Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Coordenação de Pós-Graduação em Ciência da Computação

NoSQL-GeoServices: Um *Framework* para Interoperabilidade de Dados Geográficos Armazenados em Sistemas NoSQL

Odilon Francisco de Lima Junior

Dissertação submetida à Coordenação do Curso de Pós-Graduação em Ciência da Computação da Universidade Federal de Campina Grande - Campus I como parte dos requisitos necessários para obtenção do grau de Mestre em Ciência da Computação.

Área de Concentração: Ciência da Computação
Linha de Pesquisa: Sistemas de Informação e Banco de Dados

Dr. Cláudio de Souza Baptista e Dr. Carlos Eduardo Santos Pires
(Orientadores)

Campina Grande, Paraíba, Brasil

FICHA CATALOGRÁFICA ELABORADA PELA BIBLIOTECA CENTRAL DA UFCG

L732n Lima Junior, Odilon Francisco
NoSQL-GeoServices: um framework para interoperabilidade de dados geográficos armazenados em sistemas nosql / Odilon Francisco Lima Junior. – Campina Grande, 2012.
71 f. : il. color.

Dissertação (Mestrado em Ciências da Computação) – Universidade Federal de Campina Grande, Centro de Engenharia Elétrica e Informática.
Orientadores: Prof. Dr. Cláudio de Souza Baptista , Carlos Eduardo Santos Pires.
Referências.

1. Interoperabilidade. 2. NoSQL. 3. OGC. 4. Dados Espaciais. I. Título.

CDU 004.65 (043)

"NOSQL-GEOSERVICES: UM FRAMEWORK PARA INTEROPERABILIDADE DE DADOS GEOGRÁFICOS ARMAZENADOS EM SISTEMAS NOSQL"

ODILON FRANCISCO DE LIMA JUNIOR

DISSERTAÇÃO APROVADA EM 10/09/2012

Cláudio de Souza Baptista

CLÁUDIO DE SOUZA BAPTISTA, Ph.D, UFCG
Orientador(a)

Carlos Eduardo Santos Pires

CARLOS EDUARDO SANTOS PIRES, Dr., UFCG
Orientador(a)

ULRICH SCHIEL, Dr., UFCG
Examinador(a)

Damires Yluska S Fernandes

DAMIRES YLUSKA DE SOUZA FERNANDES, D.Sc, IFPB
Examinador(a)

CAMPINA GRANDE - PB

Resumo

O grande volume de informação georreferenciada produzida na Web 2.0, especialmente devido à popularização de dispositivos de coleta, tais como *smartphones*, GPS e câmeras, tem aumentado a procura por novas formas de armazenamento de dados. Nesse contexto, sistemas NoSQL têm sido uma abordagem promissora. A necessidade de reunir todo o espectro de informação georreferenciada nos leva ao tradicional problema de interoperabilidade entre bancos de dados espaciais SQL e sistemas NoSQL. Em outras palavras, o problema endereçado é o da interoperabilidade de dados geográficos a partir de fontes de informação heterogêneas. Este trabalho propõe uma solução para a interoperabilidade de dados geográficos armazenados em bancos de dados SQL e sistemas NoSQL usando serviços de interoperabilidade WMS e WFS da OGC. Experimentos conduzidos utilizando o banco de dados espacial PostgreSQL-PostGIS e os sistemas NoSQL CouchDB-GeoCouch e MongoDB mostraram que é possível enviar consultas usando a mesma sintaxe para bancos de dados espaciais SQL e NoSQL de uma forma simples e transparente para a aplicação do usuário.

Palavras-Chave: Interoperabilidade,NoSQL,OGC,Dados Espaciais

Abstract

The large volume of georeferenced information produced in the Web 2.0, specially given the popularity of collection devices, such as smartphones, GPS and cameras, has increased the demand for new techniques for data storage. In this context, NoSQL systems have been a promising approach. The need to bring together the entire spectrum of georeferenced information takes us to the traditional problem of interoperability between SQL spatial databases and NoSQL systems. In other words, the problem addressed is the interoperability of spatial data from heterogeneous data sources. This work proposes a solution for the interoperability of spatial data stored in SQL databases and NoSQL systems using OGC interoperability services such as WMS and WFS. Experiments conducted using the spatial database PostgreSQL-PostGIS and NoSQL systems such as CouchDB-GeoCouch and MongoDB demonstrated that it is possible to submit queries using the same syntax for SQL spatial databases and NoSQL systems in a simple and transparent manner for the user's application.

Keywords: Interoperability,NoSQL,OGC,Spatial Data

Agradecimentos

Aos meus orientadores, Cláudio Baptista e Carlos Eduardo, pela motivação, ensinamentos, paciência e colaboração para efetivar mais esse marco em minha formação.

A meus pais, Odilon e Dora, minha filha Sofia e minha irmã Juliana pelo apoio e compreensão.

Aos amigos do Laboratório de Sistemas de Informação, pelas trocas de experiências e momentos de descontração. Em especial a Maxwell Guimarães, Tiago Eduardo, Vinícius Veloso e Hugo.

Aos professores Damires Yluska e Ulrich Schiel, por aceitarem o convite para participarem da banca do meu mestrado.

Aos professores e funcionários do Programa de Pós-Graduação em Ciência da Computação da UFCG pela contribuição e auxílio.

Por fim, agradeço a Hewlett-Packard (HP), pelo apoio financeiro.

Conteúdo

1 Introdução **1**

1.1 Objetivos 3

1.2 Estrutura da Dissertação 4

1.3 Trabalhos Publicados 4

2 Fundamentação Teórica **5**

2.1 Sistemas de Informações Geográficas 5

2.1.1 Dados Espaciais 6

2.1.2 Serviços de Interoperabilidade de Dados Espaciais 7

2.2 Sistemas NoSQL 16

2.2.1 Características Globais e Conceitos Teóricos 18

2.2.2 Sistemas NoSQL para Domínio Espacial 21

2.3 Considerações Finais 24

3 Trabalhos Relacionados **26**

3.1 Considerações Finais 30

4 *Framework* NoSQL-GeoServices **32**

4.1 Arquitetura 32

4.1.1 Modelo de Arquitetura 33

4.1.2 Camada de Aplicação 33

4.1.3 Camada de Serviços 36

4.1.4 Extensibilidade do *Framework* 39

4.2 Considerações sobre a Implementação 42

4.2.1 GetCapabilities 44

4.2.2 GetMap 44

4.2.3 GetFeatureInfo 46
4.2.4 DescribeFeatureType 47
4.2.5 GetFeature 47
4.3 Considerações Finais 48

5 Avaliação Funcional **50**
5.1 Configuração do Ambiente 50
5.2 Verificação dos Serviços Implementados 51
5.2.1 Interoperabilidade entre Bancos de Dados SQL e Sistemas NoSQL . 57
5.3 Considerações Finais 60

6 Conclusão **62**
6.1 Contribuições 63
6.2 Trabalhos Futuros 63

Lista de Símbolos

ACID - *Atomicidade, Consistência, Isolamento e Durabilidade*

API - *Application Programming Interface*

BASE - *Basically Available, Soft state, Eventual consistency*

BSON - *Binary JavaScript Object Notation*

CAP - *Consistency, Availability, Partition tolerance*

CAS - *Client Access Service*

CGI - *Commom Gateway Interface*

CRS - *Coordinate Reference System*

DXS - *Data Exchange Service*

EPSG - *European Petroleum Survey Group*

GDAL - *Geospatial Data Abstraction Library*

GML - *Geography Markup Language*

HTTP - *HyperText Transfer Protocol*

IDE - *Infraestrutura de Dados Espaciais*

JSON - *JavaScript Object Notation*

JSP - *JavaServer Pages*

LSDI - *Local Spatial Data Infrastructure*

MBR - *Minimum Bounding Rectangle*

MIME - *Multipurpose Internet Mail Extensions*

MVCC - *Multi-Version Concurrency Control*

OGC - *Open Geospatial Consortium*

OLTP - *Online Transaction Processing*

OWS - *OGC Web Services*

REST - *REpresentational State Transfer*

SGBD - *Sistema de Gerenciamento de Banco de Dados*

SIG - *Sistema de Informação Geográfica*

SLD - *Styled Layer Descriptor*

SQL - *Structured Query Language*

SRS - *Spatial Reference System*

URI - *Uniform Resource Identifier*

URL - *Uniform Resource Locator*

WCL - *WMS Connectivity Layer*

WMS - *Web Map Service*

WFS - *Web Feature Service*

XML - *eXtensible Markup Language*

Lista de Figuras

Figura 2.1 Paradigma dos quatro universos (Fonte: Camara *et al.* [CCD+05]) . 6

Figura 4.1 Interoperabilidade entre Sistemas de Gerenciamento de Banco de Dados Espaciais e Sistemas NoSQL Espaciais 34

Figura 4.2 Tela de configuração do serviço WMS no *NoSQL-GeoServices*. . . . 35

Figura 4.3 Visualizador de mapas GEO-STAT 36

Figura 4.4 Arquitetura MVC do *NoSQL-GeoServices* 38

Figura 4.5 Diagrama de sequência de uma requisição GetMap no *NoSQL-GeoServices* 40

Figura 4.6 Principais pontos de extensão do *framework NoSQL-GeoServices*. . . 41

Figura 4.7 Interface do *NoSQL-GeoServices* que permite a extensão para outros sistemas de armazenamento. 43

Figura 5.1 Configuração de conexão ao *NoSQL-GeoServices* no Quantum GIS. . 52

Figura 5.2 Lista de camadas disponíveis no CouchDB através *NoSQL-GeoServices*. 52

Figura 5.3 Resultado de uma requisição *GetMap*, feita ao *NoSQL-GeoServices*, solicitando os municípios, as rodovias e os focos de incêndio registrados em 2012. 53

Figura 5.4 Mapa com os focos de incêndio armazenados no MongoDB e fornecidos pelo *NoSQL-GeoServices* usando o GEO-STAT. 54

Figura 5.5 As Figuras 5.5a, 5.5b e 5.5c mostram as informações do servidor WFS recuperadas por meio de uma requisição *GetCapabilities*. 55

Figura 5.6 Campos recuperados de uma camada armazenada no CouchDB através da requisição *DescribeFeatureType*. 56

Figura 5.7 Resultado da aplicação da consulta: $"NOME\_CAPITAL" <> 'Recife'$. 56

Figura 5.8 Consulta espacial: "*Quais os focos de incêndio que estejam dentro de uma dada região de Mata Atlântica*". 57
Figura 5.10 Resultado da consulta *GetFeature* especificada no Exemplo 5.1. . . . 58
Figura 5.9 Consulta espacial no Kosmo SIG acessando o *NoSQL-GeoServices*. . 59
Figura 5.11 Resultado da consulta *GetFeature* especificada no Exemplo 5.2. . . . 60

Lista de Exemplos

2.1 Exemplo de filtro usando id da feição (*feature id*). 11

2.2 Exemplo de filtro espacial. 13

2.3 Exemplo de filtro de comparação. 14

2.4 Exemplo de filtro lógico. 14

2.5 Exemplo de inserção de feição usando WFS-T. 15

2.6 Exemplo de atualização de feição usando WFS-T. 15

2.7 Exemplo de exclusão de feição usando WFS-T. 16

2.8 Documento no formato GeoJSON. 23

4.1 URL de conexão com o serviço WMS do *NoSQL-GeoServices*. 35

4.2 URL de conexão com o serviço WFS do *NoSQL-GeoServices*. 36

4.3 Exemplo de requisição *GetCapabilities* (WMS) enviada ao *NoSQL-GeoServices*. 44

4.4 Exemplo de requisição *GetCapabilities* (WFS) enviada ao *NoSQL-GeoServices*. 44

4.5 Exemplo de requisição *GetMap* (WMS) enviada ao *NoSQL-GeoServices*. . 46

4.6 Exemplo de requisição *GetFeatureInfo* (WMS) enviada ao *NoSQL-GeoServices*. 46

4.7 Exemplo de requisição *DescribeFeatureType* (WFS) enviada ao *NoSQL-GeoServices*. 47

4.8 Exemplo de requisição *GetFeature* (WFS) enviada ao *NoSQL-GeoServices*. 48

5.1 Exemplo de requisição *GetFeature* enviada ao *NoSQL-GeoServices* solicitando os focos de incêndio dentro de região circular (*dwithin*) fornecida pelo usuário. 58

5.2 Exemplo de requisição *GetFeature* enviada ao *NoSQL-GeoServices* solicitando os focos de incêndio dentro uma região retangular (*bbox*) fornecida pelo usuário. 60

Capítulo 1

Introdução

Em virtude do grande volume de dados gerados atualmente na Internet novas formas de armazenamento e processamento de dados são necessárias. Muitas aplicações da Web 2.0 possuem conjuntos de dados que crescem a um ritmo acelerado. Estamos na era das redes sociais que geram uma enorme quantidade de informação. Por exemplo, sites de mídia social (como Twitter, Facebook e LinkedIn) geram petabytes de informação que precisam ser armazenadas para posterior consulta [Rus11].

Avanços na tecnologia de sensores, o aumento da largura de banda e a popularidade de dispositivos móveis que podem se conectar à Internet criaram um ambiente em que muitas aplicações precisam armazenar mais dados do que muitos Sistemas Gerenciadores de Bancos de Dados (SGBDs) tradicionais foram projetados para tratar [CD10].

Nesse contexto, surge a necessidade de que bancos de dados tornem-se cada vez mais escaláveis. Escalar (*scaling*) um banco de dados pode se resumir à escolha entre escalabilidade vertical (*scaling up*), através da aquisição de um super-computador; ou escalabilidade horizontal (*scaling out*), pelo particionamento dos dados em várias máquinas [CD10].

Escalabilidade horizontal é a capacidade de distribuir tanto os dados quanto a carga de operações sobre muitos servidores, sem memória RAM ou discos compartilhados entre os servidores. Na escalabilidade vertical, um sistema de banco de dados utiliza muitos núcleos e/ou CPUs que compartilham RAM e disco. Sendo essa uma das desvantagens da abordagem vertical, pois existe uma limitação no número de núcleos que podem compartilhar memória [Cat11].

Para aplicações web de grande porte, escalabilidade horizontal é uma alternativa mais

viável tanto economicamente quanto em termos de flexibilidade, pois, para adicionar espaço de armazenamento ou aumentar o desempenho, basta comprar um *commodity server* (computador de baixo custo) e adicioná-lo ao *cluster* [CD10].

As arquiteturas tradicionais dos Sistemas de Gerenciamento de Banco de Dados para armazenamento de dados estruturados tem se mostrado insuficientes para lidar com este enorme volume de dados, conhecidos como *big data* [Lai09]. Para tais aplicações, tem-se utilizado sistemas NoSQL que oferecem técnicas de armazenamento distribuído através de MapReduce [DG08].

Além dos problemas relativos à escalabilidade, os sistemas NoSQL foram motivados por outras necessidades, como utilização de uma estrutura mais flexível que permita adicionar novos atributos em registros específicos sem ter que seguir um esquema de dados pré-definido.

Por outro lado, a onipresença da dimensão espacial nos dados aliado à popularidade de aplicações espaciais como também dos dispositivos com suporte à coleta de dados georreferenciados, como *smartphones*, GPS e câmeras fotográficas, tem aumentado este volume de informação gerado.

Em virtude desses avanços tecnológicos em dispositivos para aquisição da localização geográfica tem surgido vários serviços e redes sociais baseadas na localização, e.g. Foursquare[1], Gowalla[2], Loopt[3] e Brightkite[4]. O próprio Twitter já permite fazer o *geocoding* (georreferenciamento) de *tweets*. Para satisfazer à demanda dessas aplicações, alguns sistemas NoSQL, como o CouchDB-GeoCouch e o MongoDB, já provêem suporte para dados espaciais.

Portanto, há claramente a necessidade premente de se juntar toda esta gama de informação georreferenciada advinda de redes sociais como Twitter e Foursquare, com as informações georreferenciadas tradicionais, armazenadas em bancos de dados espaciais ou Infraestruturas de Dados Espaciais (IDE). Por exemplo, pode-se querer visualizar dados de *checkin* e *checkout* de um determinado grupo de usuários dentro de uma determinada região.

O problema tratado neste trabalho envolve a interoperabilidade entre SGBDs espaciais

[1]https://foursquare.com/

[2]http://gowalla.com/

[3]http://www.loopt.com/

[4]http://brightkite.com/

objeto-relacionais como, por exemplo, PostgreSQL-PostGIS ou Oracle Spatial, com sistemas NoSQL habilitados espacialmente, como o CouchDB-GeoCouch ou MongoDB. Em outras palavras, endereçou-se a problemática da interoperabilidade de dados geográficos proveniente de fontes de informação heterogêneas e distribuídas.

1.1 Objetivos

O objetivo do trabalho é desenvolver o *framework NoSQL-GeoServices*, que possibilite a interoperabilidade de dados armazenados em bancos de dados espaciais SQL e sistemas NoSQL habilitados espacialmente.

Os objetivos específicos deste trabalho são:

- Projeto e implementação de uma arquitetura que permita a interoperabilidade entre dados espaciais armazenados em banco de dados SQL e NoSQL, através de padrões de serviços OGC (*Open Geospatial Consortium*);
- Implementação de uma camada de serviços OGC de interoperabilidade, *Web Map Service* (WMS) [Inc06] e *Web Feature Service* (WFS) [Inc05], sobre a camada do sistema NoSQL orientado a documento com suporte espacial, de sorte a integrá-lo com outros bancos de dados espaciais através de um servidor de mapas como, por exemplo, o Geoserver[5];
- Facilitar o acesso a dados espaciais armazenados em sistemas NoSQL, de modo que qualquer aplicação cliente que implemente os serviços WMS e WFS, por exemplo, o OpenLayers[6], possa submeter uma consulta usando a mesma sintaxe para bancos de dados espaciais SQL e sistemas NoSQL de forma simples e transparente para o usuário da aplicação; e
- Permitir a extensão do *framework* desenvolvido para outros sistemas NoSQL.

[5]http://geoserver.org/

[6]http://openlayers.org/

1.2 Estrutura da Dissertação

O restante da dissertação está organizada da seguinte forma:

- Capítulo 2: contém a fundamentação teórica deste trabalho, que apresenta a terminologia básica e os principais conceitos utilizados, assim como uma introdução aos sistemas de informação geográfica e sistemas NoSQL;
- Capítulo 3: apresenta os principais trabalhos relacionados à interoperabilidade de dados espaciais, assim como os trabalhos voltados a desenvolver soluções para problemas decorrentes do processamento de dados geográficos em um ambiente distribuído;
- Capítulo 4: apresenta a arquitetura da solução proposta para interoperabilidade entre bancos de dados SQL e sistemas NoSQL;
- Capítulo 5: descreve o estudo de caso utilizando a solução proposta neste trabalho; e
- Capítulo 6: apresenta as conclusões do trabalho, recapitula as contribuições e mostra os trabalhos futuros.

1.3 Trabalhos Publicados

O seguinte trabalho foi publicado, como artigo completo, em decorrência deste trabalho de mestrado:

- "Using OGC Services to Interoperate Spatial Data Stored in SQL and NoSQL Databases" [dSBdLJdO+11].
  - XII Simpósio Brasileiro de Geoinformática (GeoInfo) 2011
  - Campos do Jordão, SP, Brasil

Capítulo 2

Fundamentação Teórica

Neste capítulo, será apresentada a terminologia básica e os conceitos fundamentais utilizados nesta dissertação. Como o objetivo deste trabalho é a interoperabilidade entre dados (espaciais e não-espaciais) armazenados em sistemas NoSQL e Sistemas Gerenciadores de Bancos de Dados Espaciais Objeto-Relacionais, será abordada uma introdução aos dados espaciais e aos serviços de interoperabilidade definidos pelo OGC (*Open Geospatial Consortium*). Além disso, serão apresentados os principais conceitos sobre sistemas NoSQL.

O restante deste capítulo está organizado da seguinte forma, na seção 2.1, é feita um introdução aos Sistemas de Informações Geográficas, dados espaciais e serviços de interoperabilidade de dados espaciais. Na seção 2.2, são apresentados alguns conceitos fundamentais, técnicas e padrões comuns aos sistemas NoSQL. Por fim, são feitas algumas considerações finais sobre o capítulo.

2.1 Sistemas de Informações Geográficas

Nesta seção, serão apresentadas formas de representação computacional do espaço geográfico. Além disso, será feita uma introdução aos serviços de interoperabilidade definidos pelo OGC.

2.1.1 Dados Espaciais

Sistemas de Informações Geográficas (SIG) são sistemas que realizam o tratamento computacional de dados geográficos [CCD+05]. A principal diferença entre um SIG e um sistema de informação tradicional é a capacidade de armazenar tanto atributos descritivos como geometrias de diferentes tipos (dados geográficos).

Aplicações que utilizam SIG necessitam de uma escolha adequada das representações computacionais para capturar a semântica do domínio da aplicação. Portanto, a escolha de uma representação computacional do espaço geográfico é uma questão fundamental em SIG. Gomes e Velho [Gom95] e Câmara [Câm95] expõem o paradigma dos quatro universos, o qual descreve quatro passos (universos) entre o mundo real e sua representação computacional (veja Figura 2.1).

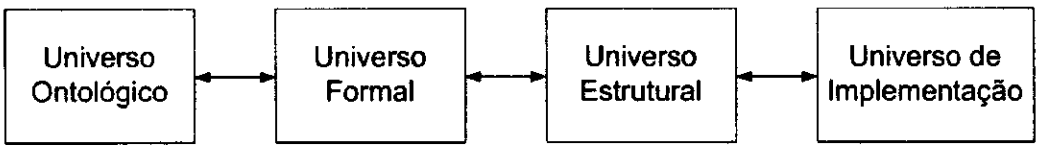


Figura 2.1: Paradigma dos quatro universos (Fonte: Camara *et al.* [CCD+05])

No universo ontológico (primeiro universo) as percepções do mundo real são materializadas em conceitos que descrevem a realidade.

O segundo passo do paradigma é o do universo formal. Neste estão os modelos lógicos e construções matemáticas que generalizam os conceitos do universo ontológico.

Em seguida, no universo estrutural as entidades do modelo formal são mapeadas para estruturas geométricas e alfanuméricas. Neste universo também são definidos algoritmos para realização de operações nas estruturas.

Por fim, o universo de implementação são realizadas escolhas como arquiteturas, linguagens e paradigmas de programação.

Apesar da importância de uma visão global dos passos para representação computacional do espaço geográfico, é necessário focar no universo em que a pesquisa será realizada. Como nosso trabalho visa a interoperabilidade entre bases de dados espaciais heterogêneas o universo de interesse é o estrutural, especificamente dados vetoriais $2D$ (para mais detalhes sobre estruturas e formas de representação de dados espaciais recomenda-se a leitura de [CCD+05]).

Na resposta de uma requisição *GetCapabilities* o servidor responde à solicitação retornando um arquivo XML (*eXtensible Markup Language*) que contém metadados sobre um servidor WMS indicando os seus dados e habilidades, como: serviços e formatos oferecidos, sistemas de referência espacial, lista de camadas/mapas disponíveis, estilos (SLD - *Styled Layer Descriptor*) e códigos específicos do fornecedor.

O documento de capacidades é dividido em duas partes: *Service* e *Capability*. A seção *Service* contém informações sobre o servidor, tais como: quem é o responsável pelo servidor, como contatá-lo, endereço do servidor, entre outras. A seção *Capability* contém informações sobre tipos de requisições aceitas, formatos de exceção e as camadas disponíveis no servidor.

GetMap

A operação *GetMap* retorna a imagem de um mapa cujos parâmetros geoespaciais e dimensionais são bem definidos. Uma solicitação *GetMap* permite que o cliente WMS possa especificar diferentes camadas, o sistema de referência espacial (SRS - *Spatial Reference System* ou CRS - *Coordinate Reference System*), a área geográfica e o formato de mapa retornado. Ao receber uma requisição *GetMap*, um servidor WMS irá satisfazer o pedido ou lançar uma exceção, de acordo com as instruções contidas na requisição *GetMap*.

Uma requisição *GetMap* é na verdade uma requisição HTTP GET que retorna uma imagem. Para entender os detalhes (parâmetros) da requisição *GetMap*, serão analisados os parâmetros passados na URL do exemplo a seguir:

http://localhost:8080/geoservcies/wms?version=1.1.1&request=getmap&layers=municipios
&styles=population&SRS=EPSG:4326&bbox=-180,-90,180,90&width=400&height=200
&format=image/png

Para entender como funciona a requisição a URL foi dividida nos seguintes trechos:

localhost:8080 - o nome ou IP da máquina e a porta em que o servidor de mapas está disponível.

geoservices/wms - programa que é executado no servidor e trata as requisições.

Os parâmetros são passados para o programa em execução no servidor para especificar seu modo de operação. A seguir, estão descritos os parâmetros (acompanhados dos respectivos valores).

- version=1.1.1 - Define qual versão da especificação WMS deverá ser usada. Dependendo da versão da especificação pode haver variações nos parâmetros que estão disponíveis ou o que eles significam.
- request=getmap - Especifica qual a requisição desejada, no caso *getmap*.
- layers=municipios - Elenca quais camadas deverão ser apresentadas no mapa. Caso mais de uma camada seja solicitada então devem ser separadas por vírgula.
- styles=population - descreve o estilo (cor, espessura de linhas, etc.) a ser aplicado em uma determinada camada. Esse parâmetro é opcional e se não for passado o servidor terá um estilo padrão especificado que será usado para desenhar o mapa.
- SRS=EPSG:4326 - A projeção exigida para o mapa, são códigos usados em um esquema chamado EPSG (*European Petroleum Survey Group*).
- bbox=-180,-90,180,90 - o Bounding Box da requisição, ou seja, as coordenadas dos pontos inferior esquerdo e superior direito que determinam o retângulo com a região a ser desenhada no mapa.
- width=400 - define o comprimento em *pixels* da imagem.
- height=200 - define a altura em *pixels* da imagem.
- format=image/png - especifica o tipo MIME (*Multipurpose Internet Mail Extensions*) da imagem, por exemplo image/gif, image/jpg, image/svg+xml são valores comuns.

Caso um valor inadequado seja passado para os parâmetros *width* e *height* a imagem resultante pode ficar "achatada".

Se todos os parâmetros obrigatórios forem informados com a requisição como resposta será fornecido um mapa no formato solicitado. Entretanto, se algum erro ocorrer (por exemplo, falta de parâmetro obrigatório) o servidor WMS enviará uma mensagem de erro.

GetFeatureInfo

A requisição *GetFeatureInfo* retorna informações sobre feições específicas mostradas em um mapa. Se um servidor WMS fornece esta operação, então um cliente WMS pode solicitar

informações sobre as feições em um mapa. A requisição *GetFeatureInfo* é formada essencialmente pelos mesmos parâmetros da requisição *GetMap*, com adição dos parâmetros X e Y referentes as coordenadas, medidas em *pixels* a partir do canto superior esquerdo, na imagem e do parâmetro *query-layers* que diz as camadas que deseja consultar. Apenas as camadas descritas como *"queryable"*=1 no XML retornado da requisição *GetCapabilities* podem ser consultadas através da requisição *GetFeatureInfo*.

Existem alguns problemas no detalhamento da requisição *GetFeatureInfo*, pois a especificação descreve apenas que os dados devem ser retornados, mas não como. Assim, servidores diferentes podem retornar formatos diferentes - dificultando a implementação de aplicações cliente.

Serviço WFS (*Web Feature Service*)

A especificação Web Feature Service (WFS) [Inc05] fornece uma interface para consultar, e em alguns casos atualizar e adicionar, feições geográficas independentemente da fonte de dados. Enquanto WMS cria um mapa com os dados, o WFS fornece acesso aos dados, normalmente codificados em GML (*Geographic Markup Language*).

Existem três tipos de requisições que um servidor WFS deve fornecer: *GetCapabilities*, *DescribeFeatureType* e *GetFeature*. Nas subseções a seguir serão detalhadas cada uma dessas requisições.

GetCapabilities

A requisição *GetCapabilities* do WFS é semelhante à operação de mesmo nome apresentada na seção anterior. Essa operação permite às aplicações clientes descobrirem quais os serviços e tipos de dados são fornecidos pelo servidor WFS.

Assim como no WMS, essa requisição retorna um documento XML. No caso do WFS o XML retornado contém as seguintes seções: *Service*, *Capabilities*, *FeatureTypeList* e *Filter Capabilities*.

A seção *Service* contém exatamente as mesmas seções que o documento *getCapabilities* do WMS. Isso porque as especificações do WMS e WFS estendem a especificação *OGC Common Implementation*.

A seção *Capabilities*, também definida na especificação *OGC Common Implementation*, lista as requisições fornecidas pelo servidor WFS. A especificação define requisições obrigatórias (*getCapabilities*, *getFeature* e *describeFeatureType*) e requisições opcionais (*transaction*, *lockFeature* e *getFeatureWithLock*). As requisições opcionais estão relacionadas com WFS transacional (em que um cliente pode modificar feições através de requisições WFS).

A seção *FeatureTypeList* contém uma lista de operações fornecidas pelo servidor (e em quais camadas as operações podem ser aplicadas) e uma lista das *FeatureTypes* disponíveis.

Por fim, a seção *Filter Capabilities* descreve os filtros fornecidos pelo servidor.

GetFeature

A operação *GetFeature* fornece um mecanismo para recuperar feições geográficas (*features*), a partir de um servidor, que satisfaçam filtros espaciais ou não-espaciais. A solicitação pode ser feita tanto por uma requisição *HTTP Get* quanto por uma requisição *HTTP Post* (com um XML no corpo da requisição). A especificação exige como formato obrigatório para resposta da requisição *GetFeature* o formato GML (*Geography Markup Language*) [Inc07]. A resposta de uma requisição *GetFeature* pode conter zero ou mais feições (*features*) que satisfaçam expressões da consulta especificadas na requisição [Inc05]. A especificação fornece ainda uma versão transacional (WFS-T) que define como criar, atualizar e excluir de feições vetoriais no servidor.

As consultas devem ser construídas usando a sintaxe definida na especificação *Filter Encoding* [Vre05]. *Filter Encoding* é uma codificação em XML usada para expressar filtros, com a mesma finalidade que a cláusula *WHERE* em SQL.

A especificação *Filter Encoding* define quatro tipos básicos de filtros que podem ser combinados para criar filtros complexos, são eles: filtros pelo identificador da feição (*feature id* - fid), filtros espaciais, filtros de comparação e filtros lógicos. A seguir serão detalhados cada um destes filtros.

Filtros pelo Identificador

O tipo mais simples de filtro é o que referencia uma feição pelo seu id (fid). Um exemplo deste tipo de filtro pode ser visto no quadro 2.1.

Exemplo 2.1: Exemplo de filtro usando id da feição (*feature id*).

```
<ogc:FeatureId fid="br_230"/>
```

Filtros Espaciais

Filtros espaciais restringem uma consulta por meio de análise geoespacial. Existem os seguintes tipos de filtros espaciais definidos pela especificação *Filter Encoding*:

- ***Equals*** - Verdadeiro se as duas geometrias são iguais espacialmente.
- ***Disjoint*** - Verdadeiro se as duas geometrias não se tocam nem se cruzam.
- ***Touches*** - Verdadeiro se e somente se os únicos pontos comuns das duas geometrias estão na união dos limites das geometrias.
- ***Overlaps*** - Verdadeiro se uma geometria sobrepuser a outra, ou seja, se a região que representa a interseção entre elas tiver a mesma dimensão que as instâncias e as instâncias não forem iguais.
- ***Crosses*** - Verdadeiro se as duas geometrias se cruzam. Para ser considerado cruzamento as seguintes condições devem ser atendidas: (*i*) A interseção das geometrias deve resultar em uma geometria não vazia cuja dimensão é menor que a dimensão máxima das geometrias de origem; (*ii*) O conjunto de interseção é interior a ambas as geometrias de origem [LOU08].
- ***Intersect*** - Verdadeiro se as duas geometrias se interceptarem. O mesmo resultado pode ser obtido com *Not Disjoint*.
- ***Contains*** - Verdadeiro se a segunda geometria está totalmente dentro da geometria do primeiro.
- ***Within*** - Verdadeiro se a primeira geometria está totalmente dentro da segunda geometria.
- ***DWithin*** - Verdadeiro se uma geometria está dentro de uma certa distância de outra geometria.

- ***BBOX*** - Verdadeiro se a geometria está dentro do envelope fornecido.

O exemplo 2.2 apresenta um filtro espacial, que recupera as feições geográficas que estão dentro do envelope fornecido.

Exemplo 2.2: Exemplo de filtro espacial.

```
<ogc:BBOX>
  <ogc:PropertyName>the_geom</ogc:PropertyName>
  <gml:Envelope srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
    <gml:lowerCorner>-75.102613 40.212597</gml:lowerCorner>
    <gml:upperCorner>-72.361859 41.512517</gml:upperCorner>
  </gml:Envelope>
</ogc:BBOX>
```

Filtros de Comparação

Um operador de comparação é usado para formar expressões que avaliam a comparação matemática entre dois argumentos. Os operadores de comparação definidos na especificação são descritos a seguir:

- ***PropertyIsEqualTo*** **(=)** - Verdadeiro se as duas expressões são equivalentes
- ***PropertyIsNotEqualTo*** **(<>)** - Verdadeiro se as duas expressões são diferentes, o oposto do *PropertyIsEqualTo*.
- ***PropertyIsLessThan*** **(<)** - Verdadeiro se a primeira expressão é menor do que o segunda.
- ***PropertyIsLessThanOrEqualTo*** **(<=)** - Verdadeiro se a primeira expressão é menor ou igual a segunda.
- ***PropertyIsGreaterThan*** **(>)** - Verdadeiro se a primeira expressão é maior do que a segunda expressão.
- ***PropertyIsGreaterThanOrEqualTo*** **(>=)** - Verdadeiro se a primeira expressão é maior ou igual a segunda.

- ***PropertyIsLike*** - Verdadeiro se a propriedade da geometria corresponde à expressão literal fornecida.

O exemplo 2.3 apresenta um filtro de comparação, que recupera as cidades (feições geográficas) da Mesorregião Agreste.

Exemplo 2.3: Exemplo de filtro de comparação.

```
<ogc:PropertyIsEqualTo>
  <ogc:PropertyName>MESORREGIAO</ogc:PropertyName>
  <ogc:Literal>Agreste</ogc:Literal>
</ogc:PropertyIsEqualTo>
```

Filtros Lógicos

Existem três operações lógicas definidas na especificação: ***And***, ***Or*** e ***Not***. Filtros lógicos podem ser usados para combinar os outros filtros. O exemplo 2.4 apresenta um filtro lógico, que recupera as cidades do Nordeste do Brasil que possuem mais de 1 milhão de habitantes.

Exemplo 2.4: Exemplo de filtro lógico.

```
<ogc:And>
  <ogc:BBOX>
    <ogc:PropertyName>the_geom</ogc:PropertyName>
    <gml:Envelope srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
      <gml:lowerCorner>-75.102613 40.212597</gml:lowerCorner>
      <gml:upperCorner>-72.361859 41.512517</gml:upperCorner>
    </gml:Envelope>
  </ogc:BBOX>
  <ogc:PropertyIsGreaterThan>
    <ogc:PropertyName>NR_HABITANTES</ogc:PropertyName>
    <ogc:Literal>1000000</ogc:Literal>
  </ogc:PropertyIsEqualTo>
</ogc:And>
```

WFS-T

Uma requisição WFS transacional (WFS-T) permite inserir, atualizar e excluir feições geográficas do servidor. O protocolo WFS implementado foi o *read-only*, ou seja, não-

transacional (usado apenas para consultar feições geográficas). A resposta de uma requisição WFS transacional é um arquivo XML com o elemento *WFS_TransactionResponse* como raiz. Para cada transação, este elemento contém um *TransactionResult*, que pode apresentar como resultados: *SUCCESS*, *FAILED* ou *PARTIAL*.

Para inserir elementos é necessário fornecer uma lista das feições geográficas que se deseja inserir. No exemplo 2.5 é mostrada a definição da inserção de uma rua usando WFS-T.

Exemplo 2.5: Exemplo de inserção de feição usando WFS-T.

```
<wfs:Insert>
  <br:estrada>
    <br:the_geom>
      <gml:MultiLineString srsName="http://www.opengis.net/gml/srs/epsg.
          xml#4326">
        <gml:lineStringMember>
          <gml:LineString>
            <gml:coordinates decimal="." cs="," ts=" ">
              494475.71056415,5433016.8189323
                  494982.70115662,5435041.95096618
            </gml:coordinates>
          </gml:LineString>
        </gml:lineStringMember>
      </gml:MultiLineString>
    </br:the_geom>
  </br:estrada>
</wfs:Insert>
```

A atualização de elementos é definida através de uma operação isolada (apenas uma atualização por vez). Na definição é fornecida uma lista de propriedades das feições com seus novos valores e um filtro é utilizado para limitar as feições que serão atualizadas com o novo valor. O exemplo 2.6 apresenta a atualização na propriedade "TIPO" da rodovia "br_230" usando WFS-T.

Exemplo 2.6: Exemplo de atualização de feição usando WFS-T.

```
<wfs:Update typeName="br:rodovias">
  <wfs:Property>
```

```
    <wfs:Name>TIPO</wfs:Name>
    <wfs:Value>BR</wfs:Value>
  </wfs:Property>
  <ogc:Filter>
    <ogc:FeatureId fid="br_230"/>
  </ogc:Filter>
</wfs:Update>
```

Transações para excluir elementos contém apenas um filtro que limita as feições a excluir. No exemplo 2.7 são excluídas todas as estradas do tipo "TRAVESSA".

Exemplo 2.7: Exemplo de exclusão de feição usando WFS-T.

```
<wfs:Delete typeName="br:estradas">
  <ogc:Filter>
    <ogc:PropertyIsEqualTo>
      <ogc:PropertyName>br:TIPO</ogc:PropertyName>
      <ogc:Literal>TRAVESSA</ogc:Literal>
    </ogc:PropertyIsEqualTo>
  </ogc:Filter>
</wfs:Delete>
```

2.2 Sistemas NoSQL

A arquitetura tradicional de sistemas dos gerenciamento de bancos de dados relacionais, originalmente projetada na década de 1970 para o processamento de dados de negócios, tem sido usada por muitas aplicações centradas em dados com grande variação de características e requisitos [SC05].

Stonebraker e Cetintemel [SC05] argumentaram que essa arquitetura tende a ser substituída por *engines* especializados de bancos de dados para os seguintes mercados: *Data warehouses*, processamento de *stream*, processamento de texto, dados semi-estruturados e bancos de dados científicos.

Nesse cenário, têm surgido alguns bancos de dados não-relacionais com recursos limitados e sem fornecer todas as propriedades ACID (Atomicidade, Consistência, Isolamento e Durabilidade), que são mais adequadas para o uso em um ambiente distribuído [Ore10]. Tais

bancos de dados são conhecidos como NoSQL (*"Not Only SQL"*) e são projetados para executar em *clusters* formados por computadores de baixo custo (*commodity servers*), portanto, têm de ser distribuídos e tolerantes a falhas.

Nos últimos anos, o movimento NoSQL atraiu um grande número de empresas e projetos, que repercurtiu na migração de bancos de dados relacionais para sistemas NoSQL por parte das grandes empresas Web.

Nesse contexto, surgiram soluções NoSQL como Cassandra [LM10] desenvolvido inicialmente para o *Facebook* e hoje também usado pelo Twitter e *Digg*, projeto *Voldermort* desenvolvido e usado pelo *LinkedIn*, além das soluções usadas em serviços de computação na nuvem como Amazon SimpleDB e *CouchDB*.

Muitas das soluções NoSQL foram inspiradas pelo *Bigtable* da Google [CDG+08] ou pelo *Dynamo* da Amazon [DHJ+07]. *Bigtable* influenciou sistemas NoSQL conhecidos como *column-stores* (e.g. *HyperTable*, *HBase*), enquanto Dynamo inspirou os sistemas NoSQL do tipo *key/values-stores* (e.g. Cassandra, Projeto *Voldemort* e Oracle NoSQL).

Existem ainda outras abordagens, como banco de dados orientados a documentos (e.g. MongoDB, CouchDB), que podem ser consideradas como *key/value-stores* com recursos adicionais.

Muitos sistemas NoSQL são projetados para ser mais escaláveis do que bancos de dados SQL tradicionais. Alguns sistemas NoSQL afirmam retornar consultas mais rápido que bancos de dados SQL, enquanto outros podem executar escritas mais rápido do que leituras [Dre11].

Não existe uma definição inteiramente aceita pela comunidade de banco de dados sobre as características de tais sistemas. Portanto, neste trabalho serão utilizados seis critérios descritos em [Cat11] para caracterizar sistemas NoSQL, a saber:

1. Habilidade de escalar horizontalmente operações de leitura e escrita em vários servidores;

2. Habilidade de replicar e distribuir (particionar) dados em vários servidores;

3. Uma interface de chamada ou protocolo simples (em contraste com SQL);

4. Um modelo mais fraco de concorrência do que as transações ACID dos sistemas de banco de dados relacionais (SQL);

5. Uso eficiente de índices distribuídos e RAM para armazenamento de dados; e

6. Capacidade de adicionar dinamicamente novos atributos nos registros de dados (*schema free* ou com restrições mais fracas).

Muitos autores têm utilizado uma definição ampla de NoSQL, incluindo qualquer sistema de banco de dados que não seja relacional (e.g. sistemas orientados a grafos e sistemas de banco de dados orientados a objetos). Ao contrário dos sistemas NoSQL, estes sistemas geralmente fornecem propriedades transacionais ACID [Cat11]. Portanto, estes sistemas estão além do escopo deste trabalho.

2.2.1 Características Globais e Conceitos Teóricos

Como os sistemas NoSQL seguem os princípios listados anteriormente, são necessários conceitos teóricos que permitam a implementação dessas funcionalidades. Alguns dos conceitos mencionados na literatura são: *MapReduce*, teorema CAP, propriedades BASE e mecanismos de particionamento dos dados.

MapReduce

MapReduce [DG08] é um conceito fundamental para alguns sistemas NoSQL. MapReduce é um modelo de programação que permite processar grandes volumes de dados distribuídos em uma rede, dividindo as tarefas em duas etapas descritas pelas funções *map* e *reduce*. Na primeira etapa, um nó mestre divide um problema em subproblemas, estes são distribuídos para outros nós da rede que executam a função *map* e produzem uma saída intermediária. Em seguida, a saída intermediária é processada por máquinas executando a função *reduce* para criar o resultado final a partir dos resultados intermediários, por exemplo alguma agregação. Tanto a função *map*, quanto a *reduce* não dependem de um estado da máquina em que são executadas, portanto, produzem saídas idênticas, em cada ambiente de execução, fornecidos os mesmos dados de entrada. Sistemas desenvolvidos seguindo esse modelo permitem que programadores sem nenhuma experiência com sistemas distribuídos ou paralelos possam utilizar recursos do sistema distribuído.

Além da implementação *MapReduce* da Google existem vários outras. Por exemplo, várias linguagens de programação (e.g. Python), *frameworks* (e.g. Apache Hadoop) e também

sistemas NoSQL (e.g. CouchDB) adotam o paradigma MapReduce. Em sistemas NoSQL, MapReduce é usado para realização de consultas ou para criação de índices.

Teorema CAP

Desde que foi proposto por Eric Brewer, o teorema CAP [Bre00] vem sendo largamente adotado por grandes empresas de comércio eletrônico (e.g. Amazon) e pelos defensores dos sistemas NoSQL [GL02]. A seguir, serão descritas cada uma das propriedades da sigla CAP (*Consistency, Availability, Partition Tolerance*):

Consistency **(Consistência) -** um sistema distribuído é considerado consistente se depois de uma operação de atualização, todos os leitores visualizam o efeito da atualização.

Availability **(Disponibilidade) -** (alta) disponibilidade significa que um sistema foi projetado de forma a permitir o funcionamento mesmo em presença de falhas.

Partition Tolerance **(Tolerância à partição) -** entendida como a habilidade de um sistema permanecer funcionando mesmo em presença de partições na rede.

O teorema CAP afirma que um sistema pode ter apenas duas em três das propriedades acima listadas. Os sistemas NoSQL em geral abdicam da consistência [Cat11], por exemplo, bancos de dados distribuídos para a Web 2.0 tipicamente enfatizam disponibilidade e tolerância a partição. Assim, um estado consistente é eventualmente alcançado em sistemas NoSQL, mas não imediatamente após uma operação.

ACID vs. BASE

Como discutido anteriormente, sistemas NoSQL geralmente não fornecem todas as propriedades transacionais ACID. Por outro lado, possuem propriedades conhecidas como BASE (*Basically Available, Soft state, Eventually consistent*) [Pri08]. As propriedades BASE podem ser resumidas da seguinte forma: um aplicativo funciona basicamente o tempo todo (*Basically Available*), não tem que ser consistente o tempo todo (*Soft state*), mas terá em algum momento eventual um estado conhecido (*Eventually consistent*).

Não existe um padrão de quais propriedades ACID os sistemas NoSQL abdicam, sendo este um dos pontos de diferenciação entre tais sistemas. Por exemplo, muitos sistemas são caracterizados como "eventualmente consistentes" (atualizações são eventualmente propagadas para todos os nós), entretanto muitos fornecem mecanismos com algum grau de consistência como o *Multi-Version Concurrency Control* (MVCC) [Cat11].

MVCC é uma técnica para gerenciamento de transações concorrentes em banco de dados. Diferentemente dos bloqueios tipicamente usados em SGBDs relacionais, MVCC permite operações de leitura e escrita paralelas. MVCC cria novas versões em um banco de dados no processo de escrita. Assim, versões anteriores podem ser consultadas e conflitos de versões podem ser resolvidos pelo sistema.

Particionamento dos Dados

Um banco de dados pode ser escalável de três diferentes formas. Pode ser escalável na quantidade de operações de leitura, no número de operações de escrita ou no tamanho do banco de dados [Ore10]. Existem duas tecnologias usadas para conseguir isso: replicação e *sharding*.

No contexto de sistemas distribuídos, replicação significa que um dado é armazenado em mais de um nó. Replicação dos dados melhora o desempenho de leituras no banco de dados, pois permite o balanceamento de carga. Além disso, torna o cluster tolerante a falhas. Se uma máquina falhar, existirá outra com os mesmos dados que possa substituí-la.

Muitas vezes os dados são replicados em diferentes *data centers*, o que faz com que os dados estejam mais próximos dos usuários (diminuição da latência). Além de permitir recuperação do sistema em casos de desastres naturais em uma região.

Entretanto, operações de escrita não são vantajosas com dados replicados. Porque quando uma operação de escrita precisa ser realizada em um banco de dados replicado existem duas opções disponíveis: (i) realizar a operação de escrita em todos os nós do cluster antes de retornar uma resposta (ii) realizar a operação apenas em um número limitado de nós e em seguida realizar a operação em outros nós da rede. A escolha por uma destas opções implica na seleção de algumas propriedades ACID, a saber: disponibilidade e consistência (ver Teorema CAP - seção 2.2.1).

Sharding é o nome dado ao processo de divisão e armazenamento dos dados em diferen-

tes máquinas. Dividindo os dados entre diferentes máquinas, torna-se possível armazenar mais dados e lidar com mais carga sem a necessidade de máquinas de grande porte [CD10]. Os *shards* (blocos de dados) também podem ser replicados por razões de confiabilidade e balanceamento de carga. A desvantagem de cenários *sharding* é que junções entre *shards* de dados não são possíveis. Portanto, aplicações cliente ou camada de *proxy* (dentro ou fora do banco de dados) tem de enviar várias requisições e realizar um pós-processamento nos resultados (por exemplo, filtrar, agregar). Muitos sistemas NoSQL adotaram *sharding* como uma característica-chave e alguns até mesmo fornecem particionamento automático e balanceamento de dados entre os nós - por exemplo, MongoDB.

2.2.2 Sistemas NoSQL para Domínio Espacial

Como visto nas seções anteriores, muitos sistemas NoSQL foram projetados para o grande volume de dados das aplicações da Web 2.0. No contexto da Web 2.0 a dimensão espacial é de particular interesse, como serviços baseados em localização (e.g. Foursquare, Gowalla, Loopt e Brightkite) além de inúmeras aplicações como Twitter, Facebook, Flickr e Panoramio que permitem ao usuário georreferenciar seu conteúdo.

Para atender a essa demanda, os sistemas NoSQL também passaram a considerar o armazenamento/consulta de dados espaciais. Os sistemas NoSQL orientados a documentos (MongoDB e CouchDB) foram os primeiros a armazenar informação espacial. Além deles, existem soluções como o SimpleGeo que fornece uma API para facilitar o desenvolvimento de sistemas baseadas em localização (pontos de interesse). O SimpleGeo estendeu o sistema NoSQL Cassandra com um índice espacial. O código atual é proprietário e o Cassandra não fornece nativamente qualquer forma de indexação espacial.

Alguns bancos de dados orientados a grafos (e.g. Neo4J[7]) permitem armazenar dados espaciais de forma nativa. Entretanto, como citado anteriormente, tais sistemas seguem o modelo transacional ACID e, portanto, estão fora do escopo deste trabalho.

Neste trabalho serão utilizados os sistemas NoSQL MongoDB e CouchDB. Tanto o MongoDB quanto o CouchDB são Sistemas NoSQL orientados a documentos e habilitados espacialmente. Ambos são *schema-free*, ou seja, não há tabelas e colunas, chaves primárias e estrangeiras, junções e relacionamentos, diferentemente dos bancos de dados tradicionais

[7]http://neo4j.org/

baseados em SQL.

Em um banco de dados relacional, deve-se especificar o esquema de todos os dados em cada tabela, e todos os dados contidos na tabela devem obedecer estritamente ao esquema. Por outro lado, em um sistema *schema-free* orientado a documentos cada documento é independente de outro e não existe uma estrutura fixa que deve ser seguida.

Isso afeta diretamente a forma como os dados são atualizados. Alterar um banco de dados SQL pode envolver uma série de dependências e problemas de integridade. O que não ocorre em um sistema NoSQL orientado a documentos. Cada documento é auto-suficiente, logo não é necessário armazenar valores nulos redundantes e podem-se definir novos campos para cada documento de forma independente.

Assim, tratamentos precisam ser realizados no nível da aplicação, ao invés de obrigar todos os dados terem o mesmo formato. Essa flexibilidade é bastante conveniente para projetos que trabalham com modelos de dados em evolução.

Em um sistema NoSQL orientado a documentos, o conceito de linha é substituído por um conceito mais flexível, o de "documento". Através de documentos embutidos é possível representar relacionamentos hierárquicos complexos em um único registro. Essa forma de modelar os dados se encaixa naturalmente com a forma na qual os desenvolvedores de linguagens orientadas a objetos modelam os dados.

Os sistemas NoSQL CouchDB e MongoDB armazenam os dados espaciais como documentos no formato GeoJSON, que é uma extensão do formato JSON (*JavaScript Object Notation*) para a codificação de estruturas de dados geográficos. O GeoJSON surgiu como um padrão simples de dados espaciais para a Web. Nesse formato, é possível representar geometrias descritas por Pontos, Multi-Pontos, Linhas, Multi-Linhas, Polígonos e Multi-Polígonos. O Trecho de Código 2.8 apresenta um documento no formato GeoJSON, no qual são armazenadas informações sobre o município de Campina Grande, localizado no estado da Paraíba.

Exemplo 2.8: Documento no formato GeoJSON.

```
{
   "_id": "453b58b521bee305d63f07ddce28f514",
   "_rev": "1-8ca3476d75564f932097757a8e0efc50",
   "type": "Feature",
   "properties": {
       "OBJECTID": 218,
       "Localidade": "Municipio",
       "CODIGO_IBG": 2504009,
       "Nome": "Campina Grande"
   },
   "geometry": {
       "type": "Point",
       "coordinates": [
           -35.89884,
           -7.21831
       ]
   }
}
```

CouchDB

No CouchDB, um banco de dados é armazenado como uma coleção de documentos JSON e toda interação é realizada usando o protocolo HTTP através de uma interface REST (*Representational State Transfer*) [Fie00]. Em outras palavras, para usar a API (*Application Programming Interface*) do CouchDB, faz-se uma requisição HTTP para o servidor CouchDB, que processa o pedido em função de um Identificador Uniforme de Recursos (URI - *Uniform Resource Identifier*), do método HTTP usado (*GET*, *POST*, *PUT*, *DELETE*) e dos dados enviados com a solicitação. Ao concluir o processamento da requisição, o servidor retorna, no formato JSON, os dados ou detalhes de algum erro que tenha sido encontrado na requisição.

Os dados são indexados e pesquisados mediante a escrita de *views* baseadas em JavaScript semelhantes a *stored procedures*. Uma *view* é formada por uma função *map* e, opcionalmente, por uma função *reduce*.

MongoDB

O MongoDB apresenta algumas similaridades com o CouchDB: fornece índices sobre coleções, não utiliza bloqueios (*lockless*) e fornece um mecanismo de consulta a documentos. Segundo Cattell [Cat11] as principais diferenças do MongoDB em relação ao CouchDB são:

- O MongoDB possui *sharding* automático, distribuindo documentos entre os servidores.
- No CouchDB, a replicação é utilizada para escalabilidade, enquanto no MongoDB é usada principalmente para tolerância a falhas.
- O MongoDB permite consultas dinâmicas com o uso automático de índices. No CouchDB os dados são indexados e pesquisados com a escrita *views Map-Reduce*.
- O CouchDB oferece MVCC (*Multi-Version Concurrency Control*) em documentos, enquanto MongoDB fornece operações atômicas em campos.

O MongoDB utiliza BSON ("*Binary JSON*") como formato de armazenamento de dados e de transferência de documentos na rede. BSON é um formato binário para representar estruturas de dados simples e *arrays* associativos (tipo abstrato de dados composto por uma coleção de pares chave/valor - também conhecidos como dicionários ou mapas). Assim como no CouchDB, o MongoDB utiliza *MapReduce* como ferramenta de agregação. MongoDB suporta índices bidimensionais geoespaciais.

2.3 Considerações Finais

Neste capítulo, foram apresentadas formas de representação computacional para dados espaciais, assim como serviços de interoperabilidade de dados propostos pelo OGC. Além disso, mostrou-se uma motivação para o surgimento dos sistemas NoSQL, bem como suas características e alguns conceitos fundamentais que tornam possível a implementação dessas características. Como visto neste capítulo, sistemas NoSQL são capazes de lidar com certos requisitos, por exemplo, escalabilidade horizontal e ausência de esquema. Por outro lado, se é necessário consistência, segurança ou consultas complexas então os SGBDs Relacionais são preferíveis [Sch11].

No capítulo seguinte, é exposta uma análise sobre os principais trabalhos relacionados com esta dissertação.

Capítulo 3

Trabalhos Relacionados

Neste capítulo, serão apresentados alguns trabalhos propostos na literatura com o objetivo de desenvolver soluções para interoperabilidade de bases de dados geográficos. Além disso, serão analisados trabalhos que focam no processamento de dados espaciais em grandes massas de dados (*Big Data*).

Interoperabilidade é a capacidade de um sistema, ou componentes de um sistema, cooperarem entre si, para permitir a portabilidade de informações. Em Sistemas de Informação Geográfica (SIG), interoperabilidade fornece meios pelos quais SIGs espacialmente distribuídos possam ser conectados em uma rede, a fim de trocar dados de forma transparente [Bis98].

A interoperabilidade é um dos desafios mais importantes relacionados a SIGs. Nos últimos anos, a pesquisa em interoperabilidade evoluiu da simples troca *off-line* de arquivos em formato padronizado até as primeiras iniciativas no tratamento de aspectos semânticos dos dados [AD06].

Com o rápido desenvolvimento de sistemas de informação e paradigmas de bancos de dados distribuídos, usuários de SIGs perceberam a necessidade de obter uma visão integrada de tais sistemas. Entretanto, como as bases de dados são modeladas para atender aos requisitos particulares das organizações, quando há necessidade de compartilhá-las, vários problemas de heterogeneidade ocorrem. Bishr [Bis98] classificou em três os tipos de heterogeneidade: heterogeneidade semântica, heterogeneidade esquemática e heterogeneidade sintática. A heterogeneidade semântica denota diferenças no significado e interpretação da mesma unidade de dado. A heterogeneidade esquemática refere-se as diferentes representações de uma mesma entidade do mundo real, por exemplo, objetos em um banco de dados são

considerados propriedades em outro, ou classes de objetos podem ter diferentes hierarquias de agregação ou generalização. Por fim, a heterogeneidade sintática ocorre devido as diferentes tecnologias para implementação de banco de dados, como os diferentes paradigmas (objeto-relacional ou orientado a objetos) e as diferentes formas de representar a geometria dos objetos (*raster* ou vetorial).

Integração e interoperabilidade de dados geográficos armazenados em diferentes fontes de informação são antigos desafios para a comunidade de dados geoespaciais, sendo abordados por uma grande quantidade de projetos de pesquisa [AS99, Bis98, BEL02, BZ09, CTdFM99, DPS98, GMZB99, UVS02].

Alguns destes trabalhos abordaram questões como formatos e representação heterogênea em conjuntos de dados espaciais, metadados espaciais para integração de dados espaciais, padrões de intercâmbio de dados espaciais, qualidade de dados e propagação de erros, entre outros.

Outros tratam da interoperabilidade semântica dos dados espaciais usando ontologias para representar os aspectos semânticos em comum das fontes de dados espaciais que serão integradas [BZ09].

Boucelma et al. [BEL02] apresentaram um sistema de mediação baseado na especificação *Web Feature Service* (WFS) para a interoperabilidade de SIGs. A arquitetura proposta para o sistema de mediação geográfica é composta principalmente de três camadas: um mediador SIG, servidores WFS e fontes de dados. Este trabalho objetivou fornecer: (*i*) uma visão integrada dos dados disponibilizados pelas fontes, e (*ii*) uma linguagem de consulta geográfica para acessar e manipular dados integrados.

No artigo Boucelma et al. [BEL02] apresentaram um sistema de integração não-materializada de dados geográficos que estende sistemas de integração de dados existentes para acomodar os operadores de SIG. O sistema desenvolvido neste trabalho obedece a uma arquitetura de mediação/*wrapper* padrão descrita na literatura, estendida com o conceito de *wrapper* derivado (usado para lidar com os operadores de SIG). Um *wrapper* derivado é um *wrapper* de uma fonte virtual que corresponde a uma fonte de dados local (*buffer*) contendo resultados da execução de um operador de SIG. *Wrappers* derivados capturam diferentes capacidades de consulta e permitem a resolução de conflitos de representação dos dados.

O tipo de configuração descrita anteriormente pode ser vista como parte de uma infraes-

trutura de dados espaciais (IDE). IDEs são uma abordagem para criação, distribuição e uso de informação geográfica de forma interoperável, divulgando dados espaciais com controle de qualidade associado, informações de metadados e descrições semânticas [DJ09].

Davis e Alves [AD06] propuseram uma arquitetura para uma infraestrutura de dados espaciais de âmbito local (*Local Spatial Data Infrastructure* - LSDI) baseada em padrões OGC (*Open Geospatial Consortium*). Esse trabalho avaliou alguns aspectos das especificações OWS (*OGC Web Services*) e dos principais serviços OGC no contexto de aplicações urbanas. Através destas aplicações, foram identificadas algumas dificuldades de implementação características de SIGs, tais como mecanismos de tolerância a falhas não-padronizados, clientes fortemente dependentes de fornecedores, entre outros.

Para realizar a avaliação dos serviços, foi implementado um protótipo baseado no modelo de serviços abstratos do OGC para um caso de uso do mundo real, para testar a utilidade dos padrões OGC para LSDI, considerando especificamente o desenvolvimento de aplicações cliente. No protótipo, foram desenvolvidos novos serviços para a infraestrutura, que facilitam o desenvolvimento de clientes de SDI de âmbito local e interoperabilidade de SIGs com características urbanas.

Um dos serviços propostos, chamado de *Data eXchange Service* (DXS), foi projetado para substituir o armazenamento local e a função persistência dos provedores de serviços por um serviço "terceirizado". Isto beneficia os computadores móveis e *thin clients*, por não exigir espaço de armazenamento local, durante o processamento de um *Web Service*. Outro serviço proposto, chamado de *Client Access Service* (CAS), realiza a mediação da conexão entre o cliente e servidor. Este serviço permite a comunicação assíncrona utilizando apenas padrões de serviço Web, além de evitar o desperdício de recursos da rede e processamento. Assim, os novos serviços constituem uma síntese de características ausentes nas tecnologias padrão.

Davis *et al.* [DJ09] apresentaram uma visão sobre os requisitos de projeto e implementação de um ambiente WMS (*Web Map Service*) para melhorar o acesso interativo às fontes de informações geográficas usando computadores móveis. Para isso propuseram a divisão da lógica do cliente WMS em duas unidades, das quais apenas uma é executada no cliente móvel (*Mobile Client* - MC), enquanto que a outra é executada por um cliente disponível através da rede (*WMS Connectivity Layer* - WCL). O cliente disponível na rede (WCL) im-

plementa a especificação WMS completa. Assim, a comunicação entre este e o cliente móvel é otimizada, utilizando protocolos mais simples e técnicas para melhorar a experiência do usuário utilizando o dispositivo móvel.

O cliente móvel é responsável pelas funções de interação com o usuário, incluindo visualização de dados. O cliente WCL recebe uma requisição do cliente móvel e realiza mediação distribuindo para os servidores WMS. Apenas o cliente WCL implementa interações de acordo com o padrão WMS, uma vez que precisa se comportar como qualquer outro cliente WMS, do ponto de vista do servidor WMS. O objetivo do trabalho foi melhorar a utilidade de ferramentas SDI relacionadas a uma abordagem de maior escalabilidade, mantendo seu potencial de interoperabilidade. A implementação proposta do cliente WMS impõe uma carga muito menor sobre o cliente móvel, preservando mais a flexibilidade do WMS para os parâmetros selecionados pelo usuário.

Nos últimos anos, a necessidade de processar e gerenciar grandes volumes de dados motivou a busca por alternativas eficientes para a realização destas tarefas. Tal necessidade tem popularizado a aplicação de tecnologias propostas para a *cloud computing* no domínio geoespacial.

Como exemplo, o modelo de programação *MapReduce* vem sendo aplicado para a realização de várias tarefas do domínio geográfico, como a geração de índices espaciais [ADBKS10] [CSHR09], processamento de consultas [JROM10] e previsão de desastres naturais [HSWW10].

O crescente volume de dados oferecidos por algumas aplicações de dados geográficos tem aumentado a busca por novas formas de armazenamento e gerenciamento deste tipo de informação. Recentemente, estas tarefas estão sendo resolvidas através de sistemas NoSQL. Os dados armazenados nestes tipos de sistemas são acessados apenas através de suas respectivas interfaces nativas, o que limita o acesso por parte dos usuários e a sua interoperabilidade com dados oferecidos através de outras de outras plataformas. Tal limitação reforça a necessidade de um *framework* que permita que dados geográficos possam ser recuperados através de serviços com interfaces abertas e padronizadas, sem a necessidade de conhecer detalhes sobre o armazenamento dos dados.

A grande popularização de dispositivos para coletar localização, como GPS, resultou na larga proliferação de serviços e aplicações baseadas em localização [NDAA11] [DGK10].

Sistemas NoSQL do tipo *key-value* permitem realizar operações com altas taxas de inserção, mas não fornecem nativamente acesso eficiente a vários atributos, sendo este um requisito fundamental para apoiar sistemas baseados em localização. Para solucionar este problema, Nishimura *et al.* [NDAA11] propuseram uma estrutura de índice multi-dimensional em camadas sobre o armazenamento de um sistema NoSQL *Key-value* (HBase). Foram usadas técnicas de linearização para transformar informações multi-dimensionais de localização em um espaço unidimensional para armazenamento no sistema NoSQL.

Miler et al. [Mil11] propuseram uma aplicação no domínio geoespacial que utiliza um sistema NoSQL (CouchDB) para armazenar os dados geográficos. A sua abordagem consiste no uso de uma arquitetura em duas camadas para a recuperação de dados a partir de dispositivos móveis. Entretanto, neste trabalho, os dados são recuperados apenas através de um navegador Web, o que dificulta a sua interoperabilidade com os dados de outras aplicações.

3.1 Considerações Finais

Neste capítulo, foram descritas algumas abordagens propostas na literatura para integração e interoperabilidade de dados geográficos bem como as soluções existente no domínio geoespacial para tratar com um grande volume de dados.

Entretanto, nenhum destes trabalhos aborda a necessidade de oferecer a interoperabilidade dos dados armazenados em sistemas NoSQL com outros conjuntos de dados existentes. Antes do desenvolvimento deste trabalho o acesso aos dados armazenados em sistemas NoSQL era realizado apenas através de suas respectivas interfaces nativas, o que limitava o seu acesso por parte dos usuários e a sua interoperabilidade com dados oferecidos através de outras plataformas. Tal limitação reforçava a necessidade de um *framework* que permitisse que dados geográficos pudessem ser recuperados através de serviços com interfaces abertas e padronizadas, sem a necessidade de conhecer detalhes sobre o armazenamento dos dados.

Portanto, a solução proposta neste trabalho padroniza o acesso a dados espaciais armazenados em sistemas NoSQL, de modo que qualquer aplicação cliente que implemente os serviços WMS e WFS, possa submeter uma consulta usando a mesma sintaxe para bancos de dados espaciais SQL e sistemas NoSQL de forma simples e transparente para o usuário da aplicação. Fornecendo assim, um mecanismo para interoperar dados espaciais em cenários

com fontes de dados heterogêneas, em que o desempenho, escalabilidade e variabilidade do esquema são preocupações fundamentais.

No próximo capítulo, será apresentado o *framework* NoSQL-GeoServices, que permite acessar através de interfaces bem definidas os dados espaciais e não-espaciais armazenados em sistemas NoSQL. Permite-se, dessa forma, a interoperabilidade destes dados com os armazenados em bancos de dados objeto-relacionais (SQL).

Capítulo 4

Framework NoSQL-GeoServices

Neste capítulo, será apresentado o *framework NoSQL-GeoServices*, que fornece uma interface padronizada para acessar dados espaciais armazenados em Sistemas NoSQL, permitindo, dessa forma, a interoperabilidade desses dados com os armazenados em bancos de dados relacionais.

O restante deste capítulo está organizado como segue. Na seção 4.1, descreve-se uma visão geral da arquitetura da solução proposta. Na seção seguinte, são apresentados detalhes de implementação dos serviços. Por fim, são apresentadas as considerações finais do capítulo na seção 4.3.

4.1 Arquitetura

Para solucionar o problema da interoperabilidade entre fontes de dados heterogêneas, especificamente entre bancos de dados espaciais objeto-relacionais e sistemas NoSQL espaciais, há pelo menos duas estratégias. A primeira, é usar uma arquitetura baseada em tradutor-mediador [Wie92], na qual se escreveria um *wrapper* para comunicação com os sistemas NoSQL espaciais e se integraria os esquemas de todos bancos de dados em um esquema comum único ao banco de dados relacional.

A segunda estratégia, consiste em implementar serviços OGC (*Open Geospatial Consortium*) de interoperabilidade, tais como *Web Map Service* (WMS) e *Web Feature Service* (WFS), sobre a camada do sistema NoSQL espacial, de sorte a integrá-lo com o banco de

dados espacial através de um servidor de mapas como, por exemplo, o Geoserver[9]. Esta segunda solução, permite que qualquer cliente que implemente os serviços WMS e WFS como, por exemplo, o OpenLayers, possa submeter uma consulta usando a mesma sintaxe para bancos de dados espaciais SQL e NoSQL. A segunda estratégia foi adotada e disponibilizada por meio de um *framework*, tornando-se a principal contribuição deste trabalho.

4.1.1 Modelo de Arquitetura

Nesta seção, é apresentada a arquitetura utilizada na solução proposta para o problema de interoperabilidade entre bancos de dados espaciais SQL e Sistemas NoSQL.

Antes de prosseguir com a descrição da arquitetura faz-se necessário definir o conceito de *framework*. Segundo Fayad e Schmidt [FS97], "*framework* é um conjunto de classes que colaboram para realizar uma responsabilidade para um domínio de um subsistema da aplicação". O principal objetivo de um *framework* é facilitar o processo de desenvolvimento de aplicações, pois permite que estas sejam desenvolvidas mais rapidamente, além de resultar em uma aplicação de qualidade superior. Esta é a definição de framework adotada neste trabalho.

A arquitetura do nosso *framework* foi projetada em três camadas: aplicação, serviços e persistência. A Figura 4.1 ilustra o mecanismo utilizado para permitir que bases heterogêneas sejam acessadas utilizando a mesma sintaxe para acessar dados de diferentes fontes de dados.

4.1.2 Camada de Aplicação

A camada de aplicação é responsável pela interação entre usuários finais e os serviços de interoperabilidade do OGC. Desta forma, a camada contém os elementos da aplicação que são visualizados pelos usuários.

No *framework NoSQL-GeoServices* a camada de apresentação é formada essencialmente por páginas JSP (*JavaServer Pages*) usadas pelo administrador do servidor de mapas para preencher informações sobre os serviços e camadas disponíveis. Estas informações são utilizadas na resposta a uma requisição *GetCapabilities*. Na Figura 4.2 é exibida a tela de con-

[9]http://geoserver.org/

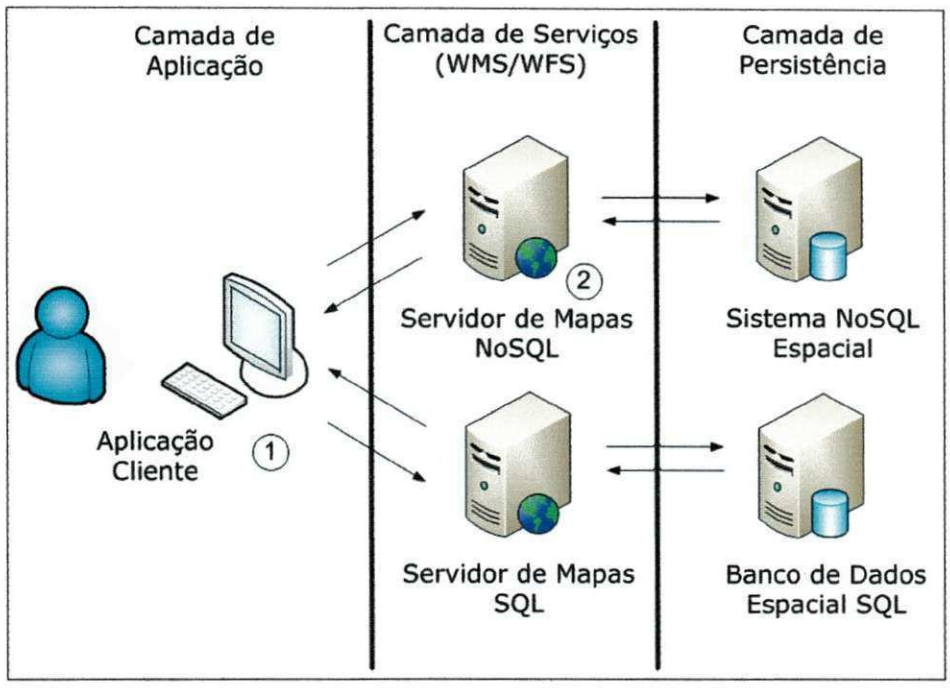


Figura 4.1: Interoperabilidade entre Sistemas de Gerenciamento de Banco de Dados Espaciais e Sistemas NoSQL Espaciais

figuração do serviço WMS no *NoSQL-GeoServices*, na qual é necessário o preenchimento de metadados do serviço, como: URL em que o serviço está disponível, título, resumo, palavras-chave, entre outros.

A camada de apresentação é acessada através do protocolo HTTP, normalmente utilizando um *browser*. As páginas JSP são executadas por um *servlet container* (servidor Web Java), como, por exemplo, Apache Tomcat[10] ou [11].

Em geral, os usuários acessam os serviços de interoperabilidade através de um visualizador de mapas (componente 1 da Figura 4.1), que interage com o servidor de mapas para atender as requisições. O visualizador de mapas é uma aplicação que permite adicionar camadas WMS e WFS a partir da configuração de conexão com um servidor WMS e WFS.

Nos experimentos realizados foram utilizadas como aplicações cliente: Quantum GIS[12],

[10]http://tomcat.apache.org/

[11]http://jetty.codehaus.org/jetty/

[12]http://www.qgis.org/

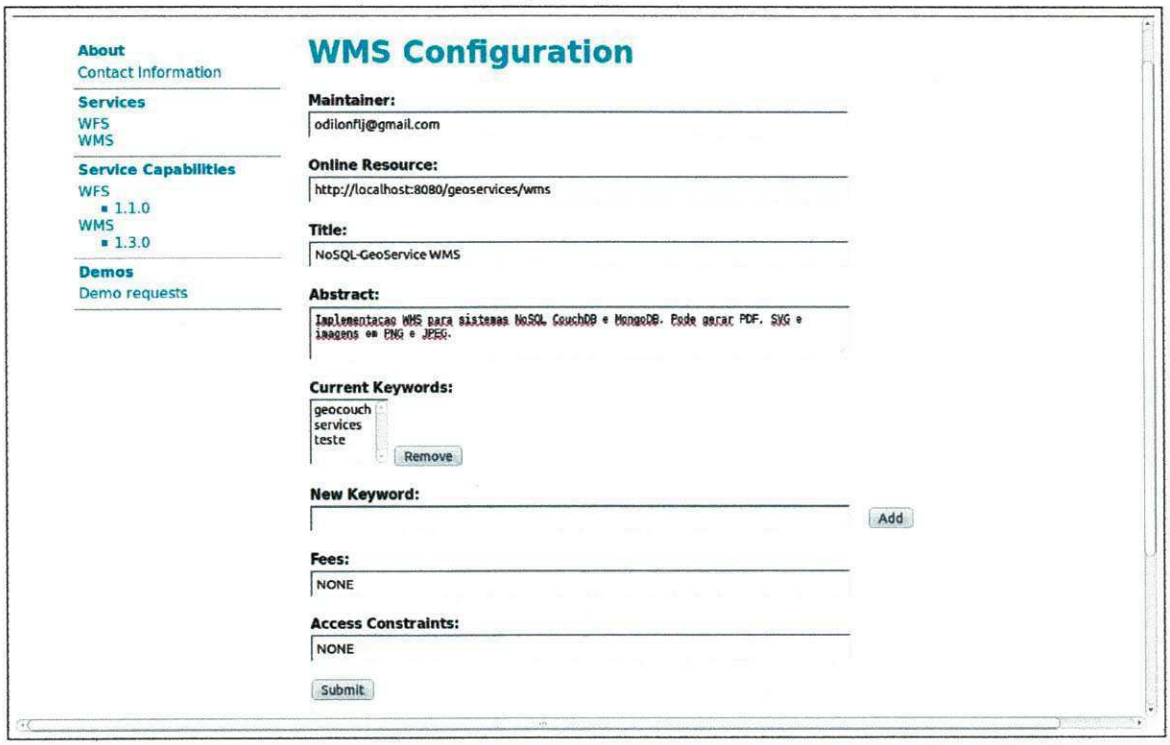


Figura 4.2: Tela de configuração do serviço WMS no *NoSQL-GeoServices*.

gvSIG[13], Kosmo GIS[14], além do módulo de visualização da ferramenta GEO-STAT (*Geographic Spatio-Temporal Analysis Tool*) [OBF12], baseado na API (*Application Programming Interface*) do Google Maps.

O módulo de visualização da ferramenta GEO-STAT possibilita a integração e visualização, de forma prática e intuitiva, de mapas disponibilizados em qualquer servidor acessível publicamente que ofereça os serviços WMS e WFS. A Figura 4.3 mostra a interface do GEO-STAT.

Para configurar uma conexão, em qualquer uma das aplicações clientes, com o servidor WMS e WFS *NoSQL-GeoServices* deve-se fornecer as URL's em que os serviços são invocados. Os exemplos 4.1 e 4.2 mostram as URL's de conexão com os serviços WMS e WFS, respectivamente.

Exemplo 4.1: URL de conexão com o serviço WMS do *NoSQL-GeoServices*.

```
http://IP:PORTA/geoservices/wms
```

[13]http://www.gvsig.org/

[14]http://www.opengis.es/

Figura 4.3: Visualizador de mapas GEO-STAT

Exemplo 4.2: URL de conexão com o serviço WFS do *NoSQL-GeoServices*.

```
http://IP:PORTA/geoservices/wfs
```

4.1.3 Camada de Serviços

A camada de serviços oferece uma interface que define como *features* podem ser acessadas pela camada de aplicação. Nossa contribuição na camada de serviços foi o desenvolvimento de um *framework* que possibilita o acesso a dados espaciais armazenados em sistemas NoSQL através dos padrões WMS e WFS.

O *framework NoSQL-GeoServices* é, em sua essência, um servidor WMS e WFS, no qual é possível submeter consultas a uma base de dados NoSQL com a mesma sintaxe usada para consultar uma base de dados SQL de forma simples e transparente (veja Figura 4.1). A sintaxe usada é a definida pelos padrões WMS e WFS e seu uso é simples porque basta invocar as operações dos serviços (*GetCapabilities*, *GetMap*, *GetFeature*, etc.) para realizar consultas espaciais e não-espaciais. A utilização dos serviços é transparente para o usuário porque funciona independentemente da fonte de dados (banco de dados relacional ou sistema

NoSQL). É importante ressaltar que a proposta de interoperabilidade SQL-NoSQL também serve para dados não-espaciais. Atualmente, para realizar as consultas em sistemas NoSQL é necessário conhecer a linguagem específica fornecida pelo fabricante do sistema. Com o *NoSQL-GeoServices* é possível utilizar uma linguagem padronizada (definida na especificação *Filter Encoding* [Vre05]) para consultar sistemas NoSQL, permitindo assim a interoperabilidade dos dados armazenados nestes sistemas com os dados disponíveis em bancos de dados SQL.

O servidor WMS/WFS *NoSQL-GeoServices* foi desenvolvido seguindo o padrão arquitetural *Model-View-Controller* (MVC) [GHJV95]. O objetivo deste padrão é separar a lógica de negócio (*Model*) da lógica de apresentação (*View*) e do controle de fluxo da aplicação (*Controller*). Para facilitar a implementação deste padrão foi utilizado o *framework* Spring (Spring Web MVC). O *framework* Spring também foi utilizado para realizar a injeção de dependências, ou seja, no *framework NoSQL-GeoServices* as dependências são fornecidas pelo *container* do Spring e não instanciadas diretamente nas classes. Isso ajuda a reduzir o acoplamento entre as entidades.

A Figura 4.4 mostra os principais módulos da arquitetura do *NoSQL-GeoServices*, componente 2 da arquitetura, descrita na Figura 4.1.

O componente *Model* é responsável por modelar as entidades (lógica de negócio) que são manipuladas pelo *framework*. O componente *View* contém os elementos responsáveis pela interface com o usuário. O componente *Controller* processa as informações fornecidas pela *View* e altera as entidades da lógica do negócio (*Model*).

Ao receber uma requisição da camada de aplicação, o Controlador do serviço analisa o pedido e o redireciona para a entidade do Modelo responsável por atender à solicitação. O serviço verifica se os atributos necessários para atender a requisição foram fornecidos. Caso algum atributo obrigatório não tenha sido fornecido, uma exceção de serviço é lançada e uma resposta no formato apropriado é gerada. Independente do resultado, o Controlador recebe uma resposta que é repassada para a camada de aplicação.

Por exemplo, quando uma requisição *GetMap* é feita, contendo todos os parâmetros obrigatórios, o *framework NoSQL-GeoServices* delega a solicitação para o módulo WMS. Este, através do módulo *Repository*, realiza uma consulta pelo *Bounding Box* no sistema NoSQL para cada camada solicitada na requisição, que devolve uma lista de documentos com os

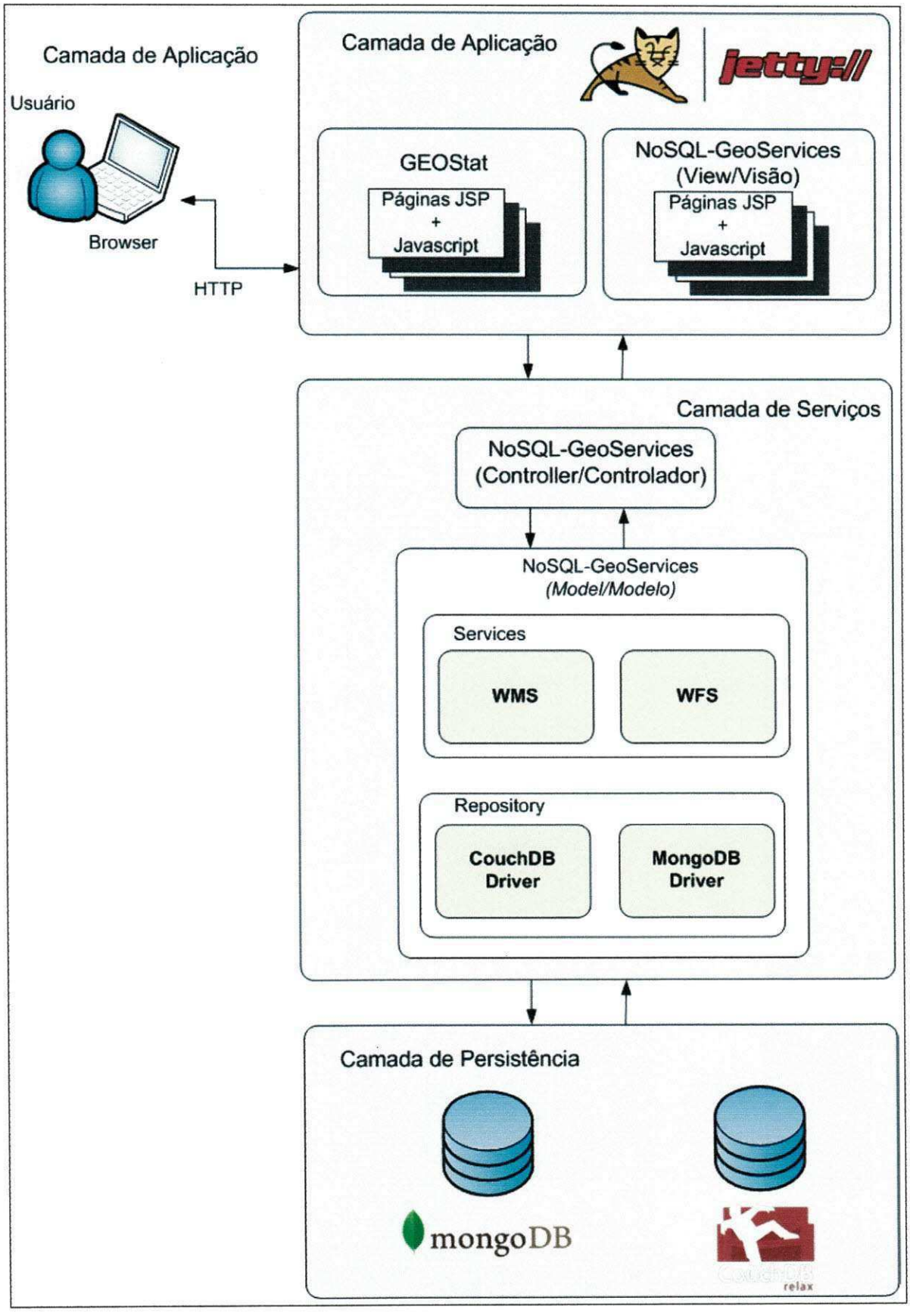


Figura 4.4: Arquitetura MVC do *NoSQL-GeoServices*

dados espaciais. O módulo *Repository* faz um *parsing* dos documentos e os transforma em uma coleção de *features*. Esta coleção é enviada para o módulo WMS que então gera o mapa solicitado. A Figura 4.5 mostra um diagrama de sequência de uma requisição *GetMap*.

Para implementar os serviços WMS e WFS, foi utilizada a biblioteca *GeoTools* [HL08]. No *framework NoSQL-GeoServices* foi implementada a versão 1.3.0 do WMS e 1.1.0 do WFS. A implementação do protocolo WMS contemplou as operações obrigatórias (*GetCapabilities* e *GetMap*) e a operação opcional *GetFeatureInfo*. O protocolo WFS implementado foi o *read-only* formado pelas operações *GetCapabilities*, *DescribeFeatureType* e *GetFeature* (não-transacional).

4.1.4 Extensibilidade do *Framework*

Os principais pontos de extensão (interfaces, abstrações e padrões de projetos) da arquitetura do *framework NoSQL-GeoServices* são:

GeoServicesDAO: Interface para as classes que implementam o padrão de projeto DAO (*Data Access Object*) , em que um objeto é responsável por abstrair e encapsular todos os acessos às fontes de dados. As classes que implementam a interface GeoServicesDAO acessam apenas a camada de persistência;

ServiceIF Interface responsável por criar objetos (*Abstract Factory*) de requisições WMS;

ServiceFactory Classe abstrata responsável por fatorar código comum às Factorys concretas;

WMSFactory Classe concreta responsável por criar objetos (*Abstract Factory*) de requisições WMS;

WFSFactory Classe concreta responsável por criar objetos (*Abstract Factory*) de requisições WFS;

A Figura 4.6 mostra o relacionamento dos pontos de extensão do *framework* descritos acima.

O *framework* NoSQL-GeoServices foi projetado de forma a possibilitar a extensão para outros sistemas de armazenamento de dados espaciais. Os usuários do *framework* podem

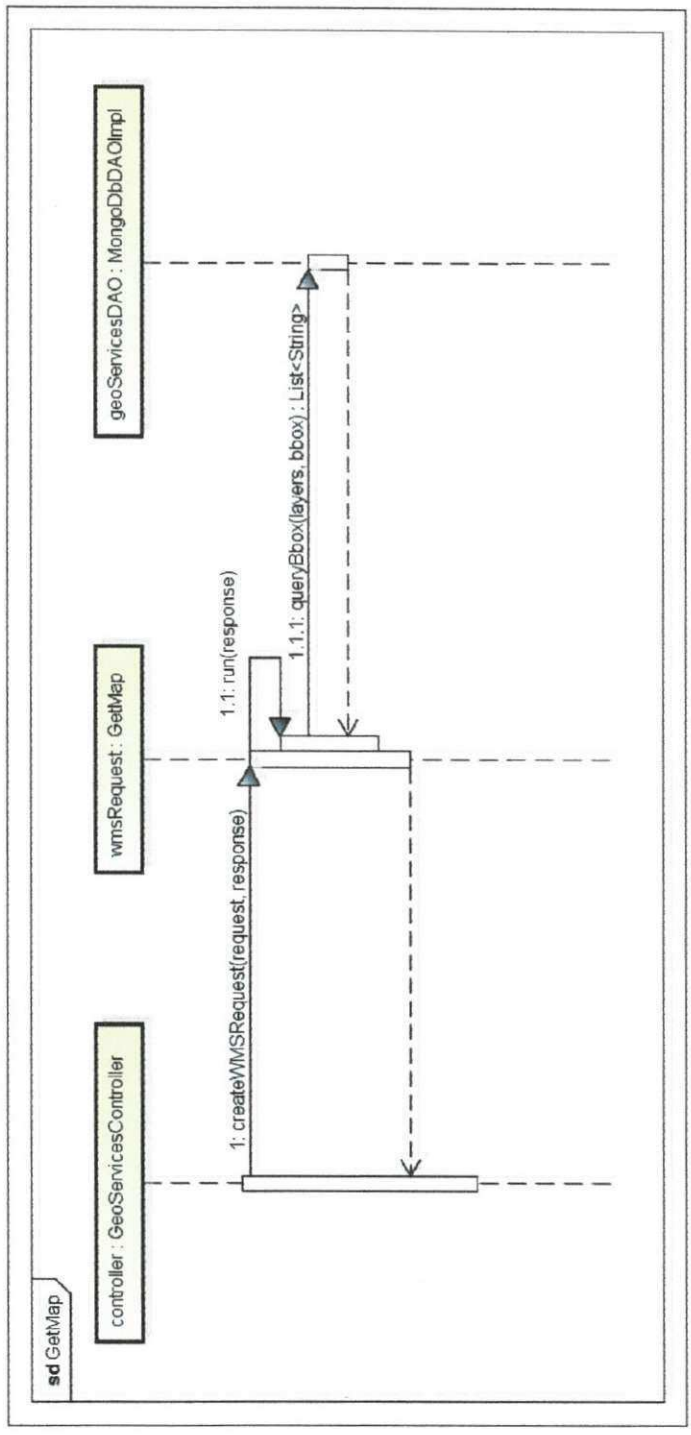


Figura 4.5: Diagrama de sequência de uma requisição GetMap no *NoSQL-GeoServices*

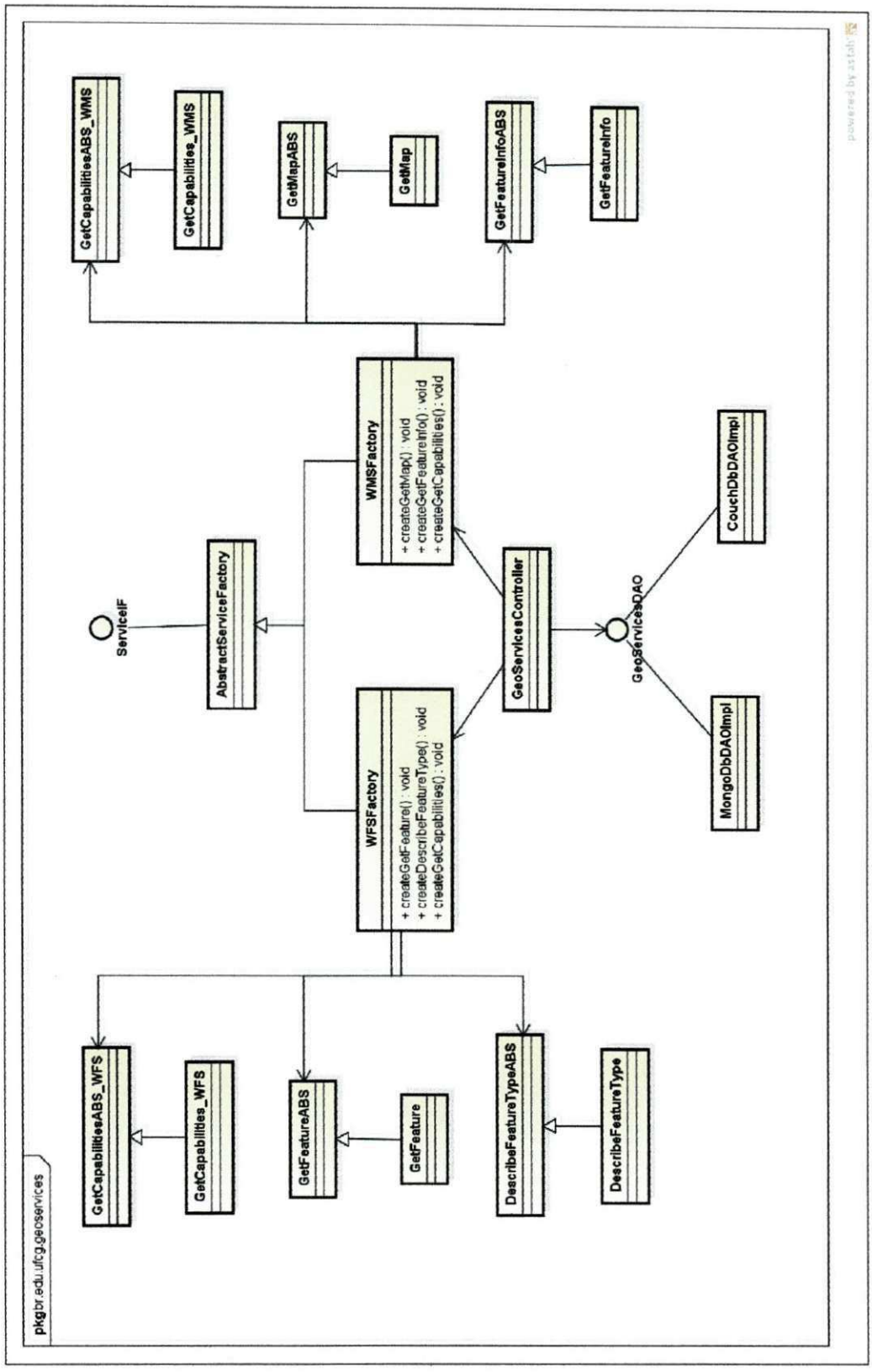


Figura 4.6: Principais pontos de extensão do *framework NoSQL-GeoServices*.

definir métodos para adaptar-ló às suas necessidades. Para utilizar outro sistema de armazenamento na camada de persistência é necessário desenvolver uma classe que implemente a interface GeoServicesDAO e configurar o Spring, que realiza a instanciação da implementação apropriada da interface, de modo que instanciação não defina classes específicas diretamente no código (fraco acoplamento). Assim, os métodos definidos pelo usuário são chamados dentro do próprio *framework*, em vez do código de aplicação do usuário. Logo, o *framework* desempenha o papel do programa principal na coordenação e sequenciamento de atividades da aplicação. Atualmente, o *framework* (módulo *Repository*) possui duas implementações da interface GeoServicesDAO: CouchDBDAOImpl e MongoDBDAOImpl.

A interface GeoServicesDAO (Figura 4.7) fornece os métodos necessários para realização das operações definidas nas especificações WMS e WFS.

A interface GeoServicesDAO define os requisitos necessários para consultar as camadas disponíveis (*getAllSpatialLayers()*), acessar e atualizar metadados do servidor, como configuração dos serviços (*getWMSConfig()* e *getWFSConfig()*) e informações de contato do responsável pelo servidor (*getContactInfo()*), assim como as consultas espaciais disponíveis (*getSpatialCapabilities()*). Caso se deseje estender o *framework* para um sistema de armazenamento que não forneça uma consulta espacial definida na interface é preciso lançar uma exceção em sua implementação informando que a consulta não pode ser realizada.

A classe concreta WMSFactory é responsável pela criação de objetos de requisições WMS (*GetCapabilities*, *GetMap* e *GetFeatureInfo*). Portanto, o usuário pode estender o *framework* definindo uma nova classe para essas requisições. Por exemplo, o usuário pode optar por utilizar uma biblioteca específica, como a JAI (*Java Advanced Imaging*), para renderizar as feições do mapa em uma requisição *GetMap*.

De modo semelhante, a classe concreta WFSFactory é responsável por criar objetos das requisições WFS. Assim, o usuário pode implementar uma versão diferente do protocolo ou alguma melhoria de desempenho.

4.2 Considerações sobre a Implementação

Nesta seção serão descritos alguns detalhes de implementação das operações definidas nos serviços WMS e WFS.

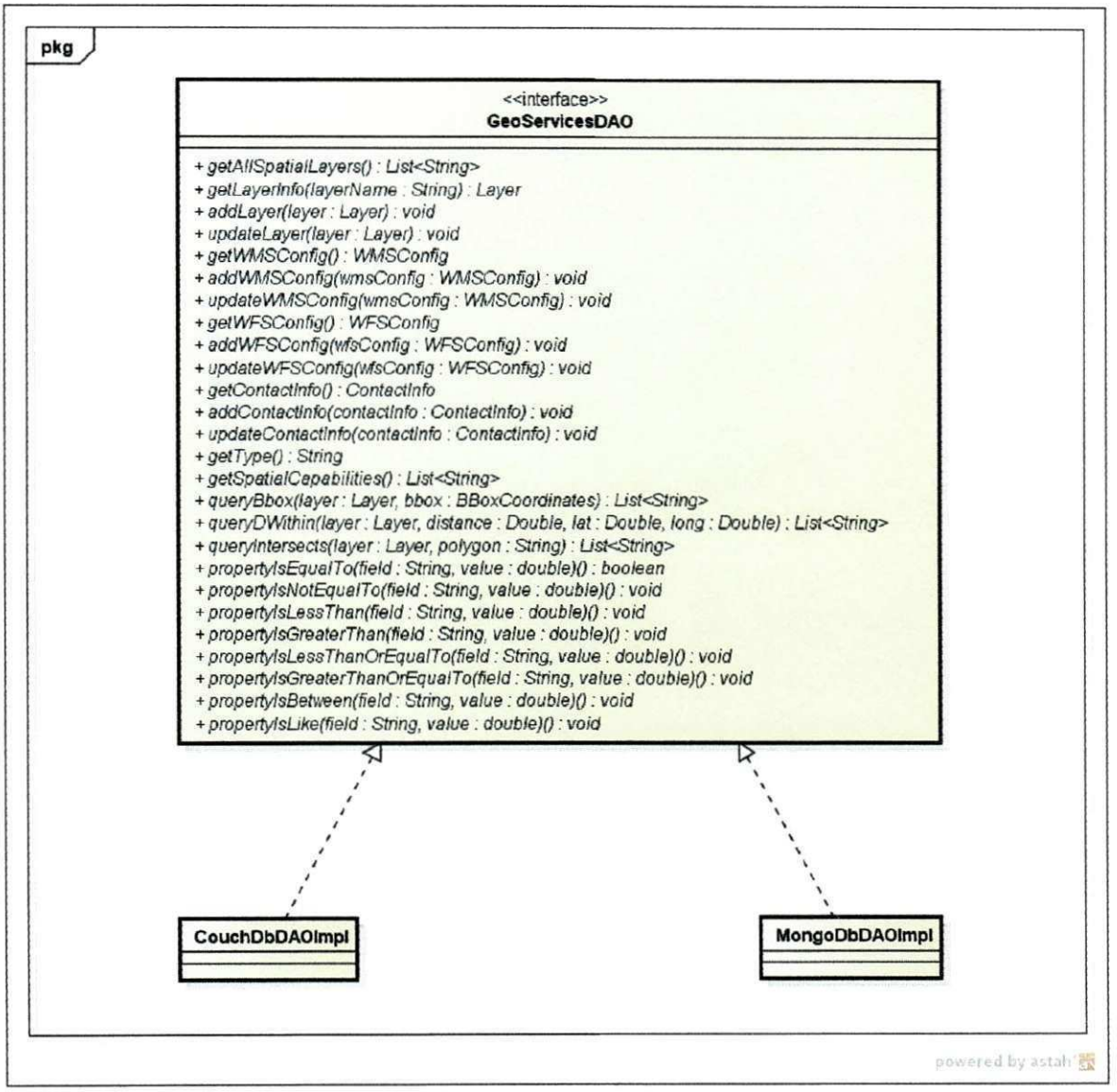


Figura 4.7: Interface do *NoSQL-GeoServices* que permite a extensão para outros sistemas de armazenamento.

4.2.1 GetCapabilities

O retorno de uma requisição à operação *GetCapabilities* obtém metadados do serviço (WMS ou WFS), descrevendo informações do servidor e valores aceitáveis dos parâmetros de requisição. Se uma requisição especificar um formato não permitido pelo servidor então uma resposta é fornecida com o formato padrão. Atualmente o servidor retorna apenas exceções em XML (*eXtensible Markup Language*), que é o formato obrigatório exigido pela especificação da OGC (WMS ou WFS). Caso uma requisição especifique um formato de exceção diferente uma exceção de serviço (XML) será emitida. Devido a diferentes decisões de projeto, sistemas NoSQL habilitados espacialmente ainda fornecem poucas opções de consultas espaciais comparadas às disponibilizadas pelos bancos de dados objeto-relacionais espaciais. Portanto, na implementação do respectivo *driver* para o *NoSQL-GeoServices* é necessário informar quais consultas espaciais o sistema NoSQL permite. Essa informação é obtida através do método $getSpatialCapabilities()$ da interface GeoServicesDAO (Figura 4.7). Nos exemplos 4.3 e 4.4 são mostradas requisições *GetCapabilities* WMS e WFS, respecitvamente.

Exemplo 4.3: Exemplo de requisição *GetCapabilities* (WMS) enviada ao *NoSQL-GeoServices.*

```
http://localhost:8080/geoservices/wms?SERVICE=wms&VERSION=1.3.0&REQUEST=
    GetCapabilities
```

Exemplo 4.4: Exemplo de requisição *GetCapabilities* (WFS) enviada ao *NoSQL-GeoServices.*

```
http://localhost:8080/geoservices/wfs?SERVICE=wfs&VERSION=1.1.0&REQUEST=
    GetCapabilities
```

4.2.2 GetMap

Ao receber uma requisição *GetMap* o servidor retorna um mapa ou emite uma exceção de serviço. Na implementação realizada, utilizou-se o sistema de coordenadas EPSG:4326 (WGS84). Este *Coordinate Reference System* é recomendado pela especificação OGC WMS por maximizar a interoperabilidade entre servidores. Nossa implementação oferece como

formatos disponíveis para requisição de mapas: JPEG, PNG, SVG e PDF. Caso o formato PNG seja requisitado pode-se definir o plano de fundo como transparente. Isso permite que diferentes requisições sejam feitas e sobrepostas produzindo um mapa composto. Caso seja feita requisição a um formato que não permite transparência, o servidor retorna uma imagem sem transparência, ignorando o atributo. Para que seja possível a implementação da operação *GetMap* é necessário que o sistema NoSQL permita consultas pelo *bounding box* (*bbox*) da camada. Nossa implementação utilizou a biblioteca *Geotools* [HL08] para renderizar o mapa a partir de uma lista de *features*.

O usuário pode definir o estilo de cada uma das camadas solicitadas no mapa através de um documento XML. Caso nenhum valor para o estilo seja fornecido é utilizado o estilo padrão.

Devido à necessidade de produzir mapas de um grande número de *features* nossa implementação utiliza os recursos oferecidos pelos sistemas NoSQL para limitar o número de dados retornados em uma consulta (parâmetro *limit*) e para avançar alguns registros/documentos no resultado da consulta (parâmetro *skip*), pois assim é possível realizar várias consultas *bbox* e gerar uma imagem para cada consulta. Em seguida, as várias imagens geradas são compostas (devido à transparência) para gerar a imagem, no formato solicitado, retornada pelo serviço. O número de consultas realizadas depende dos valores especificados para os parâmetros (definidos em arquivos de configuração) e do número de *features* retornadas pela consulta *bbox* sem definir os parâmetros. Caso a quantidade de dados retornados seja menor ou igual ao valor definido para o parâmetro *limit* então apenas uma consulta será necessária. Essa abordagem impede que todos os dados sejam carregados em memória em uma única consulta *bbox*, caso um número maior de *features* seja retornado pela consulta *bbox*.

Embora essa abordagem melhore o desempenho para um determinado número de *features*, pode gerar uma sobrecarga de requisições em função do tamanho do resultado. Nesse casos, recomenda-se o uso de *cache* de *map tiles* (i.e. *GeoWebCache*), que armazenam partes de imagens previamente geradas em disco e quando uma requisição é recebida o servidor apenas retorna a imagem solicitada, tornando o procedimento mais eficiente. O exemplo 4.5 mostra uma requisição *GetMap* submetida ao *NoSQL-GeoServices*.

No exemplo 4.5 tem-se a URL de uma requisição *GetMap* solicitando um mapa dos

municípios e rodovias do Brasil.

Exemplo 4.5: Exemplo de requisição *GetMap* (WMS) enviada ao *NoSQL-GeoServices*.

```
http://localhost:8080/geoservices/wms?SERVICE=WMS&REQUEST=GetMap
&BBOX=-73.990944,-33.750862,-32.378887,5.272225&STYLES=polygon,streams
&FORMAT=image/png&LAYERS=brasil,rodovias_br&WIDTH=1000&HEIGHT=800
```

4.2.3 GetFeatureInfo

Como visto na seção 2.1.2, *GetFeatureInfo* é uma operação opcional e aplicada apenas às camadas com atributo *queryable='1'*. A requisição *GetFeatureInfo* é formada essencialmente pelos mesmos parâmetros da requisição *GetMap* acrescidos os parâmetros das coordenadas na imagem utilizados para identificar a *feature* sobre a qual se deseja informações.

Para implementar essa requisição foram consultadas as *features* que fazem interseção com uma determinada região circular (*buffer*), sendo o raio proporcional ao tamanho da janela (1% da diagonal).

A especificação OGC WMS não define nenhum formato padrão para apresentação do resultado desta requisição. Sendo este um dos pontos da especificação criticados pela comunidade de usuários desenvolvedores, pois uma vez que não existe um formato padrão definido cada servidor de mapa é implementado para atender a requisitos particulares, dificultando a interoperabilidade. O *NoSQL-GeoServices* utiliza HTML como formato de resposta de uma requisição *GetFeatureInfo*.

No exemplo 4.6 tem-se a URL de uma requisição *GetFeatureInfo* solicitando informações sobre um município do Brasil.

Exemplo 4.6: Exemplo de requisição *GetFeatureInfo* (WMS) enviada ao *NoSQL-GeoServices*.

```
http://localhost:8080/geoservices/wms?SERVICE=WMS&REQUEST=GetFeatureInfo
&BBOX=-38.76514,-8.30297,-34.79339,-6.02699&LAYERS=brasil&WIDTH=600&
    HEIGHT=400&x=300&y=200
```

4.2.4 DescribeFeatureType

O objetivo da operação *DescribeFeatureType* é gerar um esquema de tipos de feições de um servidor. O esquema define como o servidor espera feições que serão geradas como resposta a requisições *GetFeature*. Para implementar esta operação assumiu-se que cada coleção (banco de dados) armazena um único tipo de documento com informação espacial. Uma vez que os sistemas NoSQL utilizados são *schema-free* (permitem armazenar na mesma coleção documentos com diferentes estruturas), foi necessário criar o esquema a partir dos dados (documento). Logo, o esquema retornado de uma chamada à operação *DescribeFeatureType* é criado a partir de um documento que possua informação espacial em uma dada coleção. O método $getType()$ definido na interface (Figura 4.7) é responsável por recuperar um documento com informação espacial no sistema NoSQL e construir o esquema a partir de tal documento.

O exemplo 4.7 mostra a URL de uma requisição *DescribeFeatureType* solicitando informações sobre a camada dos municípios do Brasil.

Exemplo 4.7: Exemplo de requisição *DescribeFeatureType* (WFS) enviada ao *NoSQL-GeoServices*.

```
http://localhost:8080/geoservices/wfs?REQUEST=DescribeFeatureType
&VERSION=1.1.0&TYPENAME=brasil
```

4.2.5 GetFeature

A operação *GetFeature* permite recuperar *features* que satisfaçam filtros espaciais ou não-espaciais. A requisição é feita tanto através de uma requisição HTTP POST (com um XML como corpo da requisição) como por uma requisição HTTP GET. O retorno da operação é um arquivo no formato GML. O formato padrão de retorno para a versão implementada é o GML3, mas o GML2 também pode ser requisitado. O retorno de uma requisição *GetFeature* contém zero ou mais *features* que satisfazem às expressões da consulta especificada na requisição. Ao receber uma requisição *GetFeature*, o módulo WFS realiza o *parsing* dos atributos *Query* e *Filter* presentes no XML. Em seguida, estes dados são repassados para o módulo *Repository* que utiliza o *driver* apropriado para transformá-los na respectiva consulta do sistema NoSQL alvo.

No CouchDB, as consultas não-espaciais apresentam uma limitação, pois, para realização de tais consultas é necessário definir uma *view* (uma função *map* e, alternativamente, uma função *reduce* especificada em JavaScript). Em outras palavras, no CouchDB, não é possível realizar consultas não-espaciais de forma dinâmica devido à necessidade de criação prévia de uma *view*. Entretanto, o GeoCouch permite que as consultas espaciais sejam realizadas de forma dinâmica. Por conta disso, no *NoSQL-GeoServices* apenas as consultas espaciais são realizadas de forma nativa no CouchDB. Enquanto as consultas não-espaciais foram implementadas utilizando o filtro da biblioteca *GeoTools*. Essa filtragem é realizada em memória, o que limita o tamanho do conjunto de dados em que essa filtragem pode ser realizada. A utilização do parâmetro *maxFeatures* permite que o filtro seja aplicado apenas em um subconjunto de feições de uma camada. Embora aplicar o filtro em memória seja uma solução não-escalável, ao menos permite que consultas não-espaciais sejam realizadas de forma dinâmica, o que facilita a interoperabilidade e estende as funcionalidades do CouchDB.

Já no MongoDB, é possível realizar as consultas espaciais e não-espaciais de forma dinâmica, portanto, todas as consultas encaminhadas pelo *NoSQL-GeoServices* ao MongoDB são realizadas de forma nativa.

No exemplo 4.8 é mostrada a URL de uma requisição *GetFeature* solicitando os focos de incêndio que estão dentro de uma região retangular (*bounding box*) fornecida pelo usuário.

Exemplo 4.8: Exemplo de requisição *GetFeature* (WFS) enviada ao *NoSQL-GeoServices*.

```
http://localhost:8080/geoservices/wfs?REQUEST=GetFeature&VERSION=1.1.0
&MAXFEATURES=100&TYPENAME=focos&BBOX
    =-73.990944,-33.750862,-32.378887,5.272225,EPSG:4326
```

4.3 Considerações Finais

Neste capítulo, foi apresentado o *framework NoSQL-GeoServices*, que fornece uma solução de interoperabilidade de dados armazenados em bancos de dados espaciais baseados em SQL e em Sistemas NoSQL de forma simples e transparente para o usuário. Este *framework* consiste na principal contribuição desta dissertação.

Foram apresentadas a arquitetura do sistema, algumas considerações importantes sobre a implementação e os pontos de extensão (interface) do *framework NoSQL-GeoServices*.

No capítulo a seguir, serão mostrados dois estudos de caso em que o *framework NoSQL-GeoServices* é utilizado para prover interoperabilidade entre bancos de dados espaciais SQL e NoSQL.

Capítulo 5

Avaliação Funcional

Neste capítulo, serão descritos os cenários de testes usados para verificar e validar a solução proposta neste trabalho.

O restante deste capítulo está organizado como segue. Na seção 5.1, são apresentados os detalhes da configuração do ambiente em que os testes foram executados. Na seção 5.2, são descritos os testes funcionais realizados para verificar e validar o protótipo implementado, assim como os testes para avaliar o protótipo com relação à interoperabilidade entre Sistemas de Bancos de Dados SQL e Sistemas NoSQL. Por fim, são apresentadas as considerações finais do capítulo na seção 5.3.

5.1 Configuração do Ambiente

Foram configurados dois servidores, cada um com serviços WMS e WFS. O primeiro servidor foi configurado para acessar um banco de dados SQL, enquanto o segundo usou um sistema NoSQL. Para o servidor que armazena o banco de dados SQL, foi utilizado o SGBD PostgreSQL 8.4 com a extensão espacial PostGIS, versão 1.5. Neste servidor, os serviços do OGC foram disponibilizados através do servidor WMS e WFS GeoServer 2.1.0. Para o servidor com o sistema NoSQL, usou-se o MongoDB versão 1.9 e o pacote CouchBase versão 1.1, que fornece o CouchDB com a extensão espacial GeoCouch. Neste, os serviços OGC foram disponibilizados através do servidor de mapas *NoSQL-GeoServices*.

Com base nessa configuração, foram definidos dois cenários para os testes funcionais. O objetivo do primeiro experimento foi observar a funcionalidade das solicitações feitas

aos serviços OGC WMS e WFS oferecidos pelo servidor *NoSQL-GeoServices*. O segundo experimento avaliou a possibilidade de interoperabilidade entre bancos de dados espaciais baseados em SQL e NoSQL.

No primeiro experimento, o CouchDB foi configurado para armazenar os municípios do Brasil, as rodovias que cruzam o país e os focos de incêndio detectados de janeiro a junho de 2012. O MongoDB foi configurado para armazenar os mesmos dados de focos de incêndio armazenados no CouchDB. O PostgreSQL foi configurado para armazenar os mesmos dados do CouchDB e do MongoDB.

No segundo experimento, foram armazenados no MongoDB os focos de incêndio detectados em 2012, e no PostgreSQL os dados dos municípios do Brasil e das áreas remanescentes de Mata Atlântica e da Caatinga.

Todos os registros foram armazenados usando a projeção WGS84 (EPSG:4326). Os registros utilizados são dados do mundo real, e foram obtidos a partir do Instituto Brasileiro de Geografia e Estatística (IBGE), Ministério do Meio Ambiente (MMA) e do Instituto Nacional de Pesquisas Espaciais (INPE).

5.2 Verificação dos Serviços Implementados

Foram realizados testes funcionais com o *NoSQL-GeoServices* para verificar a corretude dos serviços implementados. Os testes objetivaram explorar as requisições *GetCapabilities* e *GetMap* do serviço WMS, assim como, *DescribeFeatureType* e *GetFeature* do serviço WFS. Com o intuito de verificar o protótipo implementado, os serviços foram testados em vários clientes WMS/WFS, os quais sejam: Quantum GIS, gvSIG, Kosmo GIS além do visualizador de mapas do GEO-STAT. As respostas para cada uma das requisições foram comparadas com as fornecidas pelo GeoServer a fim de verificar e validar nossa implementação.

Em todas as aplicações clientes foram configurados os acessos para ambos os servidores, o servidor baseado no GeoServer (SQL) e o baseado no *NoSQL-GeoServices* (NoSQL). Para tal, foi necessário apenas definir um identificador (*alias*) e fornecer os parâmetros para conexão com cada servidor. A Figura 5.1 mostra a configuração de uma conexão ao *NoSQL-GeoServices* no Quantum GIS.

Uma vez que a conexão foi estabelecida, pode-se adicionar camadas geoespaciais dis-

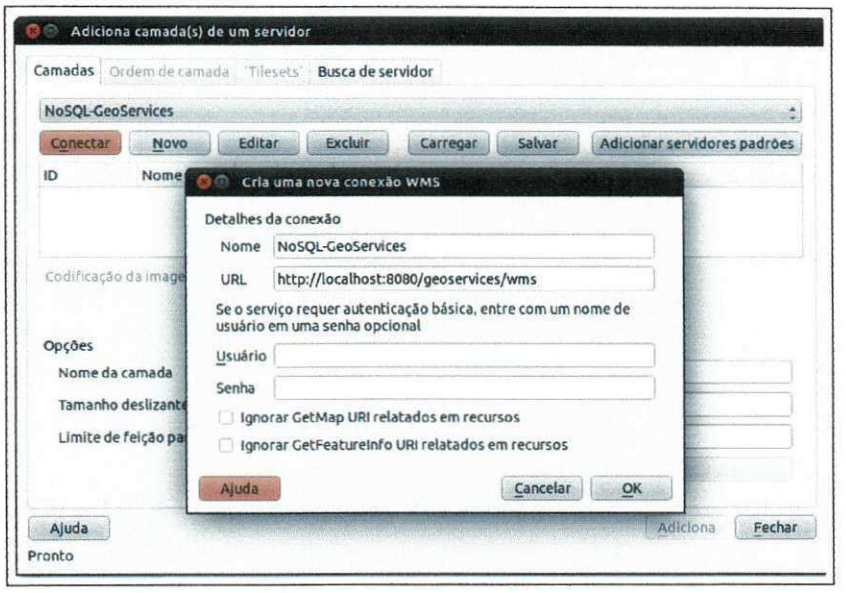


Figura 5.1: Configuração de conexão ao *NoSQL-GeoServices* no Quantum GIS.

poníveis nos servidores. Ao selecionar uma camada para ser exibida no mapa, a aplicação cliente utiliza a requisição *GetCapabilities* (WMS) para retornar a lista de camadas disponíveis, conforme mostrado na Figura 5.2 .

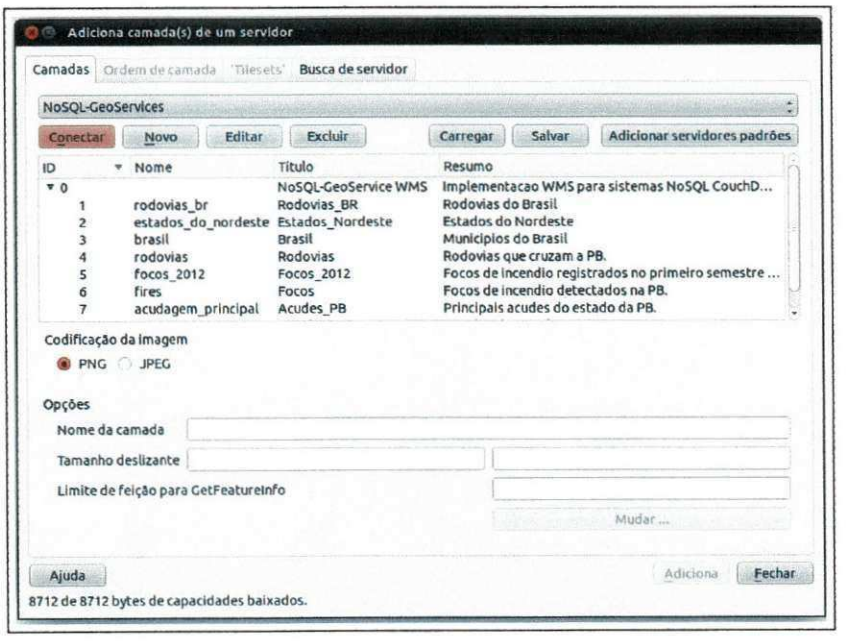


Figura 5.2: Lista de camadas disponíveis no CouchDB através *NoSQL-GeoServices*.

A Figura 5.3 mostra a adição de camadas WMS no Quantum GIS dos **municípios** do

Brasil, sobreposta pelas **rodovias** que cruzam o país e dos **focos de incêndio** detectados no primeiro semestre de 2012. Estes dados são obtidos usando a requisição *GetMap* (WMS) enviada ao servidor de mapas *NoSQL-GeoServices* configurado para acessar o CouchDB.

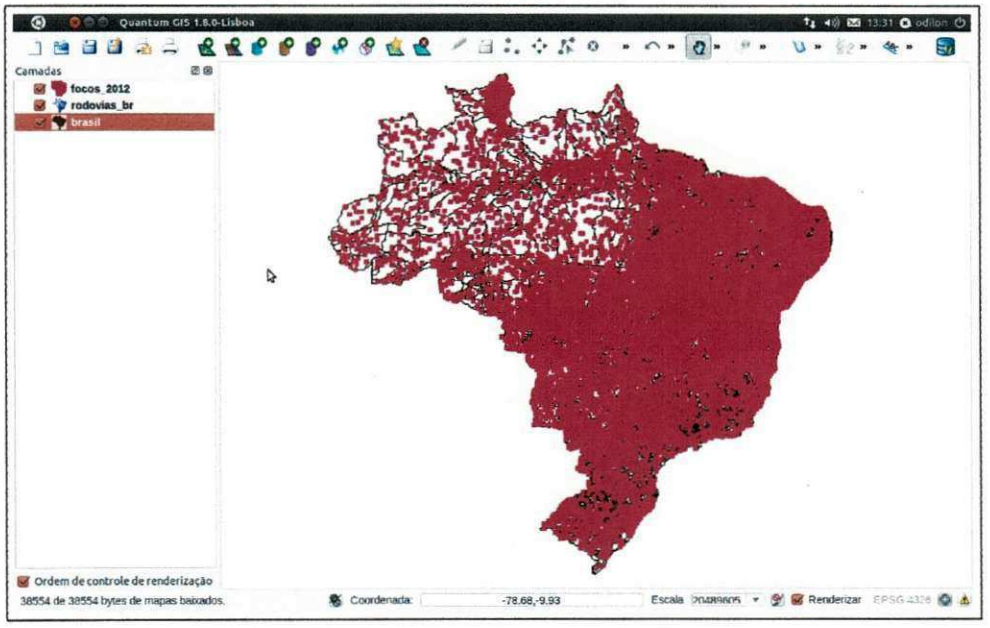


Figura 5.3: Resultado de uma requisição *GetMap*, feita ao *NoSQL-GeoServices*, solicitando os municípios, as rodovias e os focos de incêndio registrados em 2012.

O resultado de uma requisição *GetMap* (WMS) ao servidor de mapas *NoSQL-GeoServices* configurado para acessar o MongoDB usando o GEO-STAT é mostrado na Figura 5.4, em que são ilustrados os focos de incêndio detectados de janeiro a junho de 2012.

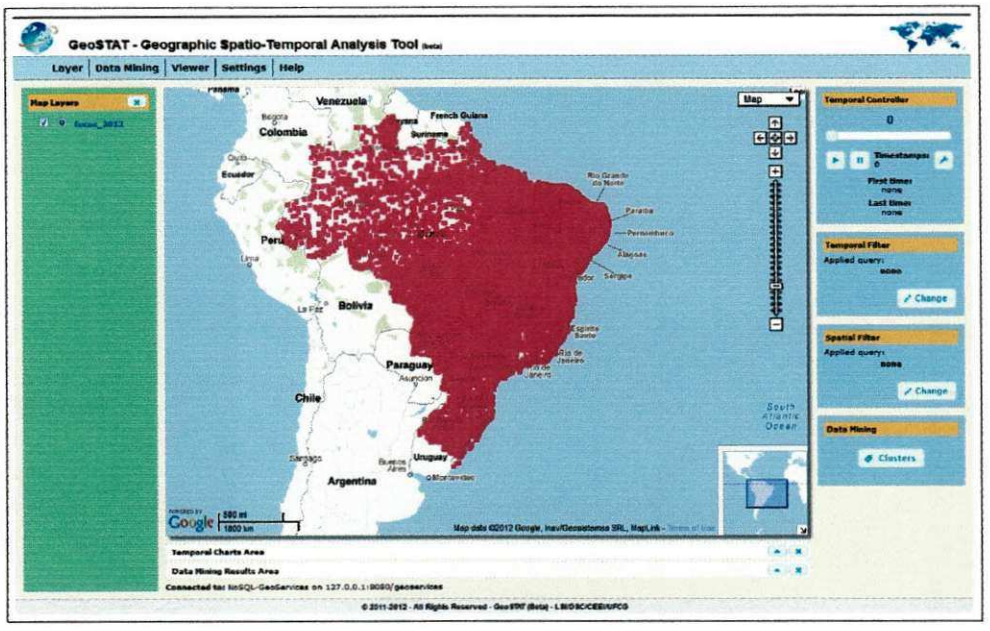
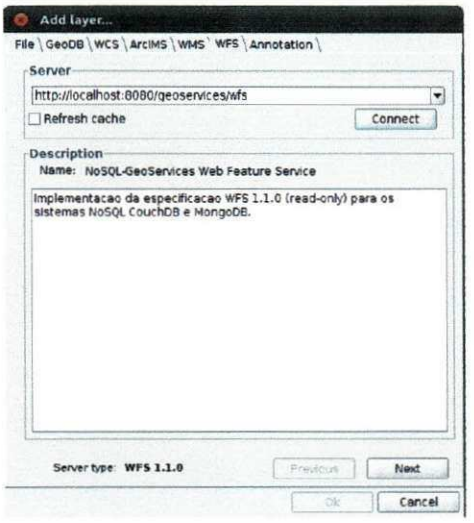
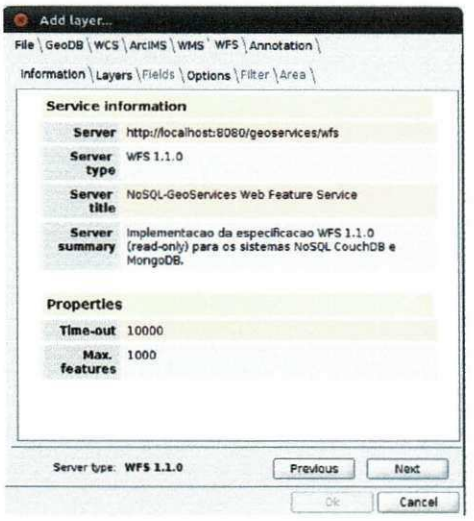


Figura 5.4: Mapa com os focos de incêndio armazenados no MongoDB e fornecidos pelo *NoSQL-GeoServices* usando o GEO-STAT.

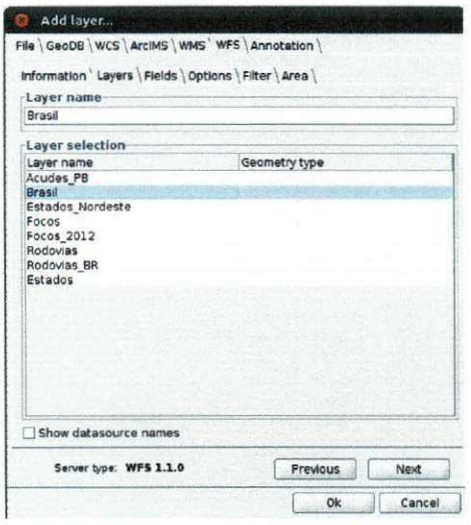
Para testar a implementação do serviço WFS foram utilizadas as aplicações gvSIG e Kosmo GIS. Essas aplicações permitem especificar consultas (filtros) em todas as camadas exibidas no mapa por meio de uma interface gráfica intuitiva. A aplicação gvSIG permite realizar apenas consultas não-espaciais, enquanto a aplicação Kosmo GIS permite realizar tanto consultas espaciais como não-espaciais. Ao realizar a conexão com o servidor WFS é feita uma requisição *GetCapabilities* para obter informações sobre o serviço, camadas e consultas disponíveis. A Figura 5.5 exibe as informações recuperadas através da requisição *GetCapabilities*.



(a) Tela de configuração de conexão com o servidor WFS.



(b) Informações sobre o servidor WFS recuperadas através do serviço *GetCapabilities*.



(c) Lista de camadas disponibilizadas pelo servidor WFS.

Figura 5.5: As Figuras 5.5a, 5.5b e 5.5c mostram as informações do servidor WFS recuperadas por meio de uma requisição *GetCapabilities*.

Para especificar as consultas as aplicações clientes utilizam a requisição *DescribeFeatureType* (WFS) para recuperar os campos de cada camada, e então utilizá-los para expressar os filtros da consulta, que são traduzidos em requisições *GetFeature* (WFS). Na Figura 5.6 são mostrados os campos de uma camada recuperados através de uma requisição *DescribeFeatureType* (WFS).

Na Figura 5.7 é exibido o resultado da seguinte consulta não-espacial: "Recupere todos

5.2.1 Interoperabilidade entre Bancos de Dados SQL e Sistemas NoSQL

Para avaliar a interoperabilidade entre os servidores WMS/WFS *NoSQL-GeoServices* e GeoServer, i.e., a interoperabilidade entre bancos de dados objeto-relacionais espaciais e sistemas NoSQL espaciais (Experimento 2). O teste de interoperabilidade foi realizado utilizando a ferramenta gvSIG para acessar, simultaneamente, as camadas: (*i*) dos municípios do Brasil armazenados no banco de dados SQL PostgreSQL disponibilizada pelo servidor GeoServer; (*ii*) e dos focos de incêndio detectados em 2012 armazenados no MongoDB e fornecida pelo *NoSQL-GeoServices*.

Em ambos os casos, foi estabelecido o limite de 15.000 feições para retorno na consulta *GetFeature* (parâmetro *maxFeatures*). Após a adição das camadas, foi aplicado um filtro não-espacial aos focos de incêndio cujo Bioma é igual a "*Caatinga*" (destacados em amarelo na Figura 5.8).

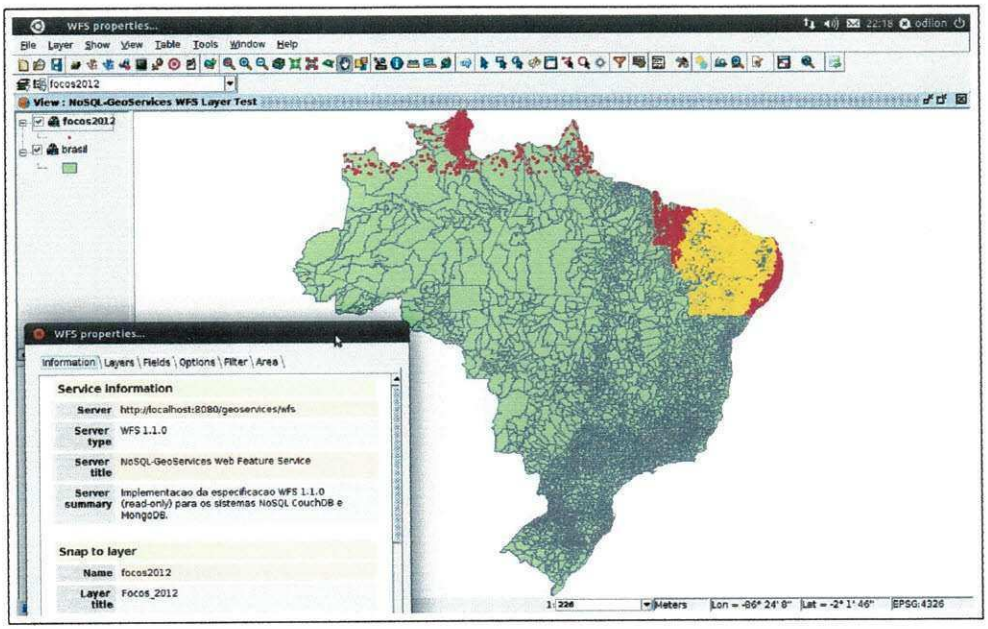


Figura 5.8: Consulta espacial: "*Quais os focos de incêndio que estejam dentro de uma dada região de Mata Atlântica*".

Para enviar uma consulta espacial ao serviço WFS implementado configurou-se o Kosmo GIS para acessar o *NoSQL-GeoServices*, camada dos focos de incêndio armazenados no

MongoDB, e aplicou-se um filtro espacial (limitado a recuperar 1000 feições) nos focos que estivessem dentro (*within*) de uma região remanescente de Mata Atlântica. O detalhe da consulta e seu resultado podem ser vistos nas Figuras 5.9a e 5.9b, respectivamente. A Figura 5.9b apresenta ainda uma camada WFS referente aos municípios do Brasil armazenada no PostgreSQL e fornecida pelo GeoServer.

O Exemplo 5.1 apresenta a URL de uma consulta pelos focos de incêndio que estão dentro de uma região circular definida pelo usuário. A Figura 5.10 mostra o resultado desta consulta.

Exemplo 5.1: Exemplo de requisição *GetFeature* enviada ao *NoSQL-GeoServices* solicitando os focos de incêndio dentro de região circular (*dwithin*) fornecida pelo usuário.

```
http://localhost:8080/geoservices/wfs?REQUEST=GetFeature&VERSION=1.1.0&
    TYPENAME=focos&OUTPUTFORMAT=GML3&FILTER=<Filter><DWithin><PropertyName
    >Geometry</PropertyName><Point><coordinates>-38.616000,-7.492000</
    coordinates></Point><Distance units='km'>10</Distance></DWithin></
    Filter>
```

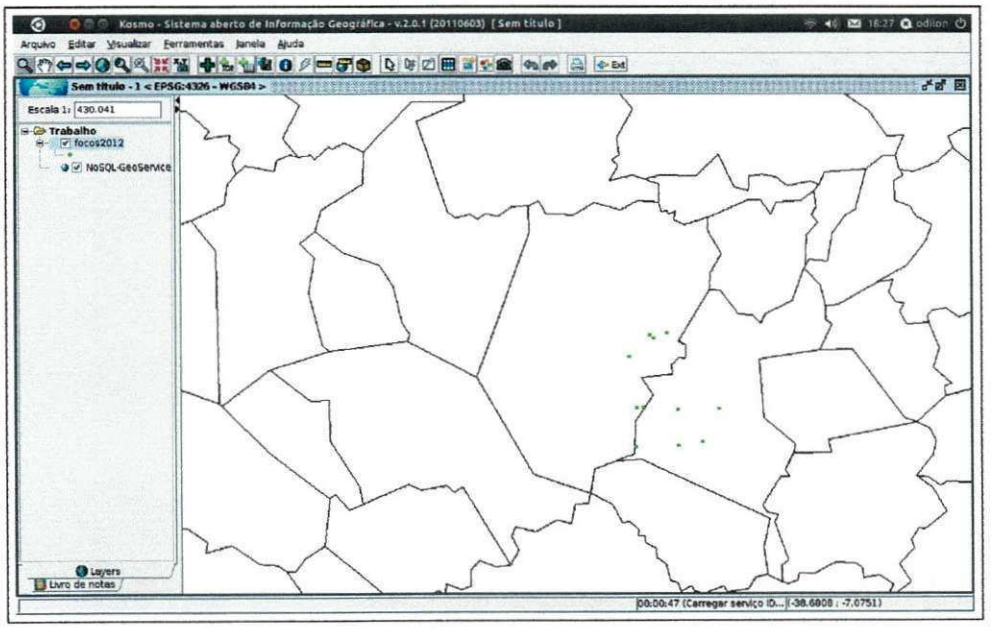
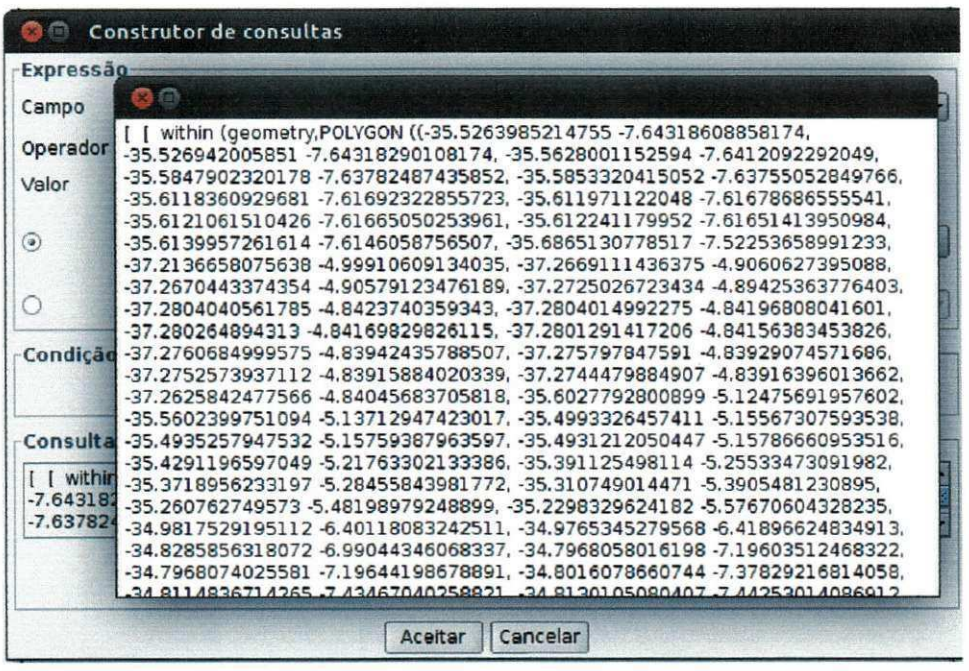
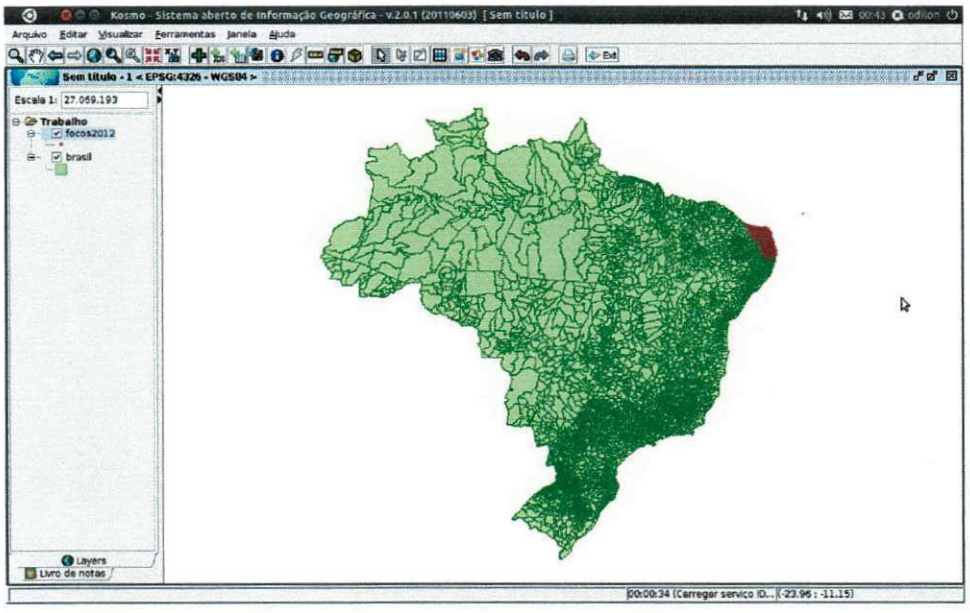


Figura 5.10: Resultado da consulta *GetFeature* especificada no Exemplo 5.1.

O Exemplo 5.2 apresenta a URL de uma consulta pelos focos de incêndio que estão dentro de uma região retangular definida pelo usuário. A Figura 5.11 mostra o resultado



(a) Consulta espacial: "*Quais os focos de incêndio que estejam dentro de uma dada região de Mata Atlântica*".



(b) Resultado da consulta espacial especificada na Figura 5.9a.

Figura 5.9: Consulta espacial no Kosmo SIG acessando o *NoSQL-GeoServices*.

fornecido pelo NoSQL-GeoServices para esta consulta.

Exemplo 5.2: Exemplo de requisição *GetFeature* enviada ao *NoSQL-GeoServices* solicitando os focos de incêndio dentro uma região retangular (*bbox*) fornecida pelo usuário.

```
http://localhost:8080/geoservices/wfs?REQUEST=GetFeature&VERSION=1.1.0&
    MAXFEATURES=100&TYPENAME=focos
&BBOX=-73.990944,-33.750862,-32.378887,5.272225,EPSG:4326
```

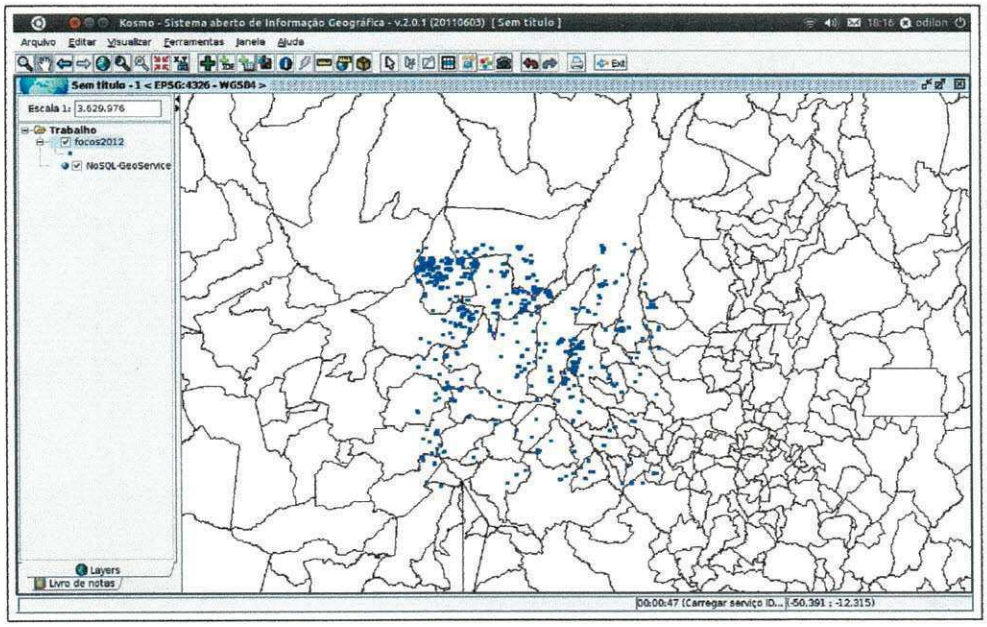


Figura 5.11: Resultado da consulta *GetFeature* especificada no Exemplo 5.2.

5.3 Considerações Finais

Os testes realizados usando as requisições WMS e WFS mostraram que o servidor de mapas *NoSQL-GeoServices* gerou o mapa solicitado de acordo com os requisitos estabelecidos na especificação WMS OGC, além de ter retornado corretamente os dados das consultas conforme a especificação OGC WFS. Portanto, podemos afirmar que o servidor WMS e WFS *NoSQL-GeoServices* funciona em conformidade com os requisitos definidos nas especificações OGC WMS e WFS.

Com esses testes funcionais mostrou-se que a interoperabilidade entre bancos de dados espaciais e sistemas NoSQL espaciais foi realizada com sucesso de forma simples e transparente para o usuário da aplicação, satisfazendo, como resultado, os testes e validando a

solução proposta.

Capítulo 6

Conclusão

Problemas com escalabilidade horizontal, desempenho de consultas em grandes massas de dados e a demanda por modelos mais simples de gerenciamento de dados (*schema free*, BASE, etc.) motivaram o surgimento dos sistemas de armazenamento de dados conhecidos como NoSQL.

A onipresença da dimensão espacial nos dados aliado à popularidade de aplicações espaciais como também dos dispositivos com suporte à coleta de dados georreferenciados motivaram o surgimento de vários serviços e redes sociais baseadas na localização. Por conta dos seus requisitos tipicamente essas aplicações da Web 2.0 utilizam sistemas NoSQL na camada de persistência.

O principal objetivo deste trabalho foi desenvolver uma solução que permitisse a interoperabilidade entre sistemas NoSQL e bancos de dados relacionais, mais especificamente o objetivo do trabalho foi desenvolver um *framework* que forneça uma camada de serviços OGC WMS e WFS para sistemas NoSQL. Dessa forma, é possível realizar consultas espaciais e não-espaciais a sistemas NoSQL de forma simples e transparente, utilizando a mesma sintaxe usada para acessar bancos de dados relacionais, ou seja, sem a necessidade de conhecer detalhes das linguagens oferecidas pelos sistemas NoSQL para acessar os dados. Bem como, pode-se estender o *framework* para dar suporte a novos sistemas NoSQL com suporte a dados espaciais.

As principais contribuições serão apresentadas na seção 6.1 e o conjunto de trabalhos futuros identificados na seção 6.2.

6.1 Contribuições

A principal contribuição deste trabalho foi o desenvolvimento do *framework* NoSQL-GeoServices, que possibilita a interoperabilidade dos dados armazenados em sistemas NoSQL com dados armazenados em outros sistemas de armazenamento disponíveis através dos serviços OGC WMS e WFS. A seguir são listados alguns aspectos relevantes do *framework* desenvolvido:

Interoperabilidade entre SQL e NoSQL: a solução permite interoperar dados armazenados em bancos de dados SQL e sistemas NoSQL. Além de realizar consultas (espaciais e não-espaciais) seguindo a mesma sintaxe independentemente de fonte de dados, seja SQL ou NoSQL.

Arquitetura orientada a serviços de interoperabilidade OGC: devido à solução seguir um padrão "de fato" garante uma maior interoperabilidade dos dados armazenados com os disponibilizados por outros sistemas.

Extensibilidade: o *framework* foi projetado de forma a facilitar o desenvolvimento de novos módulos de acesso a outros sistemas NoSQL.

Disponibilização dos serviços em infraestruturas de dados espaciais: uma das aplicações para os serviços OGC implementados sobre a camada de um sistema NoSQL é disponibilizá-los em infraestruturas de dados espaciais.

6.2 Trabalhos Futuros

Durante o desenvolvimento do trabalho percebemos a necessidade de alguns estudos mais aprofundados como trabalhos futuros, são eles:

- **Criação de benchmark:** Durante os experimentos constatou-se a ausência de um *benchmark* apropriado para comparação entre sistemas NoSQL habilitados espacialmente e bancos de dados espaciais relacionais. Desenvolver um bom *benchmark* não é uma tarefa trivial e devido à grande demanda de sistemas NoSQL (serviços de computação

Referências Bibliográficas

[AD06] Leonardo Lacerda Alves and Clodoveu A. Davis. Interoperability through web services: Evaluating ogc standards in client development for spatial data infrastructures. In Antônio Miguel Vieira Monteiro and Clodoveu A. Davis, editors, *GeoInfo*, pages 173–188. INPE, 2006.

[ADBKS10] Afsin Akdogan, Ugur Demiryurek, Farnoush Banaei-Kashani, and Cyrus Shahabi. Voronoi-based geospatial query processing with mapreduce. In *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science*, CLOUDCOM '10, pages 9–16, Washington, DC, USA, 2010. IEEE Computer Society.

[AS99] Peggy Agouris and Anthony Stefanidis, editors. *Integrated Spatial Databases, Digital Inages and GIS, International Workshop ISD '99, Portland, ME, USA, June 14-16, 1999, Selected Papers*, volume 1737 of *Lecture Notes in Computer Science*. Springer, 1999.

[BEL02] Omar Boucelma, Mehdi Essid, and Zoé Lacroix. A wfs-based mediation system for gis interoperability. In *Proceedings of the 10th ACM international symposium on Advances in geographic information systems*, GIS '02, pages 23–28, New York, NY, USA, 2002. ACM.

[Bis98] Yaser Bishr. Overcoming the semantic and other barriers to gis interoperability. *International Journal of Geographical Information Science*, 12(4):299–314, 1998.

[Bre00] Eric A. Brewer. Towards robust distributed systems (abstract). In *Procee-*

dings of the nineteenth annual ACM symposium on Principles of distributed computing, PODC '00, pages 7–, New York, NY, USA, 2000. ACM.

[BZ09] Serge Boucher and Esteban Zimányi. Leveraging owl for gis interoperability: rewards and pitfalls. In *Proceedings of the 2009 ACM symposium on Applied Computing*, SAC '09, pages 1267–1272, New York, NY, USA, 2009. ACM.

[Cat11] Rick Cattell. Scalable sql and nosql data stores. *SIGMOD Rec.*, 39:12–27, May 2011.

[CCD+05] Marco Casanova, Gilberto Câmara, Clodoveu Davis, Lúbia Vinhas, and Gilberto Ribeiro de Queiroz. *Bancos de Dados Geográficos*. MundoGEO, 2005.

[CD10] Kristina Chodorow and Michael Dirolf. *MongoDB - The Definitive Guide: Powerful and Scalable Data Storage*. O'Reilly, 2010.

[CDG+08] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.*, 26:4:1–4:26, June 2008.

[Câm95] G Câmara. *Modelos, Linguagens e Arquiteturas para Bancos de Dados Geográficos*. PhD thesis, São José dos Campos, SP: Instituto Nacional de Pesquisas Espaciais (INPE), 1995.

[CSHR09] Ariel Cary, Zhengguo Sun, Vagelis Hristidis, and Naphtali Rishe. Experiences on processing spatial data with mapreduce. In *Proceedings of the 21st International Conference on Scientific and Statistical Database Management*, SSDBM 2009, pages 302–319, Berlin, Heidelberg, 2009. Springer-Verlag.

[CTdFM99] Gilberto Câmara, Rogério Thomé, Ubirajara Moura de Freitas, and Antônio Miguel Vieira Monteiro. Interoperability in practice: Problems in semantic conversion from current technology to opengis. In Andrej Vc-

kovski, Kurt E. Brassel, and Hans-Jörg Schek, editors, *INTEROP*, volume 1580 of *Lecture Notes in Computer Science*, pages 129–138. Springer, 1999.

[dA06] Fabio Gomes de Andrade. Webs composer: Uma ferramenta baseada em ontologias para a descoberta e composiÇÃo de serviÇos na web. Master's thesis, Universidade Federal de Campina Grande, 2006.

[DG08] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51:107–113, January 2008.

[DGK10] Yerach Doytsher, Ben Galon, and Yaron Kanza. Querying geo-social data by bridging spatial networks and social networks. In *Proceedings of the 2nd ACM SIGSPATIAL International Workshop on Location Based Social Networks*, LBSN '10, pages 39–46, New York, NY, USA, 2010. ACM.

[DHJ+07] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. Dynamo: amazon's highly available key-value store. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, SOSP '07, pages 205–220, New York, NY, USA, 2007. ACM.

[DJ09] Kimo Y. J. Duarte-Figueiredo F. L. P. Davis Jr., C. A. Ogc web map service implementation challenges for mobile computers. In *17th International Conference of Geoinformatics*, 2009.

[DPS98] Thomas Devogele, Christine Parent, and Stefano Spaccapietra. On spatial database integration. *International Journal of Geographical Information Science*, 12(4):335–352, 1998.

[Dre11] Rainer Dreyer. Real-world spatial features in location-based mobile games. Master's thesis, University of Cape Town - Deparment of Computer Science, 2011.

[dSBdLJdO+11] Cláudio de Souza Baptista, Odilon Francisco de Lima Junior, Maxwell Guimarães de Oliveira, Fabio Gomes de Andrade, Tiago Eduardo da Silva, and Carlos Eduardo Santos Pires. Using ogc services to interoperate spatial data stored in sql and nosql databases. In *Anais do 12º Simpósio Brasileiro de Geoinformática, Campos do Jordão, São Paulo, Brasil*, 2011.

[Fie00] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures.* PhD thesis, 2000. AAI9980887.

[FS97] Mohamed Fayad and Douglas C. Schmidt. Object-oriented application frameworks. *Commun. ACM*, 40(10):32–38, October 1997.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[GL02] Seth Gilbert and Nancy Lynch. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *SIGACT News*, 33:51–59, June 2002.

[GMZB99] Amarnath Gupta, Richard Marciano, Ilya Zaslavsky, and Chaitanya K. Baru. Integrating gis and imagery through xml-based information mediation. In *Selected Papers from the International Workshop on Integrated Spatial Databases, Digital Inages and GIS*, ISD '99, pages 211–234, London, UK, 1999. Springer-Verlag.

[Gom95] L Gomes, J.; Velho. Abstraction paradigms for computer graphics. *The Visual Computer*, v. 11, n. 5:p. 227–239, 1995.

[HL08] G. Brent Hall and Michael G. Leahy. *Open Source Approaches in Spatial Data Handling*, pages 153–170. Springer Publishing Company, Incorporated, 1 edition, 2008.

[HSWW10] D. Hasenkamp, A. Sim, M. Wehner, and K. Wu. Finding tropical cyclones on a cloud computing cluster: Using parallel virtualization for large-scale

climate simulation analysis. In *Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science*, CLOUDCOM '10, pages 201–208, Washington, DC, USA, 2010. IEEE Computer Society.

[Inc05] Open Geospatial Consortium Inc. Web feature service implementation specification. `http://www.opengeospatial.org/standards/wfs/`, 2005.

[Inc06] Open Geospatial Consortium Inc. Web map server implementation specification. `http://www.opengeospatial.org/standards/wms/`, 2006.

[Inc07] Open Geospatial Consortium Inc. Geography markup language encoding standard. `http://www.opengeospatial.org/standards/gml/`, 2007.

[JROM10] Christine Jardak, Janne Riihijärvi, Frank Oldewurtel, and Petri Mähönen. Parallel processing of data from very large-scale wireless sensor networks. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, HPDC '10, pages 787–794, New York, NY, USA, 2010. ACM.

[Lai09] Eric Lai. No to sql? anti-database movement gains steam. *Computerworld*, 2009.

[LM10] Avinash Lakshman and Prashant Malik. Cassandra: a decentralized structured storage system. *SIGOPS Oper. Syst. Rev.*, 44:35–40, April 2010.

[LOU08] P.M.B LOURENÇO. Um estudo sobre recursos de tratamento de dados espaciais em sgbds geográficos. Master's thesis, Departamento de Cartografia do Instituto de Geociências da Universidade Federal de Minas Gerais, 2008.

[Mil11] Damir; Odobasic Drazen Miler, Mario; Medak. Two-tier architecture for

web mapping with nosql database couchdb. *Geospatial Crossroads @ GI Forum '11*, (519442):62–71, 2011.

[NDAA11] Shoji Nishimura, Sudipto Das, Divyakant Agrawal, and Amr El Abbadi. Md-hbase: A scalable multi-dimensional data infrastructure for location aware services. In *Proceedings of the 2011 IEEE 12th International Conference on Mobile Data Management - Volume 01*, MDM '11, pages 7–16, Washington, DC, USA, 2011. IEEE Computer Society.

[OBF12] Maxwell Oliveira, Cláudio Baptista, and Ana Falcão. A Web-based Environment for Analysis and Visualization of Spatio-temporal Data provided by OGC Services. In *GEOProcessing 2012: The Fourth International Conference on Advanced Geographic Information Systems, Applications, and Services*, pages 183–189, Valencia, Spain, January 2012. IARIA. ISBN: 978-1-61208-178-6.

[Ore10] Kai Orend. Analysis and classication of nosql databases and evaluation of their ability to replace an object-relational persistence layer. Master's thesis, Technical University Munich, Faculty of Informatics, 2010.

[Pri08] Dan Pritchett. Base: An acid alternative. *Queue*, 6:48–55, May 2008.

[Rus11] Matthew A. Russell. *Mining the Social Web: Analyzing Data from Facebook, Twitter, LinkedIn, and Other Social Media Sites*. O'Reilly Media, 1 edition, February 2011.

[SC05] Michael Stonebraker and Ugur Cetintemel. "one size fits all": An idea whose time has come and gone. In *Proceedings of the 21st International Conference on Data Engineering*, ICDE '05, pages 2–11, Washington, DC, USA, 2005. IEEE Computer Society.

[Sch11] J Scholz. Coping with dynamic, unstructured data sets - nosql: a buzzword or a savior? In V. Popovich & P. Zeile (Eds.) M. Schrenk, editor, *Proceedings of the Real CORP 2011*, pages pp. 121 – 129., May 2011.

[UVS02] H. Stuckenschmidt U. Visser and C. Schliede. Interoperability in gis - enabling technologies. In *5th AGILE Conference on Geographic Information Science*, pages 291–297, Palma de Mallorca Spain, 2002. Universitat de les Illes Balears.

[Vre05] Panagiotis A. Vretanos. OpenGIS Filter Encoding Implementation Specification (OGC OGC 04-095). OGC Implementation Specification, 2005.

[Wie92] Gio Wiederhold. Mediators in the architecture of future information systems. *Computer*, 25:38–49, March 1992.