistema finaliza do download e apresenta a seguinte mensagem: “Download efetuado com sucesso”.

**Excessões:**

O sistema não inicia o download do arquivo. O sistema apresenta a seguinte mensagem: “O download não pode ser iniciado no momento.”

---

**UC06 Exportar Ilha**

---

**Ator Principal:** Gerenciador

**Pré-condição:**

1. O sistema tem armazenado o registro da ilha a ser exportada

**Fluxo Principal:**

1. O usuário seleciona a opção para download da ilha.
2. O sistema busca o registro da ilha através do seu identificador.
3. O sistema constrói o arquivo e inicializa o download.
4. O sistema finaliza do download e apresenta a seguinte mensagem: “Download efetuado com sucesso”.

**Excessões:**

O sistema não inicia o download do arquivo. O sistema apresenta a seguinte mensagem: “O download não pode ser iniciado no momento.”

---

**UC07 Exportar Parâmetros de Plotagem**

---

**Ator Principal:** Gerenciador

**Pré-condição:**

1. O sistema tem armazenado em memória um ou vários registros de parâmetros de plotagem

**Fluxo Principal:**

1. O usuário seleciona a opção para download dos parâmetros de plotagem.
2. O sistema acessa os dados (parâmetros de plotagem) e inicia a construção do arquivo.
3. O sistema finaliza a construção do arquivo e inicia o seu download.
4. O sistema finaliza o download e apresenta a seguinte mensagem: “Download efetuado com sucesso”.

**Excessões:**

O sistema não inicia o download do arquivo. O sistema apresenta a seguinte mensagem: “O download não pode ser iniciado no momento.”

---

**UC08 Manter Parâmetro**

---

**Ator Principal:** Administrador

1.a.1\*. O sistema tem armazenado o registro do parâmetro selecionado.

1.b.1\*. O sistema tem armazenado o registro do parâmetro selecionado.

**Fluxo Principal:**

1. O usuário seleciona a opção cadastrar parâmetro.
2. O sistema apresenta a tela com os campos a serem preenchidos (nome identificador, nome fantasia, unidade, sigla);
3. O usuário preenche os campos e envia para o sistema.
4. O sistema finaliza o cadastro e apresenta a seguinte mensagem: “Cadastro efetuado com sucesso.”

**Fluxo Alternativo:**

- 1.a.1. O usuário seleciona a opção de edição de parâmetro.
- 1.a.2. O sistema recupera o registro do parâmetro selecionado por meio de seu identificador e preenche os campos do formulário.
- 1.a.3. O usuário realiza a edição dos dados do parâmetro e envia para o sistema.
- 1.a.4. O sistema realiza a atualização do registro e apresenta a seguinte mensagem ao usuário: “Edição realizada com sucesso”.
- 1.b.1. O usuário seleciona a opção excluir parâmetro.
- 1.b.2. O sistema busca o registro do parâmetro por meio do seu identificador e faz a seguinte pergunta ao usuário: “Tem certeza que deseja excluir esse registro?”.
- 1.b.3. O usuário confirma a exclusão.
- 1.b.4. O sistema realiza a exclusão do registro e apresenta a seguinte mensagem: “Registro excluído com sucesso”.

**Excessões:**

- 1.a.3\*. O usuário deixa algum campo obrigatório sem preencher, o sistema apresenta a mensagem: “Todos os campos obrigatórios devem ser preenchidos.”
  - 1.b.4\*. O sistema encontra alguma relação forte do registro para com outro, que inviabiliza a exclusão do mesmo, e apresenta a seguinte mensagem: “Exclusão não permitida, o registro selecionado possui uma relação que impossibilita a exclusão do mesmo”.
- 

**UC09 Manter Fator de Ajuste**

---

**Ator Principal:** Administrador**Pré-condição:**

1.a.1\*. O sistema tem armazenado o registro do fator de ajuste selecionado.

1.b.1\*. O sistema tem armazenado o registro do fator de ajuste selecionado.

**Fluxo Principal:**

1. O usuário seleciona a opção cadastrar fator de ajuste.
2. O sistema apresenta a tela com os campos a serem preenchidos (nome, valor);
3. O usuário preenche os campos e envia para o sistema.
4. O sistema finaliza o cadastro e apresenta a seguinte mensagem: “Cadastro efetuado com sucesso.”

**Fluxo Alternativo:**

- 1.a.1. O usuário seleciona a opção de edição de fator de ajuste.
  - 1.a.2. O sistema recupera o registro do parâmetro selecionado por meio de seu identificador e preenche os campos do formulário.
-

## Casos de Uso Expandidos

- 1.a.3. O usuário realiza a edição dos dados do parâmetro e envia para o sistema.
- 1.a.4. O sistema realiza a atualização do registro e apresenta a seguinte mensagem ao usuário: “Edição realizada com sucesso”.
- 1.b.1. O usuário seleciona a opção excluir parâmetro.
- 1.b.2. O sistema busca o registro do parâmetro por meio do seu identificador e faz a seguinte pergunta ao usuário: “Tem certeza que deseja excluir esse registro?”.
- 1.b.3. O usuário confirma a exclusão.
- 1.b.4. O sistema realiza a exclusão do registro e apresenta a seguinte mensagem: “Registro excluído com sucesso”.

**Excessões:**

- 1.a.3\*. O usuário deixa algum campo obrigatório sem preencher, o sistema apresenta a mensagem: “Todos os campos obrigatórios devem ser preenchidos.”.
  - 1.b.4\*. O sistema encontra alguma relação forte do registro para com outro, que inviabiliza a exclusão do mesmo, e apresenta a seguinte mensagem: “Exclusão não permitida, o registro selecionado possui uma relação que impossibilita a exclusão do mesmo”.
- 

**UC10 Parâmetro de Plotagem**

---

**Ator Principal:** Administrador**Pré-condição:**

- 1.a.1\*. O sistema tem armazenado o registro do parâmetro de plotagem selecionado.
- 1.b.1\*. O sistema tem armazenado o registro do parâmetro de plotagem selecionado.

**Fluxo Principal:**

1. O usuário seleciona a opção cadastrar parâmetro de plotagem.
2. O sistema apresenta a tela com os campos a serem preenchidos (nome, unidade, tipo, fórmula);
3. O usuário preenche os campos e envia para o sistema.
4. O sistema finaliza o cadastro e apresenta a seguinte mensagem: “Cadastro efetuado com sucesso.”.

**Fluxo Alternativo:**

- 1.a.1. O usuário seleciona a opção de edição de parâmetro de plotagem.
- 1.a.2. O sistema recupera o registro do parâmetro selecionado por meio de seu identificador e preenche os campos do formulário.
- 1.a.3. O usuário realiza a edição dos dados do parâmetro de plotagem e envia para o sistema.
- 1.a.4. O sistema realiza a atualização do registro e apresenta a seguinte mensagem ao usuário: “Edição realizada com sucesso”.
- 1.b.1. O usuário seleciona a opção excluir parâmetro de plotagem.
- 1.b.2. O sistema busca o registro do parâmetro por meio do seu identificador e faz a seguinte pergunta ao usuário: “Tem certeza que deseja excluir esse registro?”.
- 1.b.3. O usuário confirma a exclusão.
- 1.b.4. O sistema realiza a exclusão do registro e apresenta a seguinte mensagem: “Registro excluído com sucesso”.

**Excessões:**

- 1.a.3\*. O usuário deixa algum campo obrigatório sem preencher, o sistema apresenta a mensagem: “Todos os campos obrigatórios devem ser preenchidos.”.
  - 1.b.4\*. O sistema encontra alguma relação forte do registro para com outro, que inviabiliza a exclusão do mesmo, e apresenta a seguinte mensagem: “Exclusão não permitida, o registro selecionado possui uma relação que impossibilita a exclusão do mesmo”.
-



**UC11 Manter Usuários**

---

**Ator Principal:** Administrador**Pré-condição:**

1.a.1\*. O sistema tem armazenado o registro do usuário selecionado.

1.b.1\*. O sistema tem armazenado o registro do usuário selecionado.

**Fluxo Principal:**

1. O usuário seleciona a opção cadastrar usuário.
2. O sistema apresenta a tela com os campos a serem preenchidos (nome, senha, perfil);
3. O usuário preenche os campos e envia para o sistema.
4. O sistema finaliza o cadastro e apresenta a seguinte mensagem: “Cadastro efetuado com sucesso.”.

**Fluxo Alternativo:**

- 1.a.1. O usuário seleciona a opção de edição de usuário.
- 1.a.2. O sistema recupera o registro do usuário selecionado por meio de seu identificador e preenche os campos do formulário.
- 1.a.3. O usuário realiza a edição dos dados do usuário e envia para o sistema.
- 1.a.4. O sistema realiza a atualização do registro e apresenta a seguinte mensagem ao usuário: “Edição realizada com sucesso”.
- 1.b.1. O usuário seleciona a opção excluir um usuário.
- 1.b.2. O sistema busca o registro do usuário por meio do seu identificador e faz a seguinte pergunta ao usuário: “Tem certeza que deseja excluir esse registro?”.
- 1.b.3. O usuário confirma a exclusão.
- 1.b.4. O sistema realiza a exclusão do registro e apresenta a seguinte mensagem: “Registro excluído com sucesso”.

**Excessões:**

- 1.a.3\*. O usuário deixa algum campo obrigatório sem preencher, o sistema apresenta a mensagem: “Todos os campos obrigatórios devem ser preenchidos.”.
  - 1.b.4\*. O sistema encontra alguma relação forte do registro para com outro, que inviabiliza a exclusão do mesmo, e apresenta a seguinte mensagem: “Exclusão não permitida, o registro selecionado possui uma relação que impossibilita a exclusão do mesmo”.
- 

**UC12 Manter Classes**

---

**Ator Principal:** Administrador**Pré-condição:**

1.a.1\*. O sistema tem armazenado o registro da classe selecionado.

1.b.1\*. O sistema tem armazenado o registro da classe selecionado.

**Fluxo Principal:**

1. O usuário seleciona a opção cadastrar classe.
  2. O sistema apresenta a tela com os campos a serem preenchidos (nome);
  3. O usuário preenche os campos e envia para o sistema.
  4. O sistema finaliza o cadastro e apresenta a seguinte mensagem: “Cadastro efetuado com sucesso.”.
-

## Casos de Uso Expandidos

**Fluxo Alternativo:**

- 1.a.1. O usuário seleciona a opção de edição de arquivo padrão.
- 1.a.2. O sistema recupera o registro do usuário selecionado por meio de seu identificador e preenche os campos do formulário.
- 1.a.3. O usuário realiza a edição dos dados do usuário e envia para o sistema.
- 1.a.4. O sistema realiza a atualização do registro e apresenta a seguinte mensagem ao usuário: “Edição realizada com sucesso”.
- 1.b.1. O usuário seleciona a opção excluir arquivo padrão.
- 1.b.2. O sistema busca o registro do arquivo padrão por meio do seu identificador e faz a seguinte pergunta ao usuário: “Tem certeza que deseja excluir esse registro?”.
- 1.b.3. O usuário confirma a exclusão.
- 1.b.4. O sistema realiza a exclusão do registro e apresenta a seguinte mensagem: “Registro excluído com sucesso”.

**Excessões:**

- 1.a.3\*. O usuário deixa algum campo obrigatório sem preencher, o sistema apresenta a mensagem: “Todos os campos obrigatórios devem ser preenchidos.”.
  - 1.b.4\*. O sistema encontra alguma relação forte do registro para com outro, que inviabiliza a exclusão do mesmo, e apresenta a seguinte mensagem: “Exclusão não permitida, o registro selecionado possui uma relação que impossibilita a exclusão do mesmo”.
- 

**UC13 Manter Arquivo Padrão**

---

**Ator Principal:** Administrador**Pré-condição:**

- 1.a.1\*. O sistema tem armazenado o registro do arquivo padrão selecionado.
- 1.b.1\*. O sistema tem armazenado o registro do arquivo padrão selecionado.

**Fluxo Principal:**

1. O usuário seleciona a opção cadastrar arquivo padrão.
2. O sistema apresenta a tela com os campos a serem preenchidos (nome, arquivo);
3. O usuário preenche os campos e envia para o sistema.
4. O sistema finaliza o cadastro e apresenta a seguinte mensagem: “Cadastro efetuado com sucesso.”.

## APÊNDICE B – Diagramas de Sequência

Figura 51 – Diagrama de Sequência - Cadastrar Ilha.

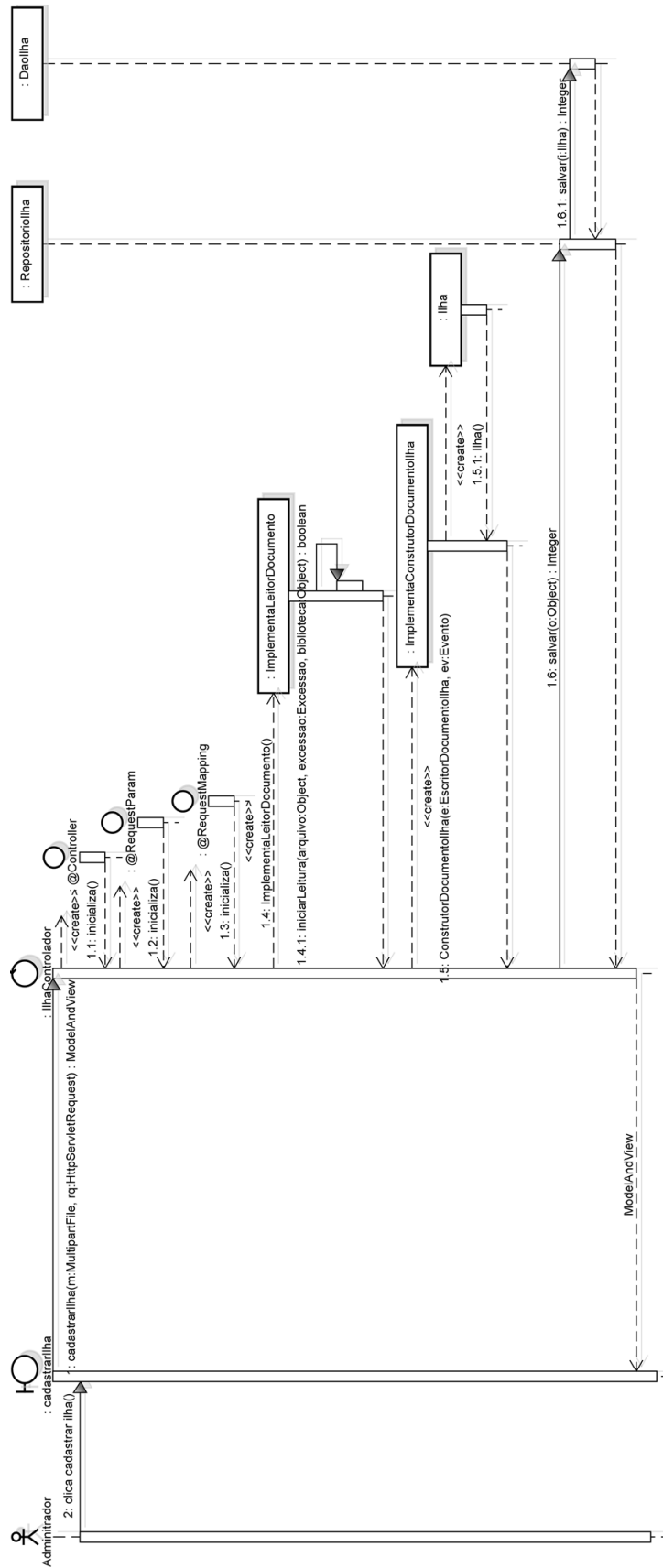


Figura 52 – Diagrama de Sequência - Importar Arquivo.

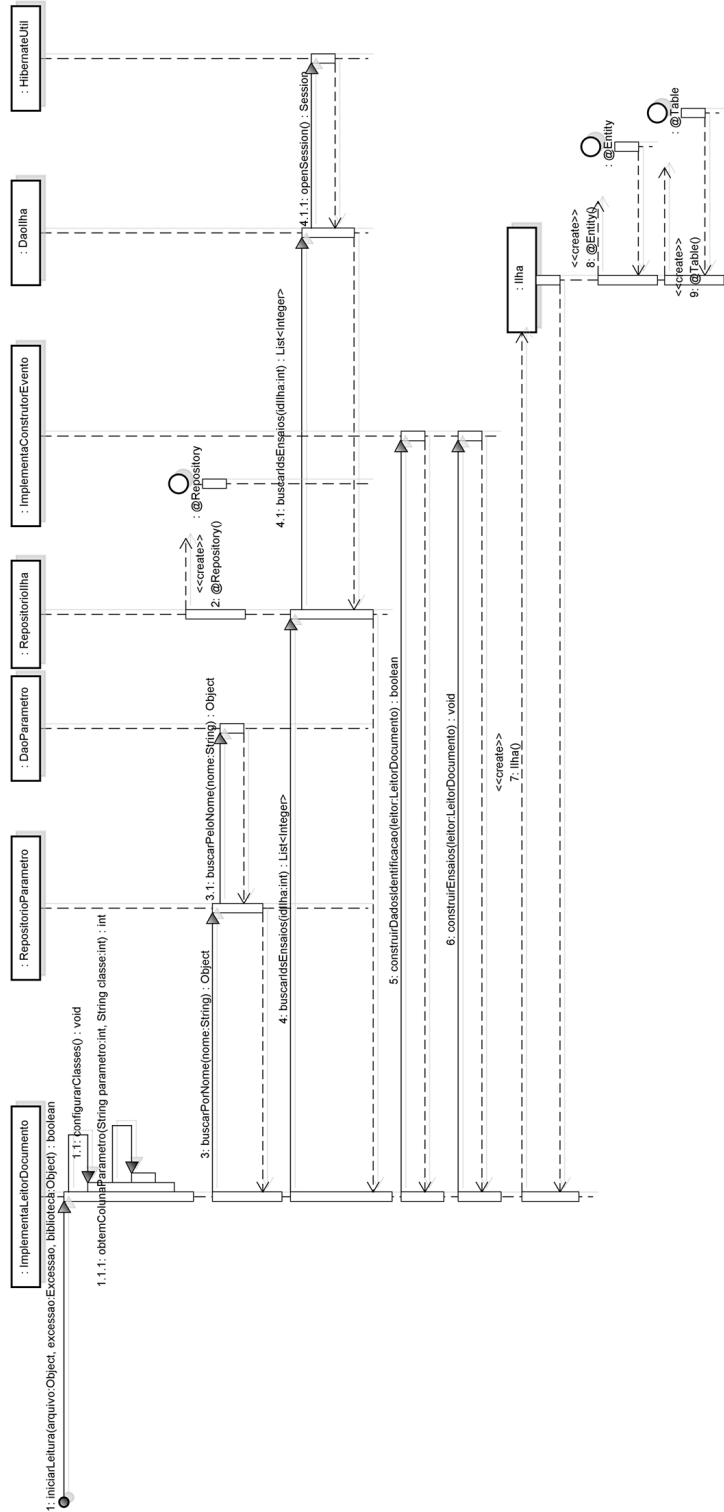


Figura 53 – Diagrama de Sequência - Manter Fator de Ajuste.

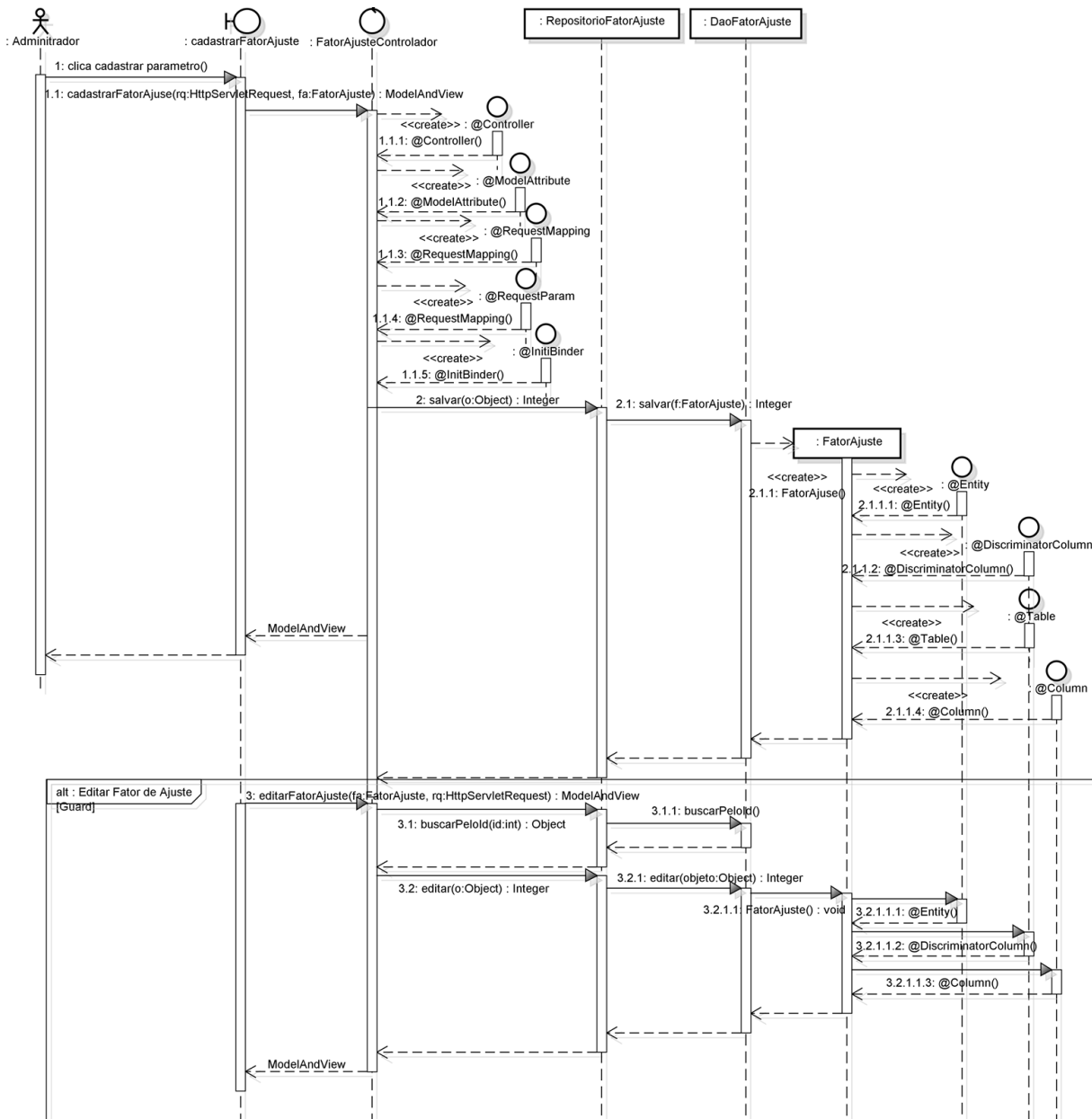


Figura 54 – Diagrama de Sequência - Manter Parâmetro de Plotagem.

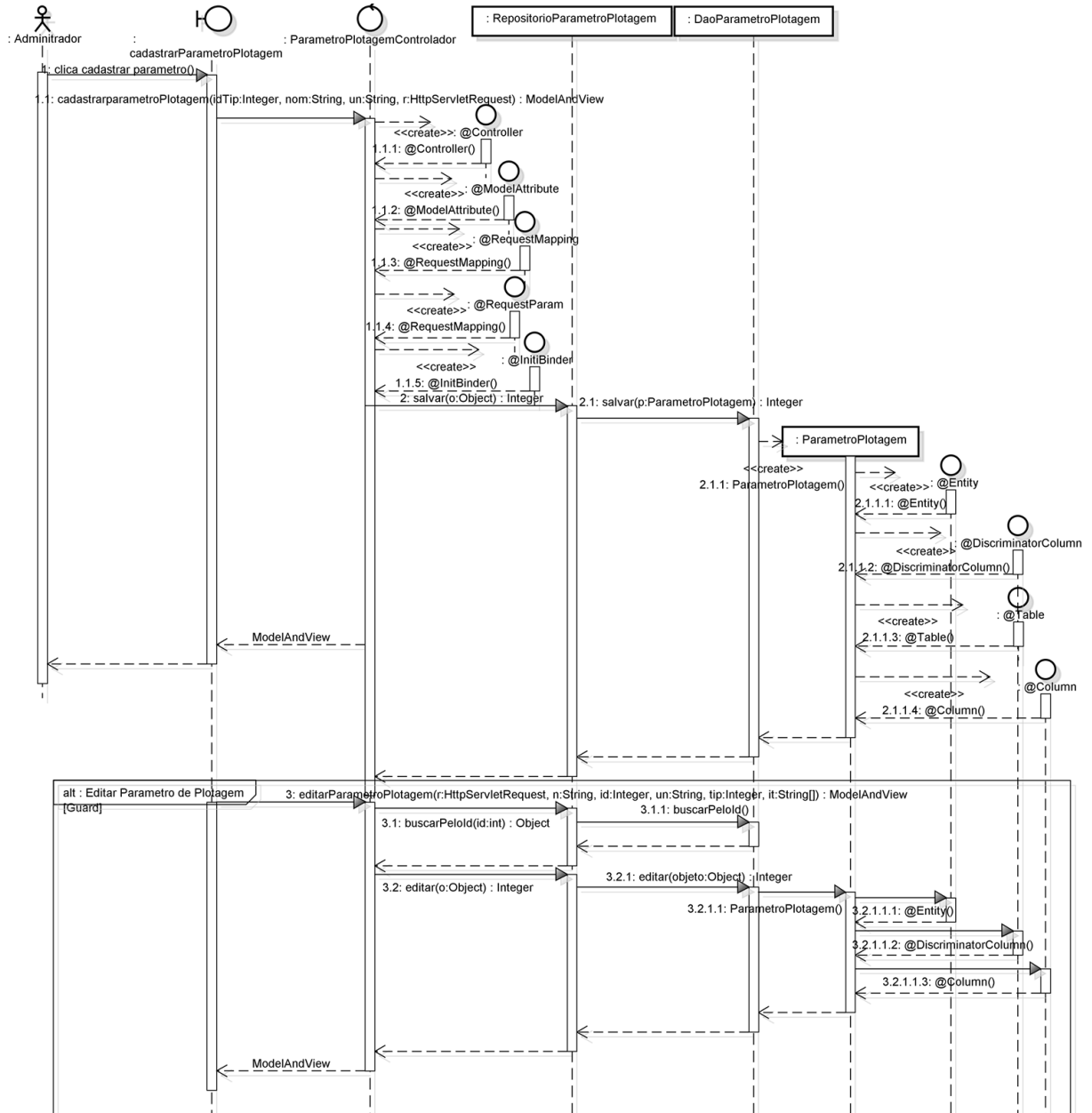
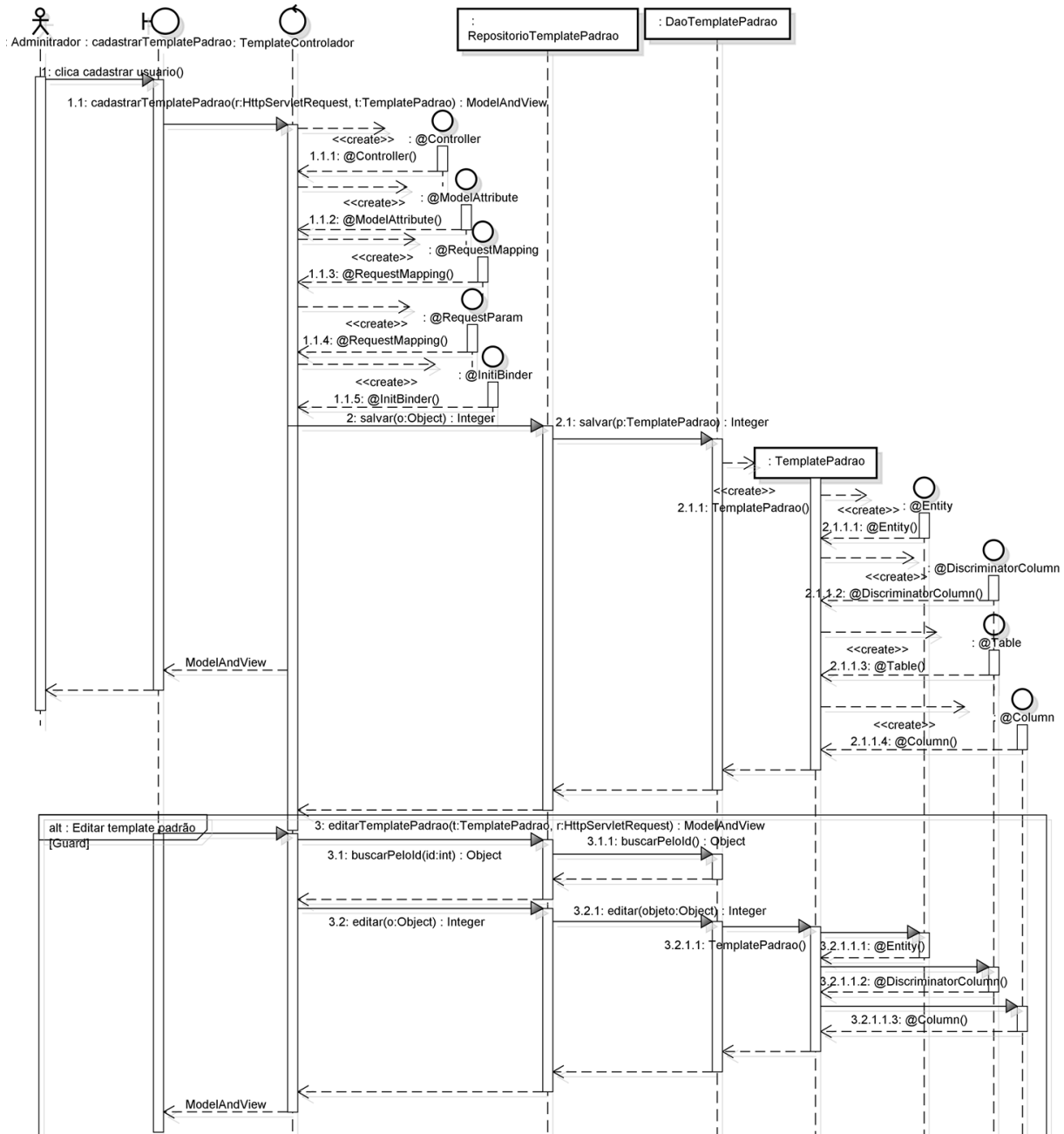


Figura 55 – Diagrama de Sequência - Template Padrão.





# Anexos



# ANEXO A – Template Padrão de Resultados

Figura 56 – Ensaio organizado em uma planilha a ser carregada e interpretada pelo sistema.

1	Classe:									
2	CPTU									
3	Nível de água:									
4	Profundidade CPTU	Resistência de Ponta Corrigida	Atrito Lateral	Poropressão $u_1$	Poropressão $u_2$	Peso Específico da Água	Peso Específico Natural do Solo	Pressão Hidrostática CPTU	Tensão Vertical Total	Tensão Vertical Efetiva
5	ProfunCPTU	ResistPontaCorrigida	AtritoLateral	Poropressao1	Poropressao2		PesoEspecNatSolo	PressaoHidroCPTU	TensaoVerticalTotal	
6	Profundidade	$q_r$	$f_s$	$u_1$	$u_2$	$\gamma_w$	$\gamma_{nat}$	$U_0$	$\sigma_{vd}$	$\sigma'_{vd}$
7	(m)	(kPa)	(kPa)	(kPa)	(kPa)	(kN/m <sup>3</sup> )	(kN/m <sup>3</sup> )	(kPa)	(kPa)	(kPa)
8	0,108	4,23	0,068	-0,184	-0,253	9,81	13	-3,26	1,40	4,66
9	0,132	3,018	0,068	-0,356	0,106	9,81	13	-3,02	1,72	4,74
10	0,156	1,295	0,068	-0,757	5,826	9,81	13	-2,79	2,03	4,81
11	0,179	5,588	0,068	1,913	3,862	9,81	13	-2,56	2,33	4,89
12	0,203	30,776	0,068	0,269	2,548	9,81	13	-2,32	2,64	4,96
13	0,225	9,888	0,068	4,538	4,574	9,81	13	-2,11	2,93	5,03
14	0,248	3,045	0,068	1,744	3,292	9,81	13	-1,88	3,22	5,11
15	0,271	10,501	0,068	0,214	3,977	9,81	13	-1,66	3,52	5,18
16	0,293	15,764	0,068	1,351	5,71	9,81	13	-1,44	3,81	5,25
17	0,317	3,045	0,832	5,458	4,149	9,81	13	-1,21	4,12	5,33
18	0,34	3,045	0,467	3,336	2,649	9,81	13	-0,98	4,42	5,40
19	0,363	6,782	1,261	5,867	3,888	9,81	13	-0,76	4,72	5,47
20	0,385	6,782	1,278	5,779	3,217	9,81	13	-0,54	5,01	5,54
21	0,408	0,397	1,278	2,967	2,815	9,81	13	-0,31	5,30	5,62
22	0,432	2,931	1,329	2,269	2,911	9,81	13	-0,08	5,62	5,69
23	0,455	81,559	1,587	4,26	4,525	9,81	13	0,15	5,92	5,77
24	0,477	41,011	0,178	8,865	8,134	9,81	13	0,36	6,20	5,84
25	0,5	112,847	0,124	40,782	0,531	9,81	13	0,59	6,50	5,91
26	0,524	112,478	0,842	39,476	-6,014	9,81	13	0,82	6,81	5,99
27	0,533	113,803	3,055	40,869	-10,862	9,81	13	0,91	6,93	6,02