UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA - DAINF
CURSO DE ESPECIALIZAÇÃO EM TECNOLOGIA JAVA VII

ELTON EIJI SASAKI

PROTÓTIPO DE SISTEMA DE ENTREGAS E COLETAS PARA MOTOBOYS

MONOGRAFIA DE ESPECIALIZAÇÃO

CURITIBA
2012

ELTON EIJI SASAKI

PROTÓTIPO DE SISTEMA DE ENTREGAS E COLETAS PARA MOTOBOYS

Monografia de especialização apresentada ao curso de Especialização em Tecnologia Java da Universidade Tecnológica Federal do Paraná como requisito parcial para a obtenção do título de especialista.

Orientador: Prof. Dr. João Alberto Fabro
Co-orientador: Prof. Me. Wilson Horstmeyer Bogado

CURITIBA

2012

AGRADECIMENTOS

Ao Professor Orientador Dr. João Alberto Fabro por todo o seu apoio durante o processo de criação e escrita na realização deste trabalho. Aos Professores que compuseram a banca de defesa da monografia, Professor Robson Linhares e Professor Paulo Bordin, pelas críticas e comentários sobre o texto da monografia. A todos os professores do Curso de Especialização em Tecnologia Java que transmitiram, com muito trabalho e dedicação, os conhecimentos para a realização desta monografia.

Não é só o progresso, o desenvolvimento da cultura intelectual, da riqueza e do poder da nossa pátria que desejamos, é também a crescente prosperidade de todos os povos do nosso continente.
(RIO BRANCO, Barão do)

RESUMO

SASAKI, Elton Eiji. **Protótipo de Sistema de Entregas e Coletas para Motoboys**. 2012. 52 páginas. Monografia de Especialização em Tecnologia Java VII - Universidade Tecnológica Federal do Paraná. Curitiba, 2012.

Na era em que as pessoas carregam a web no bolso, a forma como se faz a comunicação para o uso de serviços de entregas e coletas precisa ser repensado. A utilização da telefonia celular digital, juntamente com a difusão da conectividade sem fio, significa que as pessoas tem a sua disposição uma ferramenta que possibilita ao solicitante de um serviço de entrega ou coleta enviar sua localização atual, via google maps, para automatizar o serviço de entrega ou coleta de motoboys. Este projeto apresenta um protótipo de um sistema para auxílio no serviço de coletas e entregas de motoboys. Tal sistema é composto por um aplicativo web service, o qual está hospedado em um Servidor Privado Virtual (Virtual Private Server – VPS), responsável por armazenar os dados dos usuários do serviço e por comunicar a localização do solicitante de serviço de entrega ou coleta ao motoboy. Adicionalmente, o sistema é composto por um aplicativo para dispositivos móveis da plataforma Android, responsável por auxiliar o motoboy, via Google Maps, no serviço de entrega e coletas de produtos.
Palavras-chave: 1. Mobilidade. 2. Serviço de entregas e coletas de produtos. 3. Dispositivos Móveis Android.

ABSTRACT

SASAKI, Elton Eiji. **Prototype of a Pick-up and Delivery System for Motoboys**. 2012. 52 pages. Monografia de Especialização em Tecnologia Java VII - Universidade Tecnológica Federal do Paraná. Curitiba, 2012.

In the era where people carry the web in their pockets, the way people communicate, in order to use pick-up and delivery services, needs to be rethought. The use of mobile devices, along with the diffusion of wireless connectivity, means that people have at their disposal a tool that allows pick-up and delivery service customers to send their current location through Google Maps in order to automatize the pick up and delivery services offered by motoboys. This project presents a prototype system that aids the pick-up and delivery services of motoboys. This system consists of a web service application, which is hosted on a Virtual Private Server (VPS), responsible for storing the data of service users and for communicating the location of a pick-up and delivery service to the motoboy. Additionally, the system consists of an application for Android mobile devices, responsible for aiding the motoboy, through Google Maps, in the pick-up and delivery service of items.

Keywords: 1. Mobility 2. Pick-up and delivery service of items. 3. Android mobile devices.

LISTA DE ILUSTRAÇÕES

Figura 1 – Imagem da tela do solicitante de serviço (classe SolicitanteLocationActivity).....29

Figura 2 – Imagem da tela do solicitante de serviço apontando a sua localização atual com o ícone azul e apontando o local do serviço de entrega ou coleta com o ícone em formato de "gota" (classe SolicitanteLocationActivity).....33

Figura 3 – Imagem da tela do solicitante de serviço mostrando uma mensagem em forma de pop-up (classe SolicitanteLocationActivity).....36

Figura 4 – Imagem da tela do motoboy apontando o local do serviço de entrega ou coleta com o ícone em formato de "gota" (classe MotoboyLocationActivity).....39

Figura 5 – Diagrama de casos de uso do protótipo do sistema de entregas e coletas para motoboy.....45

Figura 6 – Diagrama de sequência do solicitante de serviço.....46

Figura 7 – Diagrama de sequência do motoboy.....47

Figura 8 – Diagrama de classes do aplicativo *Web Service*.....48

Figura 9 – Diagrama de classes do aplicativo *Android*.....49

LISTA DE ACRÔNIMOS

VPS	*Virtual Private Server* (Servidor Virtual Privado)
XML	*Extensible Markup Language*
JSON	*JavaScript Object Notation*
HTTP	*Hypertext Transfer Protocol*
REST	*Representational State Transfer*
SOAP	*Simple Object Access Protocol*
J2ME	*Java Platform, Micro Edition*
POX	*Plain Old XML*
JAX-WS	*Java API for XML Web Services*
JAXB	*Java Architecture for XML Binding*
SDK	*Software Development Kit*
UI	*User Interface*
JSF	*Java Server Faces*
JVM	*Java Virtual Machine*
JDK	*Java Development Kit*
JRE	*Java Runtime Environment*
JEE	*Java Entreprise Edition*
JNDI	*Java Naming and Directory Interface*
SMS	*Short Message Service*
API	*Application Programming Interface*

SUMÁRIO

1 INTRODUÇÃO 11
1.1 CONTEXTUALIZAÇÃO 11
1.2 OBJETIVOS 11
1.2.1 Objetivos Específicos 11
1.3 JUSTIFICATIVA 12
1.4 METODOLOGIA EMPREGADA 12
1.5 ORGANIZAÇÃO DO TRABALHO.....12
2 ESTUDOS DAS TECNOLOGIAS 14
2.1 WEB SERVICE 14
2.1.1 Soap-Based Web Service 14
2.1.2 Rest-Based Web Service – JSON e XML 14
2.1.2.1 Biblioteca Gson 15
2.1.3 Publicação de Mensagem do Web Service na Tela do Android 16
2.1.4 Publicação de Mensagem do Web Service na Tela do Android via AsyncTask 16
2.1.5 Consumo do Rest-based Json Web Service 17
2.2 JAVA SERVER FACES (JSF) 18
2.2.1 PrimeFaces 18
2.3 SERVIDOR PRIVADO VIRTUAL (VIRTUAL PRIVATE SERVER – VPS) 18
2.3.1 Java Development Kit (JDK) e Java Runtime Environment 7 (JRE) 19
2.3.2 PostgreSQL 19
2.3.3 GlassFish 19
2.4 ANDROID.....20
2.4.1 Arquivo AndroidManifest.xml 20
2.4.2 Classe Activity 20
2.5 GOOGLE MAPS 21
2.5.2 Classe MyLocationOverlay 21
2.5.3 Classe MapView 21
2.5.4 Classe LocationManager 21
2.5.5 Classe LocationListener 22
2.5.5.1 Classe Location 22
2.5.5.2 Classe Geocoder 22
2.5.5.3 Classe Context 22
2.5.5.4 Classe GeoPoint 22
3 DESENVOLVIMENTO DO PROTÓTIPO 24
3.1 WEB SERVICE 24
3.1.1 Rest-based Json Web Service.....25
3.1.2 Utilização dos Tipos de Dados Retornados pelo Web Service 25
3.1.2.1 Tipo de dado enviado ao web service para processamento: 26
3.2 VIRTUAL PRIVATE SERVER (VPS) 27
3.2.1 GlassFish 27
3.3 SMS (SHORT MESSAGE SERVICE) 27
3.3.1 Funcionamento do SMS no Aplicativo Android 28
3.4 APLICATIVO ANDROID 28
3.4.1 Levantamento de Requisitos 29
3.4.2 Diagrama de Casos de Uso.....29
3.4.3 Classe MyLocationOverlay 30
3.4.4 Chave da API do Google Maps 31
3.4.5 Activity 31
3.4.6 Classe SolicitanteLocationActivity 32

3.4.6.1 Classe LocalAtualListener 33
3.4.6.1.1 Classe Geocoder (geocodificação reversa) 34
3.4.7 Botão “Localizar” da Classe SolicitanteLocationActivity 35
3.4.7.1 Classe DestinoGeocodListener 35
3.4.7.1.1 Classe Geocoder (geocodificação) 36
3.4.7.1.2 Posicionamento do ícone no mapa (Overlay) 38
3.4.7.2 Web Service 40
3.4.8 Botão “Enviar” 42
3.4.9 Classe MotoboyLocationActivity 42
3.4.9.1 Web service e formatação de endereço no método “doInBackground” ….43
3.4.9.1.1 Web service no método “doInBackground” 43
3.4.9.1.2 Formatação do endereço no método “doInBackground” 43
3.4.9.2 Classe DestinoGeocodListener 44
3.4.9.3 Botão “Aceitar” 44
3.5 TRATAMENTO DE AUSÊNCIA DE CONECTIVIDADE COM O WEB SERVICE ….45
3.6 DIAGRAMAS DE SEQUÊNCIA 46
3.6.1 Descrição do Diagrama de Sequência do Solicitante de Serviço 46
.....46
3.6.2 Descrição do Diagrama de Sequência do Motoboy 47
3.7 DIAGRAMAS DE CLASSES 48
3.7.1 Diagrama de classes do aplicativo Web Service.....48
3.7.2 Diagrama de classes do aplicativo Android.....49
4 CONCLUSÕES 51
4.1 TRABALHOS FUTUROS.....51
5 REFERÊNCIAS BIBLIOGRÁFICAS 53

1 INTRODUÇÃO

Na era em que as pessoas carregam a web no bolso, a forma como se faz a comunicação para o uso de serviços de entregas e coletas precisa ser repensado. A utilização da telefonia celular digital, juntamente com a difusão da conectividade sem fio, significa que as pessoas tem a sua disposição uma ferramenta que possibilita ao solicitante de um serviço de entrega ou coleta enviar sua localização atual, via *Google Maps*, para automatizar o serviço de entrega ou coleta de motoboys.

1.1 CONTEXTUALIZAÇÃO

Atualmente, há muitos entregadores, conhecidos popularmente como motoboys, que ficam horas em tempo de espera durante o expediente de trabalho, pois aguardam por uma chamada para entregar ou coletar algum documento ou produto para alguma empresa. Há muitas empresas, que necessitam da entrega ou coleta de seus documentos ou produtos. Há, também, muitas pessoas físicas, que precisam da entrega ou coleta de seus documentos ou produtos, tanto para resolver problemas pessoais quanto para presentear uma pessoa querida.

É necessário o desenvolvimento de um aplicativo que pessoas físicas, empresas e entregadores possam se comunicar para agilizar o processo de coletas e entregas.

1.2 OBJETIVOS

Desenvolver um protótipo de sistema de entregas e coletas para motoboys, com o intuito de auxiliar a comunicação entre o solicitante de serviço (empresa ou pessoa física) e o ofertante de serviço (motoboy).

1.2.1 Objetivos Específicos

- desenvolver um sistema que facilite a solicitação de serviço de entregas e coletas

por meio de um aplicativo para dispositivos móveis *Android*;

- implantar uma aplicação *web service* para transmitir e obter dados, como por exemplo, o dado de endereço da localização do serviço de entrega ou coleta;
- obter o endereço da localização do serviço de entrega ou coleta por meio um aplicativo *Android* que se integre com a aplicação do *Google Maps*;
- implantar o serviço de mensagens *SMS* para o envio de solicitação de serviço ao motoboy e envio de confirmação de serviço ao solicitante.

1.3 JUSTIFICATIVA

A criação de um aplicativo de entrega e coletas é uma ferramenta que beneficia o entregador, pois recebe mais pedidos de entrega ou coleta com a utilização do aplicativo. O mesmo ocorre com a empresa ou pessoa física, pois ambos têm a sua disposição mais opções de entregadores com a adoção dessa ferramenta. Desta forma, o desenvolvimento de um aplicativo para *smartphones* se torna interessante, pois agiliza a comunicação entre o solicitante de serviço e o motoboy.

1.4 METODOLOGIA EMPREGADA

Foi realizado inicialmente um estudo das tecnologias envolvidas (comunicação via *web services*, desenvolvimento de aplicativo para dispositivos móveis da plataforma *Android*, uso de mapas e geolocalização).

Em seguida foi desenvolvido um *web service*, responsável pela comunicação de dados entre os aplicativos *Android* do solicitante de serviço e do motoboy.

Ao final, foi desenvolvido um aplicativo *Android*, utilizando os componentes do *Google Maps* integrados aos dispositivos móveis da plataforma *Android*.

1.5 ORGANIZAÇÃO DO TRABALHO

No capítulo 2 são apresentados os estudos das tecnologias que compõe o protótipo do sistema de entregas e coletas para motoboys. No capítulo 3, descreve-se a

integração das tecnologias estudadas no capítulo anterior, expondo a construção do protótipo do sistema. No capítulo 4, encontra-se as conclusões e perspectivas de trabalhos futuros.

2 ESTUDOS DAS TECNOLOGIAS

Este capítulo apresenta um referencial teórico sobre as tecnologias que fazem parte deste trabalho. As tecnologias utilizadas neste trabalho são *web service* (seção 2.1), *Java Server Faces* (seção 2.2), Servidor Privado Virtual (seção 2.3), plataforma *Android* (seção 2.4) e aplicação *Google Maps* (seção 2.5).

2.1 WEB SERVICE

Web service é um componente de software armazenado em um computador que pode ser acessado, via rede, por uma aplicação (ou outro componente de *software*) em outro computador. *Web services* comunicam-se usando tecnologias como *XML* (*Extensible Markup Language*), *JSON* (*JavaScript Object Notation*) e *HTTP* (*Hypertext Transfer Protocol*) (DEITEL; DEITEL, 2012, A, p. 1300).

2.1.1 Soap-Based Web Service

"*SOAP-Based web service* é um protocolo de plataforma-independente que usa *XML* para interagir com *web services*" (DEITEL; DEITEL, 2012, A, p. 1302). Esse tipo de *web service* permite a comunicação com um aplicativo cliente, mesmo se o cliente e o *web service* estejam escritos em linguagens de programação diferentes.

De acordo com os autores do livro "*Android* em Ação", recomenda-se utilizar *REST-Based web services* em *smartphones* para que se obtenha controle total sobre o cliente e o servidor, pois *SOAP-BASED web services* adicionam bastante excesso de recursos aos *smartphones* (ABLESON; COLLINS; SEN, 2012).

2.1.2 Rest-Based Web Service – JSON e XML

Em busca de um tipo de *web service* com "melhor desempenho, e um design mais simples" (ABLESON; COLLINS; SEN, 2012), foi encontrado o livro "*Java*: *how to program*", 9 edição, dos autores Paul e Harvey Deitel, editora Prentice Hall, que introduz

dois tipos de *web services*. O primeiro é o tipo *REST-Based XML web service* e o segundo tipo é o *REST-Based JSON web service*. O *REST-Based web service* (*Representational State Transfer*) é “uma arquitetura de rede que usa o mecanismo tradicional de requisição/resposta da web como *GET* e *POST*” (DEITEL; DEITEL, 2012, A, p. 1302).

O segundo tipo *REST-Based JSON web service*, apresentado no livro *Java*: *how to program*, utiliza *JSON*, que é um subconjunto da linguagem de programação *JavaScript*, e seus componentes – objetos, *arrays*, *strings*, números – podem ser facilmente mapeados em construções em *Java* e em outras linguagens de programação. As bibliotecas padrão *Java* atualmente não disponibilizam de funcionalidades para trabalhar com *JSON*, mas há várias bibliotecas em código aberto (open-source) *JSON* para *Java* e outras linguagens.

O *JSON* foi projetado como um formato de troca de “dados”, e o *XML* foi projetado primeiramente como um formato de intercâmbio de documentos. A maioria das linguagens de programação não mapeiam diretamente as estruturas de dados contidas nas construções em *XML* – já que a distinção entre elementos e atributos não está presente nas estruturas de dados de grande parte das linguagens de programação (DEITEL; DEITEL, 2012, A, p. 1320).

O *JSON* e o *XML* diferenciam-se no processo de leitura de strings e números. No *JSON* não é permitido que strings e números estejam expressos por conta própria em um agrupamento de dados em um arquivo. Assim, torna-se obrigatório a composição de encapsulamento (*getters* e *setters*) dos tipos de dados, que são passados via *web service*. Esse encapsulamento dos tipos de dados deve ser feito tanto no lado da aplicação *web service* quanto no lado da aplicação cliente. Dessa forma toda vez que uma aplicação cliente invocar o *web service*, um objeto é criado contendo os tipos de dados encapsulados a serem comunicados. Em seguida, um objeto da biblioteca *Gson* é criado, e chama-se o método “*toJson*” para que converta o objeto que encapsula os tipos de dados em uma representação de *String* em *JSON*. Essa *String* é retornada, como resposta, do *web service*, à aplicação cliente (DEITEL; DEITEL, 2012, A, p. 1322).

2.1.2.1 Biblioteca Gson

A biblioteca *Gson* é uma biblioteca que pode ser usada para converter Objetos *Java* em suas representações em *JSON* (GSON, 2012).

2.1.3 Publicação de Mensagem do Web Service na Tela do Android

Na publicação dos tipos de dados resultantes da requisição feita ao *web service* na tela *UI* (*User Interface*) do dispositivo móvel (aplicação cliente), o sistema operacional *Android* precisa de um tratamento peculiar que difere-se de sistemas operacionais *desktop*, como *Windows* e *Linux*. Nesses dois sistemas operacionais *desktop*, a publicação dos tipos de dados retornado por um *web service* é feito sem a necessidade de utilização de *threads*.

Porém, para publicar uma mensagem retornada pelo método *web service* na tela do dispositivo *Android*, é necessário tratar a atualização da interface gráfica, levando em consideração a *thread* principal que controla a tela do *Android*. Até a versão 2.2 do *Android* (*Froyo - API* 8) utiliza-se a classe *android.os.Handler*, que permite o envio ou agendamento uma mensagem para ser executada depois de um intervalo de tempo. Isso é feito por uma handler, que está vinculada à thread principal da plataforma *Android*." (LECHETA, 2010. p. 353). Na atualização da interface gráfica da tela, uma *thread* diferente é gerenciada pela classe *Handler* para realizar algum processamento assíncrono ao da thread principal, que controla a tela.

2.1.4 Publicação de Mensagem do Web Service na Tela do Android via AsyncTask

A partir da versão 2.3 do *Android* (*GINGERBREAD – API* 9) foi introduzida a classe *AsyncTask*. Igualmente à classe *Handler*, essa classe permite realizar operações em segundo plano em relação à thread principal do sistema *Android* e publica os resultados na *thread UI* sem ter que manipular *threads* e/ou *handlers* (*AsyncTask*, 2012). Os detalhes de manipulação de threads são gerenciados pela classe *AsyncTask*. Adicionalmente, a classe *AsyncTask* gerencia a comunicação de resultados gerados dentro dela à *thread* da *UI* (DEITEL; DEITEL; MORGANO, B, 2012, p. 263).

As regras a serem seguidas para utilização apropriada da classe *AsyncTask* são (AsyncTask, 2012):

1. A *AsyncTask* deve ser carregada na *thread* da *UI*;
2. A instância de *AsyncTask* deve ser criada na *thread* da *UI*;

3. A execução de *AsyncTask* deve ser invocada na *thread* da *UI*;
4. A *AsyncTask* pode ser executada somente uma vez (uma exceção será lançada se uma segunda execução é feita).

Na utilização da classe *AsyncTask*, deve-se aninhá-la na classe *Activity* do *Android*, responsável por publicar a mensagem retornada pelo *web service*. Cada classe *Activity* do *Android* representa uma única tela *UI* de interação com o usuário. A classe *AsyncTask* é ativada ao instanciá-la e, logo em seguida, chamando seu método "*execute*" (DEITEL; DEITEL; MORGANO, 2012, B, p. 271). Internamente, na classe *AsyncTask* há o método "*doInBackground*", onde "é criado o acesso ao *web service* no local endereçado na *URL* instanciada no próprio método "*doInBackground*" (DEITEL; DEITEL; MORGANO, 2012, B, p. 449). Nesse método, "*doInBackground*", também, é feita a conversão da *JSON string* para um objeto de uma classe genérica (*Class*<T>), contendo os mesmos tipos de dados encapsulados (*getters* e *setters*) no lado do *web service*. O método "*onPostExecute*" possui a função de publicar na tela *UI* do *Android* a mensagem convertida no objeto da classe genérica (ANDROID DEVELOPERS, 2012).

2.1.5 Consumo do Rest-based Json Web Service

No consumo do *REST-Based JSON web service*, a aplicação cliente é responsável por recuperar a mensagem desse *web service*. Para isso uma *String* de uma *URL* (*Uniform Resource Locator*) é usada para invocá-lo. A partir dessa *String* de *URL* é criada um objeto. Desse objeto utiliza-se o método *openStream* para invocar o *web service.* Em seguida, passa-se esse objeto de *URL* para um *InputStream*. É por meio do *InputStream* que o cliente pode ler a mensagem do *web service*. Esse *InputStream* é empacotado em um objeto *InputStreamReader* para ser passado como argumento no método "*fromJson*" da classe *Gson*, que recebe como argumentos uma instância do objeto *InputStreamReader*, e um objeto de uma classe genérica. Esse método é responsável pela passagem da instância do objeto *InputStreamReader* para uma *JSON String*, e pela conversão dessa *JSON String* em um objeto de uma classe genérica idêntica ao contido nos tipos de dados encapsulados (*getters* e *setters*) no lado do *web service* (DEITEL; DEITEL, 2012, A, p. 1324).

2.2 JAVA SERVER FACES (JSF)

Para acessar os dados dos usuários registrados no sistema do protótipo de serviço de entregas e coletas, foi desenvolvido um aplicativo web, baseado na tecnologia *Java Server Faces* (*JSF*). "*JSF* é um *framework* de aplicativos *web* baseado em *Java* (*Java-based web application framework*) que destina-se a simplificar o desenvolvimento integrado de *UIs* em *web*" (Java Server Faces – Wikipedia, 2012). Essa tecnologia é utilizada para "a construção de interfaces do lado do servidor do usuário" (Java Server Faces, 2012).

Por meio desse aplicativo *web*, além da possibilidade de acessar os dados dos usuários do sistema, que estão armazenados em um banco de dados instalado em um servidor, é possível adicionar, atualizar e excluir dados dos usuários.

2.2.1 PrimeFaces

Foi utilizado o componente do *JSF*, *PrimeFaces*, no desenvolvimento do aplicativo *web*, para enriquecer o *design* da *UI*.

"*PrimeFaces* é um componente de *JSF* em código aberto (*open-source*) com várias extensões" (ÇIVICI, 2012, p. 11). Além de ser uma biblioteca leve que requere apenas um *jar* (*Java Archive*), não requere dependências (PRIMEFACES, 2012). Para utilizá-lo adiciona-se o *primefaces-3.3.jar* (disponível para download em: <http://primefaces.org/downloads.html>) no *classpath* do projeto e importando-se o *namespace beans* (PRIMEFACES, 2012).

2.3 SERVIDOR PRIVADO VIRTUAL (VIRTUAL PRIVATE SERVER – VPS)

Para a fase de testes do sistema do protótipo, foi contratado o serviço de um provedor de Servidores Privados Virtuais (*Virtual Private Servers – VPS*). O aplicativo *web service* foi instalado no *VPS*, tendo em mente a possibilidade de testar o funcionalidade do sistema do protótipo ao ar livre.

2.3.1 Java Development Kit (JDK) e Java Runtime Environment 7 (JRE)

JDK 7 é um superconjunto do *JRE* 7 (*Java Runtime Environment*), e contém tudo o que está no *JRE* 7. O *JDK* 7 também contém ferramentas, como os compiladores e depuradores, que são necessários para o desenvolvimento de *applets* e aplicações. O *JRE* 7 fornece as bibliotecas, o *Java Virtual Machine* (*JVM*), e outros componentes para executar *applets* e aplicativos escritos na linguagem de programação *Java* (JAVA SE DOCUMENTATION - 2012).

2.3.2 PostgreSQL

O *PostgreSQL* é um sistema de banco de dados objeto-relacional em código aberto (ABOUT POSTGRESQL – 2012). Este sistema de banco de dados é responsável pelo armazenamento dos dados dos usuários do protótipo do sistema de entregas e coletas. Foi selecionado este sistema de banco de dados por ser uma ferramenta provida pela empresa provedora do *VPS*, e por ser comumente recomendado pelos professores do curso de especialização em tecnologia *Java* para a realização dos trabalhos das disciplinas deste curso.

2.3.3 GlassFish

"*GlassFish* é um servidor de aplicações em código aberto compatível com *Java Enterprise Edition*" (*JEE*) (GLASSFISH QUICK START GUIDE – 2012). No *GlassFish* foi implementado o aplicativo *web service* do protótipo do sistema de serviço de entregas e coletas.

O *Administration Console* do *GlassFish* (por padrão configurado para acesso na porta 4848) facilita a configuração com o banco de dados *PostgreSQL*, por meio da criação de um *Connection Pool*, que permite ao servidor o gerenciamento de um número limitado de conexões de banco de dados e o compartilhamento dessas conexões de acordo com a demanda de solicitações" (DEITEL; DEITEL, 2012, A, p. 1279).

Adicionalmente, o *Administration Console* facilita na conexão com o banco de

dados pelo *web service*. Para isso, um *Data Source Name* deve ser criado, associando-o ao *Connection Pool* que gerencia as conexões com o banco de dados. O *Data Source Name* deve ser especificado como um *JNDI* (*Java Naming* and *Directory Interface*), que é uma tecnologia para localizar componentes de aplicações (como o banco de dados do sistema do protótipo) (DEITEL; DEITEL, 2012, A, p. 1281) Para utilizá-lo, declara-se o *Data Source Name* na classe do aplicativo que interage com o banco de dados (DEITEL; DEITEL, 2012, A, p. 1282).

2.4 ANDROID

Android é a plataforma para dispositivos móveis mais popular do mundo desenvolvido pela empresa *Google* (ANDROID, 2012), sendo, portanto, esta a plataforma selecionada para o uso neste projeto.

A seguir é feita uma apresentação sobre o arquivo *AndroidManifest.xml* e a classe *Activity,* pois são importantes para a configuração e programação nesta plataforma.

2.4.1 Arquivo AndroidManifest.xml

O arquivo *AndroidManifest.xml* é o âmago de um aplicativo Android, pois contém todos as configurações essenciais para a execução de um aplicativo. É um arquivo obrigatório e sempre deve estar localizado na raiz do projeto do aplicativo (LECHETA, 2010, p. 74).

2.4.2 Classe Activity

A classe *Activity* é a tela da aplicação que se apresenta graficamente ao usuário do aplicativo *Android*. Essa classe é responsável pelo controle do estado e eventos da tela” (LECHETA, 2010, p. 57). Toda classe construída para representar a tela de um aplicativo *Android* deve herdar (*extend*) da classe *Activity*.

2.5 GOOGLE MAPS

A plataforma *Android* possui total integração com os serviços de geolocalização e mapeamento da aplicação *Google Maps*. Nesta seção apresenta-se as classes principais que provêem os serviços do *Google Maps*, utilizados no aplicativo *Android* do protótipo do sistema de entregas e coletas.

2.5.2 Classe MyLocationOverlay

A classe *MyLocationOverlay* é um objeto gráfico de sobreposição (*overlay*) desenhado no mapa para mostrar a localização física do usuário em um mapa (2. ANDROID DEVELOPERS, 2012).

2.5.3 Classe MapView

A classe *MapView* é um view que exibe um mapa (com os dados obtidos a partir do serviço *Google Maps*). O mapa pode ser exibido de várias maneiras: *setSatellite* (*boolean*), *setTraffic* (*boolean*), e *setStreetView* (*boolean*) (ANDROID DEVELOPERS, 2012).

Um *MapView* só pode ser construído por uma classe que herda da classe *MapActivity*. Isso se explica porque o *Mapview* depende de threads que acessam a rede e sistema em segundo plano (ANDROID DEVELOPERS, 2012).

2.5.4 Classe LocationManager

A classe *LocationManager* fornece acesso aos serviços de localização do sistema. Os serviços de localização permitem que aplicativos obtenham atualizações periódicas da localização geográfica do dispositivo (ANDROID DEVELOPERS, 2012).

2.5.5 Classe LocationListener

A classe *LocationListener* é usada para receber notificações da classe *LocationManager* quando a localização geográfica do dispositivo for alterada (ANDROID DEVELOPERS, 2012).

2.5.5.1 Classe Location

A classe *Location* representa uma localização geográfica, contendo uma latitude, uma longitude, um *timestamp UTC*, e, opcionalmente, informações sobre altitude, velocidade (ANDROID DEVELOPERS, 2012).

2.5.5.2 Classe Geocoder

A classe *Geocoder* trata da geocodificação e geocodificação reversa. A geocodificação é o processo de transformar um endereço de uma rua ou outra descrição de uma localização em um sistema de coordenadas (latitude, longitude). A geocodificação reversa é o processo de transformar coordenadas (latitude, longitude) em um endereço (ANDROID DEVELOPERS, 2012).

2.5.5.3 Classe Context

A classe *Context* é uma interface para informações globais sobre um ambiente de aplicação. Esta é uma classe abstrata, cuja implementação é fornecida pelo sistema *Android*. Ele permite acesso a recursos específicos de aplicativos e classes (ANDROID DEVELOPERS, 2012).

2.5.5.4 Classe GeoPoint

A classe *GeoPoint* é imutável, e representa um par de latitude e longitude,

armazenados como coordenadas em microgrаus (*microdegrees*) do tipo *int* (*ANDROID* DEVELOPERS, 2012). Obtém-se as coordenadas em microgrаus multiplicando-se a latitude e longitude por 1.000.000 (1E6). Um exemplo da utilização da latitude e longitude em microgrаus encontra-se na seção 3.4.5.1.1 Classe Geocoder (geocodificação).

3 DESENVOLVIMENTO DO PROTÓTIPO

Neste capítulo descreve-se a integração das tecnologias estudadas no capítulo anterior, expondo a construção do protótipo do sistema. As tecnologias descritas são *web service* (seção 3.1), servidor virtual privado (seção 3.2), *SMS* (seção 3.3), aplicativo *Android* (seção 3.4), tratamento de ausência de conectividade com o *web service* (seção 3.5). Adicionalmente, apresenta-se diagramas de sequência (seção 3.6) do aplicativo *Android* do solicitande de serviço e do motoboy, e diagramas de classes (seção 3.7) do aplicativo instalado no servidor virtual privado e do aplicativo *Android.*

Além de ter-se pensado em desenvolver um protótipo de um sistema que auxilie o serviço de entregas e colegas dos motoboys, procurou-se no desenvolvimento desse protótipo contribuir junto à comunidade acadêmica com um aplicativo para dispositivos móveis *Android*, que se integre, via rede sem fio, com um aplicativo hospedado em um Servidor Privado Virtual (*VPS*). Para isso foi utilizado uma solução, conhecida como *web service*, que permite a comunicação entre dois aplicativos. O *web service* é responsável, principalmente, pela comunicação de dados de geolocalização, dos usuários do sistema, providos pela aplicação *Google Maps*. Adicionalmente, o *SMS* (*Short Message Service*) foi selecionado para a comunicação de envio de mensagens. As mensagens de *SMS* são enviadas para a solicitação de serviço de entrega ou coleta feito pelo solicitante ao motoboy, e envio de confirmação de serviço de entrega ou coleta pelo motoboy ao solicitante.

3.1 WEB SERVICE

O tipo de *web service* utilizado para a comunicação entre o aplicativo do dispositivo móvel e o aplicativo hospedado no servidor foi o *REST-Based* (*Representational State Transfer*) *JSON Web Service*. Optou-se por este tipo de *web service* pelo fato de ter sido o serviço que mais adequou-se na comunicação entre os dois aplicativos.

Primeiramente, foi feita uma análise sobre qual tipo de *web service* seria mais adequado na integração de um aplicativo hospedado em um servidor e outro aplicativo instalado em um dispositivo móvel.

Após a leitura da seção chamada “*SOAP* ou não *SOAP*, eis a questão”, do livro “*Android* em Ação”, foi tomada a decisão de descartar o *web service* do tipo *SOAP-Based*.

O tipo de *web service*, *REST-Based XML*, que utiliza a *API Java*, *JAX-WS* (*Java API* for *XML Web Services*), não foi utilizado por ter falhado ao ser executado no lado do cliente da aplicação, que neste caso é o aplicativo instalado no *Android*.

3.1.1 Rest-based Json Web Service

Ao realizar testes sobre o *REST-Based JSON web service*, obteve-se sucesso na comunicação da aplicação *web service*, hospedada no servidor *Glassfish*, com o aplicativo cliente instalado no *Android*. Contudo, essa comunicação ocorreu somente na versão 4.0 do *Android* (*ICE_CREAM_SANDWICH - API* 14). Já na versão 2.2 do *Android* (*FROYO - API* 8), a biblioteca Gson (*com.google.gson.Gson*) não foi reconhecida pelo aplicativo *Android*.

Já na versão 4.0 do *Android*, a biblioteca *Gson* foi reconhecida com sucesso. Consequentemente, foi esse o tipo de *web service* adotado para fazer a comunicação entre o aplicativo do dispositivo móvel e o aplicativo hospedado no servidor.

3.1.2 Utilização dos Tipos de Dados Retornados pelo Web Service

Os tipos de dados encapsulados, contidos na classe genérica, para utilização no *web service* do protótipo são:

1. um tipo de dado *int*, para o atributo 'fone';
2. um tipo de dado *String*, para o atributo 'endereco';
3. um tipo de dado *String*, para o atributo 'usuario'.
4. Um tipo de dado *Timestamp*, para o atributo 'hora'.

Esses quatro tipos de dados são retornados como resultado da requisição feita pelo aplicativo cliente *Android* ao aplicativo *web service*.

No contexto do sistema do protótipo, tanto o aplicativo *Android* instalado no

dispositivo do motoboy, quanto o aplicativo *Android* instalado no dispositivo do solicitante, fazem a requisição ao aplicativo *web service*.

Esses quatro tipos de dados são utilizados da seguinte forma pelo aplicativo *Android* instalado no dispositivo do motoboy:

1. tipo de dado *int*, para o atributo 'fone', é utilizado para que o motoboy envie uma mensagem *SMS* aceitando a solicitação de serviço ao solicitante;
2. tipo de dado *String*, para o atributo 'endereço', é utilizado para apontar a localização de serviço de entrega e coleta no *Google Maps*;
3. tipo de dado *Timestamp*, para o atributo 'hora', é utilizado para registrar a hora em que foi enviada a solicitação de serviço ao motoboy. Essa hora é utiliza da cancelar o serviço de entrega ou coleta ao passar-se dez minutos do envio da solicitação do serviço. Nesse caso, passados os dez minutos da solicitação, o motoboy ao abrir o aplicativo Android será avisado por uma mensagem do tipo *pop-up* que a solicitação do serviço de entrega ou coleta foi cancelada.

No aplicativo *Android* instalado no dispositivo do solicitante, ao receber o resultado de requisição ao *web service*, somente um tipo de dado é utilizado:

- tipo de dado *int*, para o atributo 'fone' do motoboy, é utilizado para que o solicitante envie uma mensagem *SMS* solicitando o serviço ao motoboys.

3.1.2.1 Tipo de dado enviado ao web service para processamento:

- tipo de dado *String*, para o atributo 'endereço', é enviado como parâmetro ao *web service* para que seja salvo no banco de dados (*PostgreSQL*). Como explicado acima, esse endereço é, posteriormente, utilizado pelo aplicativo instalado no dispositivo *Android* do motoboy para apontar a localização do serviço de entrega ou coleta no mapa.

3.2 VIRTUAL PRIVATE SERVER (VPS)

O sistema operacional instalado no *VPS* é um *Linux Cent OS* 6 (x86_64 *bit*), 650 *MB RAM*, e CPU 1.8 GHz. A empresa provedora do serviço *VPS* chama-se *MochaHost*. O preço pago por um plano com duração de seis meses foi R$ 263 (US$ 142, com cotação do dólar comercial no valor de R$ 1,85, em 15/04/2012). A empresa provedora do *VPS* proveu a instalação do *Java Virtual Machine* (*JVM*), *Java Development Kit* (*JDK*), *Java Runtime Environment* (*JRE*), e o sistema de banco de dados PostgreSQL. O *IP* do *VPS* utilizado para o sistema do protótipo é: 204.93.157.62. Acessa-se os *web services* instalado no *VPS* pela porta 8080 do servidor *GlassFish*.

Optou-se pela utilização do *VPS* para dispor de uma ferramenta que permitisse testar ao ar livre a comunicação dos aplicativos *Android* do solicitante de serviço e do motoboy por meio do aplicativo *web service* instalado no *VPS*.

3.2.1 GlassFish

O servidor *GlassFish* não foi instalado no sistema *Linux CentOS* do *VPS* pela empresa provedora do serviço. Ao invés do GlassFish, a empresa provedora havia instalado o *Apache Tomcat*, que é uma implementação de *software* em código aberto das tecnologias *Java Servlet* e *JavaServer Pages* (APACHE TOMCAT – 2012). Porém, decidiu-se instalar manualmente o *GlassFish* no sistema *Linux CentOS* do *VPS*, em razão de fornecer o *Administration Console*, que é uma interface no browser para configuração, administração e monitoramento do servidor *GlassFish* (ORACLE, 2012. p. 1-4).

3.3 SMS (SHORT MESSAGE SERVICE)

No protótipo do sistema de entregas e coletas para motoboys há somente dois usuários, o motoboy e o solicitante de serviço. Na utilização do protótipo do sistema, a parte principal do ciclo de utilização do sistema se dá no momento em que o solicitante envia um *SMS* ao motoboy solicitando o serviço de entrega ou coleta. Esse SMS é enviado no momento em que o solicitante faz a solicitação de serviço por meio do

aplicativo *Android* instalado no dispositivo do solicitante.

Um *SMS*, também, é enviado quando o motoboy confirma a solicitação de serviço de entrega ou coleta na localização desejada, por meio do aplicativo *Android* instalado no dispositivo.

3.3.1 Funcionamento do *SMS* no Aplicativo *Android*

Primeiramente, declarou-se no arquivo *AndroidManifest.xml* aplicativo Android a permissão, *SEND_SMS*, para que o aplicativo *Android* tenha permissão enviar mensagens *SMS* (LECHETA, 2010, p. 554):

- <uses-permission android:name="android.permission.SEND_SMS" />

Foi utilizada uma classe, chamada *SMS*, fornecida pelo livro "*Google Android: aprenda a criar aplicativos para dispositivos móveis com o Android SDK*", do autor Ricardo Lecheta. Essa classe contém métodos para receber e enviar um *SMS*.

Para enviar um *SMS*, a classe *SMS* dispõe do método "*enviarSms(contexto, destino, mensagem)*". O primeiro parâmetro "contexto" é uma classe do pacote android.content, que provê acesso à informação sobre o ambiente o qual o aplicativo está rodando e permite o acesso à vários serviços *Android* (DEITEL; DEITEL; MORGANO, 2012, B, p. 132). O segundo parâmetro recebe o número para o envio de um *SMS*. E o terceiro parâmetro recebe o texto da mensagem.

3.4 APLICATIVO ANDROID

Tanto o aplicativo *Android* do solicitante de serviço quanto o aplicativo *Android* do motoboy foram construídos com a aplicação *Google Maps*, a qual está integrada à plataforma *Android*. A aplicação *Google Maps* é um componente fundamental do protótipo do sistema, pois o sistema como um todo circunda as funcionalidades do *Google Maps*.

3.4.1 Levantamento de Requisitos

- permitir requisição de serviço via *web service* pelo dispositivo móvel;
- utilizar mensagem *SMS* para solicitação e confirmação de serviço de entregas e coletas;
- utilizar serviço de mapeamento do *Google Maps* para envio e retorno de localização.

3.4.2 Diagrama de Casos de Uso

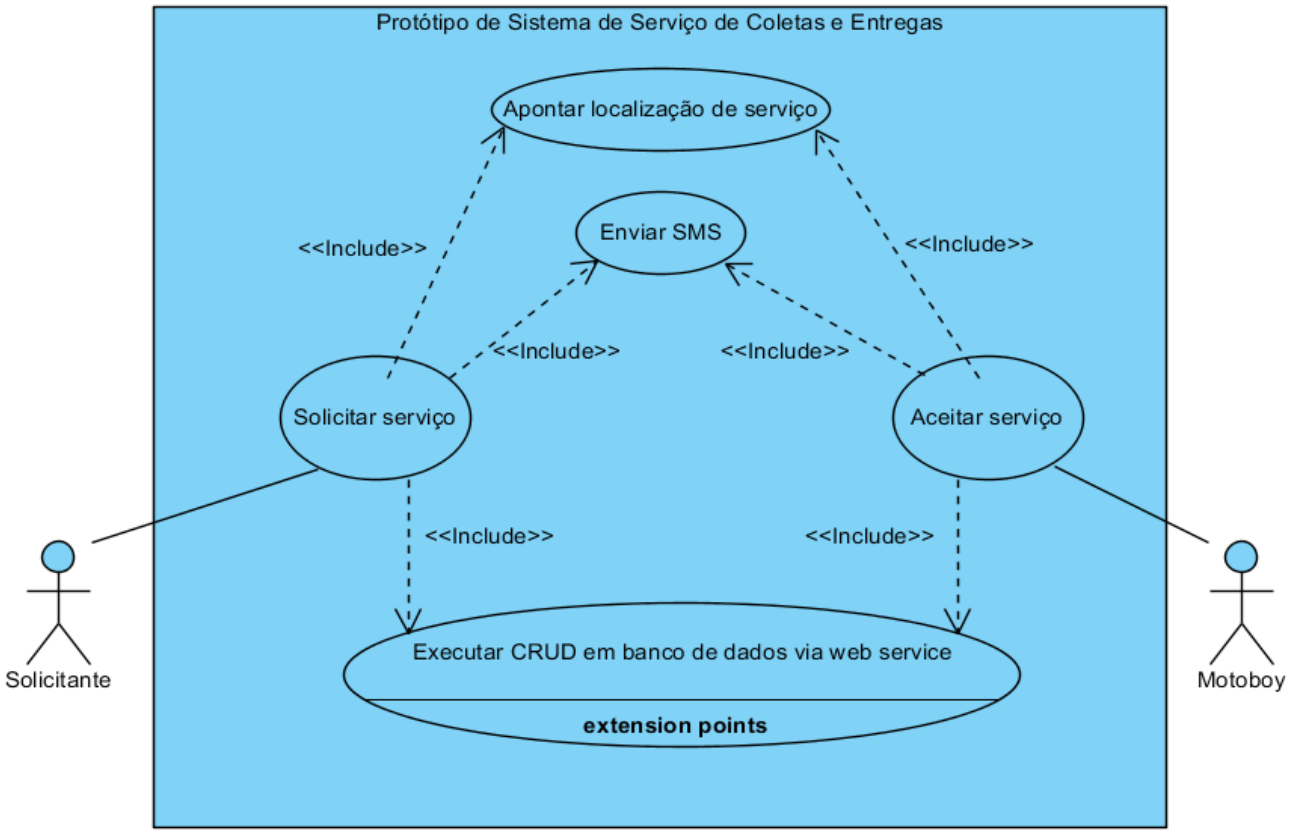

Figura 5 – Diagrama de casos de uso do protótipo do sistema de entregas e coletas para motoboy

O solicitante de serviço, ao abrir o aplicativo *Android*, tem a possibilidade de editar o endereço publicado no campo de texto da tela do aplicativo. Esse endereço, que

representa a localização aproximada do dispositivo móvel, é publicado automaticamente no momento em que o solicitante de serviço abre o aplicativo. Após a edição do endereço, o solicitante localiza no mapa o local de serviço ao pressionar o botão “Localizar” na tela do aplicativo. O local de serviço é apontado no mapa com um ícone. Ao pressionar o botão “Enviar”, o solicitante faz a solicitação de serviço ao motoboy. Essa solicitação é feita por meio do envio de uma mensagem de *SMS* ao motoboy.

O motoboy, ao abrir o aplicativo *Android*, é apresentado a uma tela, que publica automaticamente o endereço da localização do serviço de entrega ou coleta, e, também, que aponta automaticamente no mapa o local do serviço com um ícone. Ao pressionar o botão “Aceitar”, o motoboy aceita o serviço de entrega ou coleta enviando uma mensagem de *SMS* ao solicitante confirmando o serviço.

O solicitante de serviço, ao abrir o aplicativo *Android*, automaticamente visualiza sua localização atual, sinalizada por um ícone centralizado no mapa da aplicação *Google Maps*. A classe responsável por mostrar a localização atual do solicitante de serviço é chamada *MyLocationOverlay*.

3.4.3 Classe MyLocationOverlay

A classe *MyLocationOverlay* é criada ao instanciar-se um novo objeto, preenchendo-se o construtor dessa classe com os seguintes parâmetros:

- uma *Activity*, representada pela tela do aplicativo *Android*, que no caso é a classe SolicitanteLocationActivity.
- mapview.getOverlays().add(myLocationOverlay);
- e um *MapView*, responsável por exibir o mapa na tela.

O ícone que sinaliza a localização atual do usuário do aplicativo que faz parte da classe *MyLocationOverlay* é desenhada no mapa ao adicionar-se a instância da classe *MyLocationOverlay* ao *MapView*:

3.4.4 Chave da API do Google Maps

Para exibir dados do *Google Maps* em um *MapView*, deve-se obter o certificado MD5. No sistema operacional *Linux*, digita-se o seguinte comando no terminal para obter o certificado MD5:

- on terminal type: keytool -v -list -alias androiddebugkey -keystore /home/elton/.android/debug.keystore -storepass android -keypass android

Em seguida copia e cola-se o certificado MD5 na página de inscrição do serviço da *API* do *Google Maps* no seguinte endereço:

- https://developers.google.com/android/maps-api-signup

Depois de ter obtido uma chave da *API* do *Google Maps*, é preciso referenciá-lo no *XML* de layout a partir de um atributo especial – *android:apiKey* – que é uma *tag* do *MapView*:

- <com.google.android.maps.MapView
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:apiKey="0CdICKRyg-SdjK8du5vjAJyHee-pXmr67y_qLbw"/>

3.4.5 Activity

Há duas Activities que fazem parte do protótipo do sistema de auxílio a entregas e coletas para motoboys. A primeira *Activity*, chamada SolicitanteLocationActivity, é uma tela utilizada pelo solicitante de serviço. A segunda *Activity*, MotoboyLocationActivity, é uma tela utilizada pelo motoboy. Essas duas *Activities* utilizam a funcionalidade da classe *MyLocationOverlay* para sinalizar no mapa a localização atual do usuário.

3.4.6 Classe SolicitanteLocationActivity

A classe SolicitanteLocationActivity, que representa a tela do aplicativo *Android* do solicitante de serviço, utiliza uma instância da classe *LocationManager.* Essa instância obtém atualizações periódicas da localização geográfica do dispositivo. Utiliza-se uma instância da classe *LocationManager* da seguinte forma (ANDROID DEVELOPERS, 2012):

- LocationManager locationManager =
  (LocationManager) this.getSystemService(Context.LOCATION_SERVICE);

O aplicativo *Android* do solicitante de serviço faz somente uma atualização da localização geográfica do dispositivo. Isso ocorre em razão de assumir-se que o local de serviço de entrega ou coleta se encontra na localização atual do solicitante. Para obter uma atualização única da localização geográfica do dispositivo utiliza-se o método "*requestSingleUpdate*(*String provider*, *LocationListener listener*, *Looper looper*)". No aplicativo Android, esse método é preenchido da seguinte forma:

- locationManager.requestSingleUpdate(LocationManager.NETWORK_PROVIDER,
  localAtualListener, null);

O primeiro parâmetro do método "*requestSingleUpdate*" é preenchido com "*LocationManager.NETWORK_PROVIDER*". Esse parâmetro indica a utilização da rede 3G provida por uma operadora de serviço (o dispositivo para testar o aplicativo *Android* do protótipo utiliza a rede da operadora TIM).

O segundo parâmetro é uma instância da classe *LocalAtualListener*, que implementa a interface da classe *LocationListener*. Essa classe é utilizada para receber notificações da classe *LocationManager*, quando a localização for alterada. Como a instância da classe LocalAtualListener foi registrada no serviço de localização da classe *LocationManager* pelo método "*requestSingleUpdate*", somente uma única atualização de localização é acionada (ANDROID DEVELOPERS, 2012).

3.4.6.1 Classe LocalAtualListener

A classe *LocalAtualListener*, que foi construída para o aplicativo *Android*, implementa os métodos da interface da classe *LocationListener*. A classe LocalAtualListener, ao obter uma atualização única da localização geográfica do dispositivo, publica o endereço da localização geográfica em um *EditText* – campo editável de texto – localizado acima do mapa na tela da classe SolicitanteLocationActivity (Figura 1). Como havia sido mencionado anteriormente, essa atualização é feita somente uma vez pelo método "*requestSingleUpdate*". Assim sendo, somente o endereço será publicado somente uma vez no campo editável de texto. O *EditText* permite que o solicitante de serviço edite o endereço publicado para um endereço exato da localização do serviço de entrega ou coleta.

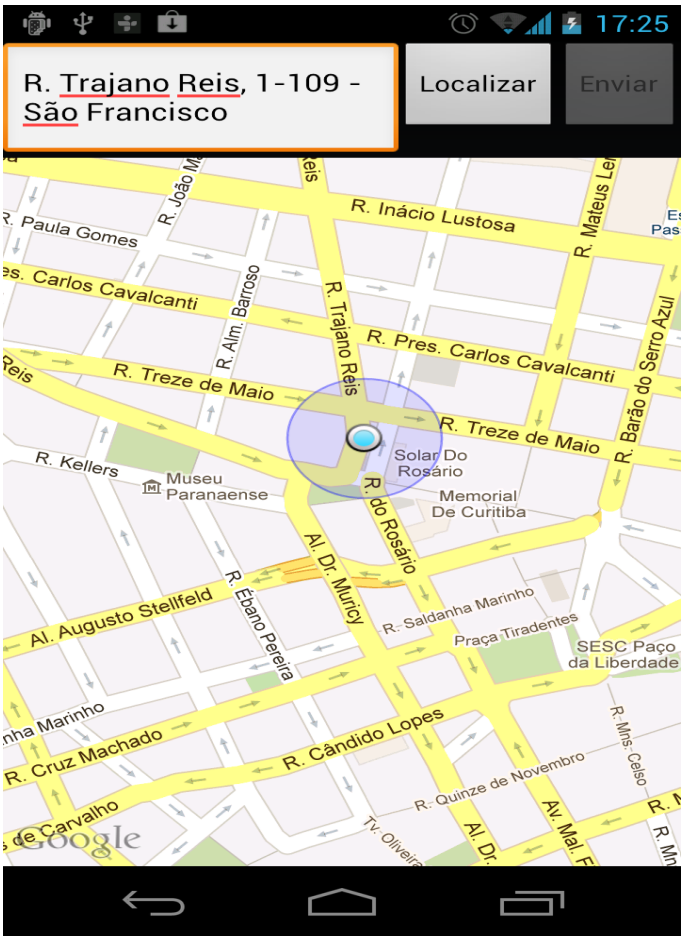

Figura 1 – Imagem da tela do solicitante de serviço apontando a sua localização atual com o ícone azul (classe SolicitanteLocationActivity)

O método “*onLocationChanged* (*Location location*)” implementada a classe *LocationListener*) é chamado quando o método “*requestSingleUpdate*” é acionado. No método *“onLocationChanged* (*Location location*)”, é feito o processo no qual obtém-se um texto (*String*) contendo o endereço equivalente à latitude e longitude da localização geográfica do dispositivo.

O parâmetro do método “*onLocationChanged*” é a classe *Location*, que contém uma latitude e longitude da localização do dispositivo. Obtém-se a latitude e longitude pelos métodos “*getLatitude*()” e “*getLongitude*()” da classe Location.

3.4.6.1.1 *Classe Geocoder (geocodificação reversa)*

Para transformar a latitude e longitude em endereço, obtidos pelos métodos “*getLatitude*()” e “*getLongitude*()” da classe Location, utiliza-se o processo de geocodificação reversa da classe *Geocoder*.

Um objeto da classe *Geocoder* deve ser criado dentro do método “*onLocationChanged*” para iniciar o processo de geocodificação reversa:

- Geocoder geocoder = new Geocoder(context, Locale.getDefault());

Preenche-se o construtor da classe *Geocoder* com dois parâmetros:

- um objeto da classe *Context*. No caso do aplicativo do protótipo o contexto é a classe SolicitanteLocationActivity.
- um objeto da classe *Locale*, que é utilizada para adequar informações às convenções da região. No caso do aplicativo utiliza-se um objeto padrão (*Locale.getDefault*()), que retorna a constante para “*en_US*”.

Utiliza-se o método “*getFromLocation*(*double latitude*, *double longitude*, *int maxResults*)” da classe *Geocoder* para obter um objeto do tipo *List<Address>* contendo um *array* de endereços da localização geográfica do dispositivo. Preenche-se os dois primeiros parâmetros do método “*getFromLocation*” com a latitude e longitude obtidos

pelos métodos “*getLatitude*()” e *“getLongitude()”,* e o terceiro parâmetro é preenchido pelo número “1”, para que seja retornado somente um endereço:

- List<Address> endereco = geocoder.getFromLocation(latitude, longitute, 1);

O endereço retornado a um objeto *List<Address>* é passado ao *EditText* para que seja publicado no campo editável de texto da tela da classe SolicitanteLocationActivity.

- String enderecoTxt = endereco.get(0).getAddressLine(0);
- textview.setText(enderecoTxt);

3.4.7 Botão “Localizar” da Classe SolicitanteLocationActivity

Os processos feitos pela classe *MyLocationOverlay*, que aponta a localização atual do usuário, e pela classe LocalAtualListener, que publica o endereço da localização atual do usuário no campo editável de texto), ocorrem automaticamente no momento em que o solicitante de serviço abre o aplicativo *Android*.

Quando o botão “Localizar” é pressionado aciona-se a funcionalidade da classe DestinoGeocodListener, a qual é descrita logo abaixo na seção 3.4.5.1. O botão também aciona os serviços do *web service*, que inclui e obtém informações do banco de dados *PostgreSQL* instalado no *VPS*.

3.4.7.1 Classe DestinoGeocodListener

A classe DestinoGeocodListener que foi construída para ser utilizada tanto pela classe SolicitanteLocationActivity quanto pela classe MotoboyLocationActivity.

Ao pressionar o botão “Localizar” na tela da classe SolicitanteLocationActivity, o método “*geocodificar*(*TextView textView*)” da classe DestinoGeocodListener é chamado. O método “geocodificar” executa duas ações:

- faz o processo de geocodificação, onde a classe *Geocoder* é utilizada para transformar o endereço em um sistema de coordenadas (latitude, longitude).

- adiciona no mapa um ícone no formato de “gota” na cor vermelha, apontando no mapa a localização do serviço de entrega ou coleta (Figura 2).

3.4.7.1.1 Classe Geocoder (geocodificação)

Na classe DestinoGeocodListener, a classe *Geocoder* é utilizada para fazer o processo de geocodificação para transformar o endereço em um sistema de coordenadas (latitude, longitude).

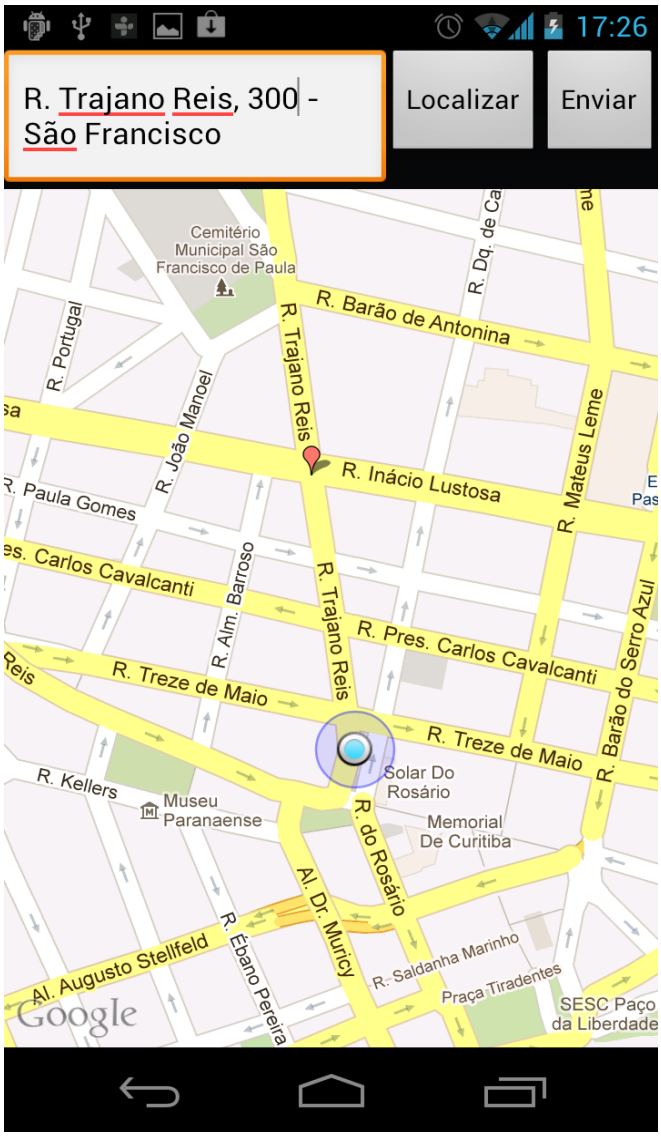

Figura 2 – Imagem da tela do solicitante de serviço apontando a sua localização atual com o ícone azul e apontando o local do serviço de entrega ou coleta com o ícone em formato de “gota” (classe SolicitanteLocationActivity)

O método “geocodificar” recebe como parâmetro o endereço escrito no campo editável de texto. Esse endereço passa pelo processo de geocodificação para transformá-lo em um sistema de coordenadas (latitude, longitude), que será utilizado para apontar no mapa, por meio do ícone em formato de “gota” (Figura 2), a localização do serviço de entrega ou coleta.

Para utilizar a classe DestinoGeocodListener, cria-se um objeto dessa classe passando cinco parâmetros ao seu construtor:

- um objeto da classe *Context*. No caso do aplicativo do protótipo o contexto é a classe SolicitanteLocationActivity;
- um objeto da classe *Drawable*, responsável por desenhar objetos gráficos na tela , representando o ícone em formato de “gota” (Figura 2) responsável por apontar no mapa a localização do serviço de entrega ou coleta;
- um objeto da classe *MapView*, representando o mapa exibido na tela;
- um objeto do tipo lista da classe *Overlay*, que representa um objeto gráfico de overlay (sobreposição) que pode ser exibido sobre um mapa;
- e um objeto da classe *TextView*, representando o campo editável de texto (*EditText*).

Para fazer o processo de geocodificação do endereço chama-se o método “*geocodificar*(*TextView textView*)”, onde passa-se como argumento o endereço preenchido no campo editável de texto, *EditText*, da tela da classe SolicitanteLocationActivity. Esse endereço deve ser passado para uma *String*:

- String enderecoTxt = textview.getText().toString();

Da mesma forma como foi feito na classe LocalAtualListener, cria-se um objeto da classe Geocoder dentro do método “*geocodificar*” da classe DestinoGeocodListener. Utiliza-se o método *“getFromLocationName* (*LocationName string*, int maxResults)” da classe *Geocoder* para que se obtenha como retorno uma lista do *Address* (*List<Address>*). Preenche-se o primeiro parâmetro do método “*getFromLocationName*” com a *String* “enderecoTXT”, e o segundo parâmetro é preenchido pelo número “1”, para que seja retornado um objeto do tipo lista contendo somente um endereço:

- List<Address> endereco = geocoder.getFromLocationName(enderecoTxt, 1);”

Passa-se a lista contendo o endereço à classe *Address* do pacote *android.location.Address*:

- Address enderecoAndroid = endereco.get(0);

Obtém-se a latitude e longitude pelos método “*getLatitude*()” e “*getLongitude*()” da classe *Address*:

- Double latitude = enderecoAndroid.getLatitude() * 1E6;
- Double longitude = enderecoAndroid.getLongitude() * 1E6;

Faz-se necessário multiplicar a latitude e longitude por 1.000.000 (1E6) para que se obtenha as coordenadas em micrograus, para que sejam passados à um objeto da classe *GeoPoint*:

- GeoPoint ponto = new GeoPoint(latitude.intValue(), longitude.intValue());

A instância “ponto” da classe *GeoPoint* é adicionada como um dos parâmetros do construtor da classe *OverlayItem*, que será descrita logo abaixo.

3.4.7.1.2 Posicionamento do ícone no mapa (Overlay)

O processo para que o ícone seja desenhado no mapa (ANDROID DEVELOPERS, 2012), para apontar o local do serviço de entrega ou coleta, inicia-se com a criação de uma instância da classe *OverlayItem*:

- OverlayItem overlayItem = new OverlayItem(ponto, enderecoTxt, fone);

Ao criar-se uma instância da classe *OverlayItem*, adicionou-se ao seu construtor os

seguintes parâmetros:

- o primeiro parâmetro recebe uma instância da classe *GeoPoint* chamada “ponto”, contendo as coordenadas (latitude, longitude) em microdegrees;
- o segundo parâmetro recebe uma *String* “enderecoTxt”, contendo o endereço do serviço de entrega ou coleta. Esse endereço aparece na tela de mensagem em forma de *pop-up* (Figura 3) ao tocar-se o dedo sobre o ícone responsável por apontar no mapa a localização do serviço de entrega ou coleta.

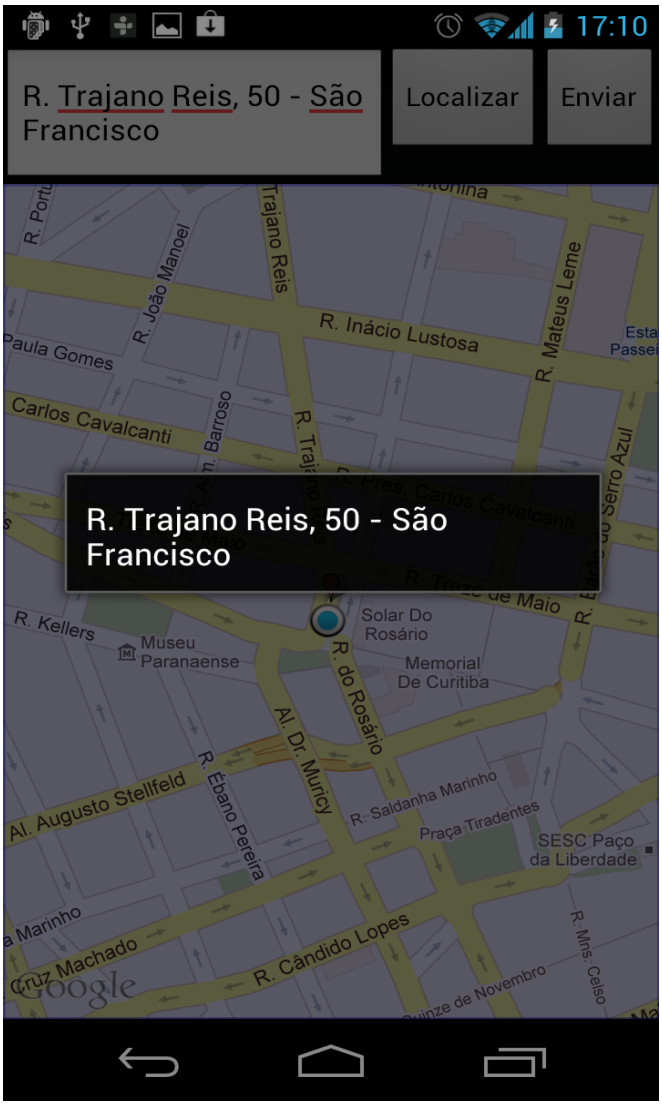

Figura 3 – Imagem da tela do solicitante de serviço mostrando uma mensagem em forma de *pop-up* (classe SolicitanteLocationActivity)

A instância “*overlayItem*”, que é um componente básico da classe *ItemizedOverlay* , deve ser adicionada a uma lista da classe *ItemizedOverlay*. A classe *ItemizedOverlay* é uma classe que consiste em uma lista de *overlayItems*. Por meio dessa classe desenha-se um ícone no mapa localizando o endereço do serviço de entrega ou coleta. Essa classe deve ser criada preenchendo-se o construtor com um objeto da classe *Drawable*, representando o ícone que aponta a localização do serviço de entrega ou coleta, e um objeto da classe *Context*, que representa a classe SolicitanteLocationActivity:

- ItemizedOverlays itemizedOverlay = new ItemizedOverlays(icone, context);

Tendo criado a instância “*itemizedOverlay*”, adiciona-se a essa lista o “*overlayItem*”:

- itemizedOverlay.addOverlay(overlayItem);

Finalmente, para que o ícone seja posicionado na localização contendo as coordenadas do endereço do serviço de entrega ou coleta, adiciona-se o “*itemizedOverlay*” a uma instância de *MapView*:

- mapview.getOverlays().add(itemizedoverlay);

Para centralizar o mapa na localização apontada pelo ícone adiciona-se a seguinte a seguinte linha ao código:

- mapview.getController().animateTo(ponto);

3.4.7.2 Web Service

Ao pressionar o botão “Localizar” aciona-se também os serviços do *web service*, que recebe duas solicitações do aplicativo *Android* do solicitante de serviço. Todo o processo de solicitação ao web service ocorre dentro do método “*doInBackground*” da classe *AsyncTask* (seção 2.1.3) que está aninhada na classe SolicitanteLocationActivity.

Primeiramente, solicita-se o envio do endereço contido no campo editável de texto (*EditText*), na parte superior da tela, para que seja armazenado no banco de dados *PostgreSQL*, que está instalado no *VPS*.

Isso é feito para que o aplicativo *Android* do motoboy possa obter via *web service* o endereço armazenado no *PostgreSQL*. Para isso preenche-se o construtor da instância da *URL* do *web service* com um parâmetro contendo uma *String* do endereço do serviço de entrega ou coleta:

- URL url = new URL(“http://204.93.157.62:8080/MapaWSVPS/” + “resources/MapaRESTJSONEndereco/” + enderecoFormatado);

O parâmetro da *URL* recebendo o endereço não aceita uma *String* contento espaços em branco. Em razão disso, substitui-se os espaços em branco por sinais de “+”:

- String endereço = “R. Trajano Reis, 50 – São Francisco”;
- String enderecoFormatado = endereco.replaceAll(" ", "+");

O conteúdo da *String* “*enderecoFormatado*” passa a conter sinais de “+” no lugar dos espaços em branco, para que seja passada como parâmetro à *URL*:

- “R.+Trajano+Reis,+50+-+São+Francisco”

A segunda solicitação ocorre para obter-se o número de celular do motoboy armazenado no *PostgreSQL*, que é utilizado para enviar uma mensagem *SMS* de solicitação de serviço ao motoboy. Para isso preenche-se o construtor da instância da *URL* do *web service* com um parâmetro contendo a String “MOTOBOY”:

- URL url = new URL(“http://204.93.157.62:8080/MapaWSVPS/” + “resources/MapaRESTJSONUsuario/MOTOBOY”);

O *web service*, ao receber a parâmetro “MOTOBOY” faz uma consulta no *PostgreSQL*, que retorna o número do celular do motoboy. Esse número é, em seguida,

retornado ao aplicativo *Android* do solicitante de serviço.

3.4.8 Botão “Enviar”

Após ter-se completado as duas solicitações feitas ao *web service*, o botão “Enviar” é habilitado dentro do método “*onPostExecute*” da classe *AsyncTask* (seção 2.1.3):

- aceitarButton.setEnabled(true);

Ao clicar no botão “Aceitar”, um *SMS* é enviado ao motoboy utilizando o método “*enviarSms*(*contexto*, *destino*, *mensagem*)” (seção 3.3):

- enviarSms(SolicitanteLocationActivity.this, foneMotoboy, mensagem);

O segundo parâmetro do método “*enviarSMS*” é preenchido pelo número do celular do motoboy obtido via *web service*. O terceiro parâmetro contém o endereço do serviço de entrega ou coleta.

3.4.9 Classe MotoboyLocationActivity

Ao abrir o aplicativo *Android*, o motoboy é apresentado pela tela da classe MotoboyLocationActivity. Automaticamente, ao abrir o aplicativo, quatro ações ocorrem:

- como ocorre, também, na classe SolicitanteLocationActivity, o ícone, da classe *MyLocationOverlay*, aponta no mapa a localização atual do usuário, que neste caso é o motoboy;
- aciona-se o serviço do *web service* para a obtenção do número do celular do solicitante de serviço e endereço do serviço de entrega ou coleta.
- aponta-se no mapa com um ícone, a localização do endereço do serviço de entrega ou coleta retornado pelo *web service*.
- e publica-se o endereço do serviço de entrega ou coleta no campo de texto

localizado na parte superior do mapa.

3.4.9.1 Web service e formatação de endereço no método “doInBackground”

O acionamento do *web service* e a formatação do endereço do serviço de entrega ou coleta retornado pelo *web service* são feitos dentro do método “*doInBackground*” da classe aninhada *AsyncTask* (seção 2.1.3).

3.4.9.1.1 *Web service no método “doInBackground”*

Para acionar o serviço do web service preenche-se o construtor da instância da *URL* com um parâmetro contendo a *String* “SOLICITANTE”:

- URL url = new URL("http://204.93.157.62:8080/MapaWSVPS/" + "resources/MapaRESTJSONUsuario/SOLICITANTE");

O *web service*, ao receber a parâmetro “SOLICITANTE” faz uma consulta no *PostgreSQL*, que retorna o número do celular do solicitante e o endereço do serviço de entrega ou coleta. Esses dados são, em seguida, retornados ao aplicativo *Android* do motoboy.

3.4.9.1.2 *Formatação do endereço no método “doInBackground”*

Na formatação do endereço faz-se dentro do método “*doInBackground*” o processo de substituição dos sinais de “+” pelos espaços em branco:

- String endereço = “R.+Trajano+Reis,+50+-+São+Francisco”;
- String enderecoFormatado = endereco.replaceAll(" ", "+");

O conteúdo da *String* “enderecoFormatado” passa a conter espaços em branco ao invés dos sinais de “+”:

- “R. Trajano Reis, 50 - São Francisco”

A formatação do endereço deve ser feita dentro do método “*doInBackground*”, pois, logo após a execução desse método, a classe aninhada *AsyncTask* executa automaticamente o método “*onPostExecute*”. Esse método deve ser usado exclusivamente para publicar resultados na *thread* da *UI*.

Dentro do método “*onPostExecute*”, é passado como parâmetro o endereço formatado para que seja publicado no campo de texto localizado na parte superior da tela do aplicativo *Android* do motoboy (classe MotoboyLocationActivity).

3.4.9.2 Classe DestinoGeocodListener

Para desenhar no mapa o ícone apontando no mapa a localização do serviço de entrega ou coleta utiliza-se novamente a funcionalidade da Classe DestinoGeocodListener (seção 3.4.5.1), que foi também utilizada pela classe SolicitanteLocationActivity do aplicativo *Android* do solicitante de serviço de entrega ou coleta.

3.4.9.3 Botão “Aceitar”

Ao pressionar o botão “Aceitar” (Figura 4) na tela da classe MotoboyLocationActivity, envia-se uma mensagem *SMS* ao dispositivo móvel do solicitante de serviço, informando a confirmação da execução do serviço de entrega ou coleta.

Caso o motoboy abra o aplicativo *Android* tendo passado dez minutos do horário em que foi feita a solicitação de serviço de entrega ou coleta, o motoboy ao abrir o aplicativo *Android* será avisado por uma mensagem do tipo *pop-up* que a solicitação de serviço foi cancelada.

O acionamento do botão “Aceitar” ou o recebimento da mensagem de cancelamento da solicitação de serviço completa o ciclo de funcionamento do protótipo do

sistema para auxílio no serviço de coletas e entregas de motoboys.

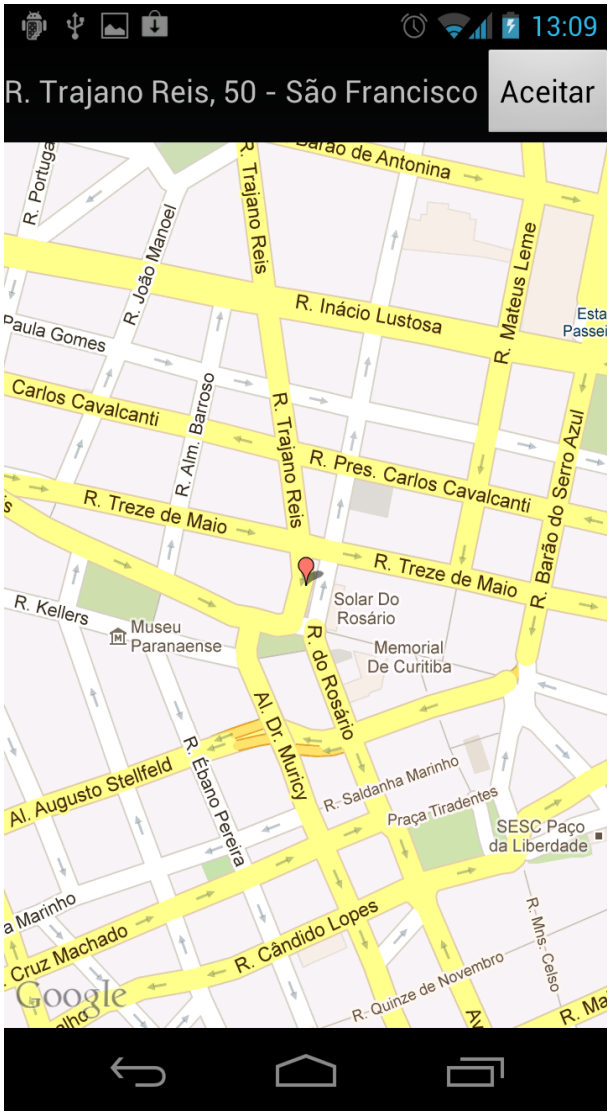

Figura 4 – Imagem da tela do motoboy apontando o local do serviço de entrega ou coleta com o ícone em formato de “gota” (classe MotoboyLocationActivity)

3.5 TRATAMENTO DE AUSÊNCIA DE CONECTIVIDADE COM O WEB SERVICE

A conectividade do aplicativo *Android* com o *web service* pode não ocorrer, caso o servidor *GlassFish* não esteja inicializado, ou caso a rede *wireless* ou rede 3G não esteja em condições favoráveis. Caso um desses casos ocorra, será lançada uma mensagem do tipo *pop-up* informando a ausência de conectividade no momento em que o aplicativo

Android do solicitante de serviço ou do motoboy faça uma requisição ao *web service*.

3.6 DIAGRAMAS DE SEQUÊNCIA

Os diagramas de sequência do uso do aplicativo *Android* tanto pelo solicitante de serviço quanto pelo motoboy são apresentados a seguir, acompanhados de uma descrição textual do processo de requisição de serviço de coleta e entrega, apresentando um resumo geral sobre o funcionamento do protótipo do sistema.

3.6.1 Descrição do Diagrama de Sequência do Solicitante de Serviço

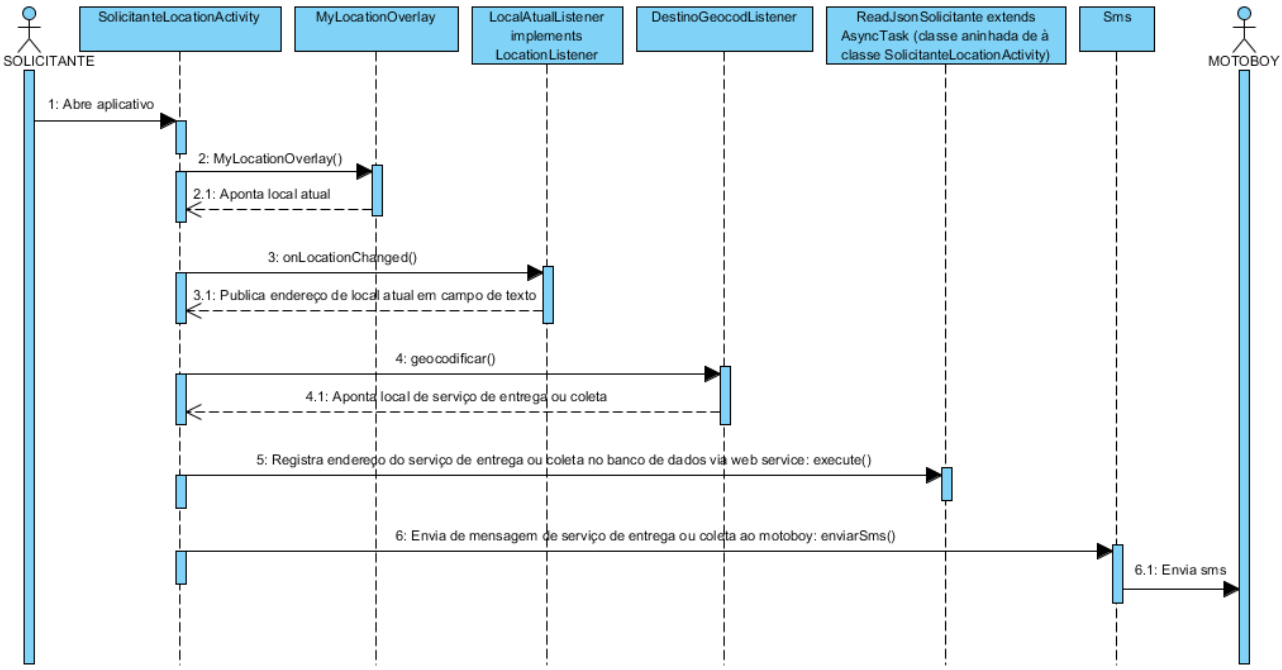

Figura 6 – Diagrama de sequência do solicitante de serviço

Duas ações ocorrem automaticamente no momento em que o solicitante de serviço abre o aplicativo *Android* de seu dispositivo móvel. A primeira ação é a adição de um ícone apontando no mapa a localização atual do solicitante feita pela instância classe da *MyLocationOverlay*. A segunda ação é a publicação do endereço da localização atual do

solicitante no campo editável de texto feita pelo método “*onLocationChanged*()” da classe LocalAtualListener.

Ao pressionar o botão “Localizar”, três ações ocorrem. A primeira ação é a adição de um ícone apontando no mapa a localização do serviço de entrega ou coleta feita pelo método “*geocodificar*()” da classe DestinoGeocodListener. A segunda ação é o registro no *PostgreSQL* do *VPS*, via web service, do endereço do serviço de entrega ou coleta feito pela classe *AsyncTask*. A terceira ação é a habilitação (*setEnabled*(*true*)) do botão “Enviar”.

Ao pressionar o botão “Enviar”, ocorre o envio de uma mensagem de *SMS* ao motoboy feito pela instância da classe *Sms*.

3.6.2 Descrição do Diagrama de Sequência do Motoboy

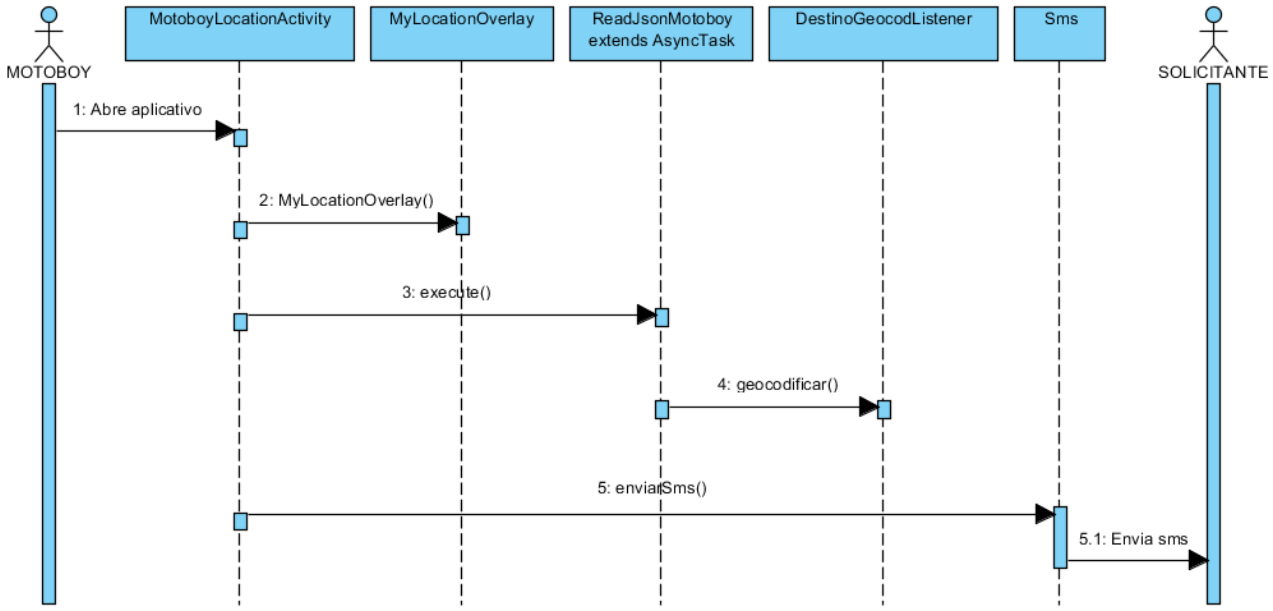

Figura 7 – Diagrama de sequência do motoboy

Quatro ações ocorrem automaticamente no momento em que o motoboy abre o aplicativo *Android* de seu dispositivo móvel. A primeira ação é a adição de um ícone apontando no mapa a localização atual do motoboy feita pela instância da classe *MyLocationOverlay*. A segunda ação é a busca de dados do solicitante de serviço armazenados no *PostgreSQL* do *VPS* feita via web service pela classe *AsyncTask*.

A terceira e a quarta ação ocorrem no método “*onPostExecute*” da classe *AsyncTask* (esse método é chamado automaticamente toda vez que o método “execute” da classe *AsyncTask* é chamado). A terceira ação é a publicação do endereço da localização do serviço de entrega ou coleta no campo de texto. A quarta ação é a adição de um ícone apontando no mapa a localização do serviço de entrega ou coleta feita pelo método “*geocodificar*()” da classe DestinoGeocodListener.

Ao pressionar o botão “Aceitar”, ocorre o envio de uma mensagem de *SMS* ao solicitante confirmando o serviço de entrega ou coleta feito pela instância da classe *Sms*.

3.7 DIAGRAMAS DE CLASSES

Nas figuras seguintes, apresenta-se diagramas de classes do aplicativo *Web Service* (Figura 8) e aplicativo *Android* (Figura 9) do protótipo do sistema.

3.7.1 Diagrama de classes do aplicativo Web Service

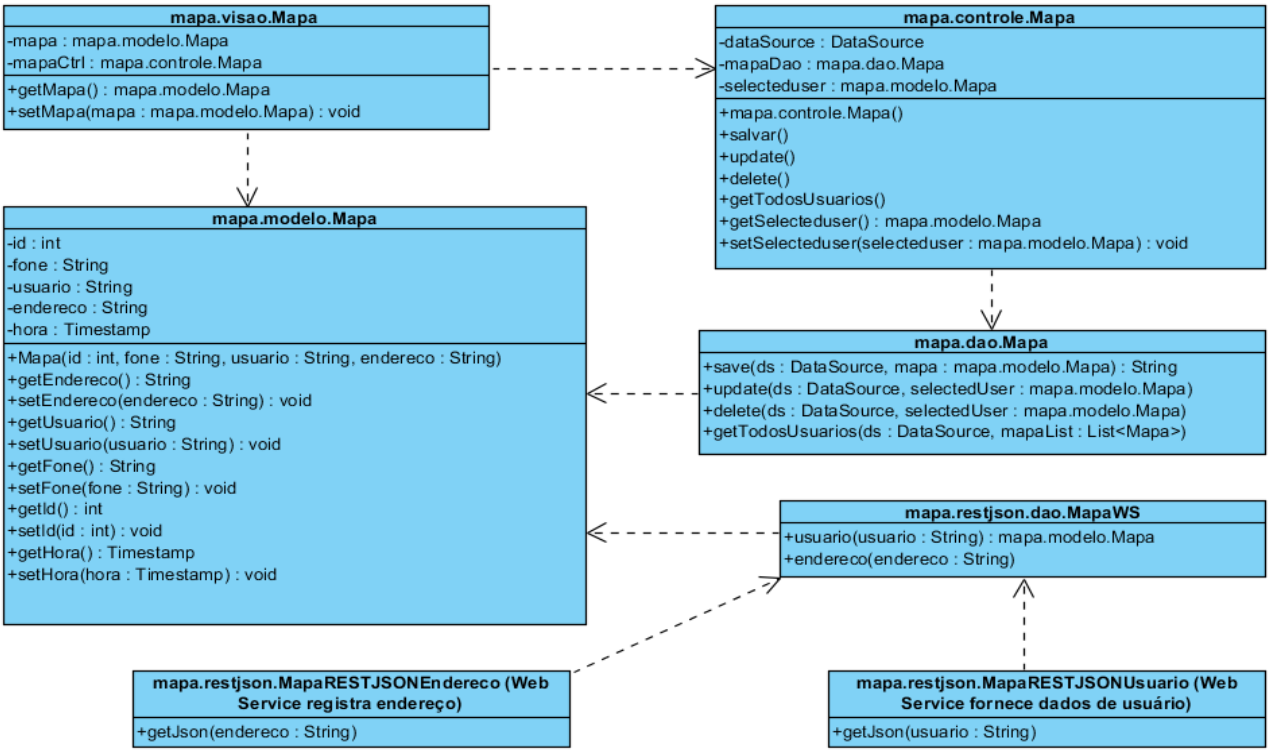

Figura 8 – Diagrama de classes do aplicativo Web Service

- a classe mapa.visao.Mapa faz a apresentação gráfica dos componentes do Java Server Faces ;
- a classe mapa.controle.Mapa faz o processamento lógico do aplicativo Java Server Faces ;
- as classes mapa.dao.Mapa e mapa.restjson.daoMapaWS fazem o processamento das requisições ao banco de dados PostgreSQL ;
- a classe mapa.restjson.MapaRESTJSONUsuario é responsável pelo acesso ao *Web Service* que recebe requisições das classes ReadJsonSolicitante e ReadJsonMotoboy do Aplicativo *Android* (Figura 8);
- a classe mapa.restjson.MapaRESTJSONEndereco é responsável pelo acesso ao *Web Service* que recebe requisições da classe ReadJsonSolicitante do aplicativo *Android* (Figura 8).

3.7.2 Diagrama de classes do aplicativo Android

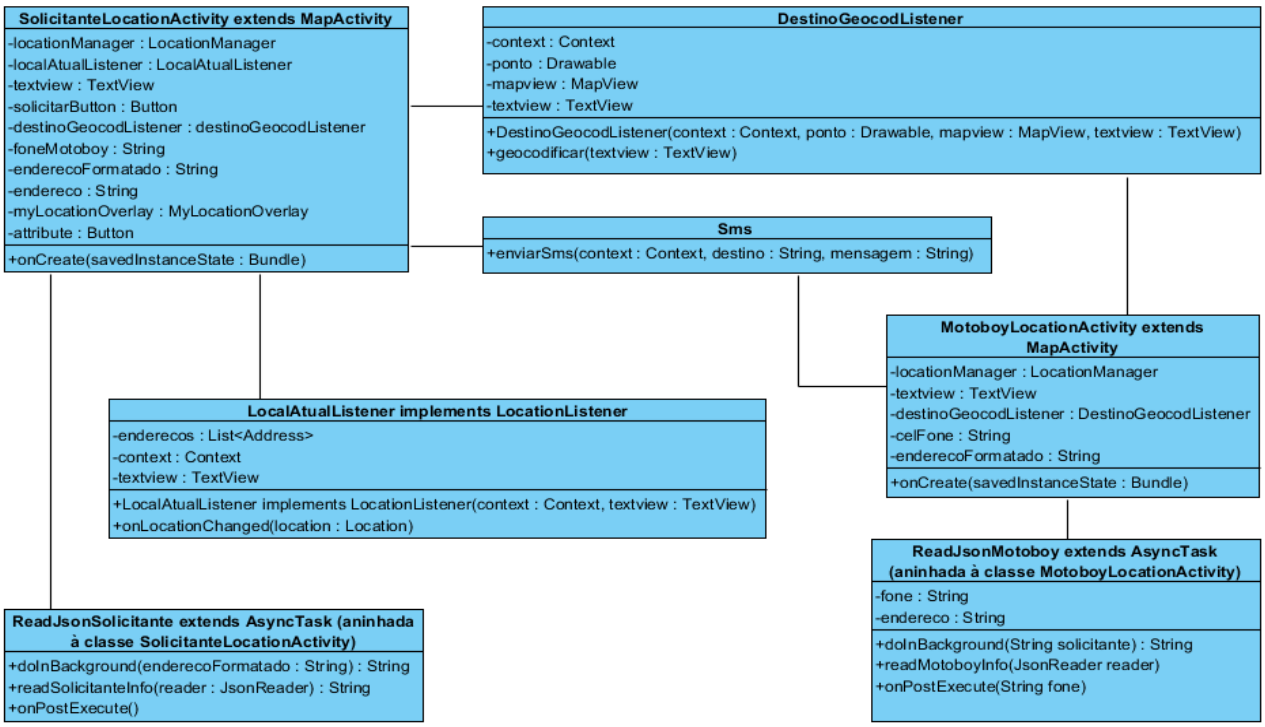

Figura 9 – Diagrama de classes do aplicativo Android

- a classe DestinoGeocodListener faz a geocodificação (transformar endereço em latitude e longitude), e adição de um ícone apontando a localização do serviço de entrega ou coleta.;
- a classe LocalAutalListener faz a geocodificação reversa (transformar latitude e longitude em endereço) da localização atual do solicitante, e publicação do endereço no EditText;
- a classe ReadJsonSolicitante é responsável pelas requisições aos Web Services:
  - mapa.restjson.MapaRESTJSONUsuario e
  - mapa.restjson.MapaRESTJSONEndereco (Figura 7) ;
- a classe ReadJsonMotoboy é responsável pela ao Web Service:
  - mapa.restjson.MapaRESTJSONUsuario (Figura 7) ;

4 CONCLUSÕES

Espera-se que o texto do Protótipo de Sistema de Entregas e Coletas apresentado acima possa contribuir para o desenvolvimento de sistemas que utilizem a aplicação do *Google Maps* da plataforma *Android* e *web services* do tipo *JSON REST-Based*.

Porém, deve-se estar ciente que o aplicativo *Android* do protótipo do sistema de entregas e coletas para motoboys é eficiente quando duas condições são satisfeitas:

- utiliza-se conexão de rede *wireless* ou 3G em condições favoráveis;
- utiliza-se o aplicativo *Android* em um local fixo, sem locomoção.

Em relação à conexão de rede, testou-se o aplicativo *Android* do protótipo em vários ambientes que disponibilizam de rede *wireless*. Nos casos em que a rede *wireless* está fraca, o aplicativo *Android* lança uma exceção com uma mensagem de falha, pois precisa de uma conexão que supra o acesso a aplicação do *Google Maps* e as requisições ao *web service*. O mesmo ocorre com a rede 3G, dependendo do ambiente, principalmente em locais fechados, em que utiliza-se o aplicativo *Android* pode ocorrer o lançamento de uma exceção com uma mensagem de falha.

Há a necessidade de utilizar-se o aplicativo *Android* em um local fixo, pois, ao testar-se o aplicativo em um automóvel em movimento, ocorre uma assincronicidade da atualização do ícone, que aponta a localização atual do usuário, com a atualização da imagem do mapa na tela. A atualização do ícone da localização atual, tanto do solicitante de serviço quanto do motoboy, é feita pela classe *MyLocationOverlay*, que fixando-se ao *GPS* de um satélite, atualiza com precisão a localização atual do usuário. Porém, a atualização da imagem do mapa na tela, que é dependente da rede 3G, não acompanha a atualização do ícone. Isso faz com que a tela mostre o ícone em uma tela vazia.

4.1 TRABALHOS FUTUROS

Tendo em vista as melhorias que devem ser feitas no protótipo do sistema, há oportunidades no sentido de adicionar recursos para torná-lo mais funcional, tais como:

- utilização do aplicativo *Android* do sistema de entregas e coletas em localidades com condições não favoráveis de conexão de rede wireless ou 3G;
- utilização do mapa da aplicação *Google Maps* em um automóvel em movimento;
- inclusão de múltiplas opções de prestadores de serviço (motoboys);
- além do envio de mensagens via *SMS* de solicitação e confirmação de serviço, implementação de envio de mensagens via rede 3G utilizando o método “sendBroadcast(intent)” da classe *BroadcastReceiver*;
- traçamento de rota no mapa do aplicativo *Android* do motoboy, delineando desde a localização geográfica atual do motoboy até o local de serviço de entrega ou coleta.

5 REFERÊNCIAS BIBLIOGRÁFICAS

ABLESON W. Frank; COLLINS Charlie; SEN Robi. Android em Ação. Rio de Janeiro: Elsevier, 2012. 3. ed.

ABOUT POSTGRESQL. Disponível em: <http://www.postgresql.org/about/>. Acesso em: 08/07/2012.

ANDROID. Disponível em: <http://www.android.com/about/>. Acesso em: 17/07/2012.

ANDROID DEVELOPERS. Disponível em: <http://developer.android.com/index.html>. Acesso em: 05/12/2011.

APACHE TOMCAT. Disponível em: <http://tomcat.apache.org/>. Acesso em: 08/07/2012.

ÇIVICI, Çağatay. Primefaces User's Guide 3.3. Turkey: Prime Teknoloji, 2012.

A. DEITEL, Paul J.; DEITEL, H. M. Java: how to program. EUA: Prentice Hall. 2012. 9 ed.

B. DEITEL, Paul J.; DEITEL, H. M.; DEITEL Abbey; MORGANO Michael. Android for Programmers: An App-Driven Approach. EUA: Pearson Education. 2012.

GLASSFISH QUICK OVERVIEW. Disponível em: <http://glassfish.java.net/public/getstarted.html>. Acesso em: 08/07/2012.

GSON. Disponível em: <http://code.google.com/p/google-gson/>. Acesso em: 07/07/2012.

JAVA SE DOCUMENTATION. Disponível em: <http://docs.oracle.com/javase/7/docs/index.html>. Acesso em: 08/07/2012.

JAVA SERVER FACES. Disponível em: <http://www.oracle.com/technetwork/java/javaee/javaserverfaces-139869.html>. Acesso

em: 07/07/2012.

JAVA SERVER FACES – WIKIPEDIA. Disponível em: <http://en.wikipedia.org/wiki/JavaServer_Faces>. Acesso em: 07/07/2012.

JAVA VIRTUAL MACHINE – WIKIPEDIA. Disponível em: <http://en.wikipedia.org/wiki/Java_virtual_machine> Acesso em: 08/07/2012.

LECHETA, Ricardo R. Google Android: aprenda a criar aplicativos para dispositivos móveis com o Android SDK. São Paulo: Novatec Editora, 2010. 2. ed.

ORACLE. GlassFish Server Open Source Edition: Quick Start Guide. EUA: Oracle. 2012.

PRIMEFACES. Disponível em: <http://primefaces.org/whyprimefaces.html>. Acesso em: 07/07/2012.

WAZLAWICK, Raul Sidney. Metodologia de Pesquisa para Ciência da Computação. Rio de Janeiro: Elsevier, 2008.