UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

G-SMS: PROTÓTIPO DE APLICAÇÃO DE ENVIO DE SMS GEOREFERENCIADAS

CARLOS ROBERTO BENDER

BLUMENAU
2011

2011/1-10

CARLOS ROBERTO BENDER

G-SMS: PROTÓTIPO DE APLICAÇÃO DE ENVIO DE SMS GEOREFERENCIADAS

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciência da Computação — Bacharelado.

Prof. Mauro Marcelo Mattos, Dr - Orientador

BLUMENAU
2011

2011/1-10

G-SMS: PROTÓTIPO DE APLICAÇÃO DE ENVIO DE SMS GEOREFERENCIADAS

Por

CARLOS ROBERTO BENDER

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

	_____
Presidente:	Prof. Mauro Marcelo Mattos, Dr – Orientador, FURB
	_____
Membro:	Prof. Francisco Adell Péricas, Mestre – FURB
	_____
Membro:	Prof. Sérgio Stringari, Mestre – FURB

Blumenau, 27 de junho de 2011

Dedico este trabalho a todos os amigos, familiares e colegas pelo incentivo, cooperação e apoio durante a realização deste.

AGRADECIMENTOS

A minha família que sempre me ajudou e me apoiou em todas as etapas de minha vida.

Aos meus amigos, pela paciência, em especial aos que sempre estiveram ao meu lado para ouvir mesmo sem entender o assunto.

Ao meu orientador, Mauro Marcelo Mattos, por ter me ajudado em todas as necessidades e pela atenção. E a todos que direta ou indiretamente me apoiaram e ajudaram para que este trabalho fosse possível.

A ambição é o puro senso de dever pois a si só não produz frutos realmente importantes para a pessoa humana, pelo contrário os frutos verdadeiros derivam do amor e da dedicação para com as pessoas e as coisas.

Albert Einstein

RESUMO

Este trabalho descreve o desenvolvimento de um protótipo de uma aplicação de envio de *SMS* georeferenciadas para *smartphone*s. Utiliza a disponibilidade de GPS e internet para obter a localização do aparelho e carregar os mapas. O usuário também pode escolher a localização utilizando a funcionalidade de *Long Press*. O *SMS* enviado pelo protótipo contém uma *URL* para o Google Maps com valores de latitude e longitude da localização. O uso da plataforma Android e da API Google Maps foram necessários para implementação deste protótipo.

Palavras-chave: Android. Georreferenciamento. SMS. Google Maps.

ABSTRACT

This paper describes the implementation of geotagged SMS application prototype for smartphones. It uses GPS and internet availability to get the device´s location and load maps. The user also can pick a location using long press functionality. The *SMS* sent contains an *URL* for the Google Maps with latitude and longitude values of the location. The use of Android platform and Google Maps Api were necessary to implement this application.

Key-words: Android. Geotagged. SMS. Google Maps.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

ADT – *Application Development Tools*

API – *Application Programming Interface*

GPS – *Global Positioning System*

IDE – *Integrated Development Environment*

JVM – *Java Virtual Machine*

LBS – *Location Based Services*

OHA – *Open Handset Alliance*

SDK – *Software Development Kit*

SMS – *Short Message Service*

UML – *Unified Modeling Language*

URL – *Uniform Resource Location*

XML - *eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

O mercado de dispositivos móveis, especialmente os celulares, está em evidência e possui um público cada vez maior, devido à facilidade de aquisição de um aparelho por qualquer pessoa. Segundo a Anatel (2010), o Brasil fechou o ano de 2009 com 173,9 milhões de telefones celulares em operação, o que representa um crescimento de 15,74% em relação à dezembro de 2008, quando existiam no país 150,6 milhões de telefones móveis em funcionamento. Portanto, um mercado totalmente aquecido.

Segundo Nemer (2006), está havendo uma rápida convergência de tecnologias no que diz respeito ao telefone móvel. A primeira geração de dispositivos móveis foi desenvolvida para satisfazer a mesma necessidade do usuário das linhas de telefones fixo: fazer e receber chamadas. No entanto, muitas funções têm sido adicionadas, tais como troca de mensagens texto/voz/multimídia, conexão Internet, tocador de música, filmadora, câmera, cartão de crédito, agenda, jornal, se estendendo a funcionalidades nunca antes imaginadas a estes pequenos dispositivos que estão evoluindo na direção dos telefones inteligentes.

Os telefones inteligentes, ou popularmente chamados de *smartphones,* agregam as mais diversas funcionalidades, mas ainda mantém as opções básicas como o serviço de mensagens curtas, no inglês *Short Message Service* (SMS). SMS é a habilidade de enviar e receber mensagens de texto entre telefones celulares, sendo que cada uma pode conter até 160 caracteres.

Os celulares estão recebendo um novo mundo de aplicações chamadas de *Location Based Systems* (LBS). Estes sistemas utilizam a informação de localização georeferenciada, para disponibilizar opções para o usuário, fornecendo informações mais relevantes ao local em que a pessoa se encontra. Segundo Virrantaus et al. (2001), LBS são serviços de informação acessados pelos dispositivos móveis através da rede móvel e utilizam a capacidade de registrar a localização atual. A maioria dos serviços LBS é capaz de obter a localização do usuário/dispositivo e utilizar essa informação para promover um serviço. Assim, os serviços que partem da posição geográfica do cliente, oferecendo a possibilidade de localizar usuários, máquinas, veículos e recursos, bem como o rastreamento dos mesmos, além de serviços sensíveis a localização. Estes sistemas, em muitos casos agregam a funcionalidade de apresentar essas informações dispostas em mapas nativos ou realizando integração com aplicações disponíveis na internet, como por exemplo, Google Maps.

Diante do exposto, foi desenvolvido um protótipo para celulares de envio de *Short*

Message Service (SMS) georeferenciadas. As mensagens contendo as informações de localização serão enviadas de um celular para outro. O dispositivo receptor terá a funcionalidade de apresentar em um mapa a localização recebida.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é desenvolver um protótipo que permite o envio de mensagens georeferenciadas a partir de um celular para outro.

Os objetivos específicos do trabalho são:

a) disponibilizar uma aplicação em dispositivos móveis para edição, associação de latitude e longitude do local de envio e o envio de SMS na plataforma Android;
b) realizar integração com a Google Maps *Application Programming Interface* (API) sendo requisitada através da mensagem recebida.

1.2 ESTRUTURA DO TRABALHO

A estrutura deste trabalho está apresentada em quatro capítulos, sendo que o segundo capítulo contém a fundamentação teórica necessária para o entendimento deste trabalho.

No terceiro capítulo é apresentado o desenvolvimento da aplicação G-SMS, onde são disponibilizadas referências para o início do desenvolvimento de uma aplicação em Android, para em seguida serem apresentados os requisitos e a especificação da aplicação desenvolvida. Esta especificação compreende os diagramas de casos de uso, classes, sequência e distribuição. O terceiro capítulo também demonstra a operacionalidade da aplicação e aborda aspectos relacionados à sua implementação, bem como os resultados obtidos.

Por fim, o quarto capítulo refere-se às conclusões do presente trabalho e sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Nas seções seguintes são apresentadas informações sobre Android, Android *Software Development Kit* (SDK), Google Maps API e trabalhos correlatos.

2.1 ANDROID

Android é uma plataforma para dispositivos móveis que utiliza o sistema operacional Linux como base. Além disso, é considerada uma plataforma móvel completa, livre e aberta. Desenvolvida inicialmente pela Google e hoje mantida pela *Open Handset Alliance* (OHA), um grupo de mais de 30 empresas de telefonia celular e tecnologia. A união dessas empresas tem como intuito acelerar e inovar o desenvolvimento dos recursos dessa plataforma, trazendo para os consumidores uma melhor experiência em termos de recursos e mais barata em termos financeiros. Mais do que a simples entrada da Google no mercado de dispositivos móveis, o projeto Android têm ambição de ser uma plataforma livre presente em diversos modelos de celulares (LECHETA, 2009).

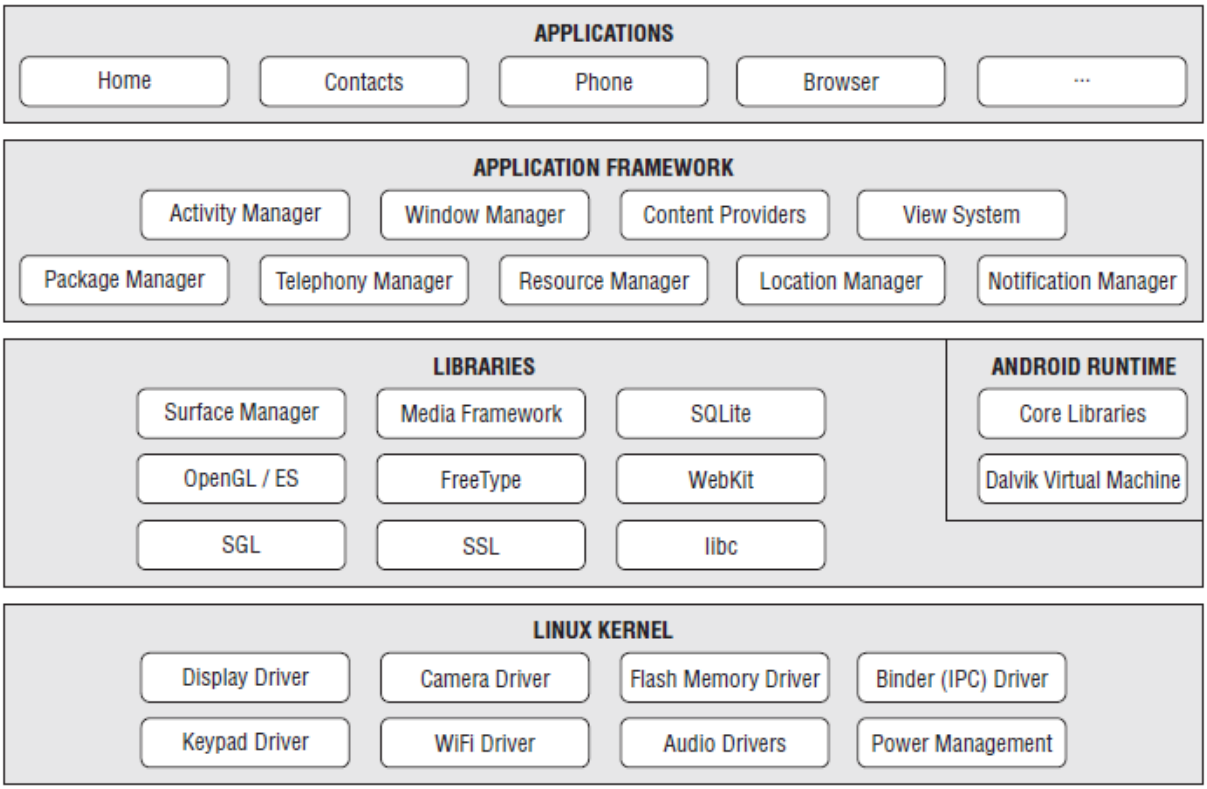
O Android é baseado no núcleo do GNU/Linux, versão 2.6. O núcleo do sistema funciona como uma camada de abstração entre o hardware e o restante da pilha de softwares da plataforma (LECHETA, 2009). O núcleo do GNU/Linux já possui vários recursos necessários para a execução de aplicações, como gerenciamento de memória, gerenciamento de processos, pilha de protocolos de rede, módulo de segurança e vários módulos de núcleo de infraestrutura. Como o sistema operacional é conhecido, também facilita o surgimento de melhorias aos *drivers* de *hardware* já existentes.

Composto por um conjunto de *software* para dispositivos móveis, um *middleware*, aplicativos e um sistema operacional, uma das grandes apostas desta plataforma é a disponibilidade do Android SDK. Este software disponibiliza os recursos necessários para criação e desenvolvimento de aplicações segundo as classes dispostas na *Application Programming Inteface* (API). Estas aplicações são escritas utilizando a linguagem de programação Java e executadas por uma máquina virtual projetada para rodar sobre um núcleo Linux e aperfeiçoada para o funcionamento em dispositivos móveis, chamada Dalvik. Portanto a plataforma Android não se utiliza de uma *Java Virtual Machine* (JVM)

convencional (GOOGLE, 2011a).

2.1.1 Arquitetura

Android possui uma arquitetura dividida em cinco componentes: Aplicativos, Framework, Bibliotecas, Ambiente de Execução e Kernel Linux. A figura 1 apresenta os cincos componentes de software que constituem o Android.



Fonte: Lee (2011).

Figura 1 - Arquitetura geral da plataforma Android

2.1.1.1 Aplicativos

A camada mais alta ou exterior da arquitetura da plataforma Android é onde são encontrados os aplicativos. Sendo estes fornecidos pela plataforma (um cliente de e-mail, navegador *Web*, calendário, mapas e outros), disponíveis no Android Market ou até mesmo desenvolvidos por programador durante sua concepção. Estes são desenvolvidos em Java e convertidos para o formato de *bytecode* capaz de ser executado pela Dalvik VM (LEE, 2011).

As aplicações podem rodar simultaneamente, através das instâncias criadas pela máquina virtual Dalvik para cada uma das aplicações, dando ao usuário o poder de alternar facilmente entre aplicações. As aplicações são executadas através de uma arquivo .dex, reconhecido pela máquina virtual utilizada.

2.1.1.2 Frameworks

A arquitetura da aplicação foi projetada para simplificar o reuso dos componentes, possibilitando os desenvolvedores terem acesso completo à mesmas APIs das aplicações chave do Android, simplificando o código através do reuso dos componentes e abstraindo parte dos procedimentos que fazem funcionar.

Os componentes chaves para desenvolvimento de aplicações Android são:

a) `activity` – base para desenvolvimento da interface visual de uma aplicação, a qual necessita da interação com um usuário. Por exemplo, uma aplicação de lista de pendências poderia ter uma `activity` que apresenta um campo com a descrição, uma segunda `activity` que representa a data limite que esta pendência deve ser finalizada. Embora trabalhem juntas para compor uma interface coesa, cada `activity` é independente uma da outra;

b) `service` – não apresenta uma interface visual para o usuário. Esta classe é responsável pelas tarefas que são executadas em segundo plano. Por exemplo, um `service` pode tocar música, enquanto o usuário vê fotos, ou pode executar tarefas cujo resultado será utilizado por uma `activity`;

c) `broadcastreceiver` – é o componente que tem função de receber e reagir à um anúncio. Por exemplo, anúncios que uma foto foi tirada, que a bateria está fraca, ou que o usuário alterou sua preferência de idioma. Também é possível que aplicativos iniciem emissões, tal como transferir dados para outro dispositivo;

d) `contentprovider` - um `contentprovider` é capaz de disponibilizar um conjunto específico de dados para outras aplicações. Estes dados podem ser armazenados no sistema de arquivo, em um banco de dados SQLite, ou em qualquer outra forma possível. Para que outras aplicações possam recupera e manipular estes dados, é necessário implementar um conjunto de métodos capazes de interagir com o tipo de

armazenamento escolhido;

e) `intent` – os componentes `service`, `activity` e `broadcastreceiver` de uma aplicação, são ativados por meio de mensagens, chamada de `intens`. Essas mensagens são utilizadas para facilitar a ligação entre componentes da aplicação ou de aplicações diferentes, em tempo de execução. O Objeto `intents`, é uma estrutura de dados passiva que possui uma descrição abstrata de uma operação a ser realizada, ou no caso de transmissões, uma descrição de algo que aconteceu e está sendo anunciado;

f) `view` – elemento utilizado para definir objetos gráficos exibidos na tela, capaz de prover interação com o usuário, tal como, botões, caixas de diálogo, mapas e outros.

2.1.1.3 Bibliotecas

Sobre o Kernel, encontra-se um conjunto de bibliotecas C/C++ que são utilizadas pelo sistema. Adaptadas para Linux, estas bibliotecas suportam diversos formatos de vídeo, áudio, assim como funções de gráficos, imagens estáticas, funções de acesso a banco de dados, entre outros. Todos esses recursos podem ser integrados às aplicações, dado que estes estão disponíveis no framework (MEIER, 2009).

2.1.1.4 Ambiente de Execução

A Dalvik *Virtual Machine* (Dalvik VM) é baseada em uma arquitetura de registradores. Esta máquina virtual foi desenvolvida com o intuito de obter maior desempenho em hardwares com pouco poder de processamento. O motivo que leva a Dalvik VM não ser considerada uma máquina virtual java é o formato dos arquivos que esta máquina virtual executa, arquivos *.dex* (Dalvik *Executable*) e não arquivos *.class*, como é feito pelas JVMs convencionais (MEIER, 2009).

A plataforma Android compila o código (*.java*) gerando arquivos *.class*, estes por sua vez são transformados pela *dx tool* em *.dex*. Este formato de *bytecode* foi desenvolvido para ocupar menos espaço e carregar rapidamente, pois é orientado a compartilhamentos de dados,

tornando-se muito útil para aplicativos voltados à dispositivos móveis.

2.1.1.5 Kernel Linux

O kernel do Linux utilizado como base do sistema operacional Android foi modificado para que o sistema fosse customizado de acordo com as necessidades e características dos dispositivos móveis, assim como novos drivers de dispositivo, modificações ao sistema de gerenciamento de energia e um sistema responsável por finalizar processos de maneira criteriosa quando há pouca memória indisponível. Além de prover as tarefas fundamentais de um sistema, como gerenciamento de memória, pilha de protocolos de rede, modelo de drivers, segurança e gerenciamento de processos.

2.2 ANDROID SDK

O desenvolvimento de uma aplicação para dispositivos móveis que utilizam a plataforma Android requer a utilização do Android SDK, que provê algumas ferramentas úteis para o desenvolvimento dos aplicativos. Estão incluídos no Android SDK o aplicativo de *debug*, bibliotecas, documentação, códigos exemplos, tutoriais e um emulador do sistema operacional Android. Ele é suportado por diversos sistemas operacionais como, Linux, Mac OS X e Windows, e pode ser facilmente integrado ao Eclipse (*Integrated Development Environment* – IDE), por meio do plugin Android *Development Tools* (ADT) *Plugin*, que facilita o desenvolvimento das aplicações por ser o IDE oficial da plataforma (LEE, 2011).

O emulador do Android SDK simula todos os recursos de *hardware* e *software* típicos de um dispositivo móvel que utiliza o sistema operacional Android. O emulador também disponibiliza várias possibilidades de depuração tais como um console em que pode-se registrar a saída do *kernel*, simular interrupções de aplicação e simular os efeitos de latência no canal de dados. Será por meio dessa ferramenta que o desenvolvimento e os testes iniciais serão realizados.

Sobre a estrutura de um projeto Android, as aplicações seguem uma base diferente das aplicações comuns de Java. Por exemplo, as aplicações Android utilizam arquivos XML responsáveis por definir os layouts que serão exibidos na tela, as classes devem ser derivadas

da classe `Activity` (atividade) e o método principal das classes é o método `onCreate`.

```
      <?xml version="1.0" encoding="utf-8"?>
      <LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
          android:orientation="vertical"
          android:layout_width="fill_parent"
          android:layout_height="fill_parent">
          <TextView android:layout_width="fill_parent"
              android:layout_height="wrap_content"
              android:text="Enter the phone number of recipient"/>
          <EditText android:id="@+id/txtPhoneNo"
              android:layout_width="fill_parent"
              android:layout_height="wrap_content"/>
          <TextView android:layout_width="fill_parent"
              android:layout_height="wrap_content"
              android:text="Message"/>
          <EditText android:id="@+id/txtMessage"
              android:layout_width="fill_parent"
              android:layout_height="150px"
              android:gravity="top"/>
          <Button android:id="@+id/btnSendSMS"
              android:layout_width="fill_parent"
              android:layout_height="wrap_content"
              android:text="Send SMS"/>
      </LinearLayout>
```

Quadro 1 - Exemplo de arquivo XML que define layout de uma tela

Para atribuir um *layout* de tela definido em um arquivo XML a uma atividade é utilizado o método `setContentView`, que recebe como parâmetro o identificador único de um arquivo do *layout* que será atribuído à atividade, de modo a representar a interface com a qual a atividade se comunica com o usuário.

O aplicativo também trabalha com diretórios específicos dentro da pasta do projeto, cada uma com suas finalidades. Os diretórios *drawable*, *layout* e *values*, por exemplo, estão dentro do diretório *res* e possuem respectivamente as funções de armazenar todas as imagens, todos os layouts e todos os valores estáticos que podem ser utilizados por um arquivo XML (STEELE; TO, 2010).

Existe também uma classe chamada R.java gerada automaticamente pela IDE. A classe mantém constantes de referências para os diversos recursos da aplicação disponibilizados dentro do diretório *res*. É através dessas referências que as demais classes interagem com os

recursos (STEELE; TO, 2010).

```
/* AUTO-GENERATED FILE.  DO NOT MODIFY.
 *
 * This class was automatically generated by the
 * aapt tool from the resource data it found.  It
 * should not be modified by hand.
 */
public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int icon=0x7f020000;
    }
    public static final class id {
        public static final int btnSendSMS=0x7f050002;
        public static final int txtMessage=0x7f050001;
        public static final int txtPhoneNo=0x7f050000;
    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
    public static final class string {
        public static final int app_name=0x7f040001;
        public static final int hello=0x7f040000;
    }
}
```

Quadro 2 – Exemplo de arquivo `R.java`

2.2.1.1 API de Localização

O Android permite que aplicações acessem os serviços de localização providos pelo dispositivo por meio do pacote de classes `android.location`. O componente central deste pacote é o serviço de sistema chamado `LocationManager`. Esta classe tem a função de monitorar eventos e atividades de localização. Por exemplo, através dessa classe é possível obter informações da posição geográfica do dispositivo e até mesmo disparar algum evento caso o dispositivo esteja nas proximidades de uma localidade específica (MURPHY, 2009).

Outra classe importante é a `LocationProvider` que visa fornecer serviços de

localização de acordo com determinados critérios estabelecidos, como por exemplo, informações de velocidade, altitude, bússola e outros.

O Android pode obter a localização do aparelho por `LocationProvider` distintos. Utilizando o `GPS_PROVIDER` o telefone irá fazer a geolocalização utilizando os dados que recebe do *Global Positioning System* (GPS). Este tipo tende a ser mais preciso, porém as vezes podem demorar mais pois necessitam de sincronização com os satélites. Além disso, ambientes fechados ou com pouca cobertura de GPS podem afetar diretamente na velocidade de obtenção da localização. No caso do `NETWORK_PROVIDER` o telefone irá tentar fazer uma triangulação das antenas, ou obter informações das redes *wi-fi* ao seu alcance.

Duas outras classes trabalham com informações de pontos geográficos fixos, a classe `Location` identifica esses pontos por meio dos valores de latitude e longitude. E a classe `Geocode` permite que valores de latitude e longitude possam ser convertidos para um endereço, podendo obter o nome e o código postal (MURPHY, 2009).

2.3 GOOGLE MAPS

Google Maps é um serviço de pesquisa e visualização de mapas e imagens de satélite da Terra gratuito fornecido pela empresa Google. Atualmente, o serviço disponibiliza mapas e rotas para diversas localizações do globo, com possibilidade de aproximação das imagens em grandes cidades.

Em fevereiro de 2005 ainda em sua versão beta, o Google Maps foi lançado tornando-se rapidamente uma referência em serviços de mapas na Internet. A interface rica e intuitiva da aplicação permitia acesso a uma enorme base de dados contendo inúmeras imagens de satélite, mapas de cidades, bairros, ruas e avenidas dos Estados Unidos. Em maio de 2007, a Google disponibilizou consultas de endereços no Brasil e em outubro de 2007, uma versão estendida para o público brasileiro foi oferecida, com a possibilidade de se localizar restaurantes, hotéis, traçar rotas, dentre outras utilidades (AZEVEDO, 2008).

2.3.1 Google Maps API

Um dos projetos do Google que dispõe de uma forte aceitação acadêmica e comercial é

a biblioteca externa para desenvolvimento de aplicações baseadas em mapas. O Google Maps API está presente em diversas aplicações, com inúmeras funcionalidades que são enriquecidas pela apresentação dos mapas disponibilizados por este provedor, assim como, a possibilidade de traçar rotas a partir de informações de origem e destino, adicionar marcadores em pontos específicos do mapa, entre outras possibilidades.

A primeira versão da API do Google Maps foi disponibilizada em junho de 2005. Atualmente está disponível em sua terceira versão sendo mais rápida e eficaz, além ter uma compatibilidade maior com dispositivos móveis e últimas versões dos navegadores convencionais (GOOGLE, 2011b).

2.3.2 Google Maps no Android

A plataforma Android está fortemente integrada com os serviços do Google. Sendo assim a Google disponibilizou uma versão da Google Maps API para o projeto Android. Baseado em uma biblioteca externa, por meio das classes presentes nesta biblioteca é possível download, renderização e cache das informações de mapas obtidas, como também várias opções de exibição e controle. Os quatro tipos de visualização de mapas fornecidos são: mapas de imagens de satélite, mapas topográficos, mapas de ruas e visualização de ruas conhecido como *StreetView*. Este último disponível apenas em algumas cidades.

Apesar de ser considerada uma biblioteca externa, o projeto da Google Maps API para Android ainda não dispõe de um meio de obtenção dessa biblioteca de maneira separada ao projeto Android. Esta biblioteca é instalada durante o processo de instalação do Android SDK, sendo necessário apenas uma seleção de opção.

As classes da biblioteca de mapas estão disponíveis no pacote `com.google.android.maps`. Sendo a classe principal desta biblioteca a `MapView`, que consiste em um objeto gráfico responsável pela visualização de mapas providos pelo provedor do Google Maps, capaz de alterar entre três diferentes tipos de mapas (mapas de imagens de satélite, mapas topográficos e mapas de ruas). A `MapView` necessariamente deve ser utilizada em uma `MapActitvity` que é um tipo de `Activity` especial para esta função. A aplicação não é executada em outros tipos de `Activity`.

Com a utilização da Google Maps API, é possivel criar aplicações robustas baseadas em localização por coordenadas geográficas (latitude e longitude) que podem ser integradas a aplicação que utilizam a API de localização encontrada no Android.

Com relação aos controles, o destaque fica para a classe `MapController` capaz de alterar o nível de zoom do mapa, mudar a localização exibida entre outras funções. Também é encontrada na biblioteca do Google Maps a classe `Overlay`, utilizando esta classe é possível criar elementos de sobreposição. Por exemplo, a partir de uma imagem estática, esta pode ser desenhada por cima do mapa principal obtida a partir do serviço do Google Maps.

Outros recursos que podem ser trabalhados através das classes disponíveis no pacote da API do Google Maps, são balões informativos, que servem para exibir informações sobre um determinado local e as linhas que ligam dois pontos estabelecidos previamente, podendo obter informações da distância entre esses dois pontos.

2.4 TRABALHOS CORRELATOS

Existem aplicações que possuem características semelhantes ao proposto neste trabalho, tais como o Flutter (JUICE WIRELESS INCORPORATION, 2009), Google Latitude (GOOGLE, 2011c) e I Am Here (GEOSMS, 2010).

2.4.1 Flutter

Segundo JUICE WIRELESS INCORPORATION (2009), o Flutter é uma aplicação para IPhone que possibilita o envio de mensagens multimídia. O aplicativo permite enviar mensagens com fotos anexadas e com a georeferência referente ao local do momento do envio.

Fonte: Juice Wireless Incorporation (2009).

Figura 2 - Aplicativo Flutter para Iphone

As mensagens são enviadas a um servidor ficando disponível ao destinatário. O destinatário acessa a mensagem através de um *link*, este encaminhado pela aplicação servidora através de um SMS ao dispositivo do usuário destinatário. Caso a mensagem tenha sido enviada com os dados de localização, um mapa é apresentado com um marcador sinalizando o local. O aplicativo faz uso da Google Maps API para apresentar a localização (JUICE WIRELESS INCORPORATION, 2009).

2.4.2 Google Latitude

O Google Latitude é um serviço de *middleware* oferecido pelo Google para usuários de dispositivos móveis do mundo todo. Ele provê funcionalidade de localização de usuários através de seu dispositivo móvel e tem como principal apelo a integração com aplicações do Google como o Orkut, Google Maps, Google Earth, Google Talk, Gmail, Google Buzz e blogs. Na maioria dos casos, o objetivo é indicar a localização do usuário no momento em que escreve uma mensagem, inicia uma sessão de bate papo ou simplesmente define a melhor rota para uma determinada localidade. No caso do Orkut, por outro lado, existem outras aplicações relevantes mais voltadas ao âmbito de redes sociais. Uma vez que este serviço pode ser estendido para os amigos de um usuário, é possível, por exemplo, que ele realize o acompanhamento da localização de seus amigos em tempo real (GOOGLE, 2011c).

Através da funcionalidade chamada *Latitude Alerts* o usuário pode ser notificado

sempre que um de seus amigos está por perto. Essa funcionalidade, em conjunto com outra chamada de Histórico de Localizações permite que o usuário receba as notificações de proximidade apenas quando estiver em lugares não usuais, evitando assim notificações de amigos enquanto você está em casa ou no seu ambiente de trabalho. A funcionalidade Histórico de Localizações permite ainda que um usuário revisite os trajetos percorridos em um determinado período de tempo. O Latitude possui outra funcionalidade chamada Crachá de Localização Pública que informa aos visitantes de um blog qual é a localização de seu autor no momento e a integração com o Google Talk permite que contatos no serviço de mensagens instantâneas visualizem a localização atual de um usuário em diferentes níveis (país, estado, cidade ou bairro, por exemplo).

Fonte: Google (2011c).

Figura 3 - Google Latitude em versões para Android, RIM e iOS

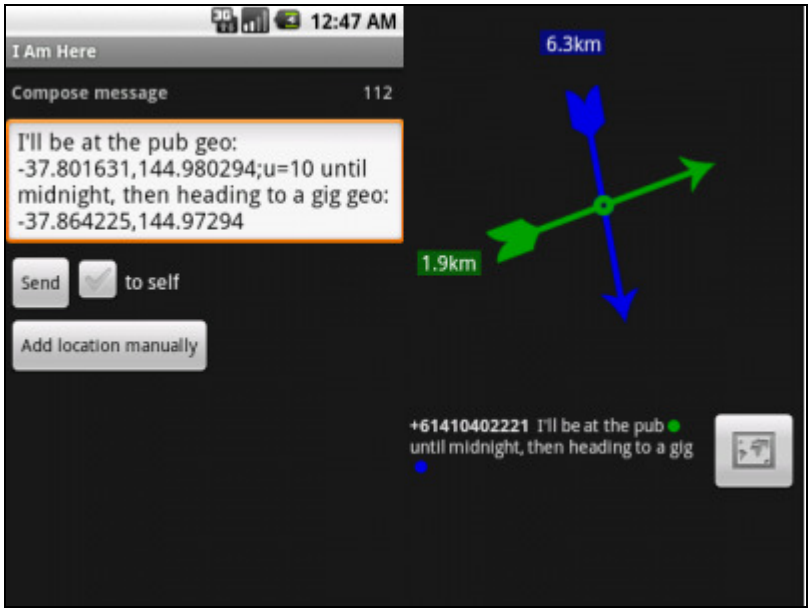
Em relação à privacidade, a Google garante que o histórico de localização de um usuário não é visível publicamente ou mesmo pelos amigos do usuário no sistema. Além disso, usuários podem apagar seu histórico (ou parte dele) a qualquer momento (GOOGLE, 2011c).

2.4.3 I Am Here

A aplicação tem como objetivo disponibilizar o envio e o recebimento de SMS com informações de georeferencia. Segundo GEOSMS (2010), o *I am here* foi desenvolvido para a plataforma Android.

A localização é enviada no SMS no formato -37.801631, 144.980294, sendo

respectivamente a latitude e longitude. Uma mesma mensagem pode apresentar mais de um conjunto de coordenadas sendo somente uma adquirida pelo GPS, as demais precisam ser digitadas pelo usuário.



Fonte: GEOSMS (2010).

Figura 4 - Telas da aplicação I am here

O destinatário ao receber a mensagem pode optar em visualizar uma bússola que aponta na direção da localização ou visualizar em mapa utilizando a integração com API Google Maps. A funcionalidade de marcadores da API é utilizada para apresentar o local exato no mapa (GEOSMS, 2010).

3 DESENVOLVIMENTO

Neste capítulo são abordadas as etapas do desenvolvimento do projeto. São ilustrados os principais requisitos, a especificação, a implementação e por fim são listados os resultados e discussões.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Os requisitos apresentados abaixo se encontram classificados em Requisitos Funcionais (RF) e Requisitos Não Funcionais (RNF). A aplicação deve:

a) possibilitar o envio de mensagem SMS contemplando no corpo da mensagem as informações de georeferenciamento (RF);
b) recuperar informações da agenda através da Contacts API do Android (RF);
c) permitir o envio de mensagem para um telefone não cadastrado na agenda (RF);
d) disponibilizar uma lista dos últimos destinatários (RF);
e) disponibilizar uma lista das últimas mensagens enviadas (RF);
f) disponibilizar a visualização da localização enviada na mensagem através de um navegador (RF);
g) a interface deverá ser definida utilizando arquivos XML (RNF);
h) a visualização dos dados georefenciados utilizando a Google Maps (RNF);
i) ser implementado utilizando o ambiente de desenvolvimento Eclipse (RNF).

3.2 ESPECIFICAÇÃO

A especificação do presente trabalho foi desenvolvida utilizando alguns dos diagramas da *Unified Modeling Language* (UML) em conjunto com a ferramenta *Enterprise Architect* versão 7.0.813 para elaboração dos casos de uso, diagramas de classe, diagrama de distribuição e diagrama de sequência.

3.2.1 Casos de uso

O trabalho disponibiliza um aplicativo para dispositivos móveis na plataforma Android para envio de mensagens georeferenciadas. Nos casos de uso são apresentados dois usuários: emissor, que envia a mensagem e receptor, que recebe.

Na figura 5 é apresentado o diagrama de casos de uso da aplicação.

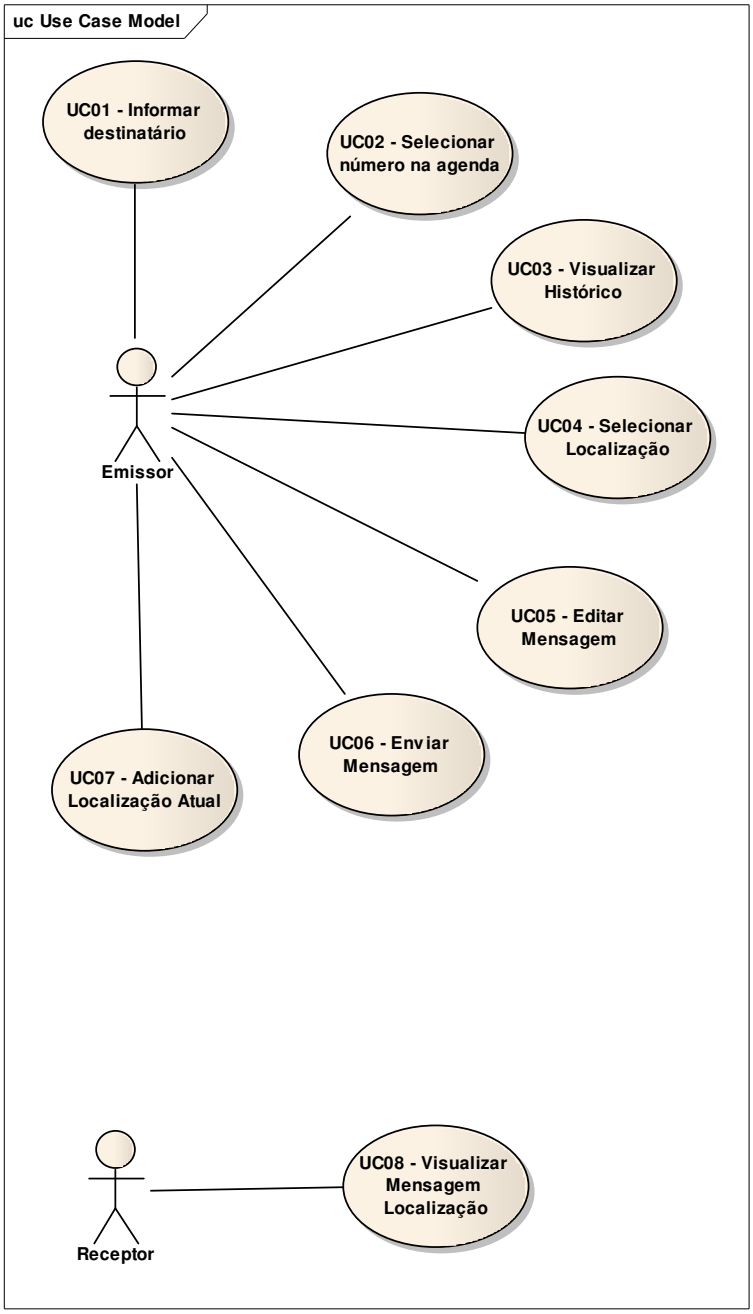


Figura 5 - Casos de uso da aplicação G-SMS

3.2.1.1 UC01 – Informar destinatário

Neste caso de uso é feito o controle do número informado de telefone de destino.

UC01 – Informar destinatário	
Pré-condição	Não possui
Cenário principal	Usuário informa número de telefone de destino
Pós-condição	Campo de número de destino preenchido

Quadro 3 - Caso de uso UC01

3.2.1.2 UC02 – Selecionar número na agenda

O caso de uso UC02, apresenta a listagem de contatos da agenda com seus respectivos contatos e permite o usuário selecionar um para preencher o campo de número de destino.

UC02 – Selecionar número na agenda	
Pré-condição	Não possui
Cenário principal	1. Usuário aperta o botão Agenda 2. Sistema lista todos os contatos que possuem números telefônicos cadastrados 3. Usuário clica em um contato da lista 4. Sistema retorna a tela principal e preenche campo número destino com o selecionado
Exceção	No passo 2 do cenário principal, caso não houver nenhum contato com número telefônico na agenda, o sistema apresentará mensagem para o usuário “Nenhum número telefônico disponível na agenda”
Pós-condição	Campo de número de destino preenchido.

Quadro 4 - Caso de uso UC02

3.2.1.3 UC03 – Visualizar histórico

Caso de uso que apresenta o histórico de mensagens enviadas e recebidas pelo

aplicativo G-SMS e que ainda estão disponíveis na memória do aparelho.

UC03 – Visualizar histórico	
Pré-condição	Não possui
Cenário principal	1. Usuário aperta o botão de agenda 2. Sistema lista os registros de mensagens enviadas e recebidas pelo aplicativo G-SMS existentes na memória do aparelho 3. Usuário clica em um dos itens da lista. 4. Sistema apresentará a mensagem com todos os detalhes. Número de destino, nome do destinatário caso este esteja cadastro na agenda e a mensagem
Exceção	No passo 2 do cenário principal, caso não houver nenhum registro de mensagem enviada ou recebida pelo aplicativo G-SMS, o sistema apresentará mensagem para o usuário "Nenhuma mensagem enviada ou recebida"
Pós-condição	Mantém-se como antes devido se tratar apenas de uma consulta

Quadro 5 - Caso de uso UC03

3.2.1.4 UC04 – Selecionar Localização

Neste caso de uso, o emissor seleciona a localização que deseja anexar no *SMS* a ser enviado para o usuário receptor. A seleção ocorre ao pressionar longamente um ponto no mapa, um alfinete é utilizado para demonstrar essa seleção. A localização pode ser qualquer uma disponível no mapa, não necessariamente a localização recuperada pelo *GPS* no mesmo momento.

UC04 – Selecionar Localização	
Pré-condição	Não possui
Cenário principal	1. Usuário aperta o botão “Selecionar Localização 2. Sistema mostra mapa animado com a localização atual do aparelho 3. Usuário navega pelo mapa e pressiona longamente a localização desejada marcando o ponto 4. Usuário aperta o botão “Usar Localização” 5. Sistema adiciona ao campo mensagem os dados de georeferenciamento
Pós-condição	Mensagem com os dados de georeferenciamento disponíveis

Quadro 6 - Caso de uso UC04

3.2.1.5 UC05 – Editar Mensagem

Este caso de uso demonstra a possibilidade do usuário emissor editar a mensagem de texto antes de ser enviada ao usuário receptor.

UC05 – Editar Mensagem	
Pré-condição	Não possui.
Cenário principal	1. Usuário seleciona campo mensagem. 2. Digita ou apaga os caracteres disponíveis no campo.
Pós-condição	Mensagem disponível editada conforme execução

Quadro 7 - Caso de uso UC05

3.2.1.6 UC06 – Enviar Mensagem

Neste caso de uso é apresentada a funcionalidade de enviar mensagem e as validações que o aplicativo realiza antes de enviar propriamente.

UC06 – Enviar Mensagem	
Pré-condição	Não possui
Cenário principal	1. Usuário clica no botão “Enviar mensagem” 2. Sistema valida se número de destino foi informado 3. Sistema valida se o campo mensagem está preenchido 4. Sistema dispara envio de mensagem
Exceção 1	No passo 2, caso o campo estiver em branco, o sistema irá apresentar mensagem erro e solicitará que seja preenchido o campo antes de clicar o botão “Enviar mensagem” novamente
Exceção 2	No passo 3, caso o campo estiver em branco, o sistema irá apresentar mensagem erro e solicitará que seja preenchido o campo antes de clicar o botão “Enviar mensagem” novamente
Pós-condição	Mensagem enviada para o número informado.

Quadro 8 - Caso de uso UC06

3.2.1.7 UC07 – Adicionar Localização Atual

Neste caso é apresentada a funcionalidade de adicionar os dados georeferenciados da localização atual na mensagem.

UC07 – Adicionar Localização Atual	
Pré-condição	Não possui.
Cenário principal	1. Usuário clica no botão “Adicionar Localização Atual” 2. Adiciona dados de georeferenciamento a mensagem
Pós-condição	Mensagem com os dados de georeferenciamento disponíveis.

Quadro 9 - Caso de uso UC07

3.2.1.8 UC08 – Visualizar Mensagem Localização

O caso de uso 8 apresenta a funcionalidade de visualizar a localização recebida sendo chamada após abrir a *Uniform Resource Location* (URL). A visualização pode ser feita através de um navegador ou aplicativo Google Maps associado.

UC08 – Visualizar Mensagem Localização	
Pré-condição	Ter recebido mensagem enviada pelo aplicativo
Cenário principal	1. Usuário receptor recebe a mensagem 2. Abre a URL enviada para o celular e visualiza a localização através do Google Maps
Pós-condição	Visualização da localização no Google Maps

Quadro 10 - Caso de uso UC08

3.2.2 Diagrama de classes

O diagrama de classes, figura 6, apresenta uma visão de como as classes desenvolvidas estão estruturadas e relacionadas. Sendo a `MensagemActivity` a *activity* principal do aplicativo. É a partir dela que as demais *activities* são iniciadas.

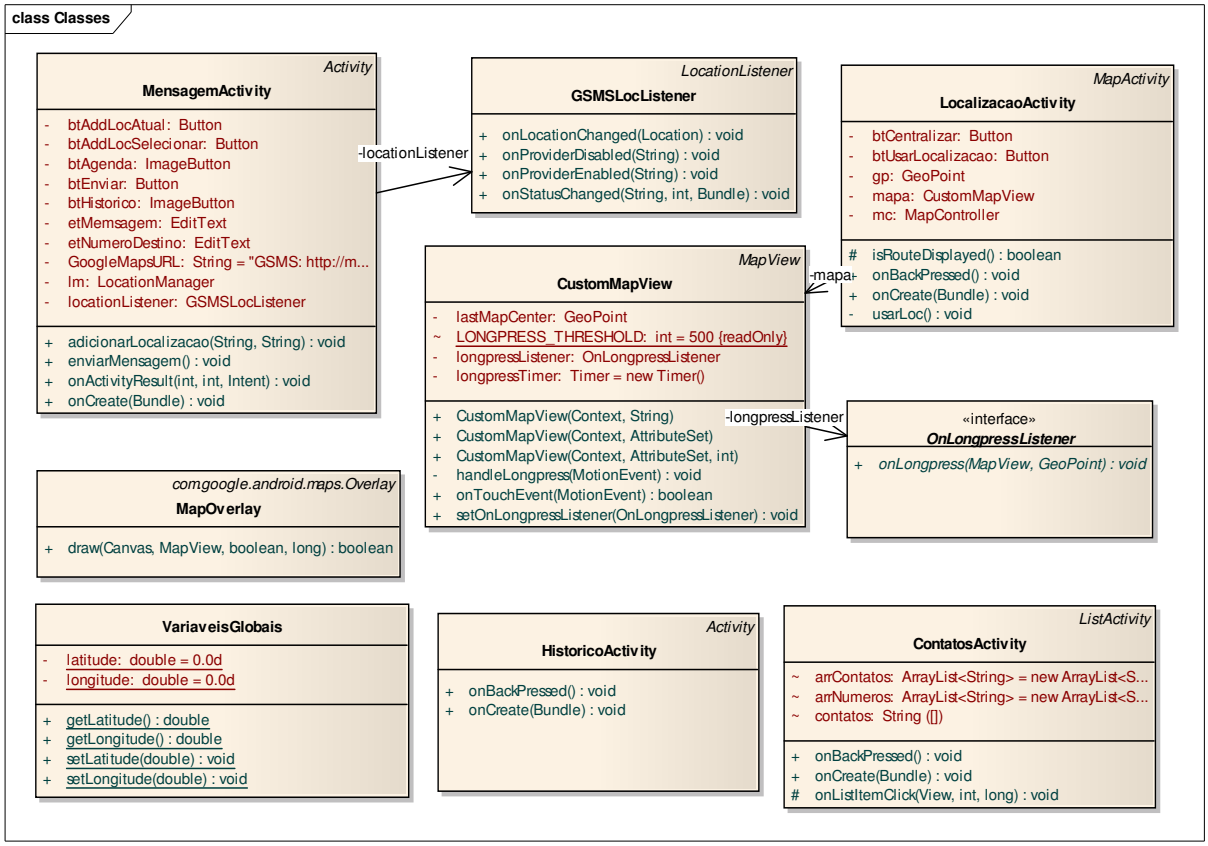


Figura 6- Diagrama de classes

A classe `MensagemActivity` possui 4 métodos sendo que o `onCreate` é chamado na inicialização. Este carrega o *layout* da tela definido no arquivo `mensagem.xml`, atribui listeners de `onClick` em cada botão da tela (`btAddLocAtual`, `btAddLocSelecionar`,

`btAgenda`, `btEnviar` e `btHistorico`) e instancia `GSMSLocListener`. Esta classe implementa os métodos da interface `LocationListener` e recebe as notificações do `LocationManager` (`lm`) quando a localização do aparelho é alterada.

`LocalizacaoActivity` é a `activity` que apresenta mapas utilizando a API do Google Maps e a classe `CustomMapView` que extende o objeto `MapView` responsável por mostrar na tela do aparelho o mapa bem como manipular vários de seus atributos como mudar o tipo de visualização de satélite para rua e realizar zoom. A criação da `CustomMapView` foi necessária devido a necessidade de realizar o tratamento da ação `longPress` utilizando a interface. Pela classe `MapView` não é possível devido a impossibilidade de associar a interface criada `OnLongpressListener`.

3.2.3 Diagrama de distribuição

O diagrama de distribuição, apresentado na figura 7, caracteriza o contexto da aplicação e o cenário de uso da mesma.

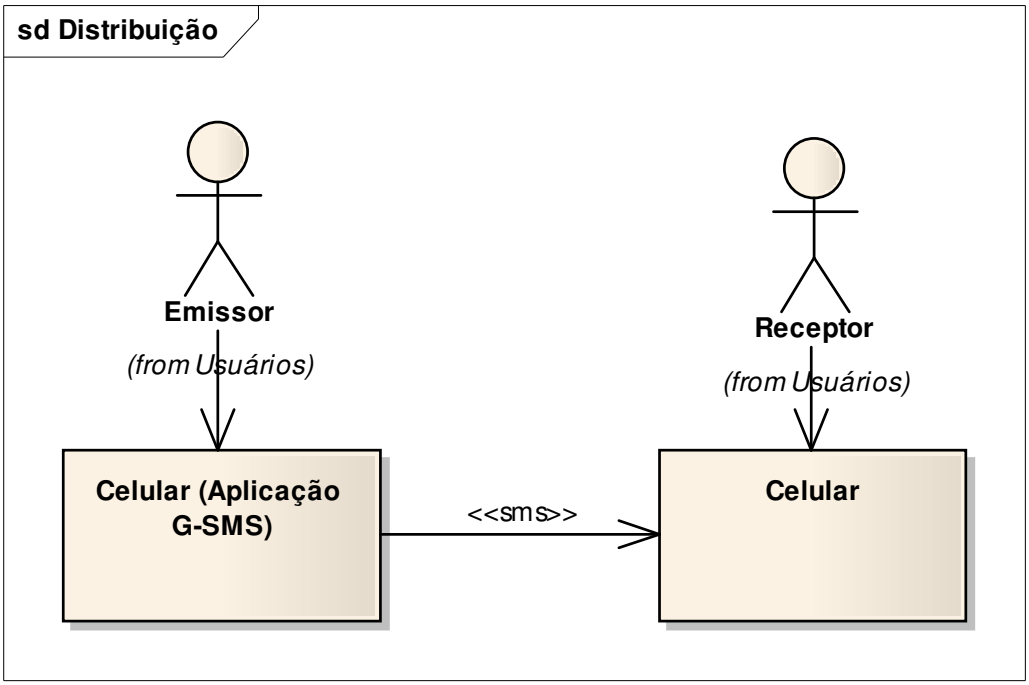


Figura 7 - Diagrama de distribuição

3.2.4 Diagrama de sequência

A figura 8 apresenta o diagrama de sequência do envio da mensagem georeferenciada. O usuário emissor ao iniciar o programa, o método `onCreate` da classe `MensagemActivity` é executado apresentando a tela principal do sistema. O método `startActivityForResult` faz a chamadas para as demais *activities* e telas do protótipo.

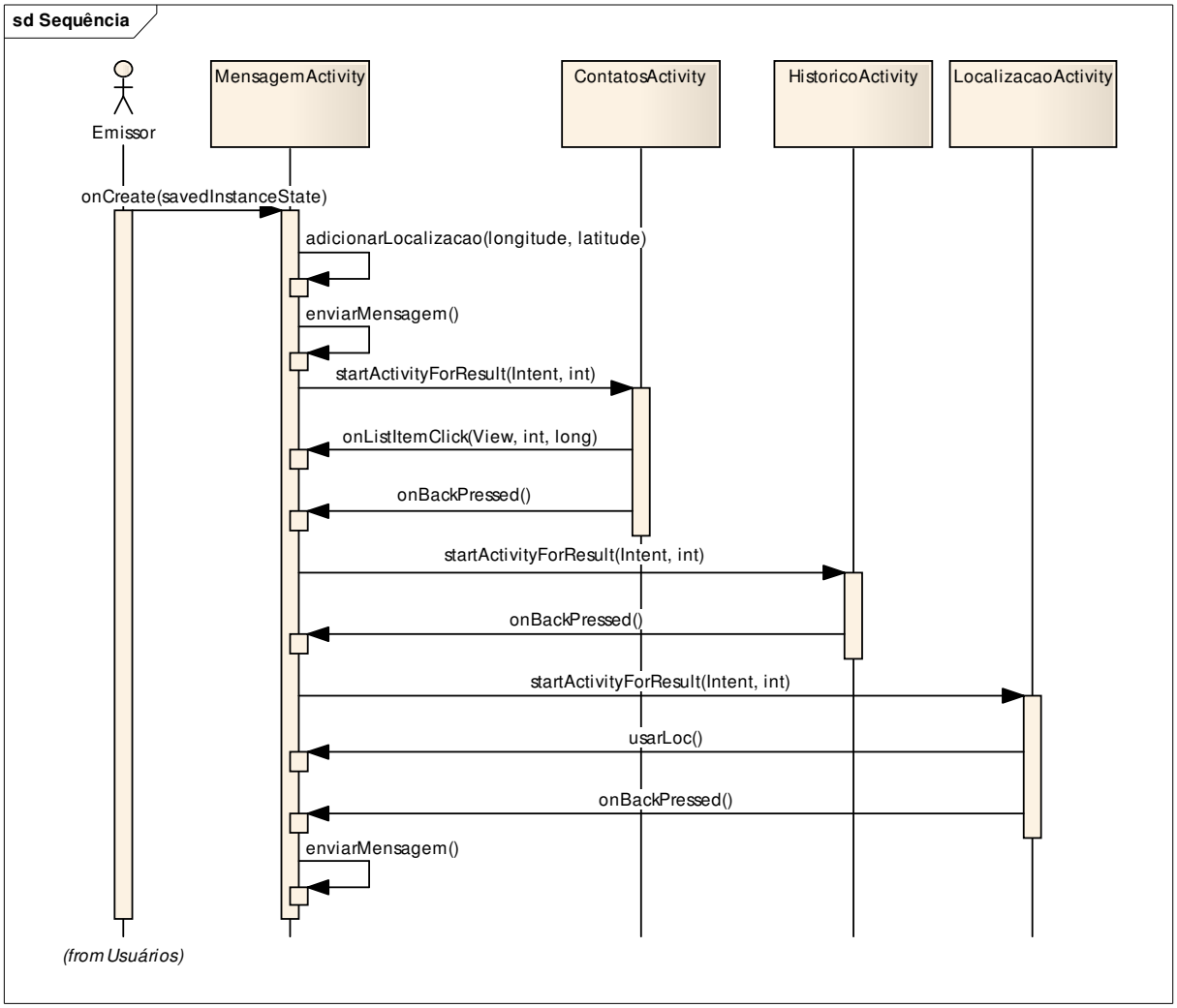


Figura 8 - Diagrama de sequência

3.3 IMPLEMENTAÇÃO

Esta seção apresenta as bibliotecas, técnicas e ferramentas utilizadas para o desenvolvimento da aplicação, junto com as descrições das telas e trechos de código para um

melhor entendimento.

3.3.1 Técnicas e ferramentas utilizadas

O protótipo G-SMS foi implementada na linguagem de programação Java utilizando-se o ambiente de desenvolvimento Eclipse IDE. Em conjunto com os recursos da linguagem de programação da aplicação foi utilizado o Android SDK, que disponibiliza todas as ferramentas e APIs necessárias para o desenvolvimento de aplicativos para a plataforma Android na linguagem de programação Java.

O pacote de desenvolvimento Android utilizado inclui um sistema operacional, um *middleware* e aplicativos de características essenciais, dentre os quais se encontram o emulador e um *debbuger*, juntamente com toda a biblioteca de desenvolvimento para dispositivos móveis que utilizam o sistema operacional Android.

A escolha do ambiente de desenvolvimento Eclipse se deu devido este ser o oficial da plataforma Android e a existência do Android *Development Tools* (ADP). Este um *plug-in* que facilita o desenvolvimento das aplicações gerando o projeto com os componentes necessários de inicio, além de disponibilizar ferramentas de *debug* e *log*.

3.3.2 Configuração da aplicação

Em todo projeto Android existe um arquivo chamado `AndroidManifest.xml`. Esse arquivo é obrigatório e é nele que são feitas as configurações de todos os recursos utilizados pela aplicação (*activities*, componentes gráficos, *layouts*, permissões, imagens, etc). O quadro 11 apresenta o conteúdo comentado do arquivo `AndroidManifest.xml` da aplicação G-SMS.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="br.com.gsms"
      android:versionCode="1"
      android:versionName="1.0">
    <!-- Menor versão compatível com a aplicação -->
    <uses-sdk android:minSdkVersion="7" />

    <!-- Ícone e nome da aplicação -->
    <application android:icon="@drawable/icon"
android:label="@string/app_name">
    <!-- Importa e disponibiliza a biblioteca do Google Maps -->
      <uses-library android:name="com.google.android.maps" />
    <!-- Activity inicial -->
        <activity android:name="br.com.gsms.MensagemActivity"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER"
/>
            </intent-filter>
        </activity>
        <!-- Demais Activities -->
        <activity android:name="br.com.gsms.HistoricoActivity"
android:label="Histórico"/>
        <activity android:name="br.com.gsms.LocalizacaoActivity"
android:label="Selecione uma Localização"/>
        <activity android:name="br.com.gsms.ContatosActivity"
android:label="Agenda"/>
    </application>
    <!-- Permissões para a aplicação -->
    <!-- Permissão necessária para consultar o histórico de SMS -->
    <uses-permission android:name="android.permission.READ_SMS"/>
    <!-- Permissão necessária para enviar SMS -->
    <uses-permission android:name="android.permission.SEND_SMS"/>
    <!-- Permissão necessária para consultar os contatos -->
    <uses-permission android:name="android.permission.READ_CONTACTS"/>
    <!-- Permissão necessária para acessar a Internet -->
    <uses-permission android:name="android.permission.INTERNET"/>
    <!-- Permissão necessária para recuperar a localização através do GPS
-->
    <uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION" />
</manifest>
```

Quadro 11 - `AndroidManifest.xml`

3.3.3 Google Maps Key

A integração do Google Maps com uma aplicação Android requer uma chave de API. A chave é utilizada para que os servidores do Google reconheçam a aplicação e permitam o acesso aos dados dos mapas. Desta maneira, a Google previne que uma aplicação acesse os servidores de maneira descontrolada deixando-os lentos e possivelmente indisponíveis para

outras aplicações.

A chave é gratuita e simples de ser solicitada. Ao executar uma aplicação Android através Eclipse é gerado um arquivo `debug.keystroke` utilizado para extrair com a ferramenta *KeyTool* disponível no Java SDK o "MD5 fingerprint". É necessário também uma conta Google para poder informar o "MD5 fingerprint" e obter uma chave de utilização da API. A API *key* deve ser informada dentro do XML de layout da tela que irá conter a classe `MapView` conforme quadro 12, que define a tela de selecionar localização.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="center_horizontal">
      <FrameLayout android:id="@+id/map_frame"
                  android:layout_weight="1"
                  android:layout_width="wrap_content"
                  android:layout_height="wrap_content">
      <br.com.gsms.utils.CustomMapView
                  android:id="@+id/mapView"
                  android:layout_width="fill_parent"
                  android:layout_height="fill_parent"
                  android:enabled="true"
                  android:clickable="true"
      <!-- Chave da API -->
      android:apiKey="0fxzS8M7KDBIGtNFHqoc8qEjgMVDrON7vHt2Zkw"/>
      </FrameLayout>
      <LinearLayout android:orientation="horizontal"
                  android:layout_width="wrap_content"
                  android:layout_height="wrap_content"
                  android:layout_marginTop="10sp"
                  android:layout_marginBottom="10sp">
            <Button android:id="@+id/btUsarLoc"
                        android:text="Usar Localização"
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"/>
            <Button android:id="@+id/btCentralizar"
                        android:text="Centralizar"
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:layout_marginLeft="10sp"/>
      </LinearLayout>
</LinearLayout>
```

Quadro 12 - XML do layout da tela de selecionar localização

No arquivo *AndroidManifest.xml* se faz necessário adicionar a permissão referente a acesso a internet para que a aplicação possa realizar o download dos mapas.

3.3.4 Operacionalidade da aplicação

A operacionalidade do protótipo é apresenta em função dos principais casos de uso da aplicação, sendo apresentadas capturas de telas e extração de código fonte para melhor ilustrar e facilitar o entendimento de cada uma das funcionalidades disponibilizadas.

Figura 9 – Tela inicial do aplicativo

Ao abrir o aplicativo (figura 9), é apresentada para o usuário a tela inicial do protótipo. Esta é a base para chamadas de todas as funcionalidades do sistema. O Quadro 13 apresenta XML, messagem.xml, que define o *layout* da tela inicial.

```
      <?xml version="1.0" encoding="utf-8"?>
      <!-- declaração do elemento LinearLayout (Raiz do layour para tela e
atividade definindo a orientação como sendo vertical e o tamanho de tela
cheia. Pois a altura (layout_height) e largura (layout_width) tem como
parâmetro "fill_parent". Preencher elemento pai, mas como esse é o
elemento inicial, a tela toda é preenchida. -->
      <LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
          android:orientation="vertical"
          android:layout_width="fill_parent"
          android:layout_height="fill_parent">
      <!-- declaração de mais um elemento LinearLayout. Com a diferença
para o parâmetro de altura (layout_height) como o valor de "wrap_content"
que garante que o elemento fique do tamanho necessário para conter o
elementos interno a ele -->
       <LinearLayout android:orientation="horizontal"
                  android:layout_width="fill_parent"
                  android:layout_height="wrap_content"
                  android:layout_marginTop="10sp">
      <!-- declaração do elemento EditText. O parâmetro id mantém o
identificador único "numeroDestine" do elemento, o inputType "phone"
garante que apenas números sejam informados -->
            <EditText android:id="@+id/numeroDestino"
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:minLines="1"
                        android:hint="Número de destino"
                        android:gravity="left"
                        android:inputType="phone"/>
      <!-- declaração do elemento ImageButton. O parâmetro scr mantém o
imagem que deve ser mostrada no botão, no caso o arquivo "addressbook"
mantido nas pastas drawable -->
            <ImageButton android:id="@+id/btAgenda"
                      android:layout_width="wrap_content"
                      android:layout_height="wrap_content"
                      android:src="@drawable/addressbook"/>
      <!-- declaração do elemento ImageButton -->
            <ImageButton android:id="@+id/btHistorico"
                      android:layout_width="wrap_content"
                      android:layout_height="wrap_content"
                      android:src="@drawable/log"/>
```

```
      </LinearLayout>
      <!-- declaração do elemento LinearLayout -->
      <LinearLayout android:orientation="horizontal"
                  android:layout_width="fill_parent"
                  android:layout_height="wrap_content">
      <!-- declaração do elemento TextView -->
            <TextView android:id="@+id/header_text"
                        android:text="Mensagem:"
                        android:textStyle="bold"
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:layout_marginLeft="5sp"/>
      </LinearLayout>
      <!-- declaração do elemento EditText. Parâmetro minLines garante que
este objeto tenha pelo menos 4 linhas de tamanho -->
      <EditText android:id="@+id/message_editor"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:minLines="4"
            android:hint="Digite a mensagem"/>
      <!-- declaração do elemento LinearLayout -->
      <LinearLayout android:orientation="horizontal"
                  android:layout_width="fill_parent"
                  android:layout_height="wrap_content">
      <!-- declaração do elemento Button -->
            <Button android:id="@+id/addLocAtual"
                        android:text="Adicionar Localização Atual"
                        android:layout_width="fill_parent"
                        android:layout_height="wrap_content"
                        android:textSize="14dp"/>
      </LinearLayout>
      <!-- declaração do elemento LinearLayout -->
      <LinearLayout android:orientation="horizontal"
                  android:layout_width="fill_parent"
                  android:layout_height="wrap_content">
      <!-- declaração do elemento Button -->
            <Button android:id="@+id/addLocSelected"
                        android:text="Selecionar Localização"
                        android:layout_width="fill_parent"
                        android:layout_height="wrap_content"
                        android:textSize="14dp"/>
```

```
      </LinearLayout>
      <!-- declaração do elemento LinearLayout -->
      <LinearLayout android:orientation="horizontal"
                  android:layout_width="fill_parent"
                  android:layout_height="wrap_content">
      <!-- declaração do elemento Button -->
            <Button android:id="@+id/enviar"
                  android:text="Enviar mensagem"
                  android:layout_width="fill_parent"
                  android:layout_height="wrap_content"
                  android:textSize="14dp"/>
      </LinearLayout>
</LinearLayout>
```

Quadro 13 - *Layout* tela inicial do programa

O aplicativo ao iniciar na activity `MensagemActivity` instancia um `LocationListener` que busca a localização atual do aparelho. O quadro 14 demonstra essa parte do código.

```
<!-- Busca o localização através do GPS -->
lm = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
locationListener = new GPSLocationListener();
lm.requestLocationUpdates(LocationManager.GPS_PROVIDER,0,0,locationListener);
```

Quadro 14- Instânciando o `LocationListener`

`GPSLocationListener` é uma classe que implementa a interface `LocationListener` conforme apresentado no quadro 15.

```
class GPSLocationListener implements LocationListener {
        public void onLocationChanged(Location loc) {
            VariaveisGlobais.setLongetitude(loc.getLongitude());
            VariaveisGlobais.setLatitude(loc.getLatitude());
        }

        public void onProviderDisabled(String provider) {
            return;
        }

        public void onProviderEnabled(String provider) {
            return;
        }

        public void onStatusChanged(String provider, int status, Bundle
extras) {
            return;
        }
}
```

Quadro 15 - Classe `GPSLocationListener`

O método `onLocationChanged` garante que sempre que houver alteração com relação os valores de latitude e longitude, estes são atualizados na classe `VariaveisGlobais`. Esta

classe é utilizada para manter valores disponíveis entre as diversas *activities* do protótipo.

O quadro 16 apresenta o código fonte da classe `VariaveisGlobais`.

```
package br.com.gsms;

public class VariaveisGlobais {

	private static double latitude = 0.0d;
	private static double longetitude = 0.0d;

	public static double getLatitude() {
		return latitude;
	}

	public static void setLatitude(double latitude) {
		VariaveisGlobais.latitude = latitude;
	}

	public static double getLongetitude() {
		return longetitude;
	}

	public static void setLongetitude(double longetitude) {
		VariaveisGlobais.longetitude = longetitude;
	}

}
```

Quadro 16 - Classe `VariaveisGlobais`

3.3.4.1 Selecionar número na agenda

O caso de uso UC02 apresenta a listagem de contatos da agenda com seus respectivos contatos e permite o usuário selecionar um para preencher o campo de número de destino ao pressionar o contato.

Figura 10 - Agenda

A base de contatos nativo do celular é acessada através do primeiro botão ao lado do campo de "número de destino". Ao clicar no botão é inicializada a *activity* `ContatosActivity` apresentada no quadro 11.

```
public class ContatosActivity extends ListActivity {

      ArrayList<String> arrContatos = new ArrayList<String>();
      ArrayList<String> arrNumeros = new ArrayList<String>();
      String[] CONTATOS;

      public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        Cursor people =
getContentResolver().query(ContactsContract.Contacts.CONTENT_URI,//Tabela
            null,
            ContactsContract.Contacts.HAS_PHONE_NUMBER + "=1", //Restrição
            null,
            ContactsContract.Contacts.DISPLAY_NAME + " ASC"); //Order By
      while(people.moveToNext()) {
          String displayName =
people.getString(people.getColumnIndex(PhoneLookup.DISPLAY_NAME));
          String contactID =
people.getString(people.getColumnIndex(PhoneLookup._ID));
                Cursor phoneNumbers =
getContentResolver().query(Phone.CONTENT_URI, null, Phone.CONTACT_ID + " =
" + contactID, null, null);
                while (phoneNumbers.moveToNext()){
```

```
                        String number =
phoneNumbers.getString(phoneNumbers.getColumnIndex(Phone.NUMBER));
                        String type =
phoneNumbers.getString(phoneNumbers.getColumnIndex(Phone.LABEL));
                        if (type == null || type.equals("")){
                              type = "Telefone";
                        }
                    arrContatos.add(displayName+" - "+type+": "+number);
                    arrNumeros.add(number);
                  }
                  phoneNumbers.close();
      }
      people.close();

      CONTATOS = arrContatos.toArray(new String[arrContatos.size()]);
      setListAdapter(new ArrayAdapter<String>(this, R.layout.contacts,
CONTATOS));
        ListView lv = getListView();

        //Listener
        lv.setOnItemClickListener(new AdapterView.OnItemClickListener() {
                @Override
                public void onItemClick(AdapterView<?> av, View v, int pos,
long id) {
                    onListItemClick(v,pos,id);
                }
            });

        lv.setTextFilterEnabled(true);
      }
```

Quadro 17 - Contatos Activity

A activity `ContatosActivity` tem a herança da classe `ListActivity`, esta que contempla os tratamentos necessários para apresentar uma lista e espera que seja atribuído valores para serem apresentados. A extração dos contatos da base nativa é realizada por um cursor definido pela seleção disposta no método `getContentResolver().query` para extrair os dados. Um restrição, `ContactsContract.Contacts.HAS_PHONE_NUMBER + "=1"`, foi utilizada para que somente contatos com telefones cadastrados sejam apresentados.

3.3.4.2 Selecionar localização

O caso de uso UC07, selecionar localização é inicializado quando o botão de mesmo nome é pressionado chamando da atividade `LocalizacaoActivity`. Esta estende a classe `MapActivity`, ou seja, é uma atividade especial voltando a apresentação de mapas.

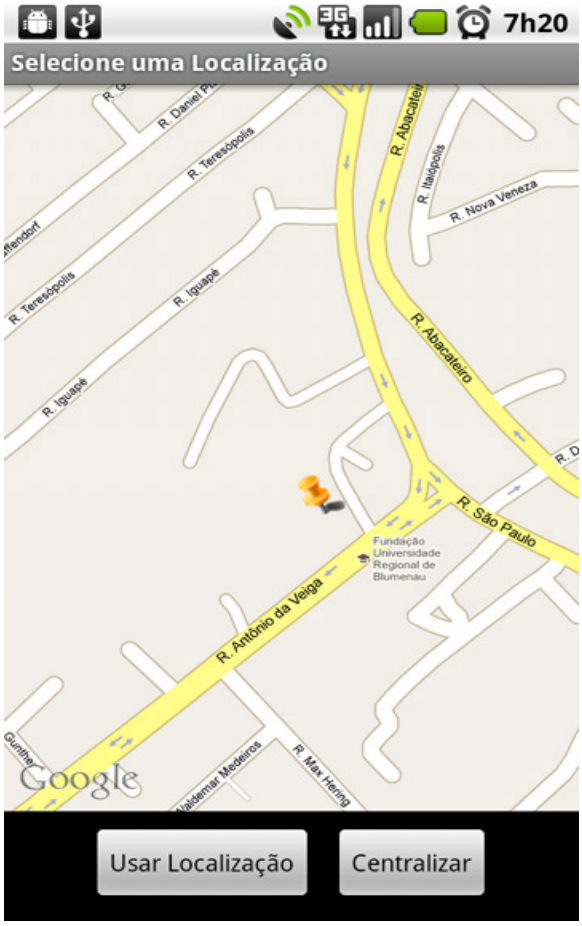


Figura 11 - Funcionalidade selecionar localização

O usuário manipula a localização efetuando um toque longo em cima do mapa. Esta ação dispara o método do `listener`, `onLongpress` atualizando o ponto que deve ser desenhado o alfinete (figura 11).

O aplicativo executa o método usarLoc apresentado no quadro 18 caso seja pressionado o botão "Usar Localização". Este método extrai a localização do ponto selecionando enviando a latitude e longitude para atividade que chamou a `LocalizacaoActivity`.

```
      private void usarLoc() {
            Intent data = new Intent();
            data.putExtra("atividade", "usarLoc");
            data.putExtra("longitude", (new Double(gp.getLongitudeE6() /
1E6)).toString());
            data.putExtra("latitude", (new Double(gp.getLatitudeE6() /
1E6)).toString());
            //data.putExtra("valor", longString);
            setResult(2, data);
            finish();
      }
```

Quadro 18 - Método `usarLoc`

Na activity `MensagemActivity` é realizado o tratamento na retorno a esta atividade

pelo método `onActivityResult`, como é apresentado no quadro 19.

```
      public void onActivityResult(int requestCode, int resultCode, Intent
data) {
            String atividade =
(String)data.getExtras().getString("atividade");
            if (atividade.equals("contatos")){
                  EditText et = (EditText)
findViewById(R.id.numeroDestino);
                  et.setText((String)data.getExtras().getString("valor"));
            }
            if (atividade.equals("usarLoc")){
                  String longitude =
(String)data.getExtras().getString("longitude");
                  String latitude =
(String)data.getExtras().getString("latitude");
                  etMemsagem.setText(etMemsagem.getText()+"
"+GoogleMapsURL+latitude+","+longitude);
            }
      }
```

Quadro 19 - Método `onActivityResult`

O método recupera a atividade do Intent, caso for igual a "usarLoc" é recuperado também a longitude e latitude. Esses valores são utilizados na montagem da URL a ser enviada. A URL é montada conforme o formato `http://maps.google.com/maps?q=-26.942124,-49.068374` sendo os valores -26.942124 e -49.068374 respectivamente a latitude e longitude. O uso da URL permite que qualquer celular com acesso a internet com um navegador compatível com os requisitos do Google Maps consiga visualizar a localização, não sendo necessário um aplicativo em especial. No caso do Android, o sistema operacional reconhece a URL sendo uma localização no Google Maps e possibilitada a escolha entre visualizar em um navegado ou na própria aplicação de mapas.

3.3.4.3 Enviar mensagem

O caso de uso UC06, enviar mensagem é iniciado após o usuário clicar no botão "Enviar mensagem" presente na tela inicial do aplicativo apresentada na figura 9. O botão tem um `listener` que ao ser pressionado chama o método `enviarMensagem` apresentado no quadro 20.

```
    //Enviar Mensagem
    public void enviarMensagem(){
        String numero = etNumeroDestino.getText().toString();
        String mesagem = etMemsagem.getText().toString();
        if (numero.length()>0 && mesagem.length()>0){
                //Instância um gerenciador de SMS
                SmsManager sms = SmsManager.getDefault();
                //Envia mensagem
                sms.sendTextMessage(numero, null, mesagem, null, null);
                //Apresenta mensagem de SMS enviado
                  Toast.makeText(getBaseContext(), "SMS enviado",
Toast.LENGTH_SHORT).show();
                  //Limpa campos
                  ((EditText)
findViewById(R.id.numeroDestino)).setText("");
                  ((EditText)
findViewById(R.id.message_editor)).setText("");
        }else{
            Toast.makeText(getBaseContext(), "Por favor, informe número e
mensagem.", Toast.LENGTH_SHORT).show();
        }
    };
```

Quadro 20 - Método enviarMensagem()

O método `enviarMensagem` recupera o valor presente nos campos “número destino” e “mensagem” da tela e utiliza o método `sendTextMessage` da classe `SmsManager` previamente instanciada para enviar o *sms*.

3.4 RESULTADOS E DISCUSSÃO

Os resultados encontrados ao término do trabalho são satisfatórios, pois o protótipo permite enviar para qualquer aparelho a localização exata em um *SMS* sendo selecionada no mapa ou recuperada através do GPS ou triangulação de antenas, consultar números na agenda telefônica do aparelho e listar as últimas mensagens enviadas e recebidas. Desta forma, o objetivo inicial foi alcançado e atende aos requisitos.

O quadro 18 apresenta um comparativo entre as principais características dos trabalhos correlatos em relação ao sistema desenvolvido.

Função	G-SMS	Flutter	Latitude	I am here
Envio de SMS	X	X		X
Extração localização atual	X	X	X	X
Selecionar localização	X			X
Uso de Google Maps	X	X	X	X
Visualização da localização recebida em um navegador	X	X		
Consulta de histórico de mensagens enviada	X			
Envia imagem em conjunto ao SMS		X		
Lista de usuários amigos			X	
Aplicação cliente para visualização da localização enviada				X

Quadro 21 - Diferenças entre os trabalhos correlatos

4 CONCLUSÕES

O protótipo desenvolvido permite aos usuários que possuem um *smartphone* com o sistema operacional Android, o envio de *SMS* georeferenciado, sendo que a visualização da localização pode ser realizada a partir de qualquer aparelho com acesso a internet, pois é utilizado a URL do Google Maps com os devidos parâmetros de localização. Desta forma os objetivos propostos foram alcançados.

O uso da IDE oficial do Android, o ambiente Eclipse, em conjunto com o plug-in ADT, ambos suportados pela Google permitiu o desenvolvimento de maneira apropriada. O Android SDK disponibiliza as bibliotecas necessárias para a extração da localização através do GPS. Inclusive, o emulador presente no SDK permite a simulação do uso do GPS entre os demais recursos disponibilizados.

Em relação aos trabalhos correlatos apresentados, verifica-se que o trabalho realizado atende parte das funcionalidades disponíveis no mercado, com visualização com ampla compatibilidade, uma vez que o único requerimento é um navegador compatível com o Google Maps. Este torna-se a principal característica e atrativo deste protótipo.

A pesquisa e implementação realizada me proporcionaram a chance de estudar a plataforma Android que tem sido utilizada amplamente em *smartphones* e *tablets*. Por exemplo, entender os processos de construção de um aplicativo para dispositivos móveis e visualizar a demanda de *softwares* para essa área, já que o mercado disponível que encontra-se em plena expansão.

4.1 EXTENSÕES

São propostas as seguintes extensões ao trabalho desenvolvido:

a) disponibilizar a aplicação para os smartphones que possuam outro sistema operacional instalado, além do android;
b) disponibilizar funcionalidade de compartilhamento de mensagem a redes sociais como Facebook e Twitter, assim como por e-mail;
c) disponibilizar função para selecionar uma localização informando o endereço.

REFERÊNCIAS BIBLIOGRÁFICAS

ANATEL. **2009 relatório.** [S.l.], 2010. Disponível em: <http:/www.anatel.gov.br>. Acesso em: 13 maio 2010.

AZEVEDO, Carlos. **Google Maps API.** [S.l.], 2008. Disponível em: <http://imasters.com.br/artigo/7832/linguagens/google-maps-api>. Acesso em: 10 abr. 2011

GEOSMS. **I am here.** [S.l.], 2010. Disponível em: <http://geosms.wordpress.com/>. Acesso em: 18 set. 2010.

GOOGLE. **Android developers**. [S.l.], [2011a]. Disponível em: <http://developer.android.com/index.html>. Acesso em: 30 mar. 2011.

GOOGLE. **API do Google Maps.** [S.l.], [2011b]. Disponível em: <http://code.google.com/intl/ptBR/apis/maps/documentation/javascript/index.html>. Acesso em: 30 mar. 2011

GOOGLE. **Google Latitude.** [S.l.], [2011c]. Disponível em: <http://www.google.com/mobile/latitude/>. Acesso em: 15 abr. 2011

JUICE WIRELESS INCORPORATION. **Flutter:** unlimited picture messaging for the iPhone. [S.l.], 2009. Disponível em: <http://www.juicecaster.com/flutter/>. Acesso em: 18 set. 2010.

LECHETA, Ricardo R. **Google Android:** aprenda a criar aplicações para dispositivos móveis com o Android SDK. São Paulo: Novatec, 2009.

LEE, Wei-Meng. **Beginning Android application development**. Indianapolis, Estados Unidos: Wiley Publishing, 2011.

MEIER, Reto. **Professional Android application development**. Indianapolis, Estados Unidos: Wiley Publishing, 2009.

MURPHY, Mark L. **Beginning Android**. Berkeley, Estados Unidos: Apress, 2009

NEMER, Amarílis C. **Estudo de usabilidade em telefones celulares.** 2006. 203 f. Trabalho Final de Mestrado Profissional em Computação – Instituto de Computação, Universidade Estaual de Campinas, Campinas. Disponível em: <http://libdigi.unicamp.br/document/?down=vtls000388132>. Acesso em: 15 set. 2010

STEELE, James; TO, Nelson. **The Android developer´s cookbook**: building aplications with the Android SDK. Boston, Estados Unidos: Pearson Education, 2010

VIRRANTAUS, Kirsi et al. **Developing GIS -** supported location-based services. [S.l.], 2001. Disponível em: <http://www.cs.jyu.fi/ai/papers/WGIS-01.pdf>. Acesso em: 25 ago. 2010.