UNIVERSIDADE ESTADUAL DE PONTA GROSSA
SETOR DE CIÊNCIAS AGRÁRIAS E DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO APLICADA

GIUVANE CONTI

“ARQUITETURA E IMPLEMENTAÇÃO DE SIG MÓVEL EMBASADO EM CONCEITOS DA INTERNET DAS COISAS”

PONTA GROSSA
2015

GIUVANE CONTI

“ARQUITETURA E IMPLEMENTAÇÃO DE SIG MÓVEL EMBASADO EM CONCEITOS DA INTERNET DAS COISAS”

Dissertação submetida ao Programa de Pós-Graduação em Computação Aplicada da Universidade Estadual de Ponta Grossa, como requisito parcial para obtenção do título de Mestre em Computação Aplicada.

Orientador: Profª Drª Selma Regina Aranha Ribeiro

PONTA GROSSA

2015

Ficha Catalográfica
Elaborada pelo Setor de Tratamento da Informação BICEN/UEPG

Conti, Giuvane
C762 "Arquitetura e implementação de SIG Móvel embasado em conceitos da Internet das Coisas"/ Giuvane Conti. Ponta Grossa, 2015.
114f.

Dissertação (Mestrado em Computação Aplicada - Área de Concentração: Computação para Tecnologias em Agricultura), Universidade Estadual de Ponta Grossa.
Orientadora: Profª Drª Selma Regina Aranha Ribeiro.

1.SIG Móvel. 2.Android. 3.Internet das Coisas. 4.Software-como-serviço (SaaS). 5.Espacialização de dados agrícolas. I.Ribeiro, Selma Regina Aranha. II. Universidade Estadual de Ponta Grossa. Mestrado em Computação Aplicada. III. T.

CDD: 006.3

TERMO DE APROVAÇÃO

Giuvane Conti

"ARQUITETURA E IMPLEMENTAÇÃO DE SIG MÓVEL EMBASADO EM CONCEITOS DA INTERNET DAS COISAS"

Dissertação aprovada como requisito parcial para obtenção do grau de Mestre no Programa de Pós-Graduação em Computação Aplicada da Universidade Estadual de Ponta Grossa, pela seguinte banca examinadora:

Orientadora:

Drª Selma Regina de Aranha Ribeiro
UEPG

Drª Maria Salete Marcon Gomes Vaz
UEPG

Dr. Claudio Leones Bazzi
UTFPR – Medianeira - PR

Ponta Grossa, 29 de outubro de 2015.

AGRADECIMENTOS

Agradeço primeiramente a Deus, pois ele sabe mais do que ninguém sobre esta caminhada e todos os momentos, bons e ruins, que passei.

Aos meus pais, Divo Conti e Ivanir Terezinha Conti, pelo apoio incondicional desde o momento em que decidi iniciar este mestrado. A minha irmã, cunhado e sobrinha Viviane Conti Carniel, Laercio Carniel e Ingrid Conti Carniel, por todo o apoio. A todos vocês, agradeço por cada visita que me fizeram em Ponta Grossa, sei o quanto era difícil para vocês se deslocarem até aqui, mas sei também que valeu cada segundo, saibam que, quando partiam, eu me sentia renovado e com forças para seguir em frente, é realmente muito bom estar junto de vocês! A minha família de Brusque, tios, primos, avó, e em especial a meu avô, Antônio Mannrich (*in memorium)*, esteja onde estiver, sinta-se agradecido. Saibam que, se hoje sou quem eu sou, e se cheguei onde cheguei é graças a vocês.

A Claudia Seffrin, pelo companheirismo compartilhado, pelo apoio e por sempre me incentivar e acreditar em meu potencial.

Agradeço a minha querida orientadora, Selma Regina Aranha Ribeiro, por todo o conhecimento e ensinamentos compartilhados, pelo exemplo de ética, pela força de vontade e também pela amizade.

Aos meus amigos de Medianeira, dos quais sempre pude contar, principalmente nestes 2 anos. É sempre muito bom rever vocês.

Aos meus professores que colaboraram substancialmente para minha formação, aos meus colegas de mestrado pelos momentos de estudo, sei que não foram poucos, e aos momentos de descontração também. A Kelly pela parceria em pesquisas realizadas, e também pela amizade que formamos. Aos Geotecs, não citarei todos pois a lista é grande, mas enfim, obrigado pelos momentos compartilhados no laboratório.

A Universidade Estadual de Ponta Grossa, pela oportunidade e confiança. E a Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) pela concessão da bolsa de pós-graduação.

A todos que contribuíram direta e indiretamente para a conclusão deste mestrado.

RESUMO

O crescimento do mercado de smartphones, combinado com as tecnologias empregadas nestes aparelhos, motiva seu uso em aplicações a campo devido ao seu potencial de georreferenciamento e acesso a internet. Em computação na nuvem, um conceito proeminente é a *Internet of Things* (IoT) ou Internet das Coisas devido ao crescimento exponencial de dados que são gerados no dia-a-dia. A ideia da centralização destes dados, provenientes de várias áreas, como: agricultura, mobilidade urbana, urbanização, governamentais, saúde, empresariais, etc., são desafios atuais que a Internet das Coisas procura solucionar. Destaca-se ainda, os bancos de dados NoSQL, os quais possuem potencial para solução de alguns destes desafios. Outra tecnologia promissora para solucionar alguns destes desafios é o padrão RESTful, um *web service* que apresenta rapidez e trabalha com quantidades reduzidas de bytes em sua comunicação. Neste contexto, esta pesquisa foi desenvolvida com o objetivo de criar um Sistema de Informação Geográfica (SIG) Móvel, que realize a coleta de dados geoespaciais (pontos, linhas e polígonos), se comunique com um servidor de aplicação, no modelo *Software as a Service* (SaaS) que irá armazenar os dados em um banco de dados MongoDB e sua arquitetura segue o conceito de Internet das Coisas. Foram realizados testes a campo na Fazenda Escola Capão-da-onça (FESCON) nos dias 11 e 13 de agosto de 2015. Um planejamento de missões, foi pré - estabelecido, para obter os melhores resultados do Sistema de Posicionamento Global (GPS). Nas referidas datas foram coletados polígonos em talhões de culturas, linhas e pontos de interesse, tais como áreas de erosão, falhas no solo e áreas de baixa produtividade. As coletas realizadas foram bem sucedidas, o aplicativo proporcionou facilidade de uso e de sincronismo dos dados com o servidor de aplicação. Os resultados, dos levantamentos à campo, foram exportados para arquivos no formato KML e apresentados em mapas, com auxílio da ferramenta Google Earth. Conclui-se que, o projeto apresenta alto potencial de uso para cadastro de propriedades agrícolas e dados espacializados e, sua arquitetura é pertinente para o desenvolvimento de projetos geoespaciais, agrícolas em SIG Móvel, que necessitem tratar dados complexos e apresentem alta disponibilidade de dados, tais como: Cadastro Ambiental Rural (CAR); Agricultura de precisão; Controle georreferenciado de dados agrícolas, recursos naturais e urbanização; e Aplicações empresariais de logística.

Palavras-chave: SIG Móvel, Android, Internet das Coisas, Software-como-serviço (SaaS), Espacialização de dados agrícolas, NoSQL.

ABSTRACT

The growth of the smartphone market, combined with the technologies used in these devices, motivates the use of this equipment in field applications because of its georeferencing potential and internet access. In cloud computing, a prominent concept is the Internet of Things (IoT) due to the exponential growth of data where are generated in day-to-day. The idea of centralization of this data, acquired from various areas, such as agriculture, urban mobility, urbanization, government, health, business, etc., are current challenges that the IoT find out to solve. It still in contrast, NoSQL databases, which have some differentiating features from relational models and they have the potential to solve some of these challenges, such as horizontal growth, high data availability and easy storage of complex structures. Another promising technology to solve some of these challenges is the RESTful, a web service that provides faster and works with fewer bytes than the communication with Simple Object Access Protocol (SOAP). In this context, this research was developed with the goal of creating a GIS Mobile application, to collect geospatial data (points, lines and polygons) and store them in an application server, in Software as a Service (SaaS) architecture, using RESTful and NoSQL. The structure follows the concept of IoT, with high potential for integration data from different areas, such as in this study. In this research were performed tests in field at the Fazenda Escola Capão-da-onça (FESCON) on 11 and 13 August 2015. A planning missions, has been pre - set to get the best results from Global Positioning System (GPS). On those dates were collected geo data on crop plots, lines and points of interest, such as erosion areas, failures in the soil and low productivity areas. The samples taken were successful and had no problems or failures, that is, the application provided facility of use and synchronization of the data with the application server. The results of the field surveys, have been exported to files in KML format and presented on thematic maps with the help of Google Earth tool. In conclusion, the project has a high potential for registration of agricultural properties and spatial geo data and, your architecture is relevant for the development of geospatial, agricultural projects in GIS Mobile area, where need to handle complex data and provide high data availability, such as Rural environmental registry (CAR, in brazil); Precision agriculture; Georeferenced control of agricultural, natural resources and urbanization data; and Business logistics applications.

Keywords: SIG Mobile, Android, Internet of Things, Software as a Service (SaaS), Geospatial agricultural data, NoSQL.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE SIGLAS

3G	Terceira Geração
4G	Quarta Geração
ACID	*Atomicity, Consistency, Isolation, Durability*
ADT	*Android Developer Tools*
A-GPS	*Assisted GPS*
API	*Application Programming Interface*
BSON	JSON Binário
CDMA	Acesso Múltiplo por Divisão de Código
DAO	*Data Access Object*
DEX	*Dalvik Executable*
DoD	Departamento de Defesa
DOP	*Dilution of Precision*
EDGE	Enhanced Date Rates For GSM
FESCON	Fazenda Escola Capão-da-Onça
FOSS4G	*Free and Open Source Software for GIS/Geospatial*
GB	*Giga Byte*
GNSS	*Global Navigation Satellite System*
GPRS	*General Packet Radio Services*
GPS	*Global Positioning System*
GSM	*Global System for Mobile Communications*
GUI	*Graphic User Interface*
HTTP	Hypertext Transport Protocol
IBSG	*Internet Business Slutions Group*
IEEE	Instituto de Engenheiros Eletricistas e Eletrônicos
IoT	*Internet of Things* ou Internet das Coisas
JAK	*Java API for KML*
JSON	*JavaScript Object Notation*
JVM	Máquina Virtual Java
KML	*Keyhole Markup Language*
MIT	*Massachusetts Institute of Technology*
MP3	*MPEG-1/2 Audio Layer 3*
NoSQL	*Not Only SQL*
OHA	*Open Handset Alliance*
PC	Computador Pessoal
REST	*REpresentational State Transfer*
RFID	Identificação por Rádio Frequência
SaaS	*Software as a Service*
SGBD	Sistema Gerenciador de Banco de Dados
SQL	Structured Query Language
SIG	Sistemas de Informação Geográfica
SO	Sistema Operacional
SOAP	*Simple Object Access Protocol*
TCP	Protocolo de Controle de Transmissão
UEPG	Universidade Estadual de Ponta Grossa

UERE	*User Equivalent Range Error*
UMTS	Sistema de Telecomunicações Móveis Universal
URI	*Uniform Resource Identifier*
USB	Universal Serial Bus
UTM	*Universal Transversa de Mercator*
WGS	*World Geodetic System*
XML	*eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

O grande volume de dados gerados por aplicações e a arquitetura utilizada pelas mesmas, podem apresentar problemas para a disponibilidade destes dados entre diversos modelos de *softwares* presentes no mercado.

Conceitos apresentados pela *Internet of Things* (IoT) ou Internet das Coisas prometem resolver este problema através de uma cidade inteligente, padronizando a forma de comunicação e acesso a estes grandes volumes de dados, que podem ser provenientes de vários segmentos como: urbanização, medicina, logística, agricultura, entre outros.

Para aplicações móveis seguirem os conceitos de cidades inteligentes da Internet das Coisas, é recomendável que seus dados relevantes sejam centralizados em um servidor de banco de dados, onde a interoperabilidade apresentada pelo web service *REpresentational State Transfer* (REST) ou Transferência de Estado Representacional, também conhecido como padrão RESTful, atende a estes conceitos, trabalhando com requisitos como: comunicação; centralização; e facilidade de acesso a dados entre cliente e servidor.

O uso de banco de dados NoSQL (Not Only SQL) em aplicações centralizadas permitem um crescimento horizontal de servidores de banco de dados apresentando maior disponibilidade e rapidez no acesso aos mesmos, características estas sugeridas pela Internet das Coisas. O uso de banco de dados NoSQL também é recomendado para o armazenamento de dados geográficos de forma prática e apresentando facilidades de integração entre os serviços geoespaciais disponíveis.

O crescimento do mercado de smartphones e computadores conectados a internet permite que o usuário acesse informações em qualquer lugar a qualquer momento, isso permite que um novo paradigma computacional seja aplicado em diversas áreas. Um bom exemplo é o uso de smartphones no mercado agrícola, onde o usuário pode coletar dados geográficos e armazená-los em um banco de dados para monitoramento, praticamente em tempo real.

O uso da computação móvel permite que usuários utilizem sistemas de informação enquanto se deslocam de um local para outro, este paradigma combinado ao uso de um Sistema de Posicionamento Global por Satélite (GNSS) e o acesso a internet apresenta grande potencial, principalmente para aplicações agrícolas que dependem de georreferenciamento, deslocamento em campo e coleta de dados para geração de mapas temáticos.

Portanto nesta pesquisa são utilizados conceitos da ciências de Sistemas de Informação Geográfica (SIG), um banco de dados NoSQL para armazenamento de dados geográficos em um servidor de aplicação e a coleta de dados geográficos em um SIG Móvel via smartphone utilizando seu recurso de GNSS - *Global Positioning System* (GPS), com sua comunicação com o servidor de aplicação utilizando o padrão RESTful, e integrando todos esses serviços com o modelo de ideias das cidades inteligentes estabelecido pela Internet das Coisas.

1.1 Objetivo Geral

O objetivo geral desta pesquisa é desenvolver uma arquitetura de um projeto voltado para aplicações geoespaciais e mobilidade, com ênfase em dados agrícolas e implementá-lo, utilizando o conceito da Internet das Coisas.

1.2 Objetivos Específicos

(1) Desenvolver a arquitetura de um SIG Móvel e de um servidor de aplicação para coleta e armazenamento de dados geoespaciais na nuvem, centralizando dados em um banco de dados NoSQL;

(2) Implementar o SIG Móvel e o servidor de aplicação;

(3) Realizar o planejamento das missões para a coleta a campo dos dados geoespaciais de um imóvel agrícola;

(4) Realizar a coleta a campo dos dados geoespaciais agrícolas;

(5) Exportar os dados geoespaciais agrícolas para mapas, em arquivos *Keyhole Markup Language* (KML) e visualizá-los no SIG desktop Google Earth.

O produto final é uma aplicação móvel (SIG Móvel) e um servidor de aplicação seguindo o modelo de *Software as a Service* (SaaS). Desta forma, espera-se que a pesquisa realizada contribua como paradigma para aplicações agrícolas que envolvem a comunicação entre um servidor de aplicação e uma aplicação móvel, baseada nos conceitos de Internet das Coisas e apresentando interoperabilidade entre servidor de aplicação e o aplicativo móvel via web service REST. O comportamento do aplicativo móvel segue o padrão de um sistema de informação geográfica móvel e os dados geográficos coletados e armazenados em um Banco de Dados NoSQL podem ser exportados em formatos utilizados em Sistemas de Informação Geográfica Desktop, como por exemplo, o *software* ArcGIS.

1.3 Motivação

A computação móvel possibilita a execução de tarefas de computação, enquanto o usuário está se deslocando de um lugar a outro ou visitando lugares diferentes de seu ambiente usual. Na computação móvel os usuários podem acessar a internet e utilizar recursos próximos enquanto se deslocam (COULOURIS *et. al.*, 2007).

A computação móvel pode ser representada como um paradigma computacional que permite que usuários acessem serviços independente de sua localização, ou seja, é um conceito que envolve processamento, mobilidade e comunicação sem fio; a ideia é ter acesso a informação em qualquer lugar e a qualquer momento (FIGUEIREDO; NAKAMURA, 2003). O crescimento do potencial computacional e componentes eletrônicos disponíveis em celulares reforça ainda mais esta ideia.

Segundo Lecheta, (2013) no ano de 2012 foi avaliado que mais de 3 bilhões de pessoas possuíam aparelho celular. A busca por celulares do tipo *smartphone* com câmera, recursos de áudio, *bluetooth*, interface visual, jogos, GNSS, acesso a internet e e-mails, e até TV digital é cada vez maior (LECHETA, 2013).

O uso do Sistema de GNSS pode ser facilmente encontrado em diversos segmentos dentro do setor agrícola e em outras áreas, como: Pulverização Química; Monitoração de Rendimento de Safras; Extensão de Safras e Rastreamento de Gado; Navegação e Monitoramento de Barcos de Pesca; Engenharia Civil; Orientação de Máquinas; Logística e Gerenciamento de Canteiros de Obras; Energia; Meio-Ambiente e Monitoramento Ambiental; Segurança Ambiental; Telecomunicações; Localização de Telefones Móveis; Rede de comunicação e aviação (SEBEM *et. al.*, 2010). Este é um recurso comum em *smartphones* e seus benefícios estão cada vez mais presentes nos aplicativos disponíveis no mercado.

Sistemas de navegação automotiva utilizam uma unidade de GNSS, mas como o GNSS se tornou popular em *smartphones*, muitas pessoas estão utilizando este serviço diretamente em seus celulares. Devido a isto, muitas companhias como a TomTom e Garmin estão investindo em aplicativos de navegação portáveis para *smpartphones* (CHANG, 2012).

Com o GNSS presente nos celulares e com a evolução significativa do *design* e miniaturização de *hardware,* nos últimos anos tornou-se possível desenvolver SIG para uso em sistemas portáteis. Tecnologias leves e de baixo custo como os receptores GNSS - GPS e as redes sem fio estimulam ainda mais este mercado. Uma áreas inovadora é o desenvolvimento de *software* portátil para *smartphones*, os sistemas podem utilizar

ativamente dados e *software,* hospedados em um servidor, como é o caso do *Google Maps*, recurso também disponível em sistemas Android. O SIG portáte ou SIG Móveis são sistemas leves, se comparados aos *desktop*, projetados para uso em trabalhos de campo (LONGLEY *et. al*., 2013).

Com o potencial computacional crescente de aparelhos celulares, o mercado corporativo busca cada vez mais incorporar aplicações móveis em seu dia-a-dia para agilizar negócios e integrar aplicações móveis em seus sistemas de *back-end*. O *Android* é a resposta da *Google* para ocupar este espaço. Os celulares não são apenas usados para atender ligações, agora são verdadeiras máquinas repletas de tecnologia (LECHETA, 2013).

Com equipamentos do tipo *smartphone* (telefone celular com GNSS - GPS integrado) o agricultor pode carregar todo banco de dados de sua lavoura, interagindo em tempo real com procedimentos de gestão e levantamento de dados, mediante a internet. Programas computacionais de tecnologia móvel possibilitam alto nível de precisão, eficiência e rapidez no mapeamento de lavouras e levantamento de pontos amostrais georreferenciados de uma gleba (AMADO; GIOTTO, 2009).

Segundo Gubbi et al, (2013) e Borgia, (2014) a próxima inovação na era da computação será fora do ambiente de trabalho tradicional. No paradigma da Internet das Coisas muitos dos objetos e informações que nos rodeiam vão estar conectados em uma rede. Tecnologias como a Identificação por Rádio Frequência (RFID), redes de sensores e a computação móvel contribuirão para atender este desafio. O resultado será uma grande quantidade de dados que necessitam ser armazenados, processados e apresentados de forma perfeita, eficiente e facilmente interpretável.

A conectividade inteligente com redes existentes é um recurso indispensável da Internet das Coisas e com o crescimento do *Wireless Fidelity* (Wi-Fi) e redes móveis de 3ª e 4ª geração (3G e 4G) para acesso a internet sem fio, a evolução para redes de informação e comunicação já é evidente. No entanto para a Internet das Coisas emergir com sucesso, o paradigma de computação deverá ir além do cenário de computação móvel tradicional, de smartphones e tecnologias portáteis, para interligar objetos diários existentes e incorpora-los de informa inteligente ao nosso ambiente (GUBBI et al, 2013). Um desafio da Internet das Coisas é no que diz respeito a tecnologia de banco de dados utilizada para o armazenamento e velocidade de disponibilidade de todo este conteúdo.

Bancos de dados relacionais tem sido a escolha padrão para o armazenamento de grandes volumes de dados, principalmente no mundo dos aplicativos corporativos, se um arquiteto de software iniciasse um projeto, provavelmente sua principal escolha seria optar

pelo uso de um banco de dados relacional. Após um longo período de domínio, atualmente surge o interesse por bancos de dados NoSQL (SADALAGE & FOWLER, 2013). Dentre alguns aspectos onde bancos de dados NoSQL superam os bancos de dados relacionais, destaca-se a aplicação com sucesso em redes que operam em *cluster*, e também o armazenamento de dados geográficos (QUEIROZ *et al.*, 2013).

1.4 Organização do Trabalho

Esta dissertação é dividida em 5 capítulos, além deste capítulo introdutório. No segundo capítulo é apresentada a revisão de literatura referente ao trabalho desenvolvido, onde são exploradas informações sobre computação móvel, A6ndroid, Internet das Coisas, *web service* REST e banco de dados NoSQL. No terceiro capítulo está a área de estudo e os procedimentos metodológicos utilizados para o desenvolvimento deste projeto, divido em etapas. No quarto capítulo estão apresentados os resultados do projeto, os quais atenderam os objetivos especificados aqui neste capítulo primeiro. No quinto capítulo apresenta-se a conclusão e considerações finais dos resultados e perspectivas de trabalhos futuros.

2 REVISÃO BIBLIOGRÁFICA

2.1 Computação móvel e *cloud computing*

Neste tópico serão descritas características da computação móvel, dispositivos móveis e algumas tecnologias que foram embasadas para a criação da arquitetura e da implementação desta dissertação.

2.1.1 Computação móvel, dispositivos móveis (smartphones/android) e suas vantagens

Segundo Mateus & Loureiro (1998), independentemente do tipo de dispositivo portátil, e a capacidade de se comunicar com a rede e com outros dispositivos móveis, tem-se a computação móvel. Que surge como uma quarta revolução computacional, antecedida pelos grandes centros de processamentos de dados na década de sessenta, surgimento dos terminais nos anos setenta e as redes de computadores na década de oitenta.

A computação móvel permite que usuários tenham acesso a serviços independente de sua localização, permitindo mudanças de localização, ou seja, mobilidade, por meio da comunicação sem fio. O objetivo principal deste modelo é disponibilizar aos usuários um ambiente computacional, com um conjunto de serviços comparáveis aos existentes, em um sistema distribuído estático, permitindo a mobilidade (MATEUS & LOUREIRO, 1998).

No contexto de computação móvel, mobilidade se refere ao uso de dispositivos móveis portáteis que realizem um conjunto de funções de aplicação, capazes de conectar-se, obter dados e fornece-los a outros usuários, aplicações e sistemas (LEE, 2005).

Segundo Marimoto, (2009), um smartphone é uma nova geração de telefones celulares, um telefone inteligente que utiliza mobilidade e conectividade, e se caracteriza pelos seguintes componentes: Acelerômetro, *bluetooth*, Wi-Fi, GPS, câmeras de alta qualidade, métodos inteligentes de escrita, acesso móvel como Sistema Global para Comunicações Móveis (GSM), Acesso Múltiplo por Divisão de Código (CDMA) e Sistema de Telecomunicações Móveis Universal (UMTS).

Os dispositivos móveis (celulares e *handhelds*), são equipamentos presentes no cotidiano das pessoas, com a finalidade de atender ao crescimento da comunicação on-line, disponibilizando portabilidade e praticidade da informação independente do lugar (DALFOVO *et al.*, 2004).

Um smartphone é capaz de: Executar um sistema operacional completo e permitir a instalação de aplicativos nativos; Comunicar-se com o Computador Pessoal (PC) via *Universal Serial Bus* (USB) e bluetooth; Conectar a web via *General Packet Radio Services* (GPRS), ou Serviços Gerais de Pacote por Rádio), *Enhanced Date Rates For GSM* (EDGE) ou 3G/4G; Manter um navegador de internet, oferecer clientes de e-mail e outros aplicativos de comunicação; Tocar MPEG-1/2 Audio Layer 3 (MP3), exibir vídeos e rodar jogos (MARIMOTO, 2009).

Segundo LEE (2005), um dispositivo móvel apresenta 4 vantagens:1) A portabilidade é definida como a capacidade de ser facilmente transportável e atualmente os dispositivos são muitas vezes mais rápidos, menores e mais poderosos, o peso do dispositivo é importante e o fato de um telefone celular pesar apenas algumas gramas é uma vantagem; 2) A usabilidade de um dispositivo móvel é a capacidade de ser utilizado por diferentes pessoas em diferentes ambientes; 3) Os dispositivos móveis servem a múltiplos propósitos e apresentam diversas funcionalidades e; 4) A conectividade de um dispositivo móvel é a capacidade de conectar as pessoas ou sistemas e transmitir e receber informação.

Diversas empresas buscam aplicações móveis para incorporar o seu dia-a-dia para agilizar seus negócios integrando aplicações móveis com seus sistemas desktop, celulares e smartphones podem ocupar um espaço. Uma solução gratuita para ocupar este nicho de mercado, diminuir fronteiras e auxiliar empresas e pessoas a incorporar estes recursos é o *Google Android* (LECHETA, 2013).

O *Android* é uma plataforma de desenvolvimento de aplicativos móveis como *smartphones*, tem seu Sistema Operacional (SO) baseado em Linux, possui código aberto, ótima interface visual, sistema GPS, diversas aplicações já instaladas, um ambiente de desenvolvimento poderoso, inovador e flexível e pode utilizar a linguagem Java para desenvolver as aplicações (LECHETA, 2013).

O sistema operacional do *Android* foi baseado no *kernel* 2.6 do *Linux*, e é responsável por gerenciar a memória, processos, *threads*, segurança, redes e *drivers*. Cada aplicativo dispara um novo processo e diversos processos podem ser executados simultaneamente, toda a segurança é baseada no *Linux*. Para cada aplicação instalada no celular é criado um usuário no sistema operacional, onde apenas este usuário pode acessar sua estrutura de diretórios (LECHETA, 2013).

Como a linguagem *Java* é utilizada para construir aplicações para o *Android*, e o sistema operacional não possui Máquina Virtual Java (JVM), o *Android* possui a máquina virtual Dalvik, que é otimizada para execução em dispositivos móveis. As classes java

(.class) são convertidas para *Dalvik Executable* (DEX) com extensão .dex e depois compactados com o *Android Package File (APK)* de extensão .apk, este sendo a aplicação final (LECHETA, 2013).

Estas características apresentadas por dispositivos móveis, possibilitam o seu uso em qualquer lugar e a qualquer momento, associadas a outras tecnologias apresentam potencial para otimizar e melhorar o setor agrícola.

2.1.2 Computação Móvel e a Agricultura

Baixas taxas de adoção e utilização de tecnologias de informação e comunicação sempre foram um fator de constrangimento do setor agrícola, porém as características e condições da computação podem mudar esta realidade. Estudos realizados a nível internacional demonstraram que o uso da tecnologia da informação e comunicação na agricultura, em sua maior parte, está relacionado a menor idade e o maior nível de instrução do empresário agrícola (NETO *et al.*, 2007).

O uso de novas tecnologias de informação e comunicação nos mais diversos setores de atividades econômicas tem instigado novos modelos de negócios, em especial para o setor agrícola e agroindustrial, destacado como “*m-Business*”. Que pode ser definido como uso de tecnologias móveis para promover troca de bens, serviços, informação e conhecimento. Ao apostar na convergência da telefonia móvel com tecnologias da Internet, novas tecnologias da informação e comunicação em qualquer lugar a qualquer momento, aumentará o potencial de uso para o setor agrícola e agroindustrial (NETO *et al.*, 2007).

Nos setores agrícola e agroindustrial pode-se encontrar alguns exemplos da utilização de dispositivos móveis e “m-Business”, como: Agricultura de precisão; Monitoramento e Controle; Identificação eletrônica animal; Assistentes pessoais digitais (PDA); Gestão de frotas e; Veículos aéreos não tripulados (NETO *et al.*, 2007).

Mesas-Carrascosa *et al.*, (2012) desenvolveram uma aplicação móvel para realizar trabalhos a campo, com uso de GPS e a câmera fotográfica do aparelho, chamado Geofoto, o autor ainda cita as vantagens do uso de smartphones para esta tarefa devido a sua facilidade de manuseio e mobilidade, seu resultado contribui para aplicação do m-business ao agronegócio.

Antonopoulou *et al.*, (2010) criaram uma solução móvel para tomada de decisão agrícola baseada na web para auxiliar os agricultores no processo de seleção de culturas alternativas adequadas para sua região, como resultado tem-se um exemplo de aplicação agrícola inteligente.

2.1.3 *Internet of Things* (IoT) ou Internet das Coisas

O conceito de *Internet of Things* (IoT) ou Internet das Coisas tem suas raízes no MIT (Massachusetts Institute of Technology), a partir de um trabalho do *Auto-ID Center*, este grupo trabalhou no campo de redes de RFID. De acordo com o Cisco *Internet Business Solutions Group* (IBSG), a definição para Internet das Coisas é simplesmente um ponto onde “coisas ou objetos” estão mais conectados a internet do que pessoas. A Cisco IBSG estima que a Internet das Coisas nasceu entre 2008 e 2009 quando o número de dispositivos conectados superou o número de pessoas (EVANS, 2011).

Ainda segundo Evans, (2011), atualmente a Internet das Coisas é composta por um conjunto disperso de redes construído para propósitos diferentes, por exemplo, carros possuem múltiplas redes para controlar a função do motor, características de segurança, sistemas de comunicação, e assim por diante; Edifícios comerciais e residenciais também possuem vários sistemas de controle para aquecimento, ventilação e ar condicionado, serviços de telefone, segurança e iluminação. Com a evolução da Internet das Coisas, essas serão conectadas com maior segurança, análise e capacidade de gestão, como exibido na Figura 1. Isso permitirá que a Internet das Coisas se torne uma “rede de redes” e auxilie pessoas.

Figura 1 – *Internet das coisas* vista como uma “rede de redes”.

Fonte: Adaptado de Evans, 2011.

Neste contexto a Internet das Coisas torna-se importante como a primeira evolução real da Internet, como um *upgrade* para levar aplicações revolucionárias que tem o potencial de melhorar drasticamente a forma como as pessoas vivem, aprendem, trabalham e se divertem, por exemplo: Pacientes poderão utilizar sensores em seu corpo para auxiliar

médicos; Sensores extremamente pequenos poderão ser colocados sobre plantas, animais e características geológicas; e Tudo isto conectado com a Internet. (EVANS, 2011).

Atzori *et al.*, (2010) sintetizou as principais áreas de pesquisa e tipos de aplicação que a Internet das Coisas pode ser utilizada, como pode ser visto na Figura 2.

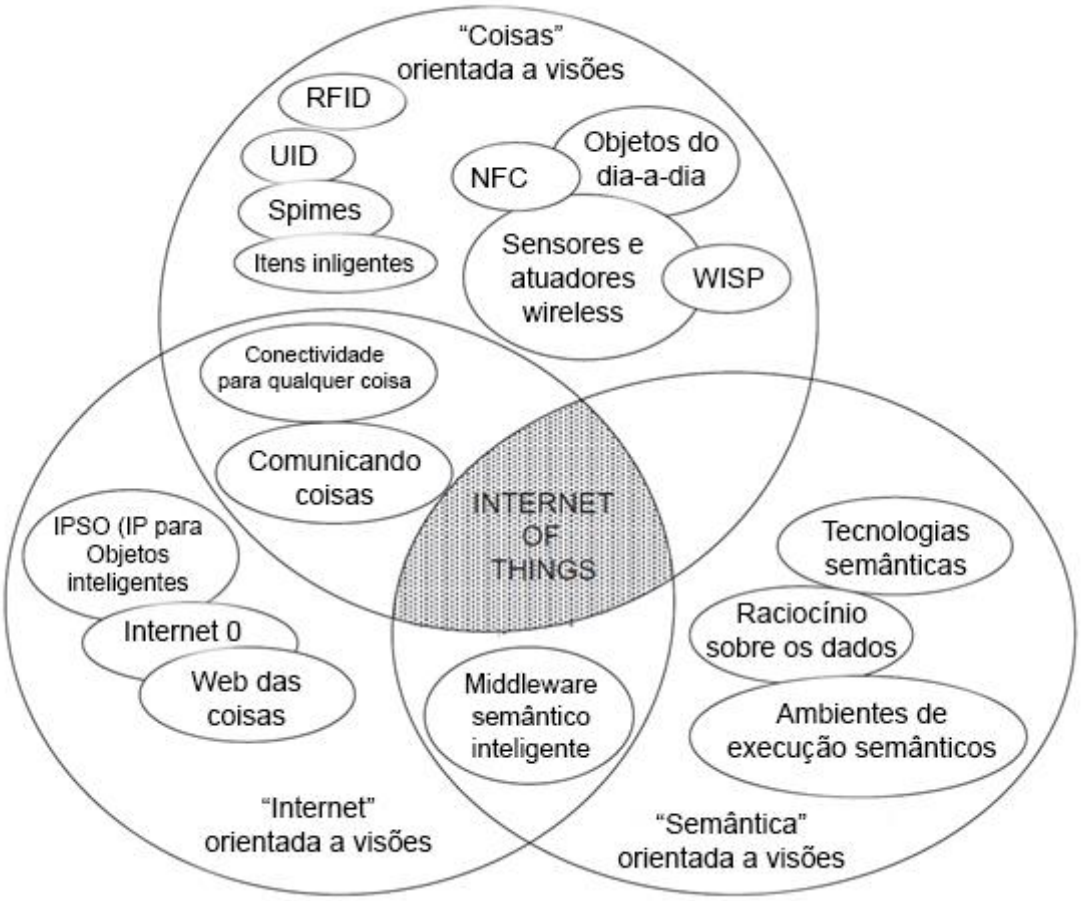


Figura 2 – O paradigma da *Internet das Coisas*

Fonte: Adaptada de Atzori et al., 2010.

Um desafio para a Internet das coisas é imposto no que diz respeito aos padrões utilizados para interligar estas redes. Segundo Evans (2011), muitos progressos ainda são necessários na área de normas, o Instituto de Engenheiros Eletricistas e Eletrônicos (IEEE) é uma organização que trabalha para resolver estes desafios, assegurando que os pacotes IPv6 podem ser encaminhados através de diferentes tipos de rede. Levando em consideração os benefícios da Internet das Coisas, resolver estes problemas é apenas uma questão de tempo.

Este esforço exigirá das empresas, governos, organizações de padrões e acadêmicos para trabalhar em conjunto de um objetivo comum, assim a Internet das Coisas

pode ganhar aceitação da população em geral, prestadores de serviços e outros os quais devem entregar aplicações que tragam valor tangível para a vida das pessoas. A Internet das Coisas tem o potencial de mudar o mundo tal como conhecemos hoje, para melhor (EVANS, 2011).

Para melhor compreensão dos conceitos da Internet das Coisas apresenta-se na Figura 3 uma arquitetura geral de camadas, desenvolvida por Pande & Pandwalkar, (2014). As camadas estão divididas em: Camada de Aplicação; Camada de Rede; e Camada de Contexto-Informado. A Camada de Contexto-Informado apresenta uma variedade de sensores tecnológicos onde os dados são coletados, como os sensores de um smartphone, por exemplo, a Camada de Rede integra os vários sensores e transfere suas informações, *web services* podem ser integrados a aplicação para realizar esta tarefa e a Camada de Aplicação apresenta as diversas funcionalidades empregadas para disponibilizar todas as informações para o usuário final.

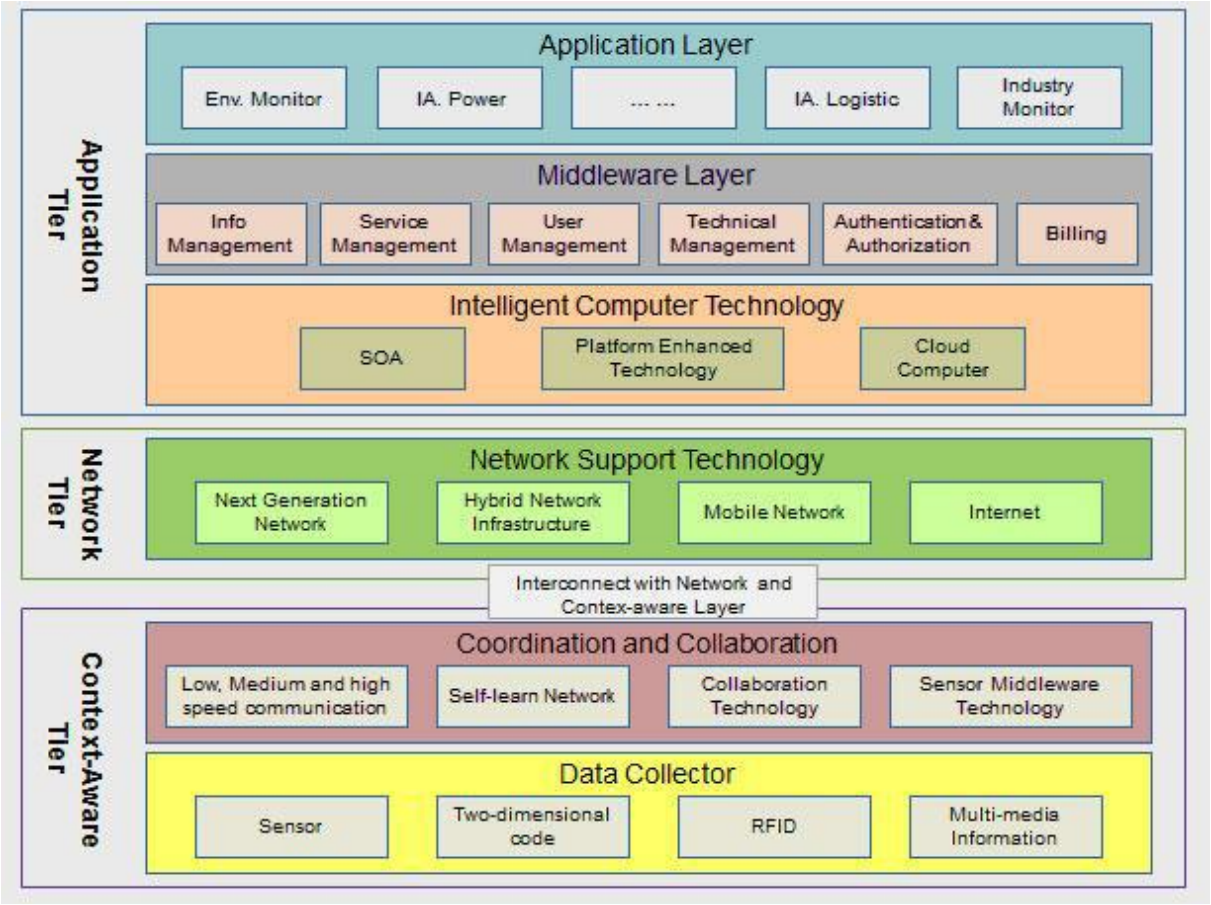


Figura 3 – *Internet das Coisas* - Arquitetura dividida em camadas.

Fonte: Pande & Pandwalker, (2014).

2.1.4 Internet das Coisas e Cloud Computing

Um dos desafios da Internet das Coisas é se adaptar a ambientes elásticos, neste contexto a *cloud computing* pode auxiliar nesta tarefa, pois pode ser definida como um ambiente elástico de execução de recursos envolvendo múltiplas partes interessadas e fornecendo um serviço calibrado de granularidade múltipla com um nível específico de qualidade (PANDE & PANDWALKER, 2014). Um modelo de desenvolvimento e uso de software baseado nos conceitos de Internet das Coisas e utilizando recursos da *cloud computing* é o Software as a Service (SaaS).

O modelo SaaS estabelece softwares com propósitos específicos que estão disponíveis para os usuários por meio da Internet, podem ser acessíveis a partir de inúmeros dispositivos por meio de uma interface*,* como um navegador Web. O usuário não administra ou controla a infraestrutura, incluindo rede, servidores, sistemas operacionais, armazenamento ou características individuais da aplicação, exceto configurações específicas, sendo assim desenvolvedores em inovação e não infraestrutura, resultando em desenvolvimento ágil de *softwares.* Com o software disponibilizado na web, qualquer usuário a qualquer momento pode acessá-lo, permitindo a integração entre unidades de uma empresa ou outros serviços de software, assim novos recursos podem ser incorporados automaticamente sem que os usuários percebam estas ações, tornando a evolução e atualização transparente dos sistemas (SOUSA *et al.*, 2009).

Segundo Pande & Pandwalker, (2014) a Internet das Coisas certamente terá que adaptar-se com questões relacionadas a elasticidade, confiabilidade e gerenciamento de dados. Os recursos da computação em nuvem podem hospedar e processar dados formando uma unidade computacional (Plataforma de processamento virtual). Nuvens especializadas podem, por exemplo, integrar sensores dedicados para fornecer recursos aprimorados e resolver questões relacionadas ao fluxo de dados, independente do tipo de fonte de dados.

Nuvens podem oferecer um apoio vital para a Internet das Coisas, a fim de lidar com uma quantidade flexível dos dados provenientes de diversas fontes de dados. Conceitos de escalabilidade e elasticidade podem ser de interesse para a Internet das Coisas, e espera-se que esta seja o maior consumidor da nuvem (PANDE & PANDWALKER, 2014).

Kaloxylos et al., (2014) desenvolveu um sistema baseado no conceito de computação na nuvem para administração de fazendas. Nikkila et al., (2010) criaram um software que integra conceitos de sistema de informação para administração de fazenda e agricultura de precisão baseado em conceitos de aplicações web.

Duro *et al.*, (2015) apresenta uma arquitetura baseada em NoSQL para solução de armazenamento rápido em aplicativos móveis que operam na nuvem e com grande escala de dados.

2.1.5 *Web service REpresentational State Transfer* (REST)

O REST é um padrão arquitetural que utiliza elementos Web para criar um sistema de hipermídia distribuído. Suas principais características são: todos seus recursos de sistema possuem um *Uniform Resource Identifier (URI)*, ou seja, um identificar único; Seus recursos são disponibilizados pelo protocolo *Hypertext Transport Protocol* (HTTP) e a comunicação entre cliente e servidor deve ser *stateless* (sem estado) (HONDA & ZARPELÃO, 2014).

A arquitetura Cliente-Servidor REST possibilita que os componentes, o cliente e o servidor se desenvolvam de forma independente, pois existe uma separação de responsabilidades bem definida. A evolução independente de cada parte torna o REST apto a suportar requerimentos de aplicações em múltiplos domínios organizacionais na escalada da Internet (FIELDING, 2000).

No REST as comunicações não devem depender de estados controlados pelo servidor. Cada requisição feita pelo cliente sempre deverá possuir todos os atributos para que ela ser processada pelo servidor sem que este último se aproveite de informações adicionais sobre o contexto da comunicação, ou seja, toda informação sobre o estado deve ser de conhecimento somente do cliente. O REST pode ser definido por quatro elementos de interface: identificação de recursos; manipulação de recursos através de representações; mensagens auto-descritivas; e hipermídia como máquina de estados da aplicação (FIELDING, 2000).

Segundo Fielding (2000), um sistema distribuído pode ser implementado utilizando três alternativas:

- Processar os dados onde eles estão localizados e enviar a resposta ao cliente em um formato específico;
- Encapsular os dados e o código de processamento e enviá-los ao cliente para que este último o execute;
- Enviar os dados brutos ao cliente juntamente com metainformações que descrevam os dados, permitindo ao cliente utilizar o processamento que o for conveniente.

O REST é um híbrido destas três alternativas, focando em um compartilhamento dos tipos de dados através de metadados, e limitando o escopo do que é revelado através de uma interface uniforme. No REST os componentes se comunicam através da transferência de representações de um recurso em um formato selecionado dinamicamente de um conjunto de tipos de dados padrões, baseado nas capacidades e desejos do componente que irá receber as informações e da natureza do recurso (FIELDING, 2000).

Uma representação, em REST, é uma sequência de bytes e mais os metadados desta representação, que descrevem os bytes, exemplos: documentos em formato texto, arquivos binários, mensagens HTTP. Uma representação consiste de dados, metadados para descrever os dados, e outro conjunto de metadados para verificação de integridade da mensagem. Os metadados são apresentados na forma de pares nome-valor, onde o nome corresponde a um padrão que define a estrutura do valor e sua semântica (FIELDING, 2000).

Informações de controle também são utilizadas na representação, com o objetivo de definir o propósito de uma mensagem entre os componentes ou parametrizar requisições e redefinir o comportamento padrão de alguns elementos, por exemplo, o comportamento de um cache pode ser alterado através dessas informações embutidas em uma requisição ou uma resposta. Estas informações também podem ser utilizadas para indicar o estado atual de um recurso solicitado, o estado desejado para um recurso ou a representação de alguma condição de erro para uma resposta (FIELDING, 2000).

Mohamed & Wijesekera, (2012) comparam o uso entre *web services* de padrão RESTful e o *Simple Object Access Protocol* (SOAP) em aplicações móveis e como conclusão apresentam o REST como o serviço que consome menos recursos e dispõe de resultados mais eficientes. Arroqui *et al.*, (2012) também compararam o uso do padrão RESTful e SOAP, porém em ambientes agrícolas com smartphones android, o web service REST apresentou uma quantidade de bytes transferidos menor que o SOAP e ainda ressalta que esta diferença é importante para ambientes onde o acesso é realizado apenas via GPRS ou 3G. Lomotey & Deters, (2012) desenvolveram um middleware chamado SOPHRA, que foi classificado pelos próprios autores como um e-saúde para centralização de dados em um hospital geriátrico, o middleware utilizou conceitos RESTful para coleta e centralização padronizada de dados em seu servidor na nuvem.

2.1.6 Web service REST e a Internet das Coisas

Encontrar uma arquitetura segura e eficiente em termos de custo, flexibilidade e escalabilidade, capaz de lidar com o cenário complexo da Internet das Coisas é um dos

principais objetivos para adoção rápida do conceito de Internet das Coisas. Um grande número de soluções técnicas de interoperabilidade torna difícil para definir o processo de desenvolvimento da Internet das Coisas. Desenvolver uma arquitetura de referencia da Internet das Coisas é um objetivo recente, sendo assim os principais organismo de normalização (3GPP, IEEE ETSI) começaram a trabalhar para esta realização e criaram um projeto de duas arquiteturas de rede conhecido como Arquitetura de Rede Hierárquica para conectividade escalável, prometendo satisfazer os requisitos da Internet das Coisas. Esta é um arquitetura de alto nível, composta de três domínios (gateway, rede e aplicação) é apresentada na Figura 4. (BORGIA, 2014).

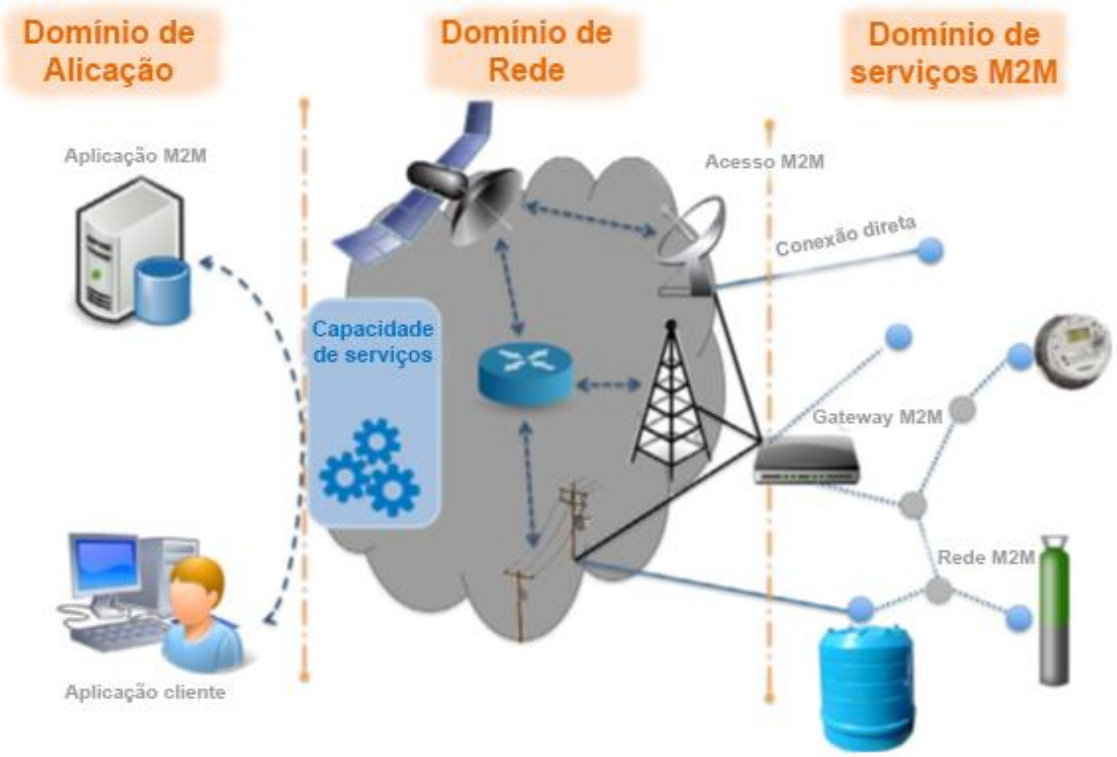


Figura 4 – Arquitetura de alto nível ETSI M2M.

Fonte: Adaptado de Borgia, 2014.

O responsável por esta arquitetura é o Comitê Técnico ETSI para comunicações *Machine-to-Machine,* a ideia é desenvolver uma arquitetura baseada em IP contando com características como: escalabilidade, desenvolvimento fácil, baixa complexidade, eficiente, compatível, utilizar interfaces normalizadas e protocolos padronizados. A arquitetura ETSI M2M é baseada no estilo arquitetônico do REST, onde qualquer entidade lógica e física é representada como um recurso que tem um estado particular que pode ser manipulado (um sensor, por exemplo, que pode ser lido e configurado). Os recursos são exclusivamente endereçáveis, e podem ser acessados com links da web utilizando navegadores e ações HTTP (BORGIA, 2014).

2.2 Sistemas de Informação Geográfica (SIG)

Neste tópico será abordado conceitos de softwares SIG e SIG Móvel e uma breve descrição sobre modelagem conceitual e espacialização de dados geográficos.

2.2.1 Sistema Global de Navegação por Satélite (GNSS)

O termo GNSS surgiu em 1991, inicialmente chamado de GNSS-1 englobando os sistemas GPS e GLONASS, posteriormente foi chamado de GNSS-2 quando agregou o sistema GALILEO. Um Sistema Global de Navegação por Satélite é formado por uma constelação de satélites com cobertura global que envia sinais de posicionamento e tempo para usuário localizados em solo, aeronaves ou transporte marítimo. Existem vários sistemas GNSS, como o GPS (dos EUA) que é utilizado pelos *smartphones* e será abordado em detalhes na sequência, o GLONASS (da Rússia), o GALILEO (da Europa) e o COMPASS (China) (SEBEM *et al.*, 2010).

Segundo Monico (2007), o GPS é um sistema de rádio navegação desenvolvido pelo Departamento de Defesa dos Estados Unidos da América – DoD (*Department of Defense* ou Departamento de Defesa), com o intuito de ser o principal sistema de navegação das forças armadas americanas. Em razão de sua alta acurácia e do grande desenvolvimento tecnológico envolvendo os receptores GPS, surgiu uma grande comunidade de usuários em diversos segmentos (navegação, posicionamento geodésico, agricultura, controle de frotas e outros).

Segundo SEBEM et al., (2010), o sistema GPS permite aos usuários determinar suas posições em coordenadas cartesianas retangulares X, Y, Z em relação ao centro de massa da terra (0, 0, 0) e posteriormente convertê-las em coordenadas elipsoidais, apresentadas como latitude, longitude e altura elipsoidal h (Figura 5).

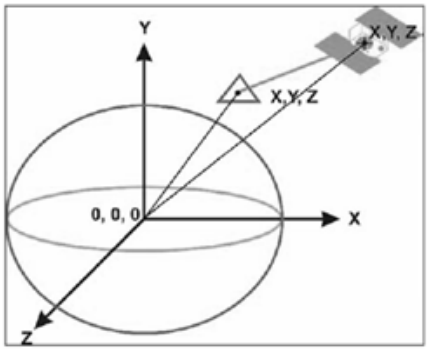


Figura 5 - O GPS e seu sistema de coordenadas.

Fonte: Robaina, 2006.

O sistema planimétrico de referência do GPS é o *World Geodetic System* 1984 (WGS 84), além das coordenadas o sistema conta com uma medida de tempo (feita por relógios atômicos), das quais existem 3 tipos de escalas de tempo: O tempo na escala de tempo do satélite; O tempo na escala de tempo do receptor e o tempo na escala de tempo controlada pelo segmento de monitoramento e controle (tempo “oficial” do sistema GPS) (SEBEM *et. al.*, 2010).

A estrutura do sistema GPS é dividida em três segmentos principais: De espaço ou espacial, onde sua estrutura final compreende em 27 satélites (24 operacionais e 3 de reserva); De controle e monitoramento, onde tem a função de realizar o monitoramento contínuo dos satélites, calcular suas posições, transmitir os dados e executar a supervisão necessária para o controle dos satélites; De usuários, onde compreende os usuários do sistema e os tipos de receptores (SEBEM *et. al.*, 2010).

No que tange o segmento espacial, Monico (2007) cita a importância da *Dilution of Precision* (DOPs) ou Diluição da Precisão, onde frequentemente é utilizada em navegação e no planejamento de observações GNSS, os valores são obtidos a partir do conceito de posicionamento por ponto dos satélites. O DOP auxilia na indicação da precisão dos resultados que serão obtidos, e depende basicamente de dois fatores:

- Da precisão da observação de pseudodistância, expressa pelo *User Equivalent Range Error (UERE)* ou erro equivalente do usuário, que é associado ao desvio-padrão da observação; e
- Da configuração geométrica dos satélites, obtidas pelos DOPs.

Para SEBEM *et al.*, (2010) o fator DOP descreve o efeito da distribuição dos satélites no espaço sobre a precisão obtida na solução de navegação, sendo o melhor valor possível para DOP igual a 1 e o pior é igual ao infinito.

Segundo Monico (2007) as seguintes designações são encontradas na literatura:

- HDOP: para posicionamento horizontal;
- VDOP: para posicionamento vertical;
- PDOP: para posicionamento tridimensional; e
- TDOP: para determinação de tempo.

O efeito combinado de posição tridimensional e tempo é denominado GDOP, é apresentado na Equação 1:

$$\boldsymbol{GDOP} = \sqrt{(\boldsymbol{PDOP})^2 + (\boldsymbol{TDOP})^2} \qquad \textbf{(1)}$$

O PDOP pode ser interpretado com o inverso do volume V de um tetraedro formado pelas posições do usuário e dos quatro satélites (Equação 2):

$$\boldsymbol{PDOP = \frac{1}{V}} \qquad \textbf{(2)}$$

Segundo Monico (2007) a melhor geometria ocorre quando o volume é maximizado, o que implica um PDOP mínimo, Na Figura 6 (a) pode-se observar que os satélites estão mais dispersos que em relação a Figura 6(b), sendo assim se conclui que o volume na Figura 6 (a) é maior que na Figura 6 (b), portanto o PDOP é melhor na Figura 6 (a) do que na Figura 6 (b) (MONICO, 2007). Para Sebem *et al.*, (2010) um GDOP maior que 6 (GDOP > 6) é considerado uma distribuição ruim e um GDOP abaixo de 6 (GDOP < 6) é considerado uma distribuição boa. Em resumo, quanto menor for o valor dos diferentes DOPs, melhor a configuração dos satélites para realizar o posicionamento.

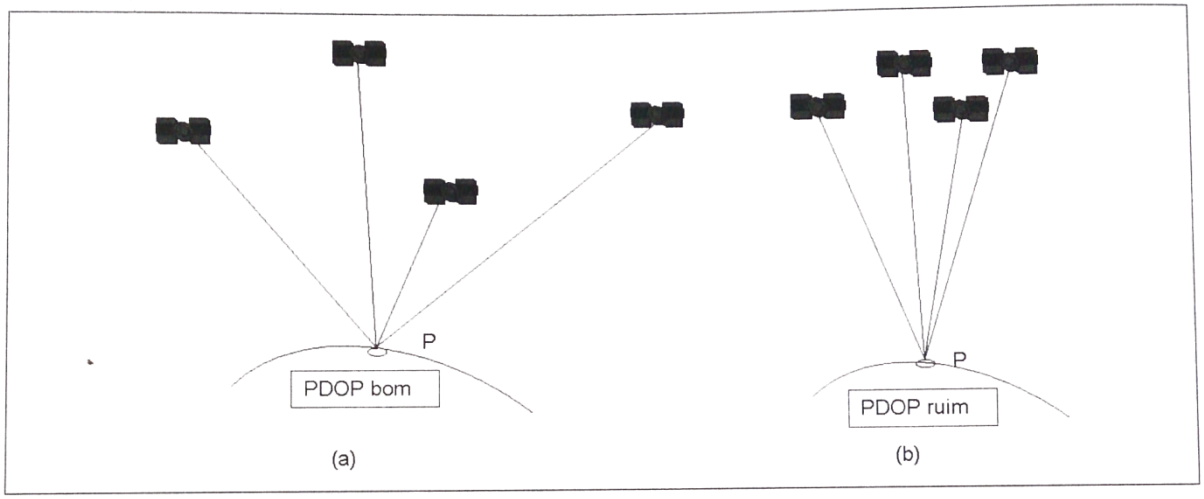


Figura 6 – Geometria dos satélites, (a) PDOP bom e (b) PDOP ruim.

Fonte: Monico (2007).

A definição dos diversos DOPs, a partir do conceito de ajustamento de observações, deixa claro que, quanto maior é o número de satélites sendo rastreados, é de se esperar que melhores serão os diversos DOPs (MONICO, 2007).

Smartphones utilizam antenas de GPS pequenas, pouco sensíveis ou chipsets GPS integrados, com menos recursos. Para compensar esta diferença de sensibilidade e permitir que o mapeamento dos satélites seja obtido de forma rápida, mesmo com um sinal fraco, os *smartphones* utilizam um sistema híbrido conhecido como *Assisted GPS* (A-GPS). Neste sistema o sinal dos satélites é combinado com informações transmitidas pela rede do celular sobre seu posicionamento atual (MARIMOTO, 2009).

2.2.2 SIG e SIG Móvel

Um SIG é um sistema computacional para captura, armazenamento, consulta, análise e exibição de dados geoespaciais. Dados geoespaciais descrevem a localização e as características de dados espaciais. A habilidade do SIG de manipular e processar dados geoespaciais, diferencia um SIG de um sistema de informação, um SIG é usado para integrar dados geoespaciais e outros dados, tabulares. Diante disso a tecnologia SIG é importante para geo-cientistas, cartógrafos, engenheiros ambientais e profissionais de planejamento urbano e ambiental (CHANG, 2012). A indústria de *software* SIG contabiliza 1 bilhão de dólares em vendas anuais (LONGLEY *et al.*, 2013).

Um SIG também pode ser traduzido como: Um repositório de mapas de meio digital; Uma ferramenta computadorizada para resolver e gerenciar problemas geográficos; Um sistema de apoio a decisão espacial; Um inventário mecanizado de feições e serviços geograficamente distribuído; Uma ferramenta para revelar o que de outra forma é invisível na informação geográfica; Uma ferramenta para realizar operações sobre dados geográficos, que são muito tediosas, onerosas ou imprecisas se feitas manualmente (LONGLEY *et al.*, 2013).

Segundo Longley et al., (2013), a anatomia do SIG é composta por seis componentes, a rede, onde ocorrem a comunicação rápida e o compartilhamento de informações, fortemente baseado na internet. O *hardware*, onde apenas o usuário interage diretamente com o SIG. O *software* que processa localmente na máquina do usuário. O banco de dados, que armazena e realiza a representação digital de aspectos de uma área específica. O gerenciamento de um SIG, onde uma organização estabelece procedimentos, linhas de comunicação, pontos de controle e outros mecanismos para atingir suas necessidades por meio do SIG. E finalmente, um SIG não é útil sem pessoas que o concebam, programem, mantenham, alimentem-no com dados e interpretem seus resultados (Figura 7).

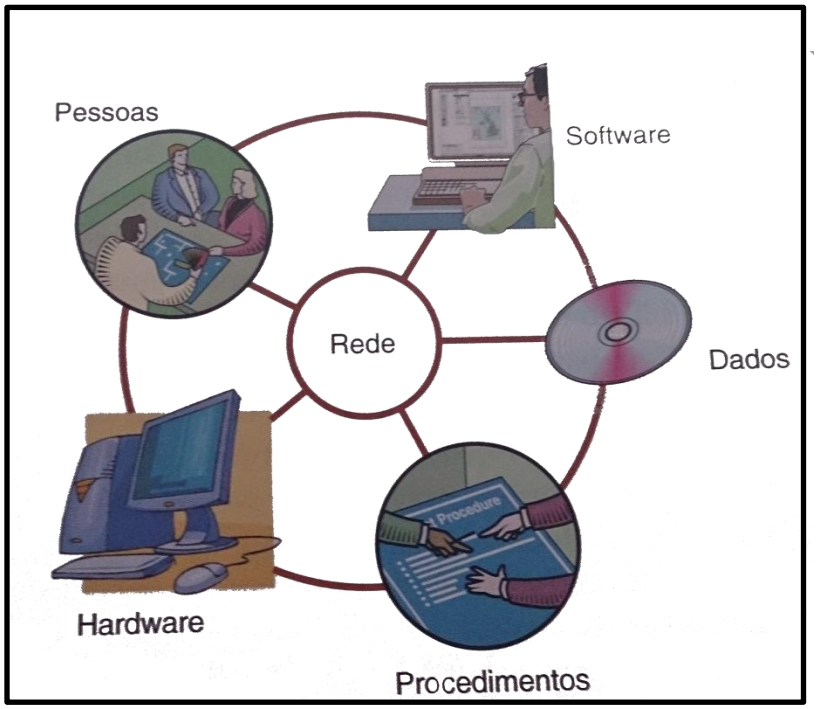


Figura 7 - Os seis componentes de um SIG.

Fonte: Longley *et al.*, (2013).

Embora as atividades do SIG, não sigam uma sequência conjunta é possível explicar, as atividades que o SIG realiza em: Aquisição de dados; Gestão de dados e atributos; Apresentação de dados; Exploração de dados; Análise de dados e Modelagem de SIG (CHANG, 2012).

Os autores Steiniger & Hunter (2013) identificam sete tipos de software de SIG: 1) Desktop; 2) Sistemas de administração de banco de dados espaciais; 3) Servidor web de mapas; 4) Servidor SIG; 5) Clientes SIG web; 6) SIG Móvel e 7) Bibliotecas e extensões. O autor ainda cita o termo *Free and Open Source Software for GIS/Geospatial* (FOSS4G) ou seja, Softwares SIG/Geoespaciais livres ou de código aberto, que geralmente são desenvolvidos por: 1) companhias; 2) indivíduos (programadores freelance ou especialistas em SIG) e 3) instituições, como institutos, universidade ou autoridades públicas, a Figura 8 apresenta um mapa das principais aplicações SIG livres disponíveis no mercado.

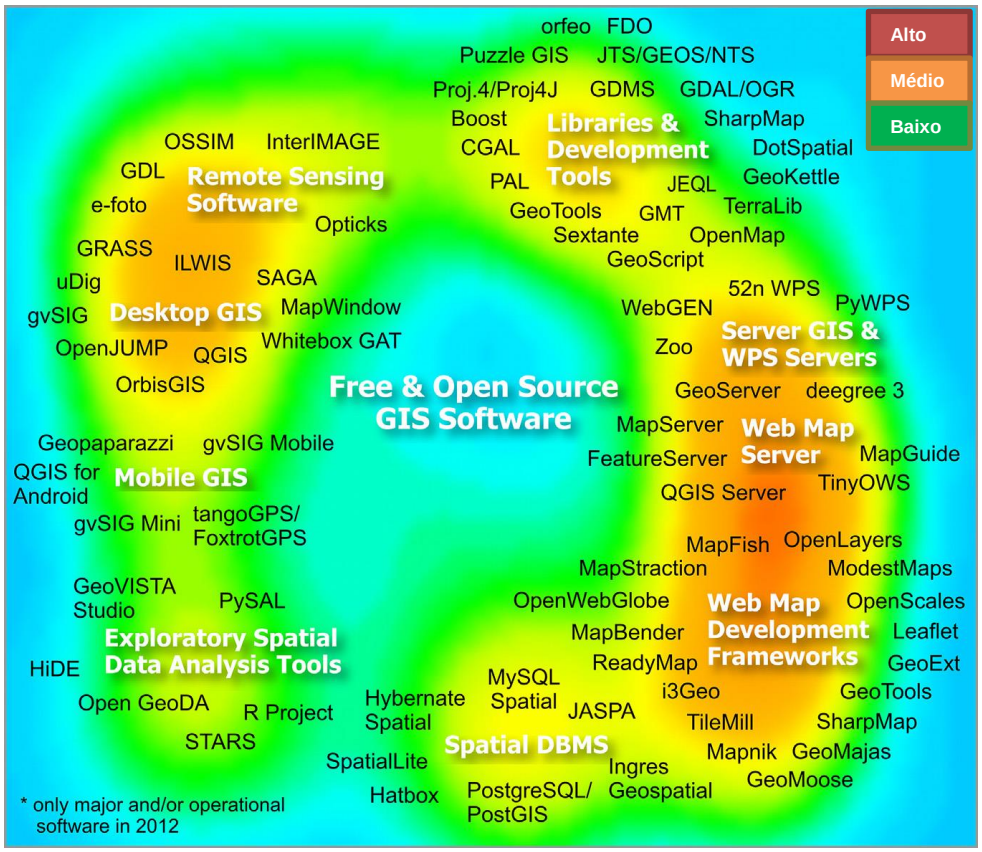


Figura 8 – Mapa de softwares SIG livres e de código aberto

Fonte: Steiniger & Hunter, (2013).

Segundo Longley *et al.*, (2013), uma das áreas mais inovadoras atualmente é o desenvolvimento de *software* portátil para smartphones, apesar de compacto esses equipamentos trabalham com alta quantidade de dados (8 *Giba Bytes* (GG) ou mais), Os sistemas operam em ambiente misto conectado/desconectado, assim podem utilizar dados e software hospedados em um servidor, um exemplo é o acesso ao *Google Maps* por meio de conexão sem fio (Figura 9).

Figura 9 - Dispositivos móveis apresentando o Google Maps, exemplo de SIG Móvel.

Fonte: http://itunes.apple.com

O SIG Móvel ou portátil é muito diferente do SIG de escritório, pois para o SIG Móvel a localização do usuário é importante e diretamente relevante à aplicação. É possível centralizar o mapa na localização do usuário para fornecer recursos à aplicação a partir de sua localização (LONGLEY *et al.*, 2013). Segundo Steiniger & Hunter, (2013) pode ser considerado SIG Móvel o aplicativo que: 1) trabalha em plataforma móvel, como tablete, *Personal Digital Assistant* (PDA) ou smartphone, e que 2) disponibilize funções espaciais que suporte aquisição e atualização de dados em campo. Geralmente suas funções são baseadas em sua capacidade de uso de localização por GPS e ligadas a um SIG desktop.

Algumas vantagens de utilizar um SIG Móvel são: Aumento progressivo da disponibilidade de aparelhos de baixo custo com recursos de localização geográfica; Proliferação de dados geograficamente referenciados, gerados com o uso da tecnologia GPS (LONGLEY *et al.*, 2013).

2.2.3 Modelagem conceitual de dados geográficos

Segundo Câmara *et al.*, (2005) modelos de dados são classificados de acordo com sua abstração, para a classificação de dados geográficos são utilizados quatro níveis de abstração:

- **Nível do mundo real**: Representa fenômenos geográficos reais, como rios, ruas e cobertura vegetal;

- **Nível de representação conceitual**: Apresenta um conceito formal com os quais as entidades geográficas para serem compreendidas pelo usuário, em um alto nível de abstração. São definidos classes básicas, contínuas e discretas, que serão criadas no banco de dados, e estão associadas a classes de representação espacial;
- **Nível de apresentação**: Utiliza ferramentas que podem especificar aspectos visuais que as entidades geográficas assumirão ao longo de seu uso em aplicações;
- **Nível de implementação**: Define padrões, formas de armazenamento e estrutura de dados para cada tipo de representação, os seus relacionamentos e suas funções e métodos.

No nível de implementação é onde são utilizadas as linguagens de definição de dados associados a um Sistema Gerenciador de Banco de Dados (SGBD) espacial. A Figura 10 apresentam os níveis de abstração de aplicações geográficas.

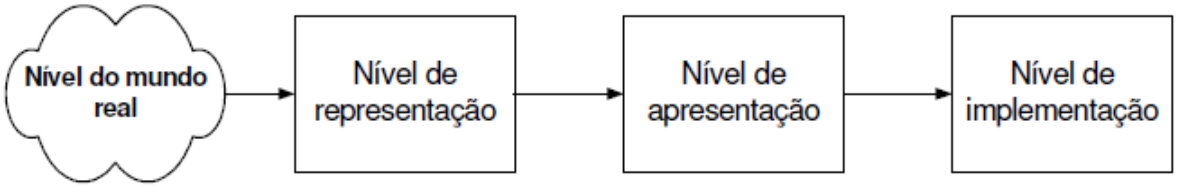


Figura 10 – Níveis de abstração de aplicações geográficas.

Fonte: Câmara *et al.*, (2005).

Uma das etapas da modelagem de dados geográficos é a espacialização de dados georreferenciados, esta etapa ocorre no nível de representação, ou seja, trata da discretização e quantização de dados do mundo real para o ambiente computacional.

Segundo Druck *et al.*, (2004), a espacialização de dados de um determinado espaço geográfico possui dois componentes, a cartografia e o geoprocessamento. Cartografia preocupa-se em apresentar um modelo de representação de dados para os processos que ocorrem no espaço geográfico. Geoprocessamento representa a área do conhecimento que utiliza técnicas matemáticas e computacionais, para tratar os processos que ocorrem no espaço geográfico. Desta forma, se estabelece uma relação interdisciplinar entre Cartografia e Geoprocessamento.

A espacialização de dados se caracteriza pela atribuição da localização geográfica, existe outras informações relacionadas porém a localização é a mais preponderante. Um objeto, como uma cidade, somente tem sua localização geográfica estabelecida quando se

é determinado um sistema de coordenadas (DRUCK *et al.*, 2004). Em sistemas SIGs é necessário a criação e uso de pontos, linhas e regiões espacializadas.

O objetivo da análise de pontos (ou eventos) é estudar a distribuição espacial de fenômenos expressos por meio de pontos localizados no espaço ou em uma determinada área. A Figura 11 apresenta um exemplo da distribuição de pontos em uma determinada área de estudo.

Figura 11 – Exemplo de dados pontuais

Fonte: Druck *et al.*, (2004).

Um padrão espacial de pontos se trata de um conjunto de dados espaciais, que possuem coordenadas de seus locais e sua referência temporal. O objeto de interesse na análise deste tipo de dados é a sua própria localização espacial. Existem formas de analisar estes dados espaciais, na Figura 12 é possível verificar a forma como estes pontos podem ser agrupados em uma representação espacial de dados.

Figura 12 – Pontos agrupados em uma apresentação espacial de dados.

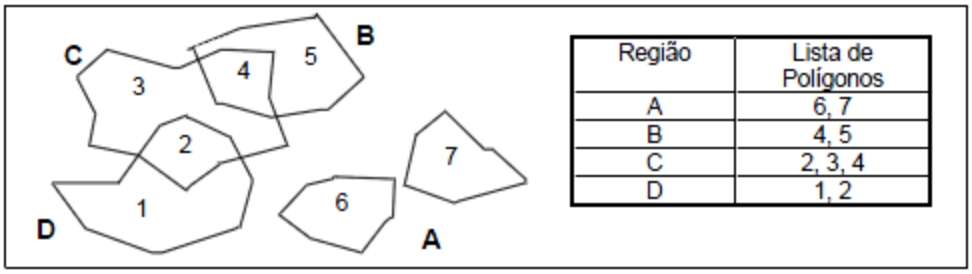
Fonte: Druck *et al.*, (2004).

Os pontos não estão associados a valores, mas sim a ocorrência de sua incidência, e normalmente possuem atributos de identificação, outra característica é que a área dos eventos não é considerada uma medida válida na distribuição destes pontos, como por

exemplo, a localização de crimes, núcleos celulares ou a ocorrência de doenças (DINIZ, 2000).

Regiões espacializadas possuem relacionamento complexo entre feições geográficas representadas por polígonos, chamada de topologia, então pode-se dizer que uma região possui um conjunto de polígonos. Por exemplo, uma região de floresta e uma outra região de floresta destruída pelo fogo são representadas por polígonos que indicam as áreas de florestas antes do incêndio e depois do incêndio. Outro exemplo, o Brasil é uma região representada por vários polígonos. Assim como ponto, linha e polígono, a cada região é dado um identificador único e o cálculo da área e perímetro são mantidos. (DRUCK *et al*., 2004)

A Figura 13 apresenta o que é o conceito de uma região especializada.



Região	Lista de Polígonos
A	6, 7
B	4, 5
C	2, 3, 4
D	1, 2

Figura 13 – Representação de regiões especializadas.

Fonte: Druck *et al.*, (2004).

Os atributos descritivos associados às feições geográficas são armazenados da mesma forma que as coordenadas, isto é, no banco de dados relacional e, o arquivo com os dados descritivos é chamado de tabela de atributos. Cada linha desta tabela possui informações descritivas de uma única feição. Na prática um identificador único realiza a ligação entre as feições geográficas e a tabela de atributos, mantendo um elo (topologia matemática) entre registro espacial e registro de atributos. Realizada esta conexão pode-se apresentar as informações descritivas sobre o mapa (DRUCK *et al.*, 2004).

Na Figura 14 a coluna “polígono” armazena o identificador único que estabelece a ligação entre dados espaciais e os dados descritivos. A tabela “Tabela de Atributos de Polígonos” trás os atributos espaciais descritos, ou não espaciais.

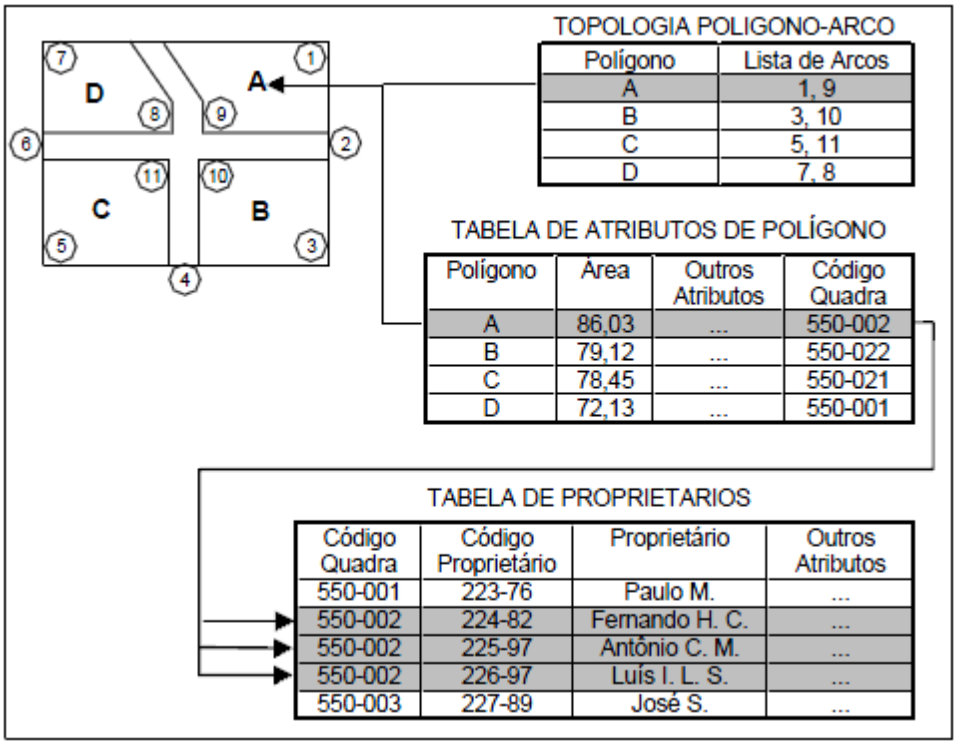


Figura 14 – Representação das informações descritivas.

Fonte: Câmara, (2005).

2.3 Banco de Dados

Neste tópico será descrito o banco de dados SQLite utilizados no projeto mobile e o banco de dados NoSQL utilizado no SaaS, e algumas de suas características.

2.3.1 Banco de Dados SQLite e NoSQL

O *SQLite* é uma biblioteca que implementa um banco de dados com Linguagem de Consulta Estruturada (SQL) local, auto-suficiente, sem servidor e sem nenhuma configuração, isto significa que você não precisa configurá-lo em seu sistema. O código fonte do *SQLite* é de domínio público (TUTORIALS POINT, 2014).

Algumas características do *SQLite* são: O *SQLite* não quer um processo servidor separado ou sistema para o operar; Não necessita nenhuma configuração ou administração necessária; Um banco de dados completo é armazenado em um arquivo de disco, de plataforma única; O *SQLite* é pequeno e leve, menos de 400kb totalmente configurado; O *SQLite* não possui dependências externas; O *SQLite* suporta os recursos de linguagem de consulta encontradas no padrão SQL92 (SQL2); O *SQLite* fornece uma API simples de usar;

SQLite é acessível em *UNIX* (Linux, Mac OS X, Android, iOS) e Windows (Win32, WinCE, WinRT) (TUTORIALS POINT, 2014).

Para o uso do *SQLite* em ambiente de desenvolvimento *Android* não é necessário uso de pacote ou *framework* externo, o próprio *kit* de desenvolvimento do *Android* possui uma API nativa para o uso do *SQLite*, e estes recursos estão disponibilizados no pacote de desenvolvimento *android.database.sqlite*.

Bancos de dados NoSQL, diferem do SQLite e de modelos relacionais, e apresentam em suas diferenças algumas soluções para os desafios da Internet das Coisas, não solucionadas por modelos relacionais.

O termo NoSQL vem do fato de que o banco de dados não utiliza SQL como linguagem de consulta. O NoSQL é manipulado por meio de shell scripts, que podem ser combinados em encadeamentos (pipelines) do UNIX. O uso do termo “NoSQL” é resultado de uma reunião realizada no dia 11 de junho de 2009, em São Francisco, nos Estados Unidos, organizada por Johan Oskarsson (SADALAGE & FOWLER, 2013).

Segundo Sadalage & Fowler, (2013), o termo “NoSQL” nunca favoreceu uma definição precisa, a chamadas inicias sobre o assunto pediam “bancos de dados não relacionais, distribuídos e de código aberto”. Existiam palestras realizadas sobre os Bancos de Dados Voldemort, Cassandra, Dynomite, HBase, Hypertable, CouchDB e MongoDB, mas o termo nunca ficou limitado a este grupo original. Não há uma definição genericamente aceita nem uma autoridade para fornecer uma, tudo o que se pode fazer é discutir algumas características comunus em bancos de dados que tendem a ser chamados de “NoSQL”.

De início o óbvio, bancos de dados NoSQL não utilizam SQL, alguns deles possuem linguagens de consulta, e faz sentido que elas sejam semelhantes ao SQL para que sejam facilmente aprendidas, no caso do CQL do Cassandra é assim “exatamente como SQL (exceto onde não é)”. Outra característica importante desses bancos de dados é que eles geralmente são projetos de código aberto. Bancos de dados NoSQL, na maioria dos casos, são orientados pela necessidade de execução em cluster, isso tem influência sobre seu modelo de dados, assim como sobre sua abordagem quanto a consistência de dados (SADALAGE & FOWLER, 2013).

Bancos de dados relacionais utilizam transações ACID (*Atomicity, Consistency, Isolation, Durability*) para lidar com consistência em todos seus modelos de banco de dados, o que é conflitante em um ambiente de clusters. Sendo assim, bancos de dados NoSQL oferecem uma gama de opções para se tratar consistência e distribuição (SADALAGE & FOWLER, 2013).

Porém, nem todos os bancos de dados NoSQL almejam a execução em *cluster*, bancos de dados NoSQL baseados em grafos, por exemplo, consistem em um estilo de distribuição semelhante ao modelo relacional, porém oferecem uma manipulação mais eficiente de dados com relacionamentos complexos (SADALAGE & FOWLER, 2013).

Os bancos de dados NoSQL atuam sem um esquema, permitindo que sejam adicionados, livremente, campos aos registros do banco de dados, sem ter de definir primeiro quaisquer mudanças em sua estrutura. Há dois motivos principais para considerar NoSQL: o primeiro é lidar com o acesso a dados cujo tamanho e desempenho demandem um cluster; o segundo é melhorar a produtividade de desenvolvimento de aplicativos utilizando um estilo de interação de dados mais conveniente (SADALAGE & FOWLER, 2013).

Mcguinnes & Kapros, (2015) apresentam um sistema de informação adaptativo a mudanças baseados em conceitos NoSQL para facilitar as mudanças no software, e citam que tecnologias de banco de dados NoSQL podem ser utilizadas para criar sistemas adaptativos devido a sua ausência de esquema e facilidade de armazenamento.

2.3.1.1 Características do NoSQL

Os NoSQL estão sendo projetados desde o início para atender a uma maior escalabilidade horizontal, ou seja, se beneficiar com a adição de novas máquinas para melhorar o desempenho do sistema de forma linear. Este tipo de escalabilidade contrasta com a vertical, onde um servidor precisa ser substituído por outro mais potente para aumentar a capacidade computacional do sistema, desta forma a escalabilidade horizontal possui menor custo de expansão (QUEIROZ *et al.*, 2013).

Alguns sistemas NoSQL, como o Cassandra (LAKSHMAN & MALIK, 2010) e o Dynamo (DECANDIA *et al.*, 2007) são descentralizados, ou seja, seus dados ficam replicados por várias máquinas, e cada uma delas é autônoma, sendo capaz de responder a qualquer consulta. Este tipo de estratégia aumenta a disponibilidade do sistema, uma vez que um único ponto não irá comprometer o funcionamento do sistema. Esta é uma característica que contrasta com a maior parte das implementações relacionais, onde, geralmente é utilizado o modelo mestre/escravo. Como neste modelo a escrita é limitada ao mestre, esta apresenta duas desvantagens: maior susceptibilidade a falhas, pois caso o nó mestre apresente uma falha o sistema ficará indisponível; maior dificuldade para obter escalabilidade horizontal, pois o nó mestre pode se tornar um gargalo para o sistema (QUEIROZ et al., 2013).

Segundo Queiroz *et al.*, (2013), outro ponto de contraste entre os NoSQL e os sistemas relacionais é o suporte a transações ACID. Muitos NoSQLs simplesmente não oferecem este recurso, considerado dispensável em algumas aplicações, principalmente pelo custo computacional envolvido para utilizá-las. Em Stonebraker (2011), esta comparação é apresentada mais a fundo.

Existe uma taxonomia dada aos NoSQL, dependendo do modelo de dados e a estratégia de armazenamento que se queira adotar. As próximas seções irão detalhas cada um dos modelos disponíveis.

2.3.1.2 Banco de Dados Orientados a Documentos

Também conhecidos como *Document Stores*, existem vários sistemas que se destacam nesta categoria, como o MongoDB e o Apache CouchDB. Com seus modelos projetados para trabalhar com documentos semi-estruturados (sem um esquema pré-definido). A representação dos documentos utiliza um notação derivada da sintaxe *JavaScript Object Notation* (JSON), um formato leve para intercâmbio de dados baseado na linguagem JavaScript. A capacidade de trabalhar com dados sem um esquema rígido combinado com sistemas de indexação eficientes tornam-se interessantes para aplicações Web 2.0. (QUEIROZ *et al.*, 2013).

No MongoDB, um banco de dados é composto por documentos no formato BSON (JSON Binário), uma versão binária dos documentos JSON, estes lembram os conceitos de tabelas e linhas dos sistemas relacionais, a diferença é que os documentos de uma coleção não precisam obrigatoriamente ter o mesmo esquema, assim os documentos de uma coleção pode possuir campos diferentes uns dos outros (Figura 15).

```
{
     "_id" : "001",
     "tipo": "hospital",
     "nome": "Santa Casa",
"endereco" : {
       "logradouro": "R. das Rosas",
           "numero": 95,
           "cidade": "Ouro Preto" }
},
{
        "_id": "999",
       "tipo": "foco_queimada",
   "long_lat": [-46.76, -20.54],
   "satelite": "GOES-12",
 "observacao": "2012-07-05 05:15:00"
}
```

Figura 15 – Coleção de documentos JSON para representação de pontos de interesse.

Fonte: Queiroz *et al.*, 2013.

No CouchDB um banco de dados é formado por uma coleção independente de documentos JSON, cada documento possui dois atributos, um identificar único e um número de revisão utilizado para versionamento e resolução de conflitos durante operações de atualização.

O protocolo usado para leitura e atualização dos documentos é baseado em uma API RESTful HTTP (RICARDSON e RUBY, 2008), isto é, uma comunicação entre clientes e o servidor CouchDB é realizada pelo protocolo HTTP utilizando métodos *GET*, *PUT*, *POST* e *DELETE*. Esta estratégia é diferente da maioria dos sistemas de banco de dados, que geralmente criam protocolos próprios com conexão via Protocolo de Controle de Transmissão (TCP) (STEVENS, 1994). Uma característica forte deste sistema é sua capacidade de replicação de dados entre nós, utilizado para manter a sincronização entre dispositivos móveis, computadores pessoais e servidores (QUEIROZ et al., 2013).

2.3.2 NoSQL e os dados geográficos

Atualmente, alguns sistemas NoSQL estão introduzindo suporte ao armazenamento e recuperação de dados espaciais. O MongoDB possui índices espaciais para dados representados por pontos, este mecanismo permite realizar consultas de localização, como N vizinhos mais próximos a determinado ponto, de consultas por intervalo, utilizando retângulo de busca com um círculo ou polígono simples como filtro, e utilizando técnicas hash sobre as coordenadas geográficas dos pontos. (QUEIROZ et. al., 2013). O *plugin* para o *software* Quantum GIS chamado MongoDBLayer (MARKUS, 2012) permite a visualização de dados armazenados no MongoDB.

O CouchDB possui ligação com as bibliotecas GDAL e GeoTools, existe também uma extensão espacial chamada GeoCouch (THOMPSON, 2011) que facilita a manipulação de documentos no formato GeoJSON (BUTLER et al., 2012), que se trata de uma especialização da notação JSON que codifica geometrias, feições e coleções de feições.

Os primeiros sistemas com dados geoespaciais desenvolvidos sobre as tecnologias *Document Stores* trabalham com o armazenamento de Pontos De Interesse (POI), o CouchDB tem sido utilizado na construção de servidores que funcionam como um *cache* de *tiles* de imagens, comuns em aplicações de mapas ao estilo do Google Maps (QUEIROZ et al., 2013).

Lee e Kang, (2015) apresentam os desafios e oportunidades que o big data geoespacial apresenta e citam a facilidade do uso do armazenamento de dados em documentos com o MongoDB. Vitolo *et al.*, (2015) utilizam bancos de dados NoSQL

combinados com frameworks OGC para implementar uma maior, mais flexível e heterogênea forma de trabalhar com dados ambientais e seu armazenamento em larga escala.

Queiroz, (2012) desenvolve uma arquitetura de software voltada aos requisitos de acesso e gerenciamento de dados de ferramentas de modelagem dinâmica espacial, e trata volumes de dados consideráveis com auxílio de banco de dados NoSQL em alguns pontos de seu middleware desenvolvido.

2.3.3 MongoDB

O MongoDB é um dos banco de dados NoSQL mais utilizados, devido a sua facilidade de instalação, sua documentação e os diversos drivers para inúmeras linguagens de programação. É orientado a documentos, escalável, livre de esquema, de alto desempenho e código aberto, escrito em C++ (CHODOROW, 2013).

Algumas de suas características são: orientação a documentos JSON/BSON; suporte a index; replicação e alta disponibilidade; auto-sharding; map/reduce flexível. Para um desenvolvedor habituado com o mundo relacional, o MondoDB poderá parecer estranho, porém com o tempo é possível notar que ele é mais simples que a maioria dos SGBDs. Seu grande diferencial é não possuir um *schema* e utilizar apenas duas dimensões: coluna e linha. Um conjunto de dados é chamado de coleção (diferente do modelo relacional, onde é chamado de tabela) (CHODOROW, 2013).

Dobre & Xhafa, (2014) apresentam soluções para “cidades inteligentes” onde todo o grande número de dados gerado todos os dias é centralizado e armazenado. Sua solução utiliza o conceito de Big Data aplicada com banco de dados MongoDB.

Papageorgas *et al.*, (2014) apresentam uma metodologia seguindo uma plataforma open source de uma rede de sensores wireless para monitoramento de casas em tempo real usando tecnologias web. O armazenamento dos dados é sugerido para ser realizado na nuvem em um banco de dados NoSQL.

3 MATERIAIS E MÉTODOS

3.1.1 Hardware utilizado

Os hardwares utilizados para o desenvolvimento desta pesquisa são os que seguem:

- Notebook Dell Inspiron 7520, com processador Intel Core i5, 2.5 GHz, 6 Gb de memória, disco rígido SATA 750 GB (5400 RPM), adaptador de rede Dell Wireless 1703 802.11 b/g/n.
- Smartphone Sony Xperia ZQ, Android OS Lollipop 5.0.2, processador quad-core 1.5 GHz Krait, chipset Qualcomm APQ064 Snapdragon S4 Pro com GPU Adreno 320, GPS A-GPS e bateria Li-Ion 2370 mAh.

3.1.2 Softwares utilizados

Os softwares utilizados para o desenvolvimento desta pesquisa são os que seguem:

- ***Android Developer Tools (ADT)***: incluindo plataforma Eclipse 4.2.1, JDT, CDT, DMF, GEF and WTP. O ADT é um projeto de código aberto disponível em http://developer.android.com. Software utilizado para o desenvolvido do SIG Móvel, devido aos recursos disponíveis no Eclipse para desenvolvimento de aplicações Android.
- **Eclipse Java EE versão Luna 4.4.1**: software utilizado para o desenvolvimento do servidor de aplicação (SaaS) na linguagem Java, devido a disponibilidade de documentação e amplo uso no mercado, com *Java SE Runtime Environment* versão 1.8.0_25.
- **Tomcat 8.0.15**: *software* utilizado como servidor para aplicação web embutido ao NetBeans IDE 8.0.2.
- **MongoDB 2.6 *Standard***: aplicação MongoDB NoSQL utilizada como base de dados no software servidor de aplicação (SaaS).
- **Robomongo 0.8.4**: *software* utilizado para visualização da base de dados NoSQL manipulada pelo servidor via MongoDB.

3.1.3 Área de estudo

A área de estudo dessa pesquisa foi a Fazenda Escola Capão-da-Onça (FESCON), administrada pela Universidade Estadual de Ponta Grossa (UEPG), no município de Ponta Grossa, PR, latitude 25° 05’ 35,7” S e longitude 50° 03’ 19” W, com altitude em média de

1041 metros. Esta região foi selecionada devido a FESCON possuir uma área experimental com cultivo de diversas culturas. Desta forma, o objeto de estudo desta pesquisa foi desenvolvido em um ambiente agrícola real. A localização da FESCON e a área de cultivo agrícola onde foram realizadas as coletas, pode ser observada na Figura 16.

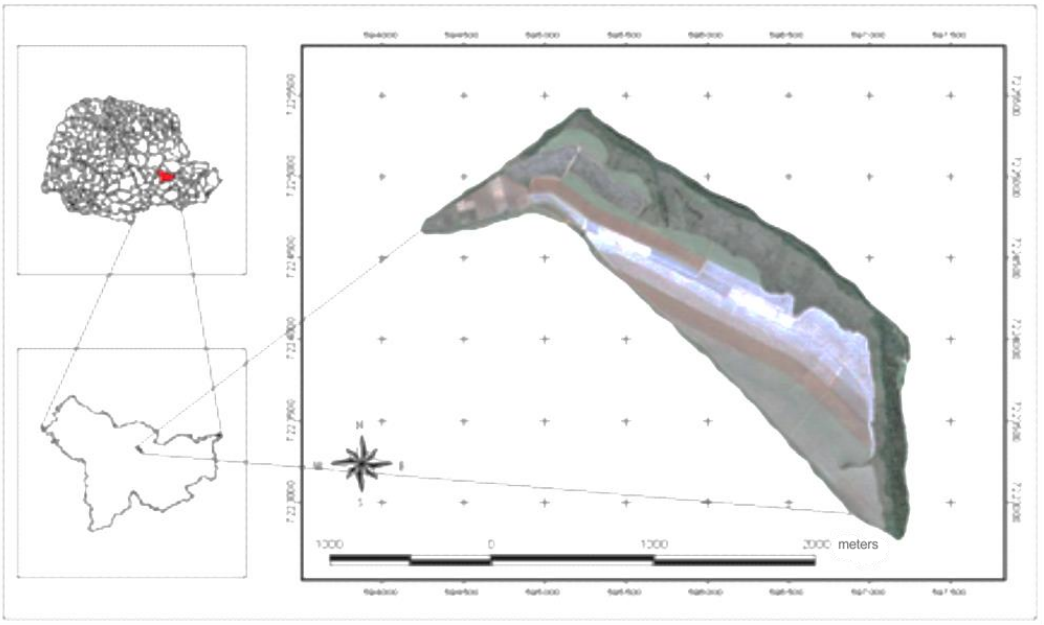


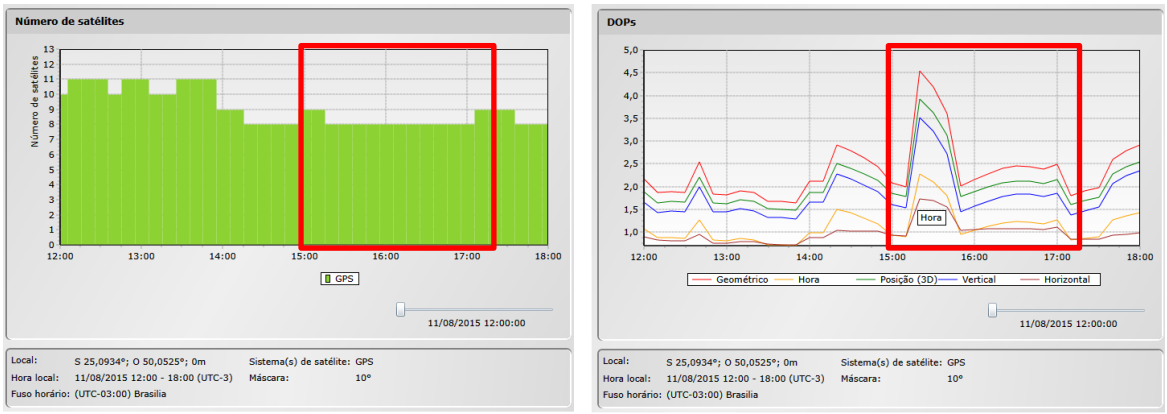
Figura 16 – Localização da Fazenda Escola Capão-da-Onça.

Fonte: Orlovski, 2013.

O planejamento das missões foram divididos em duas campanhas (janelas de observação) para realizar a coleta de dados geoespaciais, na FESCON, nos dias 11 de agosto de 2015 e no dia 13 de agosto de 2015. Como citado por Monico, (2007) na seção 2.2.2, é importante realizar o planejamento para missões, e realiza-las em horários que apresente valores DOP, entre o intervalo de 1 e 5, para incorrer em problemas de diluição de sinal, podendo prejudicar o levantamento, e obter o melhor aproveitado de sinal de um receptor GNSS.

Para as duas campanhas, o planejamento das missões foram realizado com auxílio da ferramenta *GNSS Planning* da *Spectra Precision*, disponível em http://www.spectraprecision.com/eng/support/gnss-planning.html. Possibilitando verificar o número de satélites e os valores DOPs disponíveis para todos os horários do dia e definir um plano de horários para ir a campo.

A campanha do dia 11 de agosto de 2015 foi realizada entre as 15:00 e as 17:15 horas, a Figura 17 (a) exibe o número de satélites disponíveis e a Figura 17 (b) exibe os valores DOPs referentes ao intervalo de coleta.



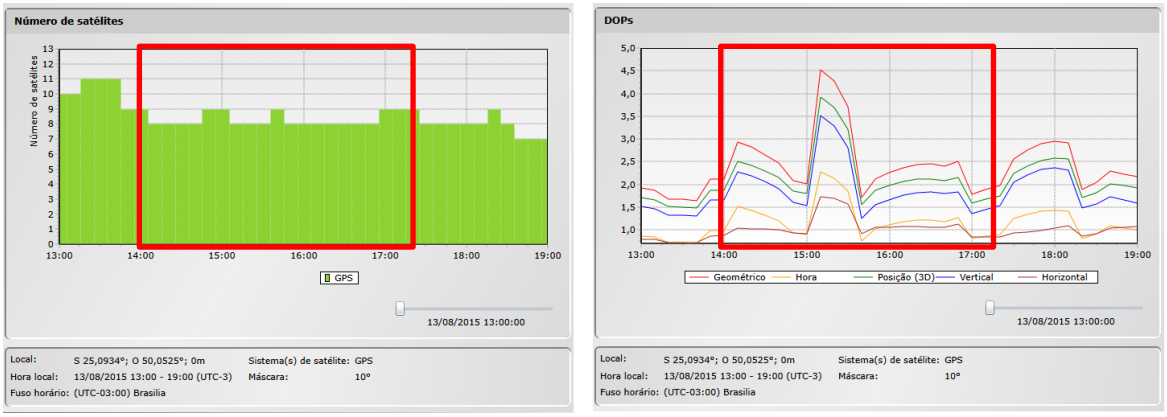
(a) Disponibilidade de satélites (b) Gráfico de DOPs

Figura 17 – Informações utilizadas no planejamento da missão do dia 11 de agosto de 2015.

Fonte: GNSS Planning (http://www.spectraprecision.com/eng/support/gnss-planning.html).

Na campanha do dia 11 de agosto de 2015 foram coletadas as seguintes áreas geoespacializadas: Nabo Forrageiro (polígono); Divisão de culturas – possível erosão (linha); Culturas diversas (polígono); Fruticultura (ponto); Estrada entre fruticultura e plantio (linha).

A campanha do dia 13 de agosto de 2015 foi realizada entre as 14:00 e as 17:15 horas; a Figura 18 (a) exibe o número de satélites disponíveis e a Figura 18 (b) exibe os valores DOPs referentes ao intervalo de coleta.



(a) Disponibilidade de satélites (b) Gráfico de DOPs

Figura 18 – Informações utilizadas no planejamento da missão do dia 13 de agosto de 2015.

Fonte: GNSS Planning (http://www.spectraprecision.com/eng/support/gnss-planning.html).

Na campanha do dia 13 de agosto de 2015 foram coletadas as áreas geoespacializadas com as seguintes descrições: Aveia (polígono); Cevada e Nabo Forrageiro (polígono); Postes (pontos); Erosão 01 (ponto); Erosão 02 (ponto); Área com falha 01 (ponto); Área com falha 02 (ponto); Madeiras de contenção (linha).

3.1.4 Modelagem do projeto

A modelagem do projeto é apresentada na Figura 19. Neste cenário se destaca o ambiente *cloud computing* ou computação na nuvem, onde os dados manipulados pelo SIG Móvel são sincronizados com o servidor de aplicação (SaaS). Kaloxylos et al., (2014) e Nikkila et al., (2010), como citados na Seção 2.1.6, desenvolveram trabalhos na área agrícola e utilizaram conceitos de *cloud computing* no desenvolvimento de suas soluções.

O servidor de aplicação, em sua arquitetura (Figura 19), utiliza comunicação via Web Service REST com seus clientes. O uso do *Web Service* REST em aplicações móveis pode ser caracterizado como comunicação RESTful. Segundo Mohamed e Wijessekera (2014) e Arroqui et al., (2012), o uso de REST é recomendado para aplicações móveis, pois consomem menos recursos e são mais eficientes.

Um diferencial para esta arquitetura (Figura 19) está no uso de um Banco de Dados NoSQL, orientado a documentos, no servidor de aplicação. Segundo Queiroz et al., (2013) o uso de bancos de dados NoSQL são promissores para aplicações em bancos de dados geográficos e, podem facilmente resolver problemas no domínio geoespacial, apresentado por soluções relacionais. Lee e Kang, (2015) citam que o uso de MongoDB no armazenamento de grandes volumes de dados geospaciais pode apresentar vantagens e oportunidades na área computacional.

Duro *et al.*, (2015) utilizaram NoSQL em sua solução móvel, justificando seu uso pela rapidez de armazenamento e disponibilidade de dados, mesmo com conexões de baixa velocidade 2g e 3g, disponibilizadas por operadoras de celular.

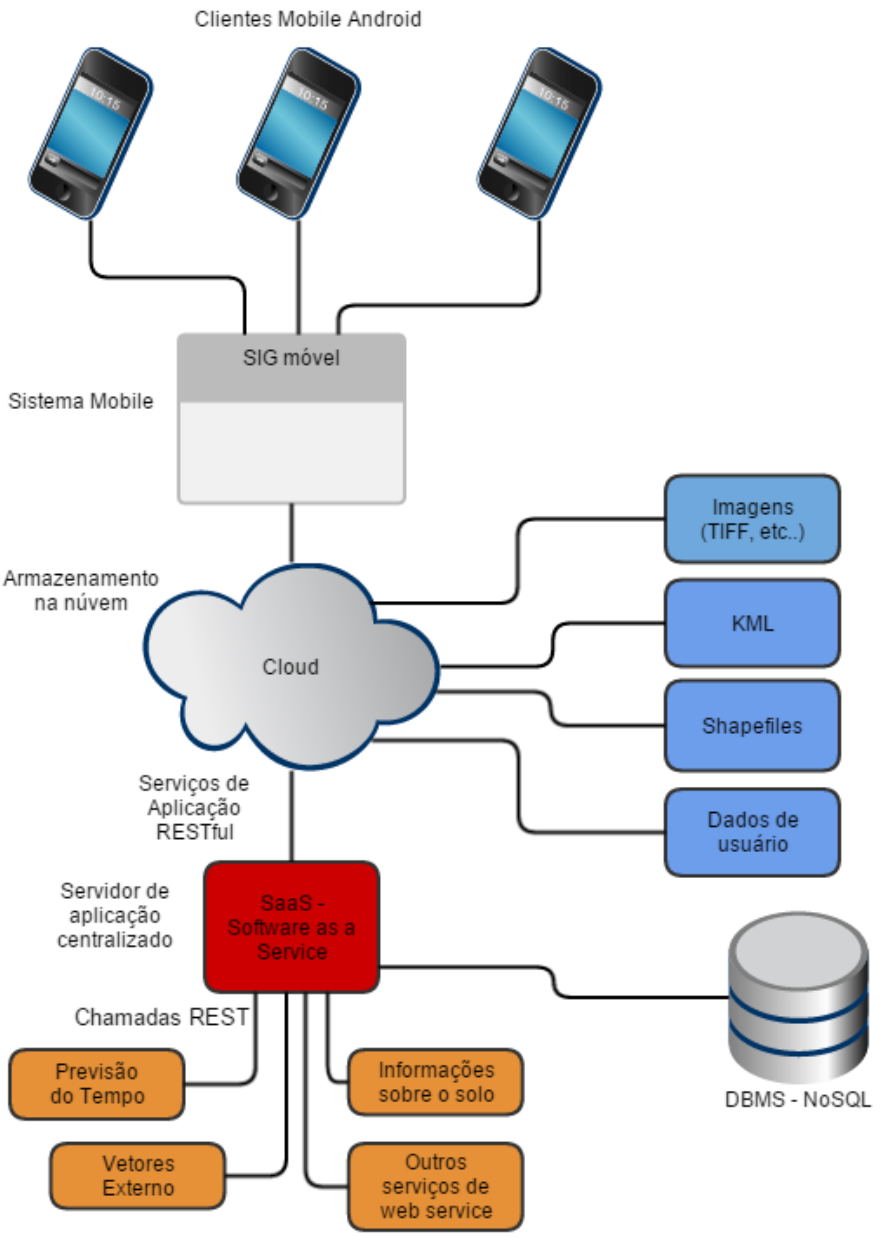


Figura 19 – Visão geral da modelagem do projeto.

Fonte: O autor.

Outra vantagem da arquitetura apresentada (Figura 19), é o uso de Web Service REST em aplicações web e móveis, com a comunicação cliente-servidor via arquivos JSON, facilmente manipulados por ambas tecnologias com várias bibliotecas disponíveis. A facilidade de acesso e disponibilidade dos mapas temáticos na nuvem, *Keyhole Markup Language* (KML) por exemplo, gerados pelo SaaS, à partir de coletas realizadas pelo SIG móvel, apresenta potencial de integração de seus resultados com outras informações em tempo real, dependendo apenas do SIG Móvel estar conectado a internet.

Na Figura 20 é apresentada uma visão geral da Arquitetura do SIG Móvel, onde seu nível hierárquico é dividido em camadas. A “Camada de usuário” apresenta o componente *Graphic User Interface* (GUI), referente aos *layouts* do software. A “Camada de aplicação” apresenta os componentes de Modelo, Controle e Visão, seguindo o modelo MVC de desenvolvimento de software. Nesta camada concentra-se a lógica desenvolvida dentro do software. O componente de visão é responsável pela lógica de comunicação entre a “camada de usuário” e a “camada de aplicação.

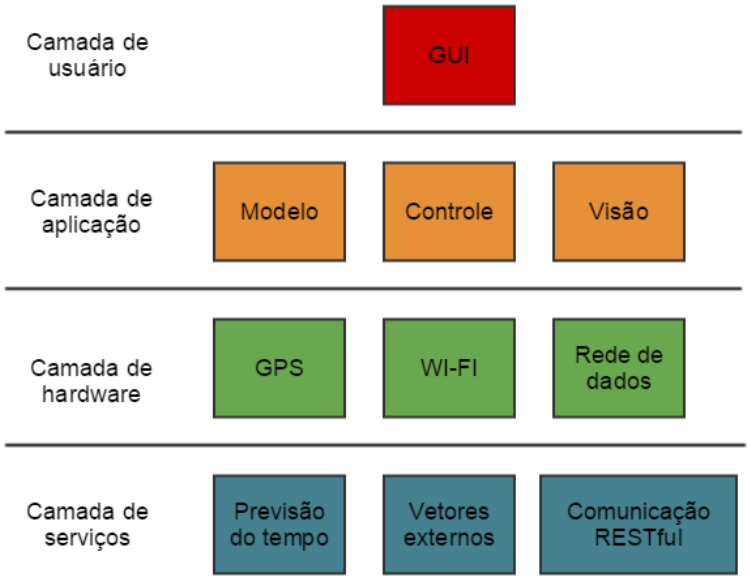


Figura 20 – Visão geral do SIG Móvel.

Fonte: O autor.

A “Camada de hardware” apresenta os componentes físicos consumidos pela aplicação, estes são o acesso à rede, WI-FI e Rede de dados, responsável pelo funcionamento do software em tempo real ou para garantir a sincronização com o servidor de aplicação. O componente GPS irá informar ao aplicativo sua localização georreferenciada (latitude, longitude, altitude e tempo da coordenada no Sistema Universal Transversa de Mercator [UTM]). Por meio dessas informações será possível coletar e armazenar pontos, linhas e polígonos de interesse. A “Camada de serviços” se refere a informações externas de interesse para a aplicação, adquiridas com comunicação via web service REST.

O fluxo de informações de entrada, processamento e saída entre o SIG Móvel e o servidor de aplicação é exibido na Figura 21. O Diagrama de fluxo de informação apresenta de forma sequencial o comportamento do SIG Móvel voltado para a coleta de dados

geoespaciais, comunicação e armazenamento de dados no servidor de aplicação. O diagrama é divido em: a) informações de entrada, que serão fornecidas pelo usuário; b) processamento destas informações, envolvendo técnicas de SIG, comunicação com web *serivce* REST e armazenamento em banco de dados NoSQL; e c) relatórios, ou seja, mapas temáticos gerados pelo *software* e que serão armazenados na nuvem e a visualização destes mapas em dispositivos móveis.

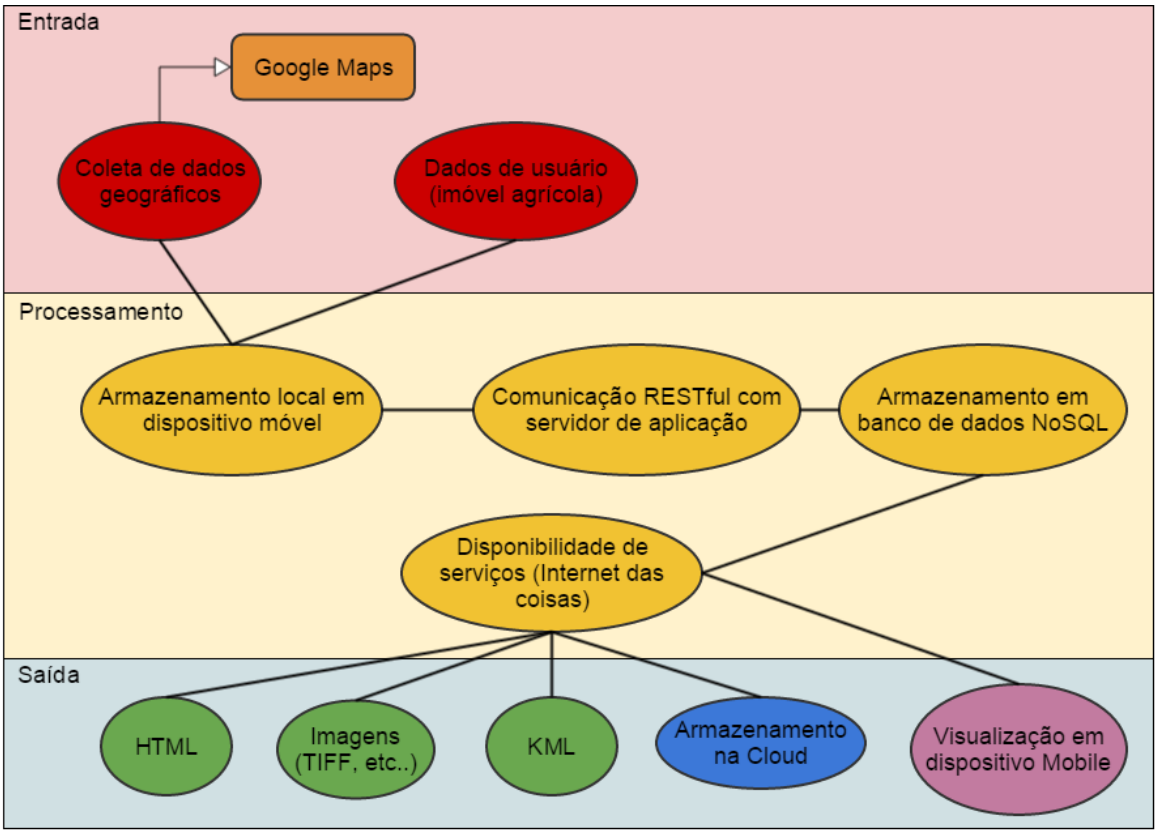


Figura 21 – Diagrama de fluxo de informação.

Fonte: O autor.

3.1.5 Bibliotecas e linguagens utilizadas

Servidor de aplicação (SaaS)

O servidor de aplicação foi desenvolvido na linguagem Java. O servidor de aplicação utilizou as bibliotecas *JAX-RS* e *Jersey* para a criação do *web service* e o *software Tomcat* como *WebContainer* para disponibilizar o servidor na internet. O padrão de desenvolvimento MVC foi seguido no desenvolvimento deste projeto.

O JAX-RS é uma especificação JSR 311 (https://jcp.org/en/jsr/detail?id=311) utilizada para o uso de *web services* com arquitetura *RESTful*. Em sua API são empregadas

anotações que simplificam a necessidade de configurações extras e auxiliam na produtividade da programação. A biblioteca que implementa a especificação JAX-RS é a *Jersey* (https://jersey.java.net/).

Para a conversão de objetos java em arquivos JSON e vice-versa, foi utilizada uma API do Google chamada *Google Gson* (https://code.google.com/p/google-gson/). Sendo assim, foram utilizados os seguintes jars para a implementação do Padrão *RESTful* no servidor de aplicação: *asm.jar; jersey-core; jersey-server; jersey-servlet; jsr311-api; jarsey-json; google-gson*.

A comunicação entre o servidor de aplicação e o sistema de banco de dados *MongoDB* é realizada com o auxílio da biblioteca *Mongo Java Driver* (http://docs.mongodb.org/ecosystem/drivers/java/). A versão da biblioteca utilizada neste projeto é a 3.0.3.

Para a geração de arquivos do tipo KML como vetores (pontos, linhas e polígonos) no servidor de aplicação, foi utilizada a biblioteca de auxílio JAK – Java API for KML, desenvolvida pela *Micromata Labs* e para a visualização dos vetores gerados foi utilizado o *Google Earth*.

SIG Móvel

O SIG Móvel foi desenvolvido em ambiente android, baseado na versão de sistema operacional *Lolipop 5.0.1*. Para a conversão de objetos java em arquivos JSON e vice-versa, assim como no servidor de aplicação, foi utilizada uma API *Google Gson*. As bibliotecas utilizadas foram: *Google Gson*; *Android support v4; Google play services.*

4 RESULTADOS E DISCUSSÕES

4.1 Softwares desenvolvidos

Esta seção apresenta o desenvolvimento e implementação do servidor de aplicação e do SIG Móvel.

4.1.1 Servidor de aplicação (SaaS - *Software as a Service*)

A Figura 22 apresenta os pacotes e classes desenvolvidas na implementação da arquitetura do Servidor de aplicação. O projeto seguiu o Padrão MVC, com destaque para a organização de classes referentes a: modelo; DAO (Data Access Object); comunicação REST; e utilitários do projeto e Gerador de KML.

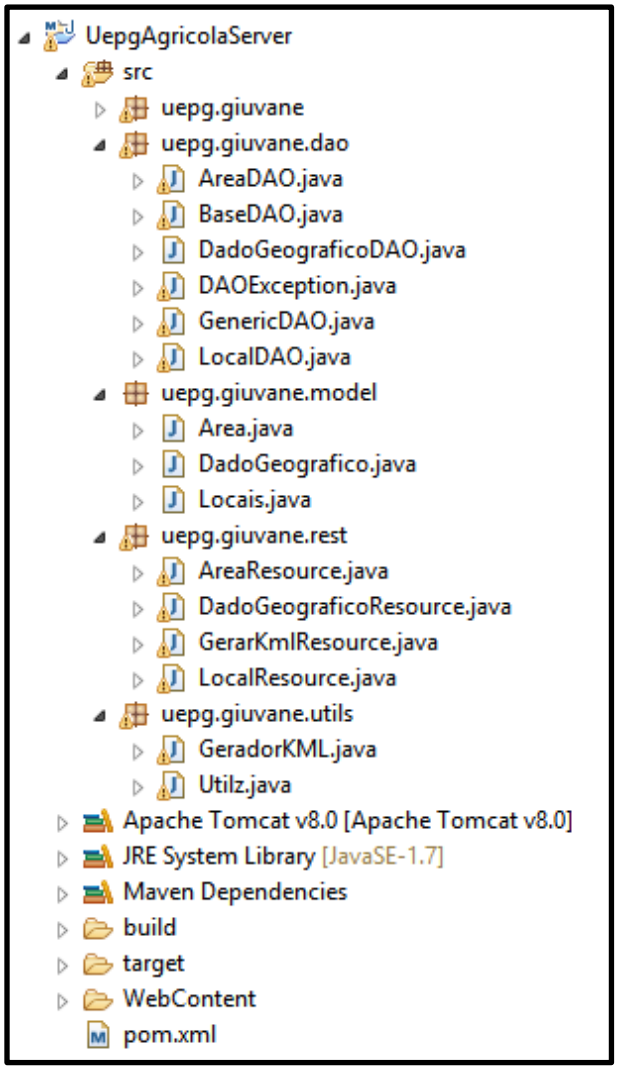


Figura 22 – Classes e pacotes pertencentes ao servidor de aplicação.

Fonte: O autor.

A Figura 23 apresenta o emprego de anotações da especificação JAX-RS, onde a anotação *@Path* define o contexto (nome concreto) do serviço para a classe e para cada método *GET.* Cada método realiza uma consulta de dados geográficos na base de dados e retornam o resultado para o aplicativo móvel. A anotação *@Produces* é responsável por definir qual é o tipo de arquivo que será utilizado na comunicação entre o servidor e o cliente, no caso deste exemplo e do restante do projeto, foi definida a comunicação por arquivos JSON.

```
import javax.ws.rs.Consumes;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.QueryParam;

import com.google.gson.Gson;
import com.google.gson.JsonArray;
import com.google.gson.JsonParser;

@Path("/dadogeografico")
public class DadoGeograficoResource {
    private DadoGeograficoDAO dadoGeograficoDAO;

    @GET
    @Path("/buscarTodos")
    @Produces("application/json")
    public List<DadoGeografico> selTodos() throws DAOException{
        dadoGeograficoDAO = new DadoGeograficoDAO();

        return dadoGeograficoDAO.buscarTodos();
    }

    @GET
    @Path("/buscarTodosGSON")
    @Produces("application/json")
    public String selTodosGSON() throws DAOException{
        dadoGeograficoDAO = new DadoGeograficoDAO();

        return new Gson().toJson(dadoGeograficoDAO.buscarTodos());
    }
```

Figura 23 – Uso de anotações para definir um serviço REST em uma classe e métodos GET em Java.

Fonte: O autor.

Como resultado do emprego da especificação JAX-RS, o contexto de comunicação definido pelos métodos *setTodos* e *selTodosGSON* (Figura 23) são os seguintes:

- http://localhost:8080/uepgagricolaserver/dadogeografico/buscarTodos;
- http://localhost:8080/uepgagricolaserver/dadogeografico/busarTodosGSON.

Para apresentar e elucidar os métodos de cadastro e exclusão de dados geográficos e as anotações utilizadas para modelagem da classe, a Figura 24 exibe a modelagem destes recursos.

Neste cenário a anotação *@POST* define que, nesta comunicação, o cliente que invocar este método ao servidor, obrigatoriamente passará um objeto do tipo *DadoGeografico* como parâmetro. A anotação *@Consumes* define qual é o formato do arquivo passado por parâmetro e, convertido na comunicação entre cliente e servidor.

```
@DELETE
@Path("/excluir")
@Produces("application/json")
public String excluir(@QueryParam("id") int id) throws DAOException{
    dadoGeograficoDAO = new DadoGeograficoDAO();

    return dadoGeograficoDAO.excluir(id);
}

@POST
@Path("/cadastrar")
@Produces("application/json")
@Consumes("application/json")
public String cadastrar(DadoGeografico dadoGeografico) throws DAOException {
    dadoGeograficoDAO = new DadoGeograficoDAO();

    return dadoGeograficoDAO.cadastrar(dadoGeografico);
}
```

Figura 24 – Uso de anotações para definir um serviço REST para métodos POST e DELETE em Java.

Fonte: O autor.

No Apêndice B apresenta-se o código fonte completo da classe DadoGeograficoResource.

4.1.2 Comunicação entre servidor de aplicação e base de dados NoSQL

Na Figura 25, é apresentado um trecho da classe *DadoGeograficoDAO,* com uso das bibliotecas *Mongo Java Driver* e *Gson. Esta classe* é responsável pela persistência dos dados de local, área e dado geográfico, na base de dados. Este processo é realizado após a comunicação entre servidor de aplicação e aplicativo móvel, quando o servidor de aplicação recupera o objeto *DadoGeografico* enviado pelo aplicativo móvel via chamada REST. O Apêndice C exibe o código fonte completo da classe *DadoGeograficoDAO*.

```
import com.google.gson.Gson;
import com.mongodb.BasicDBObject;
import com.mongodb.DBObject;

public class DadoGeograficoDAO {
    private GenericDAO genericDao;
    private DBObject localDB;

    public DadoGeograficoDAO() throws DAOException
    {
        genericDao = new GenericDAO("DadoGeografico");
    }

    public String cadastrar(DadoGeografico dadoGeografico) {
        String msg = "";

        try {
            localDB = new BasicDBObject("codDadoGeografico", dadoGeografico.getCodDadoGeografico()).
                    append("codArea", dadoGeografico.getCodArea()).
                    append("latitude", dadoGeografico.getLatitude()).
                    append("longitude", dadoGeografico.getLongitude()).
                    append("altitude", dadoGeografico.getAltitude()).
                    append("timeCoordenada", dadoGeografico.getTimeCoordenada()).
                    append("hora", dadoGeografico.getHora()).
                    append("sincronizado", dadoGeografico.getSincronizado());

            genericDao = new GenericDAO("DadoGeografico");

            this.genericDao.save(localDB);

            msg = "Dado Geográfico cadastrado com sucesso";
        } catch (DAOException ex) {
            msg = "Erro ao cadastrar Dado Geográfico";
            System.err
                    .println("Falha ao criar o objeto 'Dado Geográfico'." + ex);
            throw new ExceptionInInitializerError(ex);
        }

        return msg;
    }

    public DadoGeografico buscarPorId(String field, int id) throws DAOException
    {
        DadoGeografico dadoGeografico;
        genericDao = new GenericDAO("DadoGeografico");
        Gson gson = new Gson();
        dadoGeografico = gson.fromJson(Utilz.str2reader(genericDao.getById(field, id)), DadoGeografico.class);

        return dadoGeografico;
    }
}
```

Figura 25 – Classe DadoGeograficoDAO responsável pela persistência dos dados no banco de dados MongoDB.

Fonte: O autor.

As classes disponibilizadas pela biblioteca Mongo Java Driver para o controle das transações de banco de dados estão disponíveis em: *com.mongodb.**, como exibido nas *tags* "*import"* (Figura 26). A classe completa está disponível no Apêndice C.

4.1.3 SIG Móvel

As bibliotecas nativas do Android Lolipop 5.0.1, disponibilizam as classes para uso de GPS, WiFi, rede de dados e controle georreferenciado, tendo como base o *Google Maps*. A "Camada de aplicação" do aplicativo móvel, assim como o servidor de aplicação, visto na

Seção 4.1.1, segue o Padrão MVC em seu desenvolvimento. A Figura 26 apresenta os pacotes e classes criadas no projeto, com destaque para o pacote uepg.giuvane.appagricola.webserice, que contém as classes necessárias para consumir o Web Service REST criado no servidor.

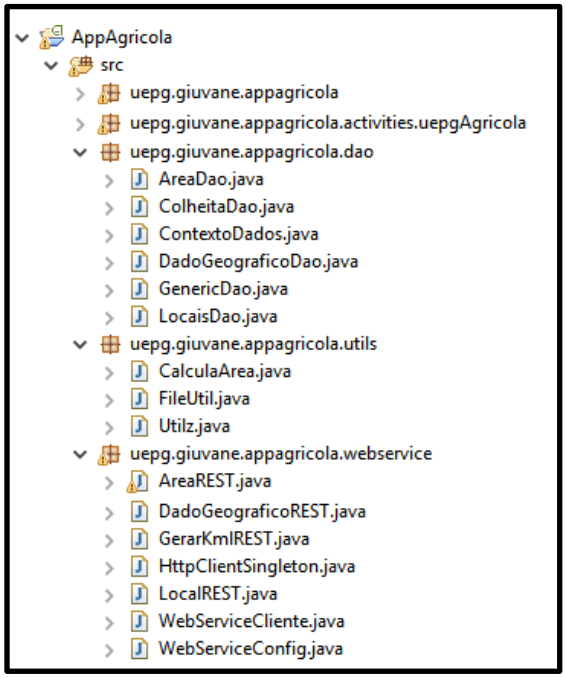
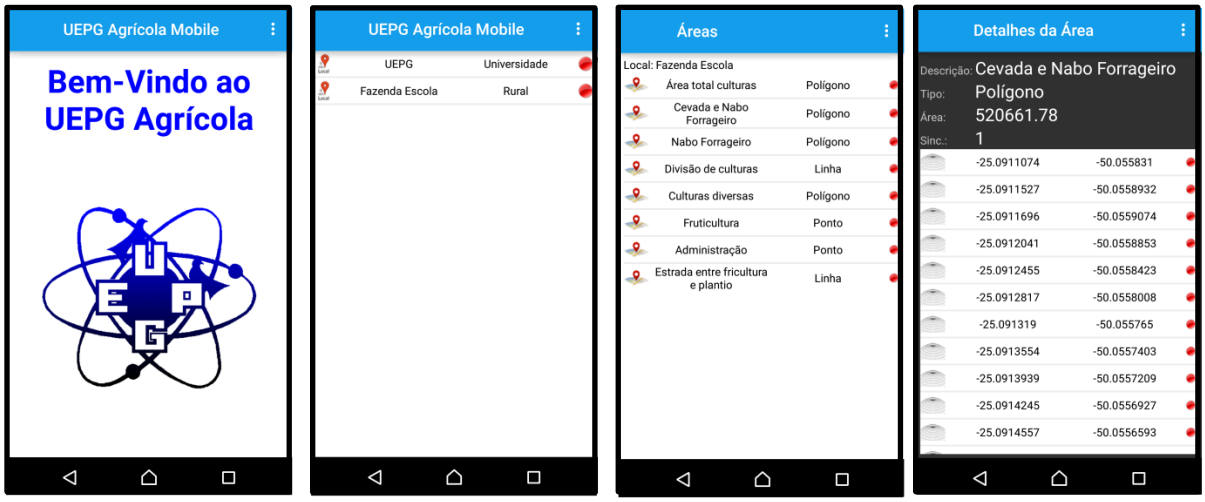


Figura 26 – Pacotes desenvolvidos no aplicativo móvel.

Fonte: O autor.

A Figura 27 discrimina a “Camada de usuário”, vista na Seção 3.1.4, onde foi apresentada a modelagem do projeto. Na Figura 27 (a), é apresentada a tela inicial do aplicativo móvel. Na Figura 27 (b), a listagem dos locais UEPG e Fazenda Escola cadastrados no aplicativo. São apresentadas na Figura 27 (c) as áreas cadastradas. Na Figura 27 (d) os dados geográficos georreferenciados coletados no talhão cevada e nabo forrageiro são apresentados.



(a) (b) (c) (d)

Figura 27 – Tela inicial do aplicativo móvel, com exemplo de Locais, Áreas e Dados Geográficos cadastrados no aplicativo móvel.

Fonte: O autor.

Os pontos em vermelho, vistos na Figura 28, ao lado das informações cadastradas no aplicativo, representam o status de sincronismo com o servidor de aplicação (vermelho = não sincronizado, verde = sincronizado e amarelo = em atualização). O sincronismo será abordado com mais detalhes na Seção 4.1.6.

O Apêndice A elucida a sequência de passos para o cadastro de um novo local e novas áreas, com coleta georreferenciada de dados geográficos, simulando uma operação do aplicativo móvel.

4.1.4 Comunicação entre Servidor de Aplicação e Aplicativo Móvel via Web Service REST

As transações da “Camada de serviços”, vista na Seção 3.1.4, entre servidor de aplicação e o aplicativo móvel via *web service,* são controladas pelo objetos “HttpClient” e “HttpParams”, disponíveis no pacote org.apache.http. Na Figura 28 é apresentado um trecho da programação da classe de configuração, responsável pelo controle das transações entre aplicativo móvel e servidor. A classe utiliza o padrão de projeto *Singleton* e ao ser instanciada, retorna um objeto do tipo “*DefaultHttpClient*”. O código fonte completo da classe *HttpClientSingleton* está disponível no Apêndice D.

```
 8 public class HttpClientSingleton {
 9
10     private static final int JSON_CONNECTION_TIMEOUT = 3000;
11     private static final int JSON_SOCKET_TIMEOUT = 5000;
12     private static HttpClientSingleton instance;
13     private HttpParams httpParameters ;
14     private DefaultHttpClient httpclient;
15
16     private void setTimeOut(HttpParams params){
17         HttpConnectionParams.setConnectionTimeout(params, JSON_CONNECTION_TIMEOUT);
18         HttpConnectionParams.setSoTimeout(params, JSON_SOCKET_TIMEOUT);
19     }
20
21     private HttpClientSingleton() {
22         httpParameters = new BasicHttpParams();
23         setTimeOut(httpParameters);
24         httpclient = new DefaultHttpClient(httpParameters);
25     }
26
27     public static DefaultHttpClient getHttpClientInstace(){
28         if(instance == null)
29             instance = new HttpClientSingleton();
30
31         return instance.httpclient;
32     }
33 }
```

Figura 28 – Classe de configuração para comunicação REST entre cliente e servidor, seguindo o padrão *Singleton*.

Fonte: O autor.

Os métodos “GET”, “POST”, “PUT” e “DELETE” são utilizados em uma interface com o *web service.* Esta interface será única para toda a aplicação, e são utilizados objetos do tipo “HttpGet”, “HttpPost”, “HttpPut” e “HttpDelete”. Na Figura 29, o método *get* da classe *WebServiceClient* é apresentado, o método recebe como parâmetro uma *string* referente a *url* responsável por uma comunicação, do tipo “GET”, com servidor de aplicação, a partir da instância gerada pelo *singleton (*Figura 28), é criado o objeto “*HttpEntity*”.

O objeto HttpEntity gera um *stream* que realiza a comunicação com servidor de aplicação, através do objeto InputStream. O código fonte completo da classe WebServiceClient está disponibilizado no Apêndice F.

```
24    public final String[] get(String url) {
25
26        String[] result = new String[2];
27        HttpGet httpget = new HttpGet(url);
28
29        try {
30            //doInBackground(httpget);
31            response = HttpClientSingleton.getHttpClientInstace().execute(httpget);
32            HttpEntity entity = response.getEntity();
33
34            if (entity != null) {
35                result[0] = String.valueOf(response.getStatusLine().getStatusCode());
36                InputStream instream = entity.getContent();
37                result[1] = toString(instream);
38                instream.close();
39                Log.i("get", "Result from post JsonPost : " + result[0] + " : " + result[1]);
40            }
41        } catch (Exception e) {
42            Log.e("UepgAgricola", "Falha ao acessar Web service", e);
43            result[0] = "0";
44            result[1] = "Falha de rede!";
45        }
46        return result;
47    }
```

Figura 29 – Método get, responsável pela comunicação REST com o método “GET” no servidor de aplicação.

Fonte: O autor.

As classes REST utilizam os métodos *get*, *post* e *delete* da classe *WebServiceClient*, A Figura 30 apresenta o método *getDadoGeografico* da classe *DadoGeograficoREST*, onde a mesma faz uso do método *get* (Figura 29). No método *getDadoGeografico* é invocado o serviço REST pelo aplicativo móvel, este método utiliza a *url* configurada no servidor de aplicação (Figura 23 e na Figura 24). Nesta classe a biblioteca *Google Gson* é utilizada para a conversão da resposta do servidor de aplicação, recebida como um arquivo *Json* e convertido em objeto *DadoGeografico*. O código fonte completo da classe *DadoGeograficoREST* é apresentado no Apêndice E.

```
 8  import com.google.gson.Gson;
 9  import com.google.gson.JsonArray;
10  import com.google.gson.JsonParser;
11
12  public class DadoGeograficoREST {
13      public DadoGeografico getDadoGeografico(int id) throws Exception {
14          String[] resposta = new WebServiceCliente().get(WebServiceConfig.URL_WS + "dadogeografico/buscar/?id=" + id);
15
16          if (resposta[0].equals("200")) {
17              Gson gson = new Gson();
18              DadoGeografico dadoGeografico = gson.fromJson(resposta[1], DadoGeografico.class);
19          return dadoGeografico;
20          } else {
21              throw new Exception(resposta[1]);
22          }
23      }
```

Figura 30 – Método *getDadoGeografico*, onde é definida a *url* chamada para o servidor de aplicação.

Fonte: O autor.

4.1.5 Captura de dados geográficos via Google Maps

Os tipos de dados geográficos georrefenciados que podem ser coletados pelo aplicativo móvel são: pontos, linhas e polígonos. A coleta pode ser realizada de duas formas: o usuário pode coletar de forma manual, com toques diretamente no mapa, ou pode optar por realizar a coleta de forma automatizada, utilizando funções do GPS em deslocamento (Figura 31).

O campo “Erro min (mt)” é aplicado às coletas automatizadas, com uso do GPS, onde este erro mínimo é definido (em metros) para descartar pontos de coleta, onde o ponto atual extrapole este valor definido. Por exemplo, se o usuário definir o erro mínimo em 50 metros, e iniciar a coleta, a cada atualização de ponto o aplicativo verifica se o ponto atual e o ponto anterior estão dentro desta distância de erro mínimo definida. Caso a distância entre os pontos extrapole este valor, a coleta do ponto atual é descartado e o aplicativo aguarda a próxima atualização de ponto e repete o processo. Utilizando este método é possível controlar possíveis erros de sincronismo do GPS e manter uma maior acurácia da coleta, visto que o GPS embarcado em smartphones ainda é limitado e pode apresentar oscilações em seu sinal.

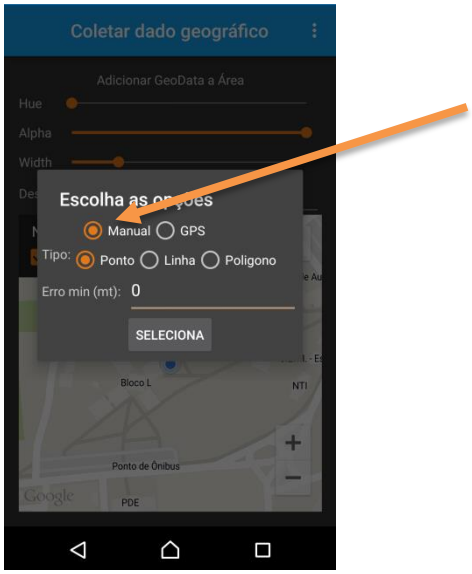


Figura 31 – Opções disponíveis para coleta de dados geográficos no aplicativo móvel.

Fonte: O autor.

A configuração para sincronismo e taxa de atualização, da “Camada de hardware”, aplicada ao GPS para coleta de dados geográficos são realizadas pela classe *LocationRequest*. Os métodos *setInterval* e *setFastestInterval* representam o tempo em milissegundos em que o GPS sincronizará sua taxa de atualização de sinal. O método

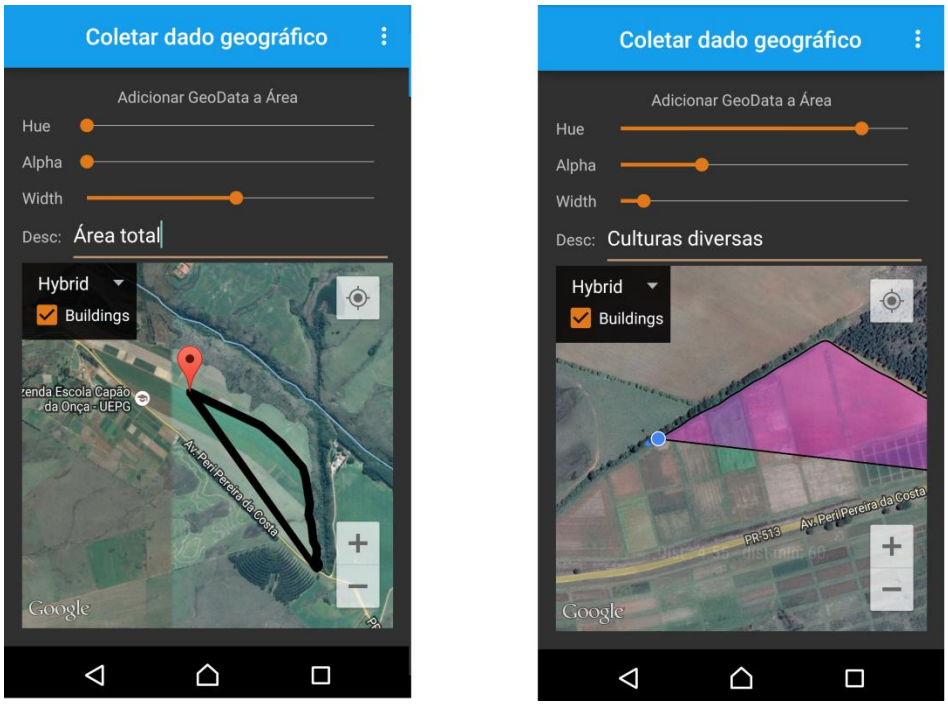
setSmallestDisplacement define que sua atualização será validada se a distância for maior que 1 metro e o método *setPriority* define qual será a prioridade de trabalho do GPS, onde a opção *LocationResquest.PRIORITY_HIGH_ACCUCARY* indica que o GPS irá utilizar a rede móvel e a rede WiFi, se disponíveis, para garantir uma maior precisão (Figura 32).

```
private LocationRequest REQUEST = LocationRequest.create()
        .setInterval(3000)          // 3 segundos
        .setFastestInterval(16)     // 16ms = 60fps
        .setSmallestDisplacement(1) // Considerar apenas deslocamentos de 1 metro
        .setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);
```

Figura 32 – Configurações aplicada ao GPS no aplicativo móvel.

Fonte: O autor.

A Figura 33 (a) apresenta o aplicativo realizando uma coleta em modo manual. A Figura 33 (b) apresenta o exemplo de uma coleta sendo realizada via GPS. Os componentes *SeekBars* intitulados*: Hue*, *Alpha* e *Width* permitem alterar a cor da linha ou do polígono, suavizar e alterar o tamanho da linha ou o tamanho da borda do polígono. Na Figura 33 (a) é possível notar um tamanho alto de borda aplicado a linha e nada suavizado. Na Figura 33 (b), a borda aplicada é fina e apresenta suavização mediana da área sendo coletada.



(a) Coleta manual (b) Coleta GPS

Figura 33 – Coleta de dados manual x coleta de dados GPS.

Fonte: O autor.

A Figura 34 apresenta o tipo de classe utilizada na configuração do mapa disponibilizado para coleta de dados geográficos, o mapa é representado pela variável *map*, que é do tipo *SupportMapFragment*, e uma variável de configuração do tipo *UiSettings* (representada pela variável *uiSettings*) é adicionada ao mapa para compor mais configurações. As seguintes configurações são aplicadas ao mapa na "Camada de hardware":

- Habilitar o uso da localização de GPS (*map.setMyLocationEnabled(true)*);
- Utilizar métodos para monitoramento de cliques no mapa (*map.setOnMapClickListener* e *map.setOnMapLongClickListener)*;
- Controlar zoom (*uiSettings.setZoomControlsEnabled(true)*);
- Habilitar bússula (*uiSettings.setCompassEnabled(true)*);
- Habilitar acelerômetro (*uiSettings.setRotateGesturesEnabled(true)*).

```
private void setUpMapIfNeeded() {
    // Verifica se o mapa necessita ser instanciado
    if (map == null) {
        // Obtem o mapa do SupportMapFragment.
        map = ((SupportMapFragment) getSupportFragmentManager().findFragmentById(R.id.mapArea))
                .getMap();

        // Habilita localização do GPS e métodos para controle de cliques no mapa
        map.setMyLocationEnabled(true);
        map.setOnMapClickListener(this);
        map.setOnMapLongClickListener(this);

        // Habilita funções no mapa
        uiSettings = map.getUiSettings();
        uiSettings.setZoomControlsEnabled(true);
        uiSettings.setCompassEnabled(true);
        uiSettings.setRotateGesturesEnabled(true);

        mColorBar.setOnSeekBarChangeListener(this);
        mAlphaBar.setOnSeekBarChangeListener(this);
        mWidthBar.setOnSeekBarChangeListener(this);
        // Check if we were successful in obtaining the map.
        if (map != null) {

        }
    }
}
```

Figura 34 – Configuração aplicada ao mapa (R.id.mapArea).

Fonte: O autor.

A Figura 35 apresenta a lógica de programação, aplicada ao mapa, para inserção de polígono. As classes responsáveis pela criação e controle de pontos, linhas e polígonos disponíveis pelo Google estão nos pacotes *com.google.android.gms.maps.model*, e as classes utilizadas são as: *Marker, MarkerOptions, Polygon, PolygonOptions, Polyline e PolylineOptions*. Um exemplo deste trecho de programação aplicado, foi visto na Figura 33 (b), onde foi aplicado ao mapa um polígono baseado em pontos coletados via localização

atual do GPS, controlado pela variável *frameworkSystemLocation*. A variável *arg0* (do tipo *Location*), disponível no pacote *com.android.gms.location* é atualizada cada movimento detectado pelo GPS, é possível visualizar a interação dos componentes *SeekBars* ao polígono inserido no mapa.

```
// Controla os polígonos adicionados ao mapa baseado na posição atual do GPS
poligonoOptions = new PolygonOptions();
pontos.add(frameworkSystemLocation = new LatLng(arg0.getLatitude(), arg0.getLongitude()));

poligonoOptions.addAll(pontos).geodesic(true);

// Calcula a área total do polígono
try {
    areaTotal = CalculaArea.calculateAreaOfGPSPolygonOnEarthInSquareMeters(pontos);
} catch (ParseException e) {
    // TODO Auto-generated catch block
    e.printStackTrace();
}

// Controla a cor do layer baseado nos SeekBar
int fillColor = Color.HSVToColor(
        mAlphaBar.getProgress(), new float[] {mColorBar.getProgress(), 1, 1});
poligono = map.addPolygon(poligonoOptions
        .strokeWidth(mWidthBar.getProgress())
        .strokeColor(Color.BLACK)
        .fillColor(fillColor));
```

Figura 35 – Lógica aplicada ao mapa para inserção de polígono.

Fonte: O autor.

O Apêndice G apresenta o documento eXtensible Markup Language (XML) com o *layout* desenvolvido na classe de coleta de dados geográficos. As seguintes interfaces foram implementadas na classe (ou *Activity*) responsável pela coleta de dados geográfico, manipulação de *SeekBars*, controle do sinal de GPS e controlar a interação de toques no mapa:

- *android.widge.Seekbar.OnSeekBarChangeListener*
- *com.google.android.gms.common.GooglePlayServicesClient.ConnectionCallbacks*
- *com.google.android.gms.location.LocationListener*
- *com.google.android.gms.maps.GoogleMap.OnMyLocationButtonClickListener*
- *com.google.android.gms.maps.GoogleMap.OnMapClickListener*
- *com.google.android.gms.maps.GoogleMap.OnMapLongClickListener*
- *android.widget.AdapterView.OnItemSelectedListener*

A visualização pós-cadastro de dados geográficos, no aplicativo móvel, é apresentada na Figura 36, onde diversos cadastros de pontos, linhas e polígonos foram realizados na FESCON. Todas as áreas cadastradas utilizaram o aplicativo em modo GPS.

Na Figura 37 (a) foram cadastrados polígonos de aveia, nabo forrageiro e culturas diversas, linhas de erosão longa e divisão de culturas e pontos de erosão. Os marcadores em vermelho, vinculados a linhas e polígonos, indicam o ponto inicial da coleta em cada área. A Figura 37 (b) apresenta um talhão de aveia, com um marcador indicando o ponto inicial da coleta.

(a) (b)

Figura 36 – Visualização de áreas cadastradas ao local Fazenda Escola no aplicativo móvel.

Fonte: O autor.

4.1.6 Sincronismo de Dados entre Aplicativo Móvel e Servidor de aplicação

Os dados cadastrados no aplicativo são automaticamente sincronizados com o servidor de aplicação quando o mesmo tiver acesso à internet. Na interface gráfica, o *status* (sincronizado, não-sincronizado e em atualização), é apresentado com um ícone, vermelho para não-sincronizado e verde para sincronizado (Figuras 27 e 37). O controle é realizado por uma *thread,* trabalhando em *background. Su*a lógica de programação é apresentada na Figura 38.

Figura 37 – Exemplo de locais cadastrados com *status* de sincronizado e não-sincronizado.

Fonte: O autor.

```
160     // Verifica se locais cadastrados estão sincronizados
161     private class SincronizaLocaisTask extends AsyncTask<Object, Object, List<Locais>>
162     {
163        LocalREST localREST = new LocalREST();
164
165        @Override
166        protected List<Locais> doInBackground(Object... params)
167        {
168            for (Locais l : locais)
169            {
170                if (l.getSincronizado().equals(1))
171                {
172
173                    try {
174                        l.setSincronizado(2);
175                        localREST.cadastrarLocal(l);
176                        localDao.update(l);
177                  } catch (Exception e) {
178                      // TODO Auto-generated catch block
179                      e.printStackTrace();
180                  }
181                }
182                if (l.getSincronizado().equals(3))
183                {
184
185                    try {
186                        l.setSincronizado(2);
187                        localREST.atualizarLocal(l.getCodLocal(), l);
188                        localDao.update(l);
189                  } catch (Exception e) {
190                      // TODO Auto-generated catch block
191                      e.printStackTrace();
192                  }
193                }
194            }
195
196              List<Locais> locaisREST = new ArrayList<Locais>();
197            try {
198                locaisREST = localREST.getListaLocais();
199            } catch (Exception e) {
200                // TODO Auto-generated catch block
201                e.printStackTrace();
202            }
203
204             // Retorna a lista de locais atualizados
205             return locaisREST;
206        } // end method doInBackground
207
208        // Atualiza a ListView com os locais cadastrados
209        @Override
210        protected void onPostExecute(List<Locais> result)
211        {
212           createListView();
213        }
214     }
215  }
```

Figura 38 – Métodos responsáveis pelo sincronismo dos locais entre o aplicativo móvel e o servidor de aplicação.

Fonte: O autor.

4.1.7 Geração de arquivos KML a partir dos dados geográficos coletados

Para a tarefa de conversão de áreas já cadastradas no banco de dados, com o auxílio do aplicativo móvel, em arquivos KML na nuvem, foram criadas duas classes (Figura 39). A primeira classe (GerarKMLResource.java) é responsável por receber a solicitação do aplicativo móvel, através de uma chamada REST e direcionar a criação da área de interesse em arquivo KML. A segunda classe (GeradorKML.java) cria objetos baseados nas classes disponibilizadas pela biblioteca JAK e gera o arquivo KML a partir destes objetos.

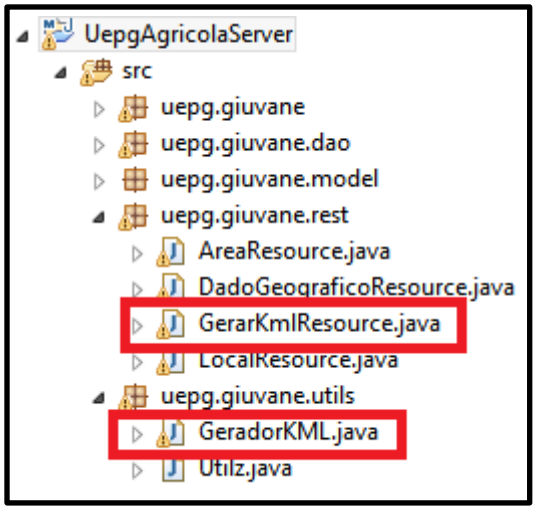


Figura 39 – Classes no servidor de aplicação responsáveis pela geração de arquivos KML.

Fonte: O autor.

A Figura 40 apresenta ó código no aplicativo móvel responsável pela chamada REST ao servidor de aplicação para criação de um arquivo KML, referente a uma área já cadastrada em seu banco de dados e sincronizada com o servidor de aplicação.

```
30    public String GerarKmlArea(Area area) throws Exception {
31
32        Gson gson = new Gson();
33        String areaJSON = gson.toJson(area);
34
35        String[] resposta = new WebServiceCliente().post(WebServiceConfig.URL_WS + "gerarkml/gerararea", areaJSON);
36        if (resposta[0].equals("200")) {
37            return resposta[1];
38        } else {
39            throw new Exception(resposta[1]);
40        }
41    }
```

Figura 40 – Método responsável pela chamada REST do aplicativo para o servidor de aplicação para geração de um arquivo KML baseado em uma área.

Fonte: O autor.

Como resultado da Figura 40, o contexto de comunicação definido pelo método GerarKmlArea, é o seguinte:

- http://localhost:8080/uepgagricolaserver/gerarkml/gerararea;

Baseado na área fornecida pelo aplicativo, via comunicação REST, o servidor de aplicação processa esta chamada e gera um arquivo georreferenciado em formato KML. A Figura 41 apresenta um trecho do código responsável pela criação do arquivo KML, de uma área do tipo linha. A geração do KML utiliza classes disponibilizadas pela biblioteca JAK. O código fonte completo pode ser visualizado no Anexo H.

```
137         else if (area.getTipo().equals(2))
138         {
139             Kml kml = KmlFactory.createKml();
140
141             Placemark placemark = KmlFactory.createPlacemark();
142             placemark.setName(area.getDescricao());
143             placemark.setDescription(area.getDescricao());
144
145             Style style = new Style();
146
147             LineStyle linhaEstilo = KmlFactory.createLineStyle();
148             linhaEstilo.setWidth(area.getWidth());
149             linhaEstilo.setColor(area.getHue().toString());
150
151             style.setLineStyle(linhaEstilo);
152
153             LineString linha = KmlFactory.createLineString();
154
155             for (DadoGeografico dg : dadosGeograficos)
156             {
157                 Coordinate coord = new Coordinate(dg.getLongitude(), dg.getLatitude());
158
159                 linha.getCoordinates().add(coord);
160
161             }
162
163             linha.addToLineStringObjectExtension(style);
164
165             linha.setExtrude(true);
166             linha.setAltitudeMode(AltitudeMode.CLAMP_TO_GROUND);
167
168             placemark.setGeometry(linha);
169
170             // Adiciona o ponto, a linha ou o polígono ao kml
171             kml.setFeature(placemark);
172         }
```

Figura 41 – Exemplo de criação de um KML baseado em uma área do tipo linha.

Fonte: O autor.

Os dados geográficos, coletados com o aplicativo móvel e sincronizados com o servidor de aplicação, ao serem submetidos ao processo de geração de KML, são convertidos em objetos do tipo Coordinate e adicionados ao objeto LineString (Figura 41). A cor, a textura e a suavidade aplicadas a área, na hora da coleta com o aplicativo móvel, são mantidos na geração do arquivo KML.

A Figura 42 apresenta a interface do *software Google Maps*, onde é exibido o resultado final do processo de aquisição, processamento e geração de arquivo KML. O polígono coletado corresponde a área da UEPG. É valido lembrar que arquivos KMLs podem ser manipulados por vários softwares SIG desktop e são amplamente utilizados na área de geoprocessamento.

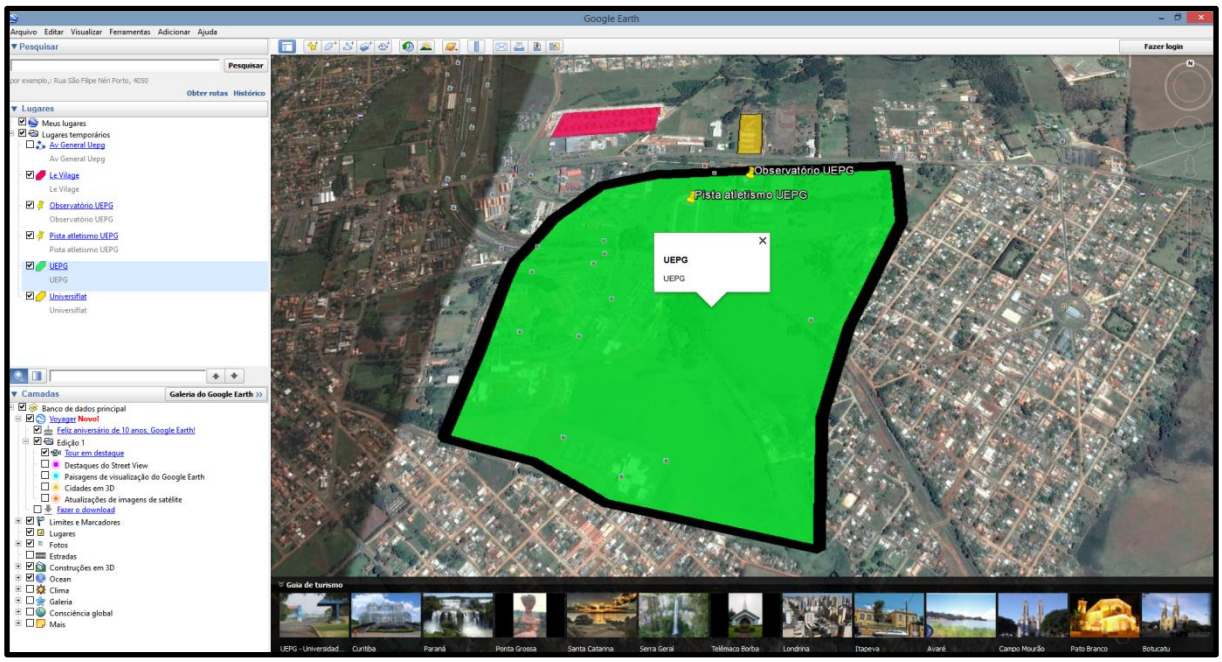


Figura 42 – Mapas temáticos georreferenciados em formato KML, gerados a partir de áreas coletadas pelo aplicativo móvel.

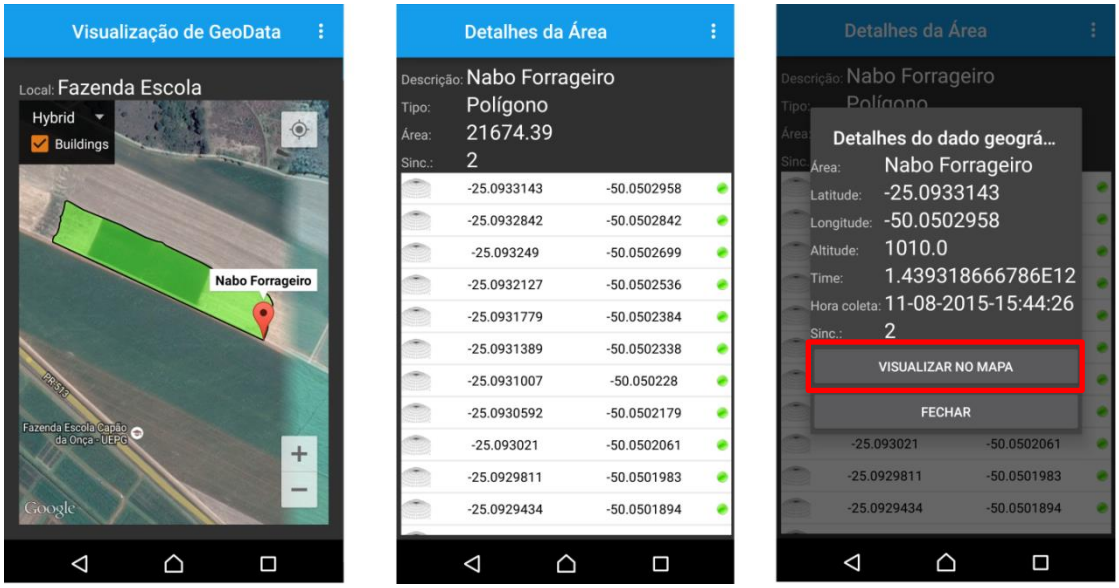
Fonte: O autor.

4.2 Campanhas realizadas no dia 11/08/2015

Na missão planejada para o dia 11 de agosto de 2015 foram coletados diversos talhões, dentre eles foi selecionado um para representar este processo. O talhão selecionado foi uma área com cultivo de Nabo Forrageiro. A coleta teve início às 15:44:26 horas e término as 15:51:02 horas.

A Figura 43 (a) apresenta o resultado final pela perspectiva do aplicativo. O resultado final apresenta um marcador no primeiro ponto coletado, uma lista de detalhes da área, exibindo todos os pontos pertencentes a área coletada e os detalhes do primeiro ponto coletado, apresentando latitude, longitude, altitude, tempo da coordenada e a hora exata em que foi realizada a coleta e com a opção de visualização apenas do ponto selecionado na lista.

A Figura 43 (b) apresenta uma foto, *in loco*, da cultura Nabo Forrageiro onde foi realizada a coleta, e a Figura 43 (c) apresenta o resultado final, a visualização do arquivo KML gerado desta área com auxílio do software Google Earth.



(a) Resultados da coleta no aplicativo móvel

(b) Foto do Nabo Forrageiro coletado

(c) KML visualizado no Google Earth

Figura 43 – Talhão de Nabo Forrageiro coletado com o aplicativo móvel.

Fonte: O autor.

Entre um talhão de Nabo Forrageiro e um talhão de Aveia, foi identificado uma precipitação, um possível resultado de erosão. Foi então realizada a coleta desta área, sendo representada geograficamente em formato de linha. O recurso "*Width*", apresentado na Seção 3.1.11, foi aplicado a linha coletada e utilizado para representar a largura real desta precipitação.

A Figura 44 (a) apresenta a linha já coletada, representando a erosão e o recurso "*Width*" aplicado a representação da linha. A Figura 44 (b) apresenta uma foto, *in loco*, da precipitação e a Figura 44 (c) exibe o resultado desta coleta. O arquivo KML gerado a partir desta coleta, e sendo visualizado no software *Google Earth*.

(a) Coleta no app (b) Imagem da área

(c) KML gerado visualizado no Google Earth

Figura 44 – Linha de divisão de culturas, possível erosão.

Fonte: O autor.

4.3 Campanhas realizadas no dia 13/08/2015

O maior talhão coletado foi nesta campanha, onde teve início as 15:10:04 horas e termino as 16:07:00 horas. Neste talhão estavam sendo cultivadas as culturas de Cevada e

Nabo Forrageiro. A Figura 45 apresenta a área total coletada pelo aplicativo móvel, com um marcador indicando o ponto inicial da coleta.

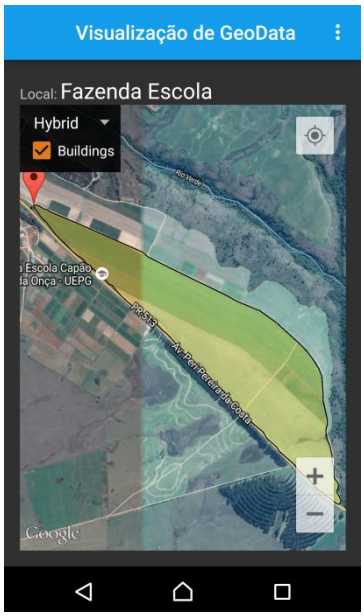


Figura 45 – Visualização pelo aplicativo do talhão de Cevada e Nabo Forrageiro coletado.

Fonte: O autor.

Ao longo do talhão de Cevada e Nabo Forrageiro também foram coletados pontos de interesse e a visão destes pontos foi registrada com imagens fotográficas (Figura 46). Foram coletados pontos onde possivelmente seria interessante realizar uma análise posterior, confrontando imagem real e ponto georreferenciado de onde a imagem foi tirada. Estes pontos foram classificados como postes, erosão e uma região com mata nativa entre duas culturas, uma área falha com erosão. Com informações extras sobre o talhão e recursos disponibilizados por softwares SIGs seria possível confrontar estes pontos e buscar explicações para estes fenômenos.

A ideia de coletas de áreas de interesse com fotografias é baseada no aplicativo Geofoto de Mesas-Carrascosa et al., (2012), onde são coletadas fotografias georreferenciadas de pontos de interesse.

Figura 46 – Talhão de Cevada e Nabo Forrageiro com pontos de interesse e suas respectivas fotografias.

Fonte: O autor.

Ao final da área experimental, onde são cultivadas diversas culturas na FESCON, encontra-se uma precipitação, onde foi implantado uma espécie de muro de contenção com toras de madeira. Para representar o muro de contenção foi realizada de uma linha, utilizando o recurso “Width” para representar a largura do muro. Além da linha foram coletados alguns pontos de interesse, com registros fotográficos, como pode ser visto na Figura 47.

Figura 47 – Muro de contenção com pontos de interesse e suas respectivas fotografias.

Fonte: O autor

4.4 Resultado Final e Mapa Gerado das Coletas

Após as coletas (dias 11 e 13 de agosto de 2015), realizadas na FESCON, e sincronização dos dados com o servidor de aplicação, foram gerados, em arquivos KML, todos os vetores coletados. O resultado do georreferenciamento de todos os pontos, linhas e talhões coletados é apresentado na Figura 48, em forma de mapa temático, utilizando o *software* Google Earth para esta tarefa.

Figura 48 – Resultado final – KML de todas áreas coletadas na FESCON apresentadas no *software* Google Earth.

Fonte: O autor.

O Quadro 1 apresenta as linhas e polígonos coletadas no dia 11/08/2015 e 13/08/2015, explicitando o tipo de área geográfica, a quantidade de pontos que compõem a área, a hora inicial e hora final da coleta,

Quadro 1 – Polígonos e linhas coletadas.

Area	Quantidade de pontos	Tipo	Cor	Data	Hora inicio	Hora fim
Nabo Forrageiro	131	Polígono	Verde	11/08/2015	15:44:26	15:51:02
Divisão culturas	13	Linha	Ciano	11/08/2015	15:52:58	15:53:28
Culturas diversas	320	Polígono	Rosa	11/08/2015	16:13:51	16:29:47
Aveia	289	Polígono	Azul	13/08/2015	14:55:16	15:07:55
Cevada e Nabo Forrageiro	1069	Polígono	Amarelo	13/08/2015	15:10:04	16:07:00
Madeiras de contenção	35	Linha	Verde	13/08/2015	16:59:43	17:01:25

A cor disponibilizada no Quadro 1 corresponde as cores utilizadas como configuração e podem ser visualizadas na Figura 49. Todas os percursos das áreas coletadas foram realizadas por caminhadas.

Utilizando o planejamento para missões, considerando horários apropriados para as mesmas (DOPs entre 1 e 5), baseado nos gráficos disponibilizados pela *Spectra Precision*, em nenhuma das áreas coletadas o aplicativo móvel precisou utilizar o recurso de “Erro mínimo” implementado, onde o software desconsidera valores que ficavam fora do intervalo em caso de oscilação do GPS. Embora não esteja documentado, em alguns testes realizados no campus da Universidade Estadual de Ponta Grossa, em horários onde alguns DOPs eram abaixo de 1, o GPS apresentou oscilação em seu sinal, exigindo o recurso de “Erro mínimo”. Porém segundo Monico (2007), a definição da janela de observação não é crítica nos dias atuais, devido ao fato de que a constelação GPS tornou-se completa em 08/12/1993 e assim os valores PDOP passaram a ser relativamente adequados.

Mesmo o sincronismo entre cliente e servidor ser aplicado utilizando uma *thread* auxiliar, não apresentando de forma direta uma ação do usuário para realizar este sincronismo, vale ressaltar que, mesmo no maior talhão, onde foram coletados 1069 pontos, em um intervalo de 3 segundos em cada ponto de coleta, com uma duração de 57 minutos, o sincronismo foi rápido, não justificando a necessidade de testes de avaliação de tempo.

Desta forma se justifica a ideia de Mohamed e Wijessekera (2014), Arroqui et al., (2012) e Honda e Zarpelão, (2014) onde é recomendado o uso do padrão RESTful para a criação de cidades inteligentes seguindo o conceito da Internet das Coisas, e ainda combinado ao armazenamento em banco de dados NoSQL, com alta disponibilidade, alto desempenho, escalonamento, rapidez na comunicação e possibilidade de crescimento horizontal apresentado por Sadalage & Fowler (2013), Queiroz et al., (2013).

5 CONCLUSÃO E TRABALHOS FUTUROS

As técnicas de *cloud computing* combinadas com o Padrão RESTful e armazenamento em bancos de dados NoSQL, orientado a documentos, resultam em técnicas que podem apresentar grande influência para o mercado de geoprocessamento e SIG. A Figura 20, da Seção 3.1.4, apresenta dados de usuário e mapas temáticos gerados por dados geográficos coletados e armazenados na nuvem, uma característica importante deste modelo apresentado e, pode ser considerada inovadora para o mercado de SIGs. A utilização dessa estrutura permite realizar a coleta de dados geográficos em tempo real, ou seja, o aplicativo disponibiliza um serviço geográfico distribuído.

O servidor de aplicação (SaaS) segue os modelos e o embasamento de integração de serviços da Internet das Coisas, onde é utilizada uma comunicação cliente-servidor via web service REST, para centralização de dados, de forma padronizada. A sincronização do aplicativo móvel com o servidor de aplicação depende da disponibilidade de internet e isto pode ser um problema, visto que a internet móvel brasileira ainda apresenta falhas, porém o Padrão RESTful utiliza menos bytes que o Padrão SOAP e sincroniza uma a quantidade de dados maior, mesmo com velocidade baixa de transferências de dados.

O uso de banco de dados NoSQL também é destaque, segundo Queiroz et al., (2013), mostra que esta modalidade de banco de dados é recomendada para o armazenamento de dados geográficos de forma prática, e também poderá facilitar a integração entre serviços geoespaciais, semelhante a projetos que utilizaram NoSQL. Ainda como em Vitolo et al., (2015) e Queiroz (2012), onde apresentaram ótimos resultados ao trabalhar com NoSQL e dados geoespaciais.

Os mapas gerados podem ser carregados, manipulados e analisados em *softwares* SIG (como o *software* ArcGIS, por exemplo), reforçando o conceito de serviço geográfico distribuído. Desta forma, a arquitetura apresenta o comportamento de “back-end”, ou seja, suas funcionalidades são integradas com outras ferramentas. Neste exemplo, *softwares* desktop. Esta é uma características importante do mercado atual e deve ser levada em consideração em qualquer aplicação móvel desenvolvida.

Outra característica é que o servidor de aplicação (SaaS) permite adicionar, de forma padronizada, quantos serviços externos forem necessários para o funcionamento da aplicação desenvolvida. Este padrão é recomendado para o uso na Internet das Coisas.

No uso do aplicativo móvel em larga escala, o NoSQL apresenta características de escalonamento e alta disponibilidade dos dados, em comparação a serviços de SQL relacional, recomendado para o cenário onde esta pesquisa foi aplicado. Assim como

Mcguinnes & Kapros, (2015) que apresentaram resultados satisfatórios no uso de NoSQL e desenvolveram um sistema adaptativo baseado nesta tecnologia e, Duro *et al.*, (2015) que apresentaram resultados com alta disponibilidade de dados e armazenamento rápido em uma solução NoSQL consumida por aplicativos móveis.

Destaca-se que a pesquisa foi essencial para a compreensão do comportamento de um software geoespacial móvel, combinando tecnologias de computação com metodologias e ciência de softwares de SIG. A arquitetura desenvolvida neste projeto é adequada como paradigma para o desenvolvimento de aplicativos voltados para o mercado geoespacial, agrícola e SIG Móvel, seguindo conceitos de integração de informações da Internet das Coisas e, com a opção de incorporar serviços externos à aplicação, com base na padronização aplicada de comunicação entre cliente-servidor e, conceitos da ciência e sistemas de informação geográfica apresentados por Longley et al., (2013). Os dados centralizados no servidor de aplicação, podem auxiliar no mapeamento em tempo real de dados provenientes da agricultura e serviços geoespaciais, e ainda podem ser correlacionados com dados provenientes de outras áreas, como: Logística, saúde, dados governamentais, mobilidade urbana, dentre outros.

A contribuição desta dissertação apresenta alto potencial de uso para cadastro de propriedades agrícolas e mapeamento das mesmas, visto que é plausível cadastrar talhões, áreas de preservação permanente, rios e pontos de interesse, como áreas de erosão, pontos em culturas onde foram detectadas pragas, falhas no solo e baixa produtividade.

As funcionalidades utilizadas neste projeto levam em consideração apenas o hardware disponível em *smartphones* para a coleta de dados geográficos, porém vale destacar que o aplicativo desenvolvido pode ser utilizado em uma placa micro controladora, onde é possível adicionar uma gama de sensores, tais como temperatura, umidade, um GPS de melhor qualidade, ou como o serviço visto de uma rede de sensores que opera em tempo real, visto em Papageorgas et al., (2014). Ainda ressalta-se que, as técnicas geoespaciais disponíveis pelo MongoDB podem ser exploradas e aplicadas a arquitetura, assim como a integração, diretamente no aplicativos, entre os pontos coletados e fotografias.

Ainda, como trabalho futuro, serviços de tomada de decisão podem ser incorporados ao projeto, exemplo visto em Antonopoulou et al., (2010), também seria possível combinar técnicas de Inteligência Artificial (IA) para realizar esta tarefa.

6 REFERÊNCIAS BILIOGRÁFICAS

AMADO, Telmo J. C., GIOTTO, Enio. **A sua lavoura na tela**. Revista A Granja, São Paulo, v. 732, dezembro, 2009.

ANTONOPOULOU, E., KARETSOS, S. T., MALIAPPIS, M., SIDERIDIS, A. B. **Web and mobile Technologies in a prototype DSS for major field crops**. Computers and Electronics in Agriculture, Elsevier, v. 70, p. 292-301, 2010.

ARROQUI, M., MATEOS, C., MACHADO, C., ZUNINO, A. **RESTful Web Services improve the efficiency of data transfer of a whole-farm simulator accessed by Android smartphones**. Computers and Electronics in Agriculture, Elsevier, v. 87, p. 14-18, 2012.

ATZORI, L., IERA, A., MORABITO, G. **The Internet of Things: A survey**. Computer Networks, Elsevier, v. 54, p. 2787-2805, 2010.

BORGIA, Eleonora. **The Internet of Things vision: Keys features, applications and open issues**. Em: “Computer Communications”. Elsevier, v. 54, p. 1-30, 2014.

BUTLER, H.; DALY, M.; DOYLE, A.; GILLIES, S.; SCHAUB, T.; SCHMIDT, C. **The GeoJSON Format Specification**. Disponível em: <http://www.geojson.org/geojson-spec.html>. Acesso: 30 de junho de 2015.

CÂMARA, G., CASANOVA, M., DAVIS, C., VINHAS, L., QUEIROZ, G. R. **Bancos de Dados Geográficos**. Curitiba: Livraria Virtual MundoGEO, 2005.

CHANG, K. **Introduction to Geographic Information Systems**. McGraw-Hill Companies, 6ª ed., New York, 2012.

CHODOROW, K. **MongoDB: The Definitive Guide**. O’REILLY, 2ª edição, 2013.

COULOURIS, George, KINDBERG, Tim, DOLLIMORE, Jean. **Sistemas Distribuídos: Conceitos e Projeto**. Bookman, 4ª edição – Porto Alegre.

DALFOVO, Oscar, ZEINDIN, Carla A., AZAMBUJA, Ricardo A., DIAS, Paulo R. A **Tecnologia do Futuro Wi-Fi (Wireless Fidelity)**. Blumenau, 2003. <http://campeche.inf.furb.br/siic/siego/docs/Artigo_Wireless_Uniplac_2003.pdf> Acessado em 08/02/2014.

DECANDIA, G., HASTORUN, D., JAMPANI, M., KAKULAPATI, G., LAKSHMAN, A., PILCHIN, A., SIVASUBRAMANIAN, S., VOSSHALL, P., VOGELS, W. **Dynamo: Amazon’s**

Highly Available Key-value Store. ACM SIGOPS Operating System Review, v. 41, n. 6, p. 205-220, Dezembro, 2007.

DINIZ, A. **Estatística Espacial**. Minas Gerais: UFMG - Universidade Federal de Minas Gerais, 2000.

DOBRE, C., XHAFA, F. **Intelligent services for Big Data science**. Future Generation Computer Systems. Elsevier, v. 37, p. 267-281, 2014.

DRUCK, S.; CARVALHO, M.S.; CÂMARA, G.; MONTEIRO, A.V.M. **Análise Espacial de Dados Geográficos**. Brasília, EMBRAPA, 2004.

DURO, F. R., BLAS, J. G., HIGUERO, D., PEREZ, O., CARRETERO, J. **CoSMiC: A hierarchical cloudlet-based storage architecture for mobile clouds**. Simulation Modelling Practice and Theory. Elsevier, v. 50, p. 3-19, 2015.

EVANS, D. **The Internet of Things: how the next evolutivon of the internet is changing everything**. White Paper, CISCO IBSG, 2011. Disponível em <http://www.cisco.com/web/about/ac79/iot/index.html>. Acessado em 19 de junho de 2015.

FIELDING, R. **Architectural Styles and the Design of Network-based Software Architectures**. 100 p. Tese (Doutorado) — University of California, 2000.

FIGUEIREDO, Carlos M. S., NAKAMURA, Eduardo. **Computação Móvel: Novas Oportunidades e Novos Desafios**. T&C Amazônia, nº 2, junho, 2003.

GUBBI, J., BUYYA, R., MARUSIC, S., PALANISWAMI, M. **Internet Of Things (IoT): A Vision, architectural elements, and future directions**. Future Generation Computer Systems, Elsevier, v. 29, p. 1645-1660, 2013.

HONDA, Natali S., ZARPELÃO, Bruno B. **Arquitetura baseada em REST (Representational State Transfer) para Cidades Inteligentes**. X SBSI, Londrina, 2014.

KALOXYLOS, A., GROUMAS, A., SARRIS, V., KATSIKAS, L., MAGDALINOS, P., ANTONIOU, E., POLITOPOULOU, Z., WALFERT, S., BROWSTER, C., EIGENMANN, R., TEROL, C. M. **A cloud-based Farm Management System: Architecture and implementation**. Computers and Electronics in Agriculture. Elsevir, v. 100, p. 168-179, 2014.

LECHETA, Ricardo R. **Google Android: Aprenda a criar aplicações para dispositivos móveis com Android SDK**. Novatec, 2013, 3ª edição – São Paulo.

LEE, J., KANG, M. **Geospatial Big Data: Challenges and Opportunities**. Big Data Research. Elsevier, Article in press, 2015.

LEE, Valentino, SCHNEIDER, Heather, SCHELL, Robbie. **Aplicações Móveis: Arquitetura, projeto e desenvolvimento**. São Paulo: Pearson Education do Brasil, 2005.

LAKSHMAN, A., MALIK, P. **Cassandra: a decentralized structured storage system**. ACM SIGOPS Operating System Review, v. 44, n. 2, p. 35-40, Abril, 2010.

LOMOTEY, R., DETERS, R. **Using a Cloud-Centric Middleware to Enable Mobile Hosting of Web Services**. Procedia Computer Science, Elsevier, v. 10, p. 634-642, 2012.

LONGLEY, Paul A., GOODCHILD, Michael F., MAGUIRE, David J., RHIND, David W. **Sistemas e ciência da informação geográfica**. Bookman, 2013, 3ª edição – Porto Alegre.

MARIMOTO, Carlos E. **Smartphones, Guia Prático**. Porto Alegre, GDH Press & Sul Editores, 2009.

MARKUS. **MongoDB Plugin for Quantum GIS**. Disponível em: <http:// http://geokoder.com/mongodb-plugin-for-quantum-gis>. Acesso: 30 de junho 2015.

MATEUS, Geraldo R., LOUREIRO, Antonio A. F. **Introdução A Computação Móvel**. Rio de Janeiro, RJ: 11a. Escola de Computação, 1998.

MCGUINNES, S., KAPROS, E. **Conceptual Independence: A design principle for the construction of adaptative information systems**. Information Systems. Elsevier, v. 47, p. 33-50, 2015.

MESAS-CARRASCOSA, F. J., CASTILLEJO-GONZÁLEZ, I. L., ORDEN, M. S., GARCÍA-FERRER, A. **Real-time mobile phone application to support land policy**. Computers and Electronics in Agriculture, v. 85, p. 109-111, 2012.

MOHAMED, K., WIJESEKERA, D. **Performance Analysis of Web Service on Mobile Devices.** Em: Procedia Computer Science. Elsevier, v. 10, p. 744-751, 2012.

MONICO, João F.G. **Posicionamento pelo GNSS: Descrição, fundamentos e aplicações**. 2ª ed. Presidente Prudente: Editora UNESP, 2007.

NETO, M. C., MAIA, J., QUEIROZ e MELLO L., FERNANDES, L. M. **Computação móvel em agricultura**. Revista de Ciências Agrárias v.30 n.1, Lisboa, jan. 2007.

NIKKILA, R., SEILONEN, I., KOSKINEN, K. **Software architecture for farm management information systems in precision agriculture**. Computers and Electronics in Agriculture. Elsevier, v. 70, p. 328-336, 2010.

ORLOVSKI, R. **Avaliação de espaçamentos das coletas de solo para a modelagem de granulometria com aplicação na agricultura de precisão**. (Dissertação) Universidade Estadual de Ponta Grossa, 2013.

PANDE, P., PADWALKAR, A. R. **Internet of Things – A Future of Internet: A Survey**. International Journal of Advance Research in Computer Science and Management Studies, v. 2, 2014.

PAPAGEORGAS, P, PIROMALIS, D., ILUOPOULOU, T., AGAVANAKIS, K., BARBAROSOU, M., PREKAS, K., ANTONAKOGLOU, K. **Wirelress Sensor Networking architecture of Polytropon: An open source scalable platform for the smart grid**. Energy Procedia, Elsevier, v. 50, p. 270-276, 2014.

PITT, Layland F., PARENT, Michael, JUNGLAS, Iris, CHAN, Anthony, SPYROPOULOU, Stavroula. **Integrating the smartphone into a sound environmental information systems strategy: Principles, practices and a research agenda**. Journal of Strategic Information Systems. Elsevier, 2011.

QUEIROZ, G. R. CellDB: Uma arquitetura para suporte a problemas de modelagem ambiental em grande escala. Tese de doutorado, São José dos Campos – INPE, 109 p., 2012.

QUEIROZ, G. R., MONTEIRO, M. V., CÂMARA, G. **Banco de dados geográficos e sistemas NoSQL: Onde estamos e para onde vamos**. Revista Brasileira de Cartografia n. 65, v. 3, p. 479-492, 2013.

ROBAINA, Adroaldo D., CATEN, Alexandre T. **Caderno Didático: Fundamentos do Sistema de Posicionamento Global - GPS**. Santa Maria, Colégio Politécnico da UFSM, 2006.

SADALAGE, Pramd J., FOWLER, Martin. **NoSQL: Um Guia Conciso para o Mundo Emergente da Persistência Poliglota**. Novatec Editora Ltda, São Paulo - SP, 2013.

SEBEM, Elódio, CATEN, Alexandre T., ROBAINA, Adroaldo D., MOREIRA, Antão L. L., PELLEGRINI, Guilherme C. **Fundamentos de cartografia e o sistema de posicionamento**

global GPS. UFSM – Santa Maria, Colégio Politécnico, Departamento de Engenharia Rural, 2010.

SONY, 2013. **Xperia ZQ | Smartphone – Sony Smartphones (Brazil)**. <http://www.sonymobile.com/br/products/phones/xperia-zq/> Acessado em 08/02/2013.

SOUSA, F. R. C. ; MOREIRA, L. O. ; MACHADO, J. C.. **Computação em Nuvem: Conceitos, Tecnologias, Aplicações e Desafios**. Em: Antônio Costa de Oliveira; Raimundo Santos Moura; Francisco Vieira de Souza. (Org.). III Escola Regional de Computação Ceará, Maranhão e Piauí (ERCEMAPI). 1ª ed. Teresina. : SBC. v. 1, p. 150-175. 2009.

STEINIGER, S., HUNTER, A. J. S. **The 2012 free and open source GIS software map – A guide to facilitate research, development, and adoption**. Computers, Environment and Urban Systems, Elsevier, v. 39, p. 136-150, 2013.

STEVENS, W. R. **TCP/IP Illustrated, Volume 1, The Protocols**. Addison-Wesley, 600 p. 1994.

STONEBRAKER, M., CETINTEMEL, U. **“One Size Fits All”: An Idea Whose Time Has Come and Gone**. International Conference on Data Engineering (ICDE ’05), 21. Proceedings... IEEE Computer Society, Washington, DC, USA, p. 2-11, 2005.

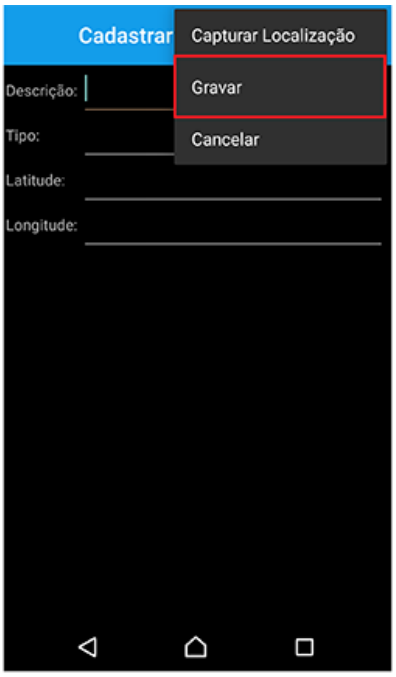
THOMPSON, M. **GEO, CouchDB & Node.js**. O’REILLY, 66 p., 2011.

TUTORIALS POINT, 2014. **SQLite Tutorial**. Disponível em <http://www.tutorialspoint.com/sqlite/sqlite_tutorial.pdf> Acesso em 12/02/2014.

VITOLO, C, ELKHATIB, Y., REUSSER, D., MACLEOD, C. J. A., BUYTAERT, W. **Web Technologies for environmental Big Data**. Environmental Modelling & Software, Elsevier, v. 63, p. 185-198, 2015.

APÊNDICE A – Cadastro de Locais e inserção de área manualmente no aplicativo móvel

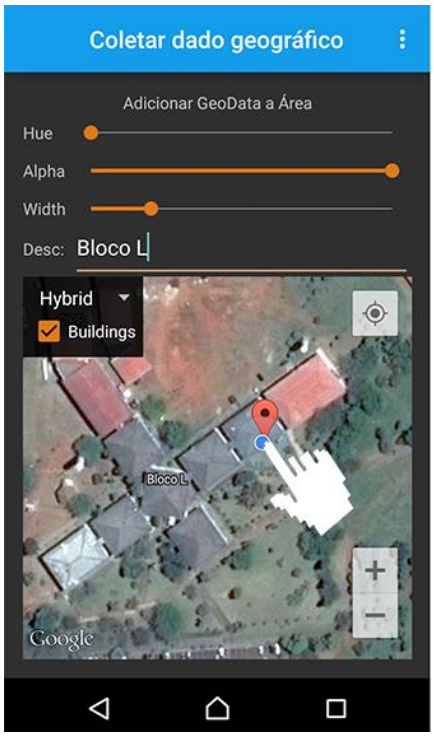
(a) Botão para cadastro de novo Local e clique para acessar o local cadastrado.



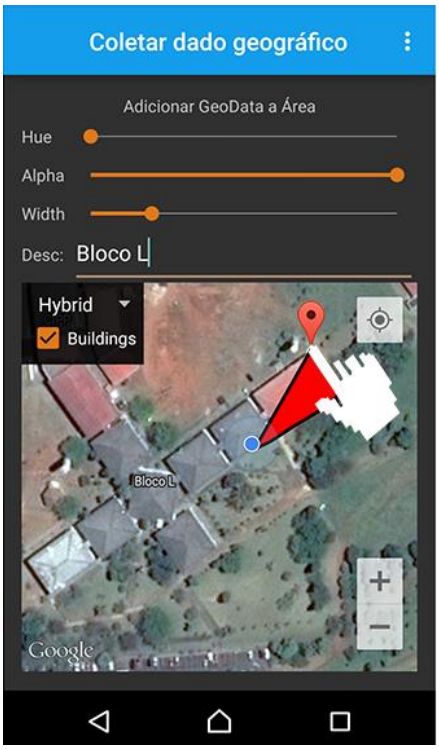
(b) Campos para serem preenchidos e menu cadastro de novo Local.

(c) Listagem de Áreas cadastradas em um Local e menu com opções disponíveis.

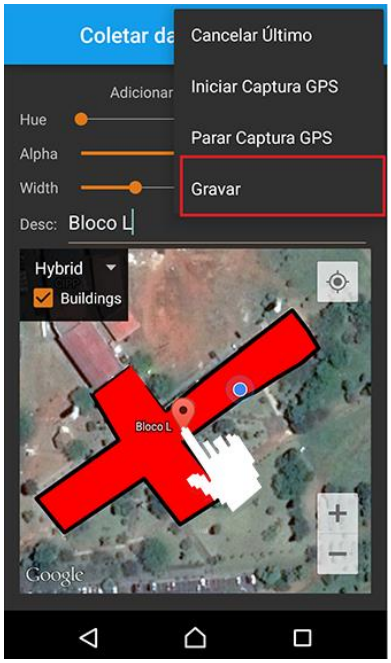
(d) Opção “Insere GeoData” acessada no menu (c) e tipos de coleta de dados geográfico disponíveis.



(e) Coleta do tipo manual de polígono, inserção do ponto inicial com clique longo no mapa.



(f) Coleta do tipo manual de polígono, coletando pontos com cliques longos no mapa.



(g) Finalizando polígono e gravando

e opções no menu para Iniciar e Parar Captura,

em caso de coleta via GPS.

APÊNDICE B – Código fonte da classe DadoGeograficoResource do servidor de aplicação

```
1  package uepg.giuvane.rest;

2  import java.util.ArrayList;
3  import java.util.List;
4  import javax.ws.rs.Consumes;
5  import javax.ws.rs.DELETE;
6  import javax.ws.rs.GET;
7  import javax.ws.rs.POST;
8  import javax.ws.rs.Path;
9  import javax.ws.rs.Produces;
10 import javax.ws.rs.QueryParam;
11 import com.google.gson.Gson;
12 import com.google.gson.JsonArray;
13 import com.google.gson.JsonParser;
14 import uepg.giuvane.dao.DAOException;
15 import uepg.giuvane.dao.DadoGeograficoDAO;
16 import uepg.giuvane.model.DadoGeografico;

17 @Path("/dadogeografico")
18 public class DadoGeograficoResource {
19 private DadoGeograficoDAO dadoGeograficoDAO;

20 @GET
21 @Path("/buscarTodos")
22 @Produces("application/json")
23 public List<DadoGeografico> selTodos() throws DAOException{
24 dadoGeograficoDAO = new DadoGeograficoDAO();
25 return dadoGeograficoDAO.buscarTodos();
26 }

27 @GET
28 @Path("/buscarTodosGSON")
```

```
29 @Produces("application/json")
30 public String selTodosGSON() throws DAOException{
31 dadoGeograficoDAO = new DadoGeograficoDAO();
32 return new Gson().toJson(dadoGeograficoDAO.buscarTodos());
33 }


34 @GET
35 @Path("/buscarListaPorId")
36 @Produces("application/json")
37 public List<DadoGeografico> getListaPorId(@QueryParam("id") int id) throws
   DAOException{
38 dadoGeograficoDAO = new DadoGeograficoDAO();
39 List<DadoGeografico> dadosGeograficos = new ArrayList<DadoGeografico>();
40 dadosGeograficos = dadoGeograficoDAO.buscarListaPorId("codAreaDg", id);
41 return dadosGeograficos;
42 }


43 @GET
44 @Path("/buscar")
45 @Produces("application/json")
46 public DadoGeografico getDadoGeografico(@QueryParam("id") int id) throws
   DAOException{
47 dadoGeograficoDAO = new DadoGeograficoDAO();
48 DadoGeografico dadoGeografico = new DadoGeografico();
49 dadoGeografico = dadoGeograficoDAO.buscarPorId("codDadoGeografico", id);
50 return dadoGeografico;
51 }


52 @GET
53 @Path("/excluir")
54 @Produces("application/json")
55 public String excluir(@QueryParam("id") int id) throws DAOException{
56 dadoGeograficoDAO = new DadoGeograficoDAO();
```

```
57 return dadoGeograficoDAO.excluir(id);
58 }


59 @POST
60 @Path("/cadastrar")
61 @Produces("application/json")
62 @Consumes("application/json")
63 public String cadastrar(DadoGeografico dadoGeografico) throws DAOException {
64 dadoGeograficoDAO = new DadoGeograficoDAO();
65 return dadoGeograficoDAO.cadastrar(dadoGeografico);
66 }


67 @POST
68 @Path("/atualizar")
69 @Produces("application/json")
70 @Consumes("application/json")
71 public String atualizar(@QueryParam("id") int id, DadoGeografico dadoGeografico)
   throws DAOException {
72 dadoGeograficoDAO = new DadoGeograficoDAO();
73 return dadoGeograficoDAO.atualizar(id, dadoGeografico);
74 }


75 @POST
76 @Path("/cadastrarLista")
77 @Produces("application/json")
78 @Consumes("application/json")
79 public String inserirLista(String listaDadosGeograficosJson) {
80 Gson gson = new Gson();
81 ArrayList<DadoGeografico> listaDadosGeograficos = new
   ArrayList<DadoGeografico>();
82 JsonParser parser = new JsonParser();
83 JsonArray array = parser.parse(listaDadosGeograficosJson).getAsJsonArray();
```

```
84  for (int i = 0; i < array.size(); i++) {
85  listaDadosGeograficos.add(gson.fromJson(array.get(i), DadoGeografico.class));
86  }
87  return null;
88  //return Banco.getBancoInstance().inserirLista(listaAreas);
89  }
90  }
```

APÊNDICE C – Código fonte da classe DadoGeograficoDAO no servidor de aplicação

```
1  package uepg.giuvane.dao;

2  import java.util.ArrayList;
3  import java.util.List;
4  import uepg.giuvane.model.DadoGeografico;
5  import uepg.giuvane.utils.Utilz;
6  import com.google.gson.Gson;
7  import com.mongodb.BasicDBObject;
8  import com.mongodb.DBObject;

9  public class DadoGeograficoDAO {
10 private GenericDAO genericDao;
11 private DBObject localDB;

12 public DadoGeograficoDAO() throws DAOException
13 {
14 genericDao = new GenericDAO("Local");
15 }

16 public String cadastrar(DadoGeografico dadoGeografico) {
17 String msg = "";
18 try {
19 localDB = new BasicDBObject("codDadoGeografico",
   dadoGeografico.getCodDadoGeografico()).
20 append("codAreaDg", dadoGeografico.getCodAreaDg()).
21 append("latitude", dadoGeografico.getLatitude()).
22 append("longitude", dadoGeografico.getLongitude()).
23 append("altitude", dadoGeografico.getAltitude()).
24 append("timeCoordenada", dadoGeografico.getTimeCoordenada()).
25 append("hora", dadoGeografico.getHora()).
26 append("sincronizado", dadoGeografico.getSincronizado());
```

```
27 this.genericDao.save(localDB);
28 msg = "Dado Geográfico cadastrado com sucesso";
29 } catch (DAOException ex) {
30 msg = "Erro ao cadastrar Dado Geográfico";
31 System.err.println("Falha ao criar o objeto 'Dado Geográfico'." + ex);
32 throw new ExceptionInInitializerError(ex);
33 }
34 return msg;
35 }


36 public String atualizar(int id, DadoGeografico dadoGeografico) {
37 String msg = "";
38 try {
39 localDB = new BasicDBObject("codDadoGeografico",
   dadoGeografico.getCodDadoGeografico()).
40 append("codAreaDg", dadoGeografico.getCodAreaDg()).
41 append("latitude", dadoGeografico.getLatitude()).
42 append("longitude", dadoGeografico.getLongitude()).
43 append("altitude", dadoGeografico.getAltitude()).
44 append("timeCoordenada", dadoGeografico.getTimeCoordenada()).
45 append("hora", dadoGeografico.getHora()).
46 append("sincronizado", dadoGeografico.getSincronizado());
47 this.genericDao.update("codDadoGeografico", id, localDB);
48 msg = "Dado Geográfico cadastrado com sucesso";
49 } catch (DAOException ex) {
50 msg = "Erro ao cadastrar Dado Geográfico";
51 System.err
52 .println("Falha ao criar o objeto 'Dado Geográfico'." + ex);
53 throw new ExceptionInInitializerError(ex);
54 }
55 return msg;
56 }


57 public String excluir(int id) {
58 String msg = "";
```

```
59 try {
60 DadoGeografico dadoGeografico = new DadoGeografico();
61 dadoGeografico = buscarPorId("codDadoGeografico", id);
62 localDB = new BasicDBObject("codDadoGeografico", id).
63 //append("_id", local.get_id()).
64 append("codAreaDg", dadoGeografico.getCodAreaDg()).
65 append("latitude", dadoGeografico.getLatitude()).
66 append("longitude", dadoGeografico.getLongitude()).
67 append("altitude", dadoGeografico.getAltitude()).
68 append("timeCoordenada", dadoGeografico.getTimeCoordenada()).
69 append("hora", dadoGeografico.getHora()).
70 append("sincronizado", dadoGeografico.getSincronizado());
71 this.genericDao.delete(localDB);
72 msg = "Dado Geográfico excluído com sucesso";
73 } catch (DAOException e) {
74 // TODO Auto-generated catch block
75 msg = "Erro ao excluir área";
76 e.printStackTrace();
77 }
78 return msg;
79 }


80 public List<DadoGeografico> buscarTodos() {
81 List<DBObject> objetos = new ArrayList<DBObject>();
82 DadoGeografico dadoGeografico;
83 List<DadoGeografico> dadosGeograficos = new ArrayList<DadoGeografico>();
84 Gson gson = new Gson();
85 try {
86 //genericDao = new GenericDAO("DadoGeografico");
87 objetos = this.genericDao.list();
88 for(int i = 0; i < objetos.size(); i++) {
89 dadoGeografico = gson.fromJson(Utilz.str2reader(objetos.get(i)),
   DadoGeografico.class);
90 dadosGeograficos.add(dadoGeografico);
91 //System.out.println(contato.getNome() + contato.getTelefone());
92 }
```

```
93  } catch (Throwable ex) {
94  System.err.println("Falha ao criar o objeto 'Lista Dado Geográfico'." + ex);
95  throw new ExceptionInInitializerError(ex);
96  }
97  return dadosGeograficos;
98  }


99  public DadoGeografico buscarPorId(String field, int id) throws DAOException
100         {
101         DadoGeografico dadoGeografico;
102         //genericDao = new GenericDAO("DadoGeografico");
103         Gson gson = new Gson();
104         dadoGeografico = gson.fromJson(Utilz.str2reader(genericDao.getById(field,
    id)), DadoGeografico.class);
105         return dadoGeografico;
106         }


107         public List<DadoGeografico> buscarListaPorId(String field, int id)
108         {
109         List<DBObject> objetos = new ArrayList<DBObject>();
110         DadoGeografico dadoGeografico;
111         List<DadoGeografico> dadosGeograficos = new
    ArrayList<DadoGeografico>();
112         Gson gson = new Gson();
113         try {
114         //genericDao = new GenericDAO("DadoGeografico");
115         objetos = this.genericDao.listById(field, id);
116         for(int i = 0; i < objetos.size(); i++) {
117         dadoGeografico = gson.fromJson(Utilz.str2reader(objetos.get(i)),
    DadoGeografico.class);
118         dadosGeograficos.add(dadoGeografico);
119         //System.out.println(contato.getNome() + contato.getTelefone());
120         }
121         } catch (Throwable ex) {
122         System.err
```

```
123         .println("Falha ao criar o objeto 'Lista Contatos'." + ex);
124         throw new ExceptionInInitializerError(ex);
125         }
126         return dadosGeograficos;
127         }
128         }
```

APÊNDICE D – Código da classe HttpClientSingleton no aplicativo móvel

```
1  package uepg.giuvane.appagricola.webservice;

2  import org.apache.http.impl.client.DefaultHttpClient;
3  import org.apache.http.params.BasicHttpParams;
4  import org.apache.http.params.HttpConnectionParams;
5  import org.apache.http.params.HttpParams;

6  public class HttpClientSingleton {

7  private static final int JSON_CONNECTION_TIMEOUT = 3000;
8  private static final int JSON_SOCKET_TIMEOUT = 5000;
9  private static HttpClientSingleton instance;
10 private HttpParams httpParameters ;
11 private DefaultHttpClient httpclient;

12 private void setTimeOut(HttpParams params){
13 HttpConnectionParams.setConnectionTimeout(params,
   JSON_CONNECTION_TIMEOUT);
14 HttpConnectionParams.setSoTimeout(params, JSON_SOCKET_TIMEOUT);
15 }
16 private HttpClientSingleton() {
17 httpParameters = new BasicHttpParams();
18 setTimeOut(httpParameters);
19 httpclient = new DefaultHttpClient(httpParameters);
20 }

21 public static DefaultHttpClient getHttpClientInstace(){
22 if(instance == null)
23 instance = new HttpClientSingleton();
24 return instance.httpclient;
25 }
```

```
26 }
```

APÊNDICE E – Código fonte da classe DadoGeograficoREST no aplicativo móvel

```
1  package uepg.giuvane.appagricola.webservice;

2  import java.util.ArrayList;
3  import java.util.List;
4  import uepg.giuvane.appagricola.classes.DadoGeografico;
5  import com.google.gson.Gson;
6  import com.google.gson.JsonArray;
7  import com.google.gson.JsonParser;

8  public class DadoGeograficoREST {
9  public DadoGeografico getDadoGeografico(int id) throws Exception {
10 String[] resposta = new WebServiceCliente().get(WebServiceConfig.URL_WS +
   "dadogeografico/buscar/?id=" + id);
11 if (resposta[0].equals("200")) {
12 Gson gson = new Gson();
13 DadoGeografico dadoGeografico = gson.fromJson(resposta[1],
   DadoGeografico.class);
14 return dadoGeografico;
15 } else {
16 throw new Exception(resposta[1]);
17 }
18 }

19 public List<DadoGeografico> getListaDadosGeograficos() throws Exception {

20 String[] resposta = new WebServiceCliente().get(WebServiceConfig.URL_WS +
   "dadogeografico/buscarTodosGSON");
21 if (resposta[0].equals("200")) {
22 Gson gson = new Gson();
23 List<DadoGeografico> listaDadosGeograficos = new ArrayList<DadoGeografico>();
24 JsonParser parser = new JsonParser();
```

```
25 JsonArray array = parser.parse(resposta[1]).getAsJsonArray();
26 for (int i = 0; i < array.size(); i++) {
27 listaDadosGeograficos.add(gson.fromJson(array.get(i), DadoGeografico.class));
28 }
29 return listaDadosGeograficos;
30 } else {
31 throw new Exception(resposta[1]);
32 }
33 }

34 public List<DadoGeografico> getListaDadosGeograficosPorId(int id) throws
   Exception
35 {
36 String[] resposta = new WebServiceCliente().get(WebServiceConfig.URL_WS +
   "dadogeografico/buscarListaPorId?id=" + id);
37 if (resposta[0].equals("200")) {
38 Gson gson = new Gson();
39 List<DadoGeografico> listaDadosGeograficos = new ArrayList<DadoGeografico>();
40 JsonParser parser = new JsonParser();
41 JsonArray array = parser.parse(resposta[1]).getAsJsonArray();

42 for (int i = 0; i < array.size(); i++) {
43 listaDadosGeograficos.add(gson.fromJson(array.get(i), DadoGeografico.class));
44 }
45 return listaDadosGeograficos;
46 } else {
47 throw new Exception(resposta[1]);
48 }
49 }

50 public String cadastrarDadoGeografico(DadoGeografico dadoGeografico) throws
   Exception {
51 Gson gson = new Gson();
52 String areaJSON = gson.toJson(dadoGeografico);
```

```
53 String[] resposta = new WebServiceCliente().post(WebServiceConfig.URL_WS +
   "dadogeografico/cadastrar", areaJSON);
54 if (resposta[0].equals("200")) {
55 return resposta[1];
56 } else {
57 throw new Exception(resposta[1]);
58 }
59 }


60 public String atualizaDadoGeografico(int id, DadoGeografico dadoGeografico) throws
   Exception {
61 Gson gson = new Gson();
62 String areaJSON = gson.toJson(dadoGeografico);
63 String[] resposta = new WebServiceCliente().post(WebServiceConfig.URL_WS +
   "dadogeografico/atualizar?id=" + id, areaJSON);
64 if (resposta[0].equals("200")) {
65 return resposta[1];
66 } else {
67 throw new Exception(resposta[1]);
68 }
69 }


70 public String excluirDadoGeografico(int id) {
71 String[] resposta = new WebServiceCliente().get(WebServiceConfig.URL_WS +
   "dadogeografico/excluir?id=" + id);
72 return resposta[1];
73 }
74 }
```

APÊNDICE F – Código fonte da classe WebServiceClient do aplicativo móvel

```
1  package uepg.giuvane.appagricola.webservice;

2  import java.io.ByteArrayOutputStream;
3  import java.io.IOException;
4  import java.io.InputStream;
5  import java.net.URI;
6  import org.apache.http.HttpEntity;
7  import org.apache.http.HttpResponse;
8  import org.apache.http.client.ClientProtocolException;
9  import org.apache.http.client.methods.HttpDelete;
10 import org.apache.http.client.methods.HttpGet;
11 import org.apache.http.client.methods.HttpPost;
12 import org.apache.http.client.methods.HttpPut;
13 import org.apache.http.entity.StringEntity;
14 import org.apache.http.params.HttpParams;
15 import org.json.JSONException;
16 import org.json.JSONObject;
17 import android.os.AsyncTask;
18 import android.os.StrictMode;
19 import android.util.Log;

20 public class WebServiceClient {

21 HttpResponse response;

22 public final String[] get(String url) {
23 String[] result = new String[2];
24 HttpGet httpget = new HttpGet(url);
25 try {
26 response = HttpClientSingleton.getHttpClientInstace().execute(httpget);
27 HttpEntity entity = response.getEntity();
```

```
28  if (entity != null) {
29  result[0] = String.valueOf(response.getStatusLine().getStatusCode());
30  InputStream instream = entity.getContent();
31  result[1] = toString(instream);
32  instream.close();
33  Log.i("get", "Result from post JsonPost : " + result[0] + " : " + result[1]);
34  }
35  } catch (Exception e) {
36  Log.e("UepgAgricola", "Falha ao acessar Web service", e);
37  result[0] = "0";
38  result[1] = "Falha de rede!";
39  }
40  return result;
41  }


42  public final String[] post(String url, String json) {
43  String[] result = new String[2];
44  try {
45  HttpPost httpPost = new HttpPost(new URI(url));
46  httpPost.setHeader("Content-type", "application/json");
47  StringEntity sEntity = new StringEntity(json, "UTF-8");
48  httpPost.setEntity(sEntity);
49  HttpResponse response;
50  response = HttpClientSingleton.getHttpClientInstace().execute(httpPost);
51  HttpEntity entity = response.getEntity();
52  if (entity != null) {
53  result[0] = String.valueOf(response.getStatusLine().getStatusCode());
54  InputStream instream = entity.getContent();
55  result[1] = toString(instream);
56  instream.close();
57  Log.d("post", "Result from post JsonPost : " + result[0] + " : " + result[1]);
58  }
59  } catch (Exception e) {
60  Log.e("UepgAgricola", "Falha ao acessar Web service", e);
61  result[0] = "0";
62  result[1] = "Falha de rede!";
```

```
63  }
64  return result;
65  }


66  private String toString(InputStream is) throws IOException {
67  byte[] bytes = new byte[1024];
68  ByteArrayOutputStream baos = new ByteArrayOutputStream();
69  int lidos;
70  while ((lidos = is.read(bytes)) > 0) {
71  baos.write(bytes, 0, lidos);
72  }
73  return new String(baos.toByteArray());
74  }
75  }
```

APÊNDICE G – Arquivo XML contendo o Layout desenvolvido para a tela de coleta de dados geográficos

```
1  <LinearLayout xmlns:tools="http://schemas.android.com/tools"
2  xmlns:android="http://schemas.android.com/apk/res/android"
3  android:layout_width="match_parent"
4  android:layout_height="match_parent"
5  android:orientation="vertical"
6  android:paddingBottom="@dimen/activity_vertical_margin"
7  android:paddingLeft="@dimen/activity_horizontal_margin"
8  android:paddingRight="@dimen/activity_horizontal_margin"
9  android:paddingTop="@dimen/activity_vertical_margin"
10 tools:context="uepg.giuvane.appagricola.activities.uepgAgricola.AdicionarGeoDataAc
   tivity" >

11 <TextView
12 android:layout_width="match_parent"
13 android:layout_height="wrap_content"
14 android:gravity="center_horizontal"
15 android:text="@string/cabecalho_adicionar_geodata" />

16 <TableLayout
17 android:layout_width="match_parent"
18 android:layout_height="wrap_content"
19 android:stretchColumns="1,2,3,4" >

20 <TableRow
21 android:layout_width="wrap_content"
22 android:layout_height="wrap_content"
23 android:gravity="center_vertical" >

24 <TextView android:text="@string/hue" />

25 <SeekBar
26 android:id="@+id/hueSeekBar"
27 android:layout_span="3" />
28 </TableRow>

29 <TableRow
30 android:layout_width="wrap_content"
31 android:layout_height="wrap_content"
32 android:gravity="center_vertical" >

33 <TextView android:text="@string/alpha" />

34 <SeekBar
35 android:id="@+id/alphaSeekBar"
36 android:layout_span="3" />
37 </TableRow>

38 <TableRow
39 android:layout_width="wrap_content"
40 android:layout_height="wrap_content"
41 android:gravity="center_vertical" >

42 <TextView android:text="@string/width" />

43 <SeekBar
```

```
44  android:id="@+id/widthSeekBar"
45  android:layout_span="3" />
46  </TableRow>

47  <TableRow
48  android:id="@+id/tableRow2"
49  android:layout_width="wrap_content"
50  android:layout_height="wrap_content" >

51  <TextView
52  android:id="@+id/tvDescAreaLabel"
53  android:layout_width="wrap_content"
54  android:layout_height="wrap_content"
55  android:text="Desc:  " />

56  <EditText
        a.  android:id="@+id/txtDescAreaDetalhes"
57  android:layout_width="wrap_content"
58  android:layout_height="wrap_content"
59  android:layout_span="3"
60  android:ems="10"
61  android:inputType="text" >

62  <requestFocus />
63  </EditText>

64  </TableRow>

65  <TableRow
66  android:id="@+id/tableRow1"
67  android:layout_width="wrap_content"
        a.  android:layout_height="wrap_content" >
68  </TableRow>

69  </TableLayout>

70  <RelativeLayout
71  android:layout_width="match_parent"
72  android:layout_height="match_parent">
73  <fragment
74  android:id="@+id/mapArea"
75  android:layout_width="match_parent"
76  android:layout_height="match_parent"
77  class="com.google.android.gms.maps.SupportMapFragment"/>
78  <!-- A set of test checkboxes. -->
79  <LinearLayout
80  android:layout_width="wrap_content"
81  android:layout_height="wrap_content"
82  android:layout_alignParentLeft="true"
83  android:layout_alignTop="@id/map"
84  android:padding="6dp"
85  android:background="#D000"
86  android:orientation="vertical">
87  <Spinner
88  android:id="@+id/layers_spinner"
```

```
89  android:layout_width="match_parent"
90  android:layout_height="wrap_content"/>


91  <CheckBox
92  android:id="@+id/buildings"
93  android:layout_width="wrap_content"
94  android:layout_height="wrap_content"
95  android:onClick="onBuildingsToggled"
96  android:checked="true"
97  android:text="Buildings"/>

98  </LinearLayout>
99  </RelativeLayout>


100 </LinearLayout>
```

APÊNDICE H - Exemplo de um arquivo KML gerado pelo servidor de aplicação.

UEPG.kml

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<ns3:kml xmlns:xal="urn:oasis:names:tc:ciq:xsdschema:xAL:2.0" xmlns:gx="http://www.google.com/kml/ext/2.2" xmlns:ns3="http://www.opengis.net/kml/2.2" xmlns:atom="http://www.w3.org/2005/Atom">
    <ns3:Placemark>
        <ns3:name>UEPG</ns3:name>
        <ns3:description>UEPG</ns3:description>
        <ns3:StyleMap>
            <ns3:Pair>
                <ns3:key>normal</ns3:key>
                <ns3:Style>
                    <ns3:LineStyle>
                        <ns3:color>FF000000</ns3:color>
                        <ns3:width>14.0</ns3:width>
                    </ns3:LineStyle>
                    <ns3:PolyStyle>
                        <ns3:color>cd2AFF00</ns3:color>
                    </ns3:PolyStyle>
                </ns3:Style>
            </ns3:Pair>
            <ns3:Pair>
                <ns3:key>highlight</ns3:key>
                <ns3:Style>
                    <ns3:LineStyle>
                        <ns3:color>FF000000</ns3:color>
                        <ns3:width>14.0</ns3:width>
                    </ns3:LineStyle>
                    <ns3:PolyStyle>
                        <ns3:color>cd2AFF00</ns3:color>
                    </ns3:PolyStyle>
                </ns3:Style>
            </ns3:Pair>
        </ns3:StyleMap>
        <ns3:Polygon>
            <ns3:altitudeMode>clampToGround</ns3:altitudeMode>
            <ns3:outerBoundaryIs>
                <ns3:LinearRing>
                    <ns3:coordinates>
                        -50.09411472827196,-25.088758652093368 -50.09400140494108,-25.090779990164094 -50.09572237730026,-25.094359835500914 -50.0965166464448,-25.097031746790226
                        -50.096573643386364,-25.100491481573826 -50.10228540748358,-25.09968629358901 -50.104402005672455,-25.098711073894446 -50.107030905783176,-25.09819704618781
                        -50.10653905570507,-25.095799334958723 -50.106122978031635,-25.09280190211188 -50.10517716407776,-25.09124303797335 -50.103815607726574,-25.0899586453132
                        -50.10201886296272,-25.089410269304842 -50.09411472827196,-25.088758652093368
                    </ns3:coordinates>
                </ns3:LinearRing>
            </ns3:outerBoundaryIs>
        </ns3:Polygon>
    </ns3:Placemark>
</ns3:kml>
```