UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ – UTFPR
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

VANDERSON COMACHIO

FUNCIONAMENTO DE BANCO DE DADOS EM ANDROID: UM ESTUDO EXPERIMENTAL UTILIZANDO SQLITE

TRABALHO DE DIPLOMAÇÃO

MEDIANEIRA
2011

VANDERSON COMACHIO

FUNCIONAMENTO DE BANCO DE DADOS EM ANDROID: UM ESTUDO EXPERIMENTAL UTILIZANDO SQLITE

Trabalho de Diplomação apresentado à disciplina de Trabalho de Diplomação, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas – CSTADS – da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para obtenção do título de Tecnólogo.

Orientador: Me. Fernando Schütz.

MEDIANEIRA

2011

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Diretoria de Graduação e Educação Profissional
Coordenação do Curso Superior de Tecnologia em
Análise e Desenvolvimento de Sistemas

TERMO DE APROVAÇÃO

FUNCIONAMENTO DE BANCO DE DADOS EM ANDROID: UM ESTUDO EXPERIMENTAL UTILIZANDO SQLITE

Por

VANDERSON COMACHIO

Este Trabalho de Diplomação (TD) foi apresentado às 13 h do dia 23 de novembro de 2011 como requisito parcial para a obtenção do título de Tecnólogo no Curso Superior de Tecnologia Análise e Desenvolvimento de Sistemas de Informação, da Universidade Tecnológica Federal do Paraná, *Campus* Medianeira. O candidato foi argüido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

Prof. *Me*. Fernando Schütz
UTFPR – *Campus* Medianeira
(Orientador)

Prof. *Me*. Claudio Leones Bazzi
UTFPR – *Campus* Medianeira
(Convidado)

Prof. Marcio Ângelo Matté
UTFPR – *Campus* Medianeira
(Convidado)

Prof. Juliano Rodrigo Lamb
UTFPR – *Campus* Medianeira
(Responsável pelas atividades de TCC)

RESUMO

COMACHIO, Vanderson. **Funcionamento de Banco de Dados em Android: Um Estudo Experimental Utilizando SQLite**. 2011. 61f. Trabalho de Conclusão de Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas de Informação. Universidade Tecnológica Federal do Paraná, Medianeira, 2011.

O amplo mercado para venda e o grande crescimento da demanda por aplicativos para dispositivos móveis está em constante crescimento. O presente trabalho explora diversas funcionalidades da plataforma Android e também a forma de armazenamento nativa do Android SDK, o *SQLite*. Um estudo experimental, na forma de protótipo o aplicativo iCADASTRO desenvolvido para demonstrar a utilização das tecnologias de programação para o Sistema Operacional Android abordadas durante a revisão bibliográfica e a integração com a API do Google Maps. Ao final são apresentados alguns trechos de código fonte e testes realizados com a aplicação.

PALAVRAS-CHAVE: Sistema Android. *SQLite*. Desenvolvimento de sistemas para dispositivos móveis. Google Maps.

ABSTRACT

COMACHIO, Vanderson. **Funcionamento de Banco de Dados em Android: Um Estudo Experimental Utilizando SQLite**. 2011. 61f. Trabalho de Conclusão de Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas de Informação. Universidade Tecnológica Federal do Paraná, Medianeira, 2011.

The large market for sale and the huge growth in demand for mobile applications is constantly growing. This paper explores various features of the Android platform as well as native storage form the Android SDK, SQLite. An experimental study in the form of the application iCADASTRO prototype developed to demonstrate the use of technology programming for the Android operating system developed during the literature review and integration with Google Maps API. At the end are some snippets of source code and tests with the application.

Keywords: IT. Android. SQLite. Google Maps.

LISTA DE FIGURAS

Figura 1 - Motorola DynaTac. 16
Figura 2 - Camadas da Arquitetura Android. 19
Figura 3 - Crescimento Android Market em relação a Apple AppStore. 23
Figura 4 - Aplicativos pagos mais baixados no Android Market. 24
Figura 5 - Google Maps dispositivo Android. 26
Figura 6 - Diagrama de casos de uso. 28
Figura 7 - Caso de uso: CadastrarGrupo. 28
Figura 8 - Caso de uso: CadastrarContato. 29
Figura 9 - Caso de uso: TracarRota. 30
Figura 10 - Caso de uso: EnviarEmail. 31
Figura 11 - Caso de uso: FazerChamada. 32
Figura 12 - Diagrama de classes. 32
Figura 13 - Diagrama de sequência AdicionarGrupo. 33
Figura 14 - Diagrama de seqüência AdicionarContato. 34
Figura 15 - Diagrama de seqüência TracarRota. 34
Figura 16 - Diagrama de seqüência EnviarEmail. 35
Figura 17 - Diagrama de seqüência FazerChamada. 36
Figura 18 - MER sistema iCADASTRO. 37
Figura 19 - Comando para obtenção chave Google Maps. 39
Figura 20 - Solicitação de assinatura para utilizar o Google Maps API. 39
Figura 21 - Estrutura do projeto no Eclipse. 40
Figura 22 – Classe Contato.java. 41
Figura 23 – Classe Grupo.java. 41
Figura 24 - Classe DatabaseHelper.java. 42
Figura 25 - Metodos onCreate e onUpgrade classe DatabaseHelper.java. 43
Figura 26 - Métodos para os grupos de contatos. 44
Figura 27 – Métodos para contatos. 45
Figura 28 - Tela Principal. 46
Figura 29 - Arquivo XML para Criação tela Principal. 46
Figura 30 - Método onCreate() Tela Principal. 48
Figura 31 - Tela Cadastrar Grupo. 49
Figura 32 - Tela Cadastro de Contatos. 49
Figura 33 - Tela Lista de Grupos. 50
Figura 34 - *AlertDialog* Editar Grupo. 50
Figura 35 - Tela Lista de Contatos. 51
Figura 36 - AlertDialog opções Contato. 51
Figura 37 - Método para Fazer Chamada. 51
Figura 38 - Método para recuperar email para envio e número para chamada. 52
Figura 39 - Método Enviar Email. 52
Figura 40 - Método chamada da tela TracarRota.java. 53

Figura 41 - Processo de Geocodificação. 53
Figura 42 - Verifição posicionamento atual dispositivo móvel. 54
Figura 43 - Método *LocationListener*. 54
Figura 44 - Método de atualização do local do dispositivo. 55
Figura 45 - Método tracarRota(). 55
Figura 46 - Passagem de coordenadas para o Google Maps. 56
Figura 47 - Classe que faz a conexão com o serviço do Google Maps. 56
Figura 48 - Método getRoute(). Fonte: autoria própria. 57
Figura 49 - Classe que armazena os pontos da rota recebidos do arquivo KML. 57
Figura 50 - Classe que armazena dados do caminho a ser traçado. 57
Figura 51 - Classe KMLHandler (a) 58
Figura 51 - Classe KMLHandler (b) 59
Figura 51 - Classe KMLHandler (c) 60
Figura 51 - Classe KMLHandler (d) 61

LISTA DE QUADROS

Quadro 1 - Pesquisa sobre Crescimento das Tecnologias móveis. 22
Quadro 2 - de vendas de Smarphones com Android segundo fabricantes. 22

LISTA DE SIGLAS

AD2P	*Advanced Audio Distribuition Profile*
ANSI	*American National Standards Institute*
API	*Application Programming Interface*
CDMA	*Code Division Multiple Access*
CLDC	*Connected Limit Device Cofiguration*
DDL	*Data Definition Language*
DML	*Data Manipulation Language*
GPS	*Global Positioning System*
GSM	*Global System for Mobile Communications*
IOS	*IPhone Operational System*
JAVA ME	*Java Micro Edition*
KML	*Keyhole Markup Language*
MIDP	*Mobile Information Device Profile*
OpenGL/ES	*OpenGL for Embedded Systems*
RIM	*Research in Motion Limited*
RMS	*Record Management System*
SDK	*Software Development Kit*
SGDB	Sistema Gerenciador de Banco de Dados
SIP	*Session Initiation Protocol*
SQL	*Structured Query Language*
TDMA	*Time Division Multiple Access*

SUMÁRIO

1 INTRODUÇÃO 11
1.1 OBJETIVO GERAL 12
1.2 OBJETIVOS ESPECÍFICOS 12
1.3 JUSTIFICATIVA 12
1.4 ESTRUTURA DO TRABALHO 14
2 REFERENCIAL TEÓRICO 15
2.1 A HISTÓRIA DOS CELULARES 15
2.2 SISTEMAS OPERACIONAIS PARA DISPOSITIVOS MÓVEIS 17
2.3 ANDROID 18
2.4 SQLITE 25
2.5 GOOGLE MAPS API 25
3 MATERIAIS E MÉTODOS 27
3.1 SISTEMA iCADASTRO 27
3.2 DESCRIÇÃO DO PROJETO 27
3.3 CASOS DE USO 28
3.4 DIAGRAMA DE CLASSES 32
3.5 DIAGRAMAS DE SEQUÊNCIA 33
4 RESULTADOS E DISCUSSÕES 38
4.1 OBTER CHAVE GOOGLE MAPS 38
4.2 APLICATIVO iCADASTRO 40
5. CONSIDERAÇÕES FINAIS 63
5.1 CONCLUSÃO 63
5.2 TRABALHOS FUTUROS 64
REFERÊNCIAS BIBLIOGRÁFICAS 65

1 INTRODUÇÃO

Um estudo sobre internet móvel realizado pela Morgan Stanley (2009) prevê que o acesso à Internet pelo celular deverá superar o acesso pelo computador em cerca de cinco anos. Este estudo aponta a popularização de *smartphones*, *e-readers* e *tablets* e a ampliação das redes 3G e 4G como os principais fatores que favorecem o aumento no acesso a internet pelo celular. A aceleração com que a Internet móvel vem crescendo atinge valores que a do desktop jamais alcançou. Nos próximos cinco anos, a Internet via dispositivo móvel deve ser mais utilizada que via desktop. No Japão o uso da internet móvel em alguns casos já superou o uso por meio de desktops (GILSOGAMO, 2009).

O total de acessos ativos à internet via celular 3G cresceu 90% na comparação entre o primeiro semestre de 2011 e de 2010, chegando a 21,3 milhões. Este número representa 76,34% dos acessos móveis – que somam 27,9 milhões – e 48,74% do total de conexões de banda larga no Brasil – que chegam a 43,7 milhões (GLOBO, 2011).

Devido a esse aumento na demanda de usuários, nota-se que a necessidade de aplicações para esse sistema operacional torna-se muito promissora no mercado de desenvolvimento.

Fazendo-se uso do Android SDK, o programador tem acesso a APIs que facilitam o desenvolvimento de aplicações assim como acesso a recursos de hardware do dispositivo.

Essas APIs fornecem recursos que tornam possível a integração de aplicações de dispositivos móveis com a *Web*, trazendo mais praticidade tanto ao usuário como ao desenvolvedor.

Através de ferramentas abertas para desenvolvimento, o desenvolvedor não tem a necessidade de pagar licença para criar aplicativos para Android.

Esta flexibilidade permite que possam ser desenvolvidas aplicações que utilizem funções nativas do Android SDK com a mesma agilidade e praticidade como as nativas do dispositivo. Um banco de dados SQLite pode armazenar dados pessoas do usuário para que possa manter uma base sobre qualquer tipo de dado a ser armazenado e possibilitando assim, desenvolver aplicativos mais voltados para as necessidades particulares de cada utilizador.

Diante da possibilidade de integração e utilização dos recursos nativos do Sistema Operacional Android, o presente trabalho de diplomação visa o desenvolvimento de um aplicativo Android, com persistência em banco de dados SQLite integrado com o Google Maps API.

1.1 OBJETIVO GERAL

Desenvolver uma aplicação, como estudo experimental, que funcione como uma agenda para dispositivos móveis, utilizando a plataforma Android SDK, a API do Google Maps e o banco de dados SQLite.

1.2 OBJETIVOS ESPECÍFICOS

Como objetivos específicos do projeto têm-se:

- Elaborar um referencial teórico sobre Android SDK, banco de dados SQLite e API do Google Maps juntamente com as ferramentas de desenvolvimento utilizadas.
- Desenvolver análise e projeto do sistema proposto como estudo experimental.
- Desenvolver o estudo experimental que faça uso da tecnologia Android SDK com SQLite.

1.3 JUSTIFICATIVA

O sistema operacional para dispositivos móveis do Google, o Android, conquistou o terceiro lugar entre as plataformas móveis mais utilizadas no mundo, com 10 milhões de aparelhos vendidos durante o segundo trimestre de 2010, elevando a participação de mercado do Android para 17.2%. Quanto aos outros

sistemas operacionais móveis, Symbian continua como líder, utilizado em 25.4 milhões de aparelhos. Em seguida aparece a RIM, fabricante do BlackBerry, com vendas de 11.2 milhões de aparelhos. A Apple aparece em quarto lugar (8.7 milhões de smartphones vendidos), seguida pela Microsoft e o Windows Mobile, com 3.1 milhões de unidades (OLHAR DIGITAL, 2010).

No contexto de aplicações para dispositivos móveis, a escassez de recursos computacionais, tais como memória, capacidade de armazenamento, inerentes a dispositivos móveis pessoais (celulares e PDAs), combinado com a falta de suporte, faz com que os desenvolvedores para estes tipos de dispositivos não desfrutem dos benefícios de um SGBD2. Muitas tecnologias para desenvolvimento móvel, assim como no início do desenvolvimento de aplicações desktop, se baseiam em arquivos para armazenar dados. Por exemplo, a plataforma Java ME possui o Record Management System (RMS). O RMS consiste em uma API que dispõe de métodos para manipular registros. Entretanto, a manipulação de dados nessa API não poupa o desenvolvedor de lidar com rotinas de baixo nível para serialização dos dados, já que a API só trabalha com arrays de bytes (DEVMEDIA, 2010).

Em contrapartida, a plataforma Android, dentre uma série de inovações e facilidades, trouxe suporte nativo ao SQLite. Uma poderosa biblioteca de banco de dados baseado em SQL (*Structured Query Language*) que atua como um “mini-SGBD”, capaz de controlar diversos bancos de dados que podem conter diversas tabelas. O desenvolvedor pode criar o banco de dados e as tabelas, assim como manipular seus dados através dos comandos DDL (*Data Definition Language*) e DML (*Data Manipulation Language*) do SQL padrão. Isso traz um ganho considerável de produtividade, pois o desenvolvedor agora pode apenas focar nas regras de negócio, tendo em vista que os serviços para persistência de dados são fornecidos pelo SQLite (DEVMEDIA, 2010).

O mercado de aplicativos para celulares que utilizam o sistema operacional Android é muito promissor, pois a demanda de usuários é grande e vem crescendo a cada ano. Por fazer uso de tecnologia Java, traz praticidade ao desenvolvedor, permitindo o uso de diversos recursos. Sendo assim, Android desperta o interesse em aprofundar o conhecimento nesta tecnologia.

1.4 ESTRUTURA DO TRABALHO

O presente trabalho divide-se em cinco capítulos, sendo que:

- o primeiro capítulo apresenta uma introdução à evolução dos celulares e o desenvolvimento das tecnologias em dispositivos móveis.
- o segundo capítulo um estudo mais aprofundado em Android e SQLite. O terceiro apresenta um estudo de caso para apresentação da forma de armazenamento de dados e recuperação em um banco de dados SQLite no Android SDK
- o quarto capítulo apresenta os resultados do desenvolvimento do aplicativo.
- no quinto capítulo estão dispostas as considerações finais referentes a todo conteúdo apresentado no presente trabalho.

.

2 REFERENCIAL TEÓRICO

Ao decorrer deste capitulo será apresentado o referencial teórico deste trabalho de diplomação.

2.1 A HISTÓRIA DOS CELULARES

Antigamente, quando o aparelho de telefone fixo surgiu, o preço de aquisição era altíssimo. Os valores para utilizar uma linha telefônica eram extremamente altos. Com o passar do tempo, o aparelho e as linhas telefônicas foram ganhando maior acessibilidade, tornando o meio de comunicação mais popular entre as pessoas. Os primeiros a utilizarem as linhas telefônicas foram as empresas passando logo após a serem utilizadas por usuários domésticos. Na década seguinte, o mesmo aconteceu com os dispositivos móveis - a novidade era tão impressionante que muitos jamais pensavam em adquirir um. E mais uma vez, a história se repetiu: com a popularização das operadoras de telefonia móvel e o barateamento dos dispositivos, não demorou para que o celular se tornasse fundamental na vida de qualquer indivíduo (CELESTINO, 2010). No Brasil, por exemplo, já ultrapassa 200 milhões o número de celulares.

Com o passar dos anos, os dispositivos móveis passaram por diversas evoluções, onde cada uma é marcada por um novo recurso ou uma nova ferramenta acoplada ao dispositivo. O pioneiro se chamava *DynaTAC*, celular fabricado pela Motorola em 1982, e surpreendia em apenas realizar e receber ligações sem a necessidade de um fio que o conectasse à linha telefônica. A Figura 01 apresenta o pioneiro dos celulares.

Figura 1 - Motorola DynaTac.
Fonte: MOTOROLA, 2011.

Apesar do peso de 1 kg e o tamanho que quase chegava às dimensões de uma garrafa de 600 ml, o dispositivo chamou a atenção de admiradores e abriu as portas para as novas gerações de celulares (CELESTINO, 2010).

Por volta de 1990, os fabricantes já estavam bastante desenvolvidos para disponibilizar aparelhos mais leves e amigáveis. Três tecnologias surgiram nesta época: TDMA, CDMA e GSM. Esta geração trouxe inovações como as mensagens SMS e durou até o fim do milênio. Logo após, os visores coloridos, mensagens multimídias (MMS), acesso à Internet através de páginas próprias denominadas WAP, implantação de câmeras digitais nos aparelhos e a possibilidade de vídeo conferências foram grandes “saltos tecnológicos” dos aparelhos (JORDÃO, 2009).

A venda de *smartphones* no Brasil cresceu mais de 102% no Brasil em 2011. Os dispositivos com tecnologia Android estão se tornando mais acessíveis financeiramente e culminando na queda de vendas de celulares tradicionais. Atualmente são comercializados 45 milhões de dispositivos móveis no Brasil. Dentre os *smartphones* mais vendidos, os dispositivos que contém o sistema operacional Android lidera o mercado de vendas no Brasil (TOZZETO, 2011).

2.2 SISTEMAS OPERACIONAIS PARA DISPOSITIVOS MÓVEIS

Os sistemas operacionais convencionais utilizados em computadores visam tornar a utilização destes equipamentos mais eficiente e conveniente. Um sistema operacional oferece diversos recursos para manipulação e execução de tarefas nativas do computador. Toda a área de acesso aos periféricos, compartilhamento de recursos e programas são feitos pelo sistema operacional (OLIVEIRA et al, 2001). Estes sistemas no ambiente de dispositivos móveis trazem consigo a mesma linha de funcionamento, procurando sempre tornar mais simples e eficaz a utilização do dispositivo. Funções como acesso rápido a e-mails, Internet e redes de relacionamento estão cada vez mais indispensáveis ao usuário de dispositivos móveis e vêm sendo implantados diretamente no sistema operacional, dispensando a instalação posterior ou manual do utilizador.

No âmbito de dispositivos móveis podem-se citar cinco grandes sistemas operacionais presentes. São eles:

- ANDROID: Sistema operacional desenvolvido pela Google. Permite que um aplicativo possa chamar qualquer funcionalidade nativa do telefone, como fazer chamadas, enviar mensagens de texto, ou usar a câmera, permitindo que os desenvolvedores criar experiências mais ricas e coesas para os usuários. Android é construído sobre o *kernel* do Linux. Além disso, utiliza uma máquina virtual personalizado que foi projetado para aperfeiçoar os recursos de memória e *hardware* em um ambiente móvel (ANDROID BRASIL, 2011).
- IOS: O iOS é o sistema operacional para dispositivos móveis da Apple, derivado do Mac OS X. Lançado primeiramente para dispositivos iPhone, o sistema também é compatível com outros aparelhos da empresa, como *iPod*, *iPad*, *iPod Touch* e *Apple TV*. O iOS é utilizado por cerca de 16% dos usuários de smartphone, sendo superado pelos sistemas operacionais Android e Symbian (TECHTUDO, 2011).
- SYMBIAN: é um sistema operacional aberto produzido pela Symbian Ltd e licenciado pelos maiores fabricantes de aparelhos celulares. É projetado para as exigências específicas dos celulares 2G, 2.5G e

3G. Este sistema operacional é exclusivo do fabricante de dispositivos móveis NOKIA (NOKIA DEVELOPER, 2011).

- WINDOWS MOBILE: O sistema Windows Mobile é um sistema operacional muito completo e seguro. Excelente para aplicações corporativas. Utiliza recursos e interfaces semelhantes ao Windows utilizados em computadores *desktop* (PALMBRASIL, 2011).

Para o desenvolvimento do presente trabalho optou-se em utilizar o sistema operacional Android SDK pelo fato de ser uma ferramenta de *software* livre e o seu crescimento traz diversas vantagens e a possibilidade de manipular e utilizar ao máximo os recursos dos dispositivos móveis.

2.3 ANDROID

O Android é um conjunto de *software*s para dispositivos móveis que inclui um sistema operacional, *middleware* e aplicativos importantes. O Android SDK fornece as ferramentas e APIs necessárias para começar a desenvolver aplicativos que executam em dispositivos com Android (GOOGLE, 2011).

Apesar de ser baseado no *kernel* do Linux, existe pouca coisa em comum com as versões disponibilizadas para desktop. À grosso modo, o Android é uma máquina virtual Java rodando sobre o *kernel* (estrutura principal que liga os aplicativos ao processamento real do sistema, é o gerenciador de recursos) Linux, dando suporte para o desenvolvimento de aplicações Java através de um conjunto de bibliotecas e serviços (PRADO, 2011). Sua arquitetura tem basicamente quatro camadas demonstradas no esquema apresentado na Figura 02.

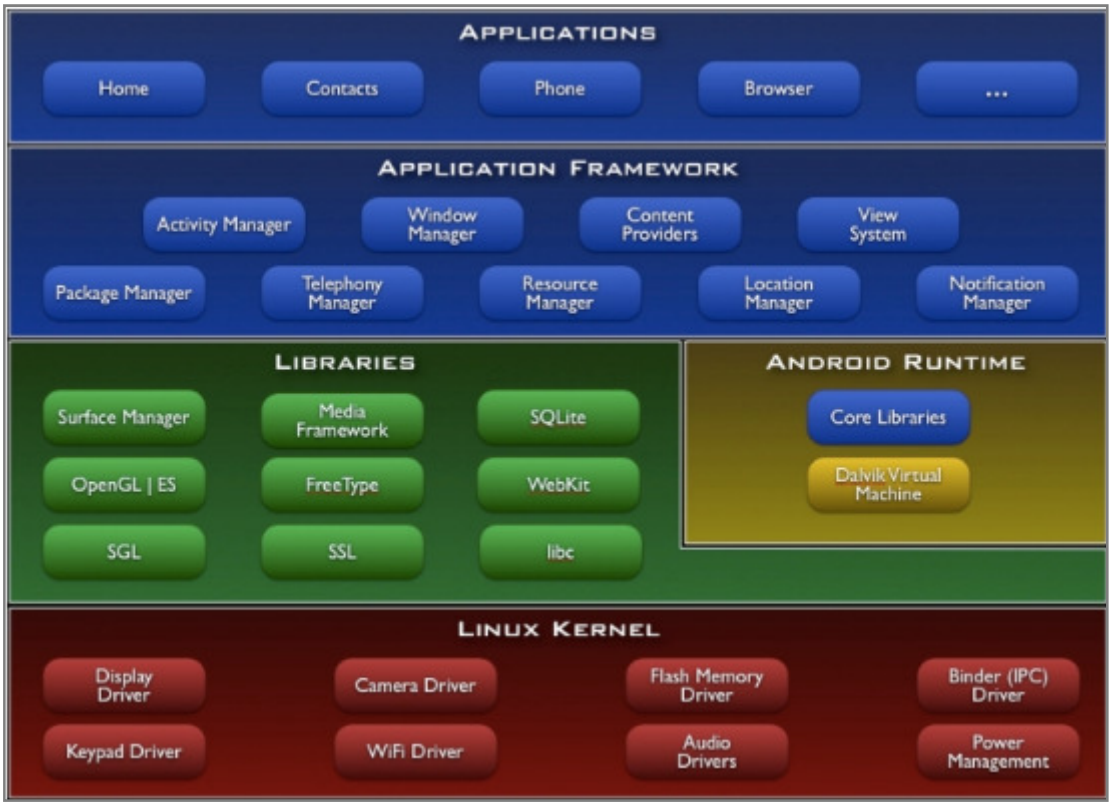

Figura 2 - Camadas da Arquitetura Android.
Fonte: RODRIGUES, 2011.

As quatro camadas demonstradas na Figura 02 são:

- Linux Kernel: O Google usou a versão 2.6 do Linux para construir o *kernel* do Android, o que inclui os programas de gerenciamento de memória, as configurações de segurança, o software de gerenciamento de energia e vários *drivers* de *hardware* (STRICKLAND, 2009).
- Bibliotecas e Serviços: Nesta camada encontra-se as principais bibliotecas utilizadas pelo Android, dentre elas a *OpenGl/ES* para trabalhar com gráficos e a SQLite que permite manipular a base de dados. Encontra-se também a Dalvik, que é uma JVM (Java Virtual Machine) para rodar o conteúdo Java. A maior parte destas bibliotecas foi desenvolvida em C++ (PRADO, 2011);
- Frameworks: Desenvolvida basicamente por completo em Java, esta camada é responsável pela interface com as aplicações Android. Ela provê um conjunto de bibliotecas para acessar os diversos recursos do dispositivo como interface gráfica, telefonia, serviço de localização (GPS), banco de dados persistente, armazenamento no cartão SD,

etc. Fornece blocos de alto nível de construção utilizados para criação de aplicações. O framework vem pré-instalado com o Android (MOBILEIN, 2010).

- Aplicações: Nesta camada que ficam as aplicações (desenvolvidas em Java) para o Android. E é um dos grandes segredos do sucesso da plataforma, já que possui mais de 250.000 aplicações no Android Market, e continua crescendo cada dia que passa (PRADO, 2011).

A máquina virtual Java rodando sobre o *kernel* do Linux presente no SO Android denomina-se Dalvik. A Dalvik não consome *bytecode* (estágio intermediário entre o código-fonte e a aplicação final) Java, mas sim *dexcode*. Para isso, o Google desenvolveu uma ferramenta, chamada "dx", que converte Java *bytecodes* (*.class) em *dexcodes* (*.dex). Além disso, desde a versão 2.2 (Froyo), o Android possui uma implementação de JIT (*Just-in-time*), que compila *dexcodes* para a arquitetura-alvo em tempo de execução, tornando a execução dos processos consideravelmente mais rápidos, já que não precisa ficar interpretando *dexcodes*. Junto com a máquina virtual Dalvik, o Android usa o *framework* Apache Harmony, desenvolvido pela Apache Software Fundation como biblioteca padrão de classes Java (PRADO, 2011).

Atualmente, o Android conta com as seguintes versões:

- Versão 1.0: Lançada apenas no mercado americano, mas sem grandes destaques (SOARES, 2011).
- Versão 1.1: Lançada em Fevereiro de 2009, foi considerada a primeira versão de verdade, pois já agrupava diversos recursos da Google nos dispositivos móveis (SOARES, 2011).
- Versão 1.5 (Cupcake): Lançada em Abril de 2009. Primeira versão em grande escala, trazia recursos de transferência de vídeos diretamente para o *YouTube*. Alem disso, suporte a teclados QWERTY, envio de fotos para o *Picasa* e presença de *widgets*. Não é mais usada nos aparelhos lançados nos dias de hoje, mas ainda está presente e possui aplicativos no Android Market (IG TECNOLOGIA, 2011);
- Versão 1.6 (Donut): Lançada em Setembro de 2009. Ainda utilizada nos aparelhos com menor performance. Trouxe consigo uma interface nova e o suporte a câmeras para fotografia e vídeo, aplicativo Android

Market facilitando o acesso as aplicações disponíveis no site oficial da Google (SOARES, 2011).

- Versão 2.0/2.1 (Eclair): Lançada em Outubro de 2009. Trouxe inúmeras inovações. Nova interface e um aplicativo de contatos suporte para câmeras com flash, email Exchange, papeis de parede animados, nova versão do Google *Maps*, suporte a HTML 5, Zoom digital, Bluetooth 2.1. Versão ainda muito utilizada em aparelhos de ponta no Brasil (IG TECNOLOGIA, 2011).
- Versão 2.2 (Froyo): Lançada em Maio de 2010. Trouxe mais velocidade, melhorias nos recursos de copiar e colar e gerenciador de downloads, Adobe Flash 10.1, instalação de aplicativos no cartão de memodia (SD Card) entre outras., atualização de aplicativos em massa dentre outras (SOARES, 2011)..
- Versão 2.3 (GingerBread): Lançado em Novembro de 2010, trouxe como pontos principais a economia de energia, suporte ao protocolo SIP para chamadas via Internet (IG TECNOLOGIA, 2011).
- Versão 3.0 (Honeycomb): Lançado em Janeiro de 2011. Versão disponível para Tablets, mas estão adaptando-a para funcionar em qualquer dispositivo com Android (SOARES, 2011).

No desenvolvimento prático do presente trabalho para fins de testes foi utilizada a versão 2.1 (*Eclair*).

Um estudo revelado pela comScore (COMSCORE, 2011), demonstra que o mercado europeu praticamente “adotou” o sistema operacional móvel da Google, pois a taxa de crescimento de seu uso é de 16,1% em um ano (julho 2010/2011). Neste mesmo período o iOS obteve um crescimento de apenas 1,2%, mantendo-se em terceiro lugar - e com uma taxa semelhante ao crescimento da RIM, na quarta posição. Os dados foram contabilizados entre a França, Alemanha, Itália, Espanha e Reino Unido (MIRANDA, 2011). A pesquisa é apresentada na Quadro 01.

Percentual de Usuários de Smartphones por Plataforma Total de assinantes de telefonia móvel no grupo EU5 (ALE, FRA, RU, ITA e ESP) Fonte: comScore MobiLens			
	Percentual dos usuários de Smartphones		
Plataforma Smartphone	jul/10	jul/11	Diferença (%)
Total de Usuários de Smartphones	100%	100%	0
Symbian	53,9%	37,8%	-1610,0%
Google	6,0%	22,3%	1620,0%
Apple	19,0%	20,3%	1,2%
RIM	2,0%	9,4%	1,5%
Microsoft	11,5%	6,7%	-4,8%

Quadro 1 - Pesquisa sobre Crescimento das Tecnologias móveis.
Fonte: Adaptado de COMSCORE, 2011.

Existem várias empresas que fabricam *smartphones* com o sistema operacional Android, e com isso gera-se uma guerra particular entre os principais concorrentes (MIRANDA, 2011). O Quadro 02 apresenta uma comparação disponibilizada pela empresa comScore, a respeito dos principais fabricantes de *smartphones* com Android.

Percentual de Usuários de Smartphones Android por Fabricante Total de assinantes de telefonia móvel no grupo EU5 (ALE, FRA, RU, ITA e ESP) Fonte: comScore MobiLens						
	Percentual de Usuários de Smartphones com Android					
Fabricante Smartphone	EU5	Reino Unido	França	Alemanha	Itália	Japão
Total Usuários de Smartphones com Android	100,00%	100,00%	100,00%	100,00%	100,00%	100,00%
HTC	34,60%	50,90%	21,80%	29,80%	24,80%	31,90%
Samsung	31,70%	21,60%	42,30%	32,40%	35,60%	33,90%
Sony Ericsson	15,10%	14,20%	17,40%	14,60%	8,30%	18,80%
LG	5,80%	3,50%	7,80%	6,80%	10,10%	3,40%
Motorola	3,60%	2,10%	3,00%	7,80%	2,40%	2,80%

Quadro 2 - de vendas de Smarphones com Android segundo fabricantes.
Fonte: Adaptado de COMSCORE, 2011

O Android Market é uma loja *online* desenvolvida pelo Google para comercialização de aplicações para a plataforma Android. Surgiu em 2008 para concorrer com os aplicativos da AppStore, da concorrente Apple.

Um estudo desenvolvido por Daniel Meehan (MEEHAN, 2011), da empresa Research2Guidance, em 15 de Agosto de 2011, apresenta graficamente o crescimento do Android Market em relação ao Apple AppStore. A Figura 03 a seguir demonstra o estudo:

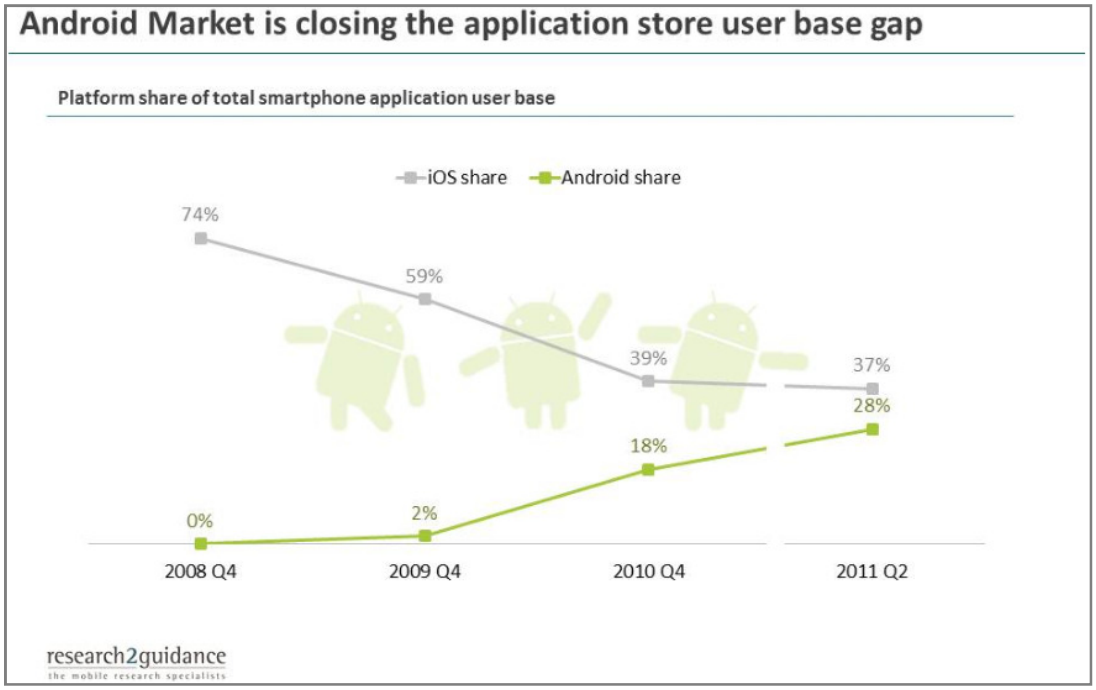

Figura 3 - Crescimento Android Market em relação a Apple AppStore.
Fonte: MEEHAN, 2011.

Observa-se na Figura 03 que em 2009 o Android apresentava apenas 2% do mercado de *smartphones* contra 59% da Apple. Mas até no segundo trimestre de 2011 a diferença caiu para apenas 9%. A Apple já foi o líder indiscutível, com a App Store. Se levados em consideração apenas os usuários de *smartphones*, o Android já está muito além, no entanto a Apple App Store é base de usuários alimentada por milhões de usuários de *iPod Touch* e *iPad*. Até o momento a Apple vendeu mais de 240 milhões de dispositivos inteligentes (iPhones Touch, iPod e iPads), deixando para trás Android com 170 milhões de dispositivos (MEEHAN, 2011).

Outro estudo desenvolvido por Egle Mikalajunaite (MIKALAJUNAITE, 2011), da mesma empresa Research2Guidance, apresenta os principais aplicativos pagos mais baixados. Na Figura 04 o gráfico com os números referentes até o final do mês de Agosto de 2011.

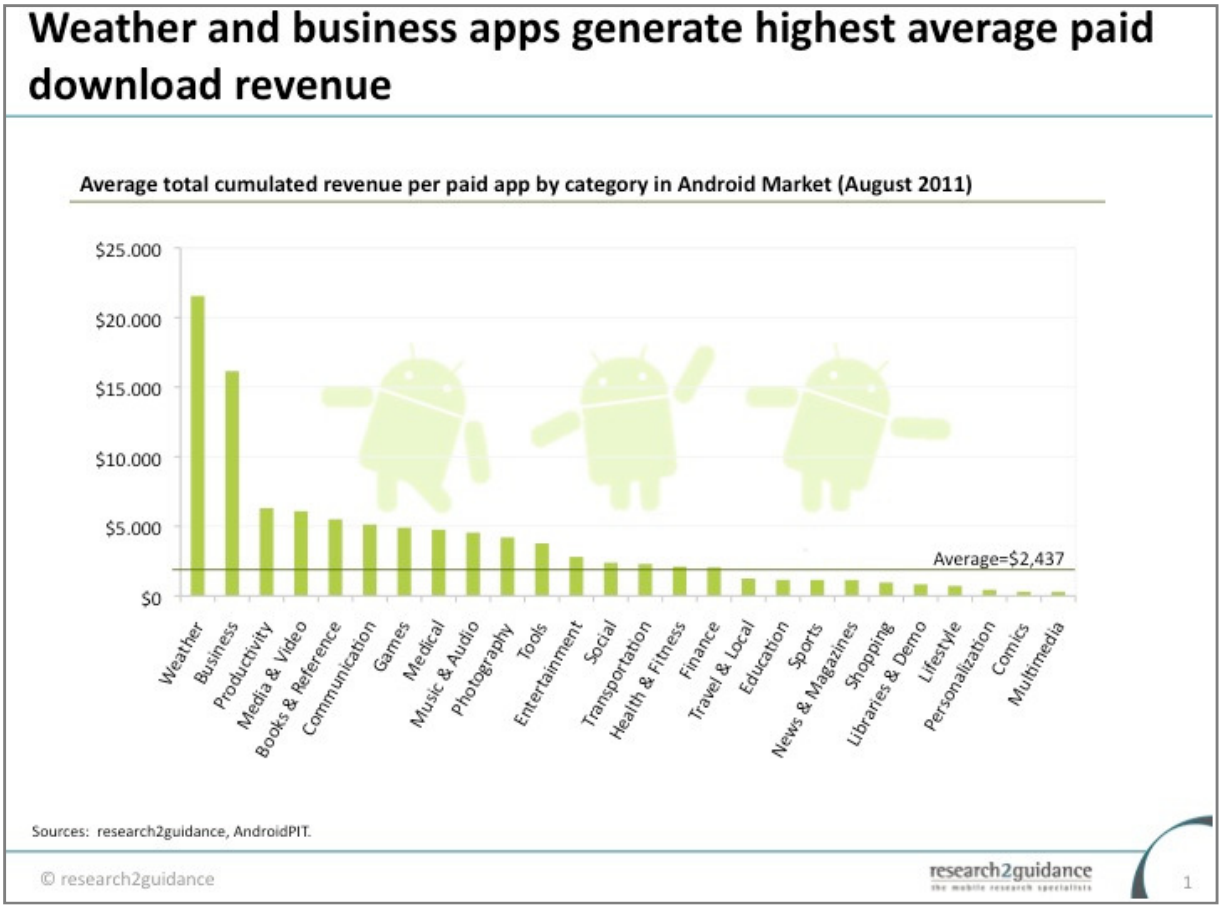

Figura 4 - Aplicativos pagos mais baixados no Android Market.
Fonte (MIKALAJUNAITE, 2011).

2.4 SQLITE

SQLite é um banco de dados *Open Source*, utilizado no Android. SQLite suporta padrão dos bancos de dados relacionais como a sintaxe SQL, operações e instruções preparadas. Além disso, requer apenas pouca memória em tempo de execução (aprox. KByte 250). A utilização do SQLite em Android não requer nenhuma configuração inicial, apenas é necessário especificar a instrução SQL para gerar o banco de dados e ele é criado automaticamente. SQLite suporta dados do tipo `TEXT` (similar a `String` em Java), `INTEGER` (semelhante a `LONG` em Java) e `REAL` (Semelhante a `Double` em Java). Todos os outros tipos devem ser convertidos em um desses tipos antes de armazená-los no banco de dados. O SQLite não valida se os campos enviados para armazenamento são iguais aos campos definidos nas colunas, cabe ao desenvolvedor validá-los (VOGEL, 2011).

Na prática, o SQLite é capaz de criar um arquivo em disco, ler e escrever diretamente sobre este arquivo. O arquivo criado possui a extensão ".db" e é capaz de manter diversas tabelas. Uma tabela é criada com o uso do comando CREATE TABLE da linguagem SQL. Os dados das tabelas são manipulados através de comandos DML (`INSERT`, `UPDATE` e `DELETE`) e são consultados com o uso do comando `SELECT` (GONÇALVES, 2011).

Por padrão, as bases de dados SQLite criadas são armazenadas no diretório "`DATA/data/APP_NAME/databases/FILENAME.db`, onde "`APP_NAME`" é o nome da aplicação e "`FILENAME`" é o nome da base de dados atribuída na hora da criação.

2.5 GOOGLE MAPS API

O Google Maps é um serviço da Google para fornecimento de informações em um espaço georeferenciado através de mapas. O Google Maps API consiste basicamente em um conjunto de bibliotecas que permitem acessar diversos recursos disponibilizados pelo Google Maps. Através dela é possível a busca e visualização de mapas (CONSTANTINI, 2009).

No Sistema Operacional Android, pelo fato de ser desenvolvido pelo Google, o aplicativo “Google Maps” já vem nativo no dispositivo móvel. A Figura 05 apresenta a interface do Google Maps em um dispositivo Android.

Figura 5 - Google Maps dispositivo Android.

Como Observado na Figura 05, o Google Maps é muito utilizado em dispositivos Android. Com os botões de zoom permite aproximar e fazer qualquer outra função como qualquer GPS.

3 MATERIAIS E MÉTODOS

Neste capítulo serão descritas as ferramentas e métodos utilizados neste trabalho de diplomação.

Para o desenvolvimento dos diagramas UML utilizou-se a ferramenta ASTAH COMMUNITY, ferramenta livre para modelagem de diagramas.

Para o desenvolvimento da aplicação prática da integração da iCADASTRO com o Google Maps utilizou-se o Eclipse Helios, versão 3.6.2 e o Android SDK, versão 10.0.0.

3.1 SISTEMA iCADASTRO

Ao decorrer desta seção será apresentada a modelagem do projeto iCADASTRO.

3.2 DESCRIÇÃO DO PROJETO

O trabalho desenvolvido consiste em um sistema de cadastro de contatos que, além dos dados tidos como “básicos” em um cadastro (nome, celular, email, endereço, numero e cidade), possibilite traçar uma rota utilizando o Google Maps API entre a posição local do dispositivo até o contato adicionado. Através desta funcionalidade acredita-se que, localização e orientação via mapa são facilitadas entre o usuário e o contato.

Assim pode-se ter como base que os requisitos principais do sistema residem no cadastro de usuário, grupos, envio de e-mails, chamadas e rotas.

3.3 CASOS DE USO

A Figura 06 mostra o diagrama de casos de uso do sistema iCADASTRO:

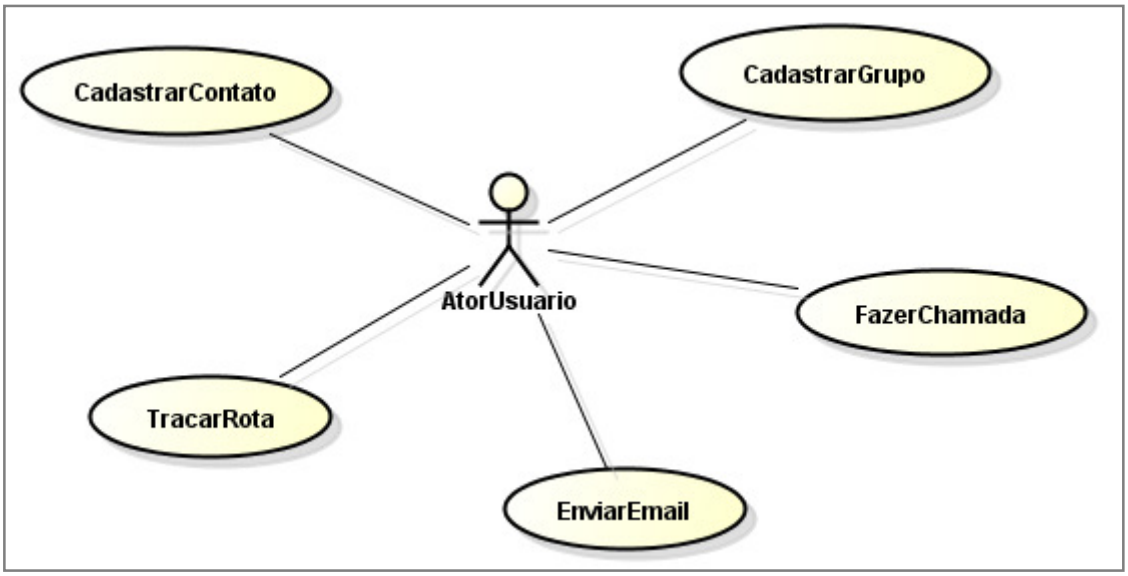

Figura 6 - Diagrama de casos de uso.

Como observado na Figura 06, o modelo de caso de uso trata o Ator Usuário como o principal ator do sistema.

Cada caso de uso, que nada mais são que representações de interações entre o ator, que neste caso é o usuário, com as demais funções do sistema, foram elaborados diagramas para tornar mais visível estas ações. Neste titulo será descrito cada caso de uso de acordo com o ator do aplicativo iCADASTRO.

Número: 01

Caso de Uso: CadastrarGrupo

Descrição: Este caso de uso descreve o cadastramento de grupos na base de dados.

Ator: AtorUsuario.

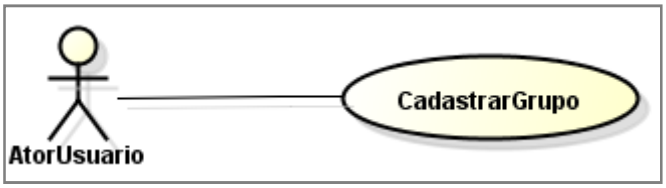

Figura 7 - Caso de uso: CadastrarGrupo.

Curso Normal:

1. Usuário informa os dados do grupo para cadastro;
2. Sistema verifica se os campos estão preenchidos;
3. Sistema exibe mensagem de sucesso ao cadastrar;
4. Sistema exibe a pagina de cadastro novamente com os campos vazios.

Curso Alternativo:

2. O sistema encontra campos obrigatórios não cadastrados;
2.1 O sistema informa a falha ao usuário e aguarda a entrada dos dados;
2.2 Encerra o caso de uso.

Número: 02

Caso de Uso: CadastrarContato

Descrição: Este caso de uso trata da inserção de contatos na base de dados.

Ator: AtorUsuario

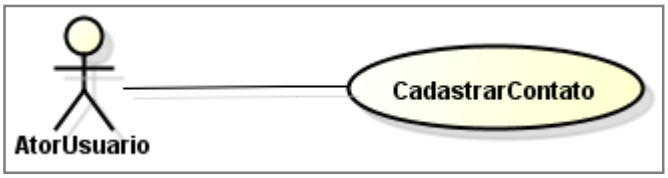

Figura 8 - Caso de uso: CadastrarContato.

Curso Normal:

1. Usuário informa os dados do contato para cadastro;
2. Usuário seleciona o grupo desejado;
3. Sistema verifica se os campos estão preenchidos;
4. Sistema exibe mensagem de sucesso ao cadastrar;
5. Sistema exibe a pagina de cadastro novamente com os campos vazios.

Curso Alternativo:

2. O sistema não encontra grupos cadastrados;
2.1 O sistema retorna um objeto nulo;
2.2 O sistema encerra o caso de uso.
3. O sistema encontra campos obrigatórios não cadastrados;
3.1 O sistema informa a falha ao usuário e aguarda a entrada dos dados;

3.2 Encerra o caso de uso.

Número: 03

Nome: TracarRota

Descrição: Este caso de uso trata da solicitação do usuário traçar uma rota até o contato cadastrado

Ator: AtorUsuario

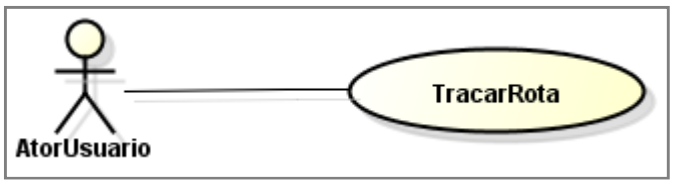

Figura 9 - Caso de uso: TracarRota.

Curso Normal:

1. O usuário solicita listagem de contatos;
2. O usuário seleciona um contato cadastrado;
3. O usuário seleciona a opção traçar rota;
4. O sistem verifica as coordenadas de origem e destino
5. O sistema traça a rota no Google Maps;

Curso Alternativo:

2. O sistema não encontra contatos cadastrados;

2.1 O sistema exibe uma mensagem de lista vazia;

2.2 Encerra o caso de uso.

4. O sistema não consegue se comunicar com o acesso ao GoogleMaps;

4.1 O sistema exibe uma mensagem de aviso ao usuário;

4.2 Encerra o caso de uso.

Número: 04

Caso de Uso: EnviarEmail

Descricao: Este caso de uso trata do envio de email para contatos cadastrados

Ator: AtorUsuario

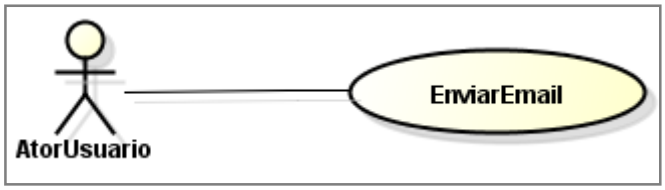

Figura 10 - Caso de uso: EnviarEmail.

Curso Normal:

1. O usuário solicita listagem de contatos;
2. O usuário seleciona um contato cadastrado;
3. O usuário solicita a opção enviar email;
4. O usuário informa dados adicionais para envio;
5. O sistema envia email para o contato;

Curso Alternativo:

2. O sistema não encontra contatos cadastrados;

2.1 O sistema exibe uma mensagem de lista vazia;

2.2 Encerra o caso de uso.

4. O usuário não informa dados adicionais ao email;

4.1 Encerra o caso de uso.

5. O sistema tenta enviar email, mas não há conexão ou algum problema de rede é identificado;

5.1 Encerra o caso de uso.

Número: 05

Caso de Uso: FazerChamada

Descrição: Este caso de uso trata da opção de realizar chamada a um contato cadastrado.

Ator: AtorUsuario

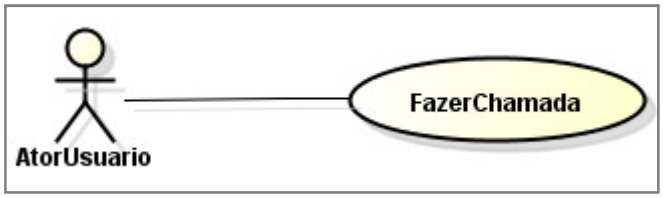

Figura 11 - Caso de uso: FazerChamada.

Curso Normal:

1. O usuário seleciona listagem de contatos;
2. O usuário seleciona um contato na lista;
3. O usuário seleciona a opção fazer chamada;
4. O sistema disca o numero contido no banco de dados e retorna como ligação ao usuário.

A descrição de casos assim como o curso normal e alternativo é de grande importância para o desenvolvedor de sistemas e aplicativos.

3.4 DIAGRAMA DE CLASSES

A Figura 12 demonstra os relacionamentos entre as classes do sistema iCADASTRO. Estas classes estão alocadas no pacote `com.tcc.pojo` dentro do sistema.

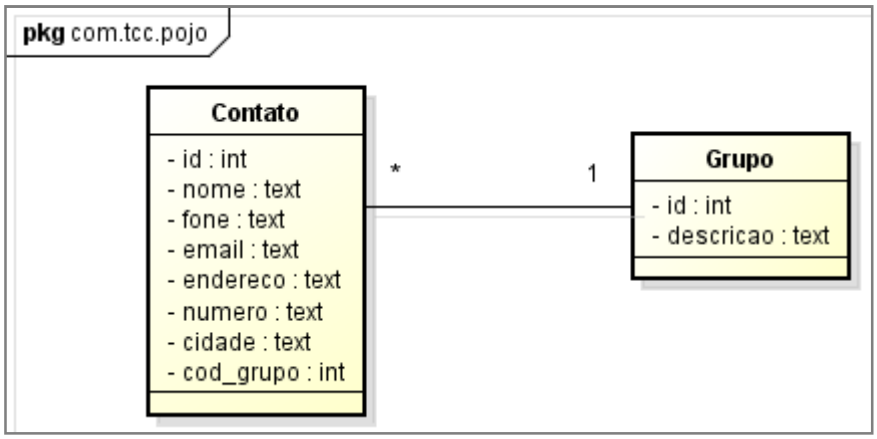

Figura 12 - Diagrama de classes.

O sistema iCADSTRO conta com duas tabelas para cadastro de contato e grupo. Um contato pode estar em um grupo, mas um grupo pode conter vários contatos.

3.5 DIAGRAMAS DE SEQUÊNCIA

Diagramas de sequência se mostram úteis para demonstrar as etapas que devem ser efetuadas para cada funcionalidade do sistema. Ligados diretamente com os casos de uso. Este título trata dos diagramas de sequência para cada caso de uso do sistema.

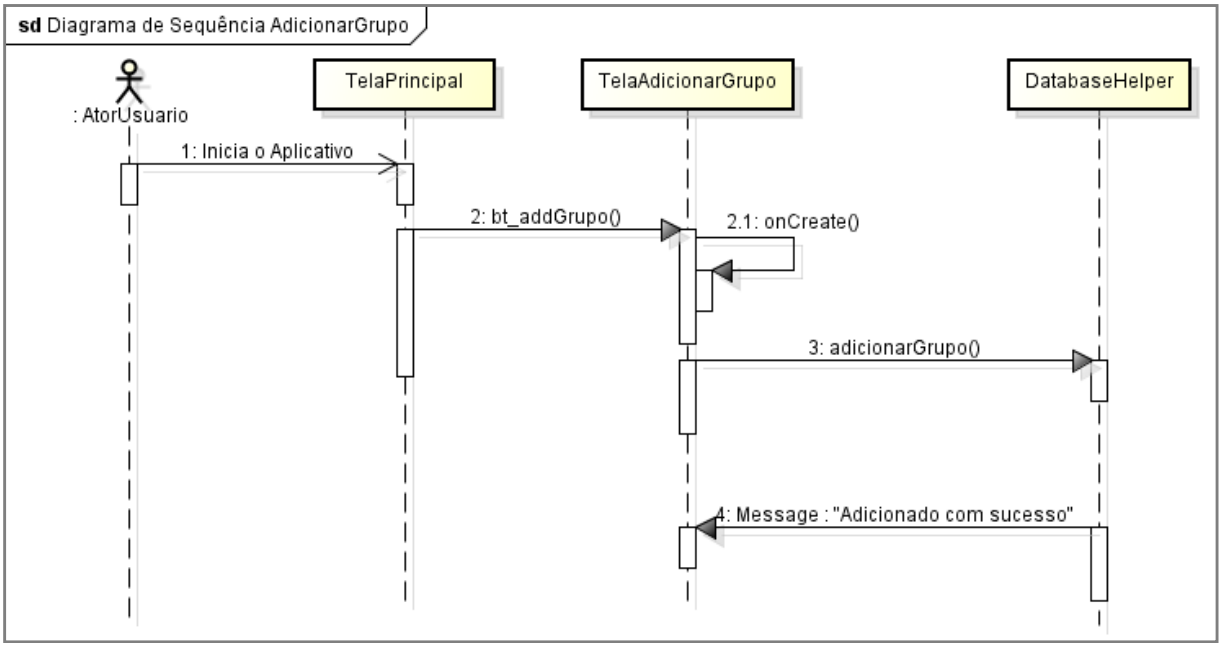

Figura 13 - Diagrama de sequência AdicionarGrupo.

A Figura 13 demonstra o diagrama de seqüência do caso de uso "*AdicionarGrupo*", apresentando os métodos de acesso as telas e de persistência na base de dados realizado pela classe `DatabaseHelper.java`.

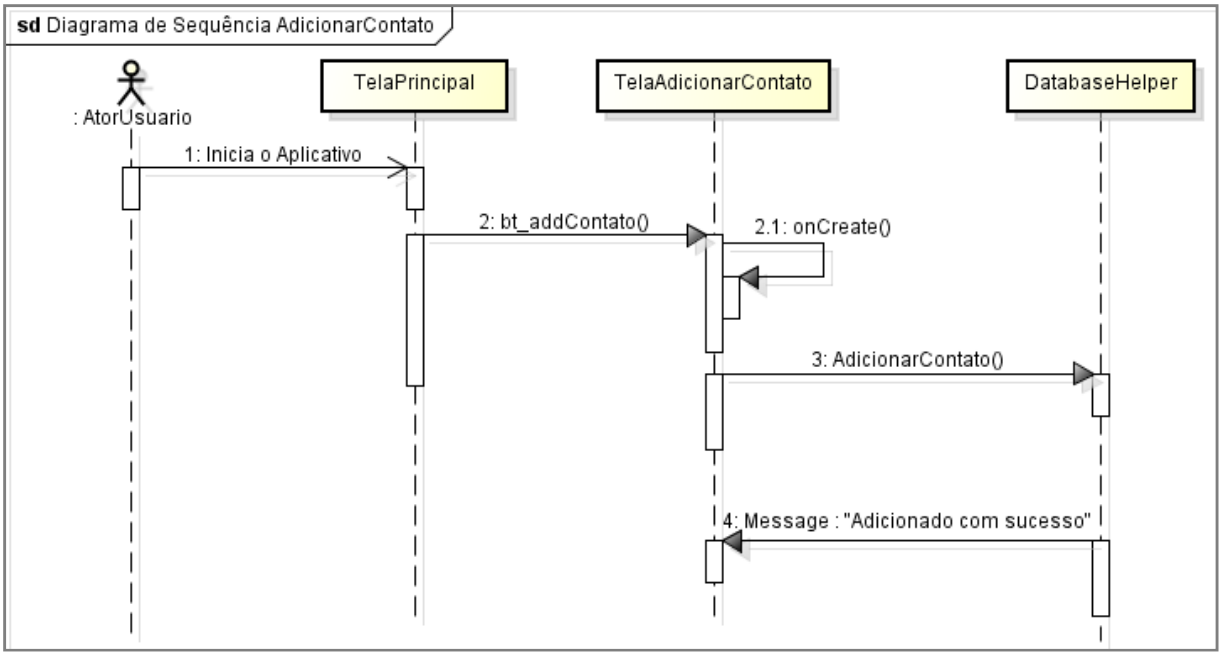

Figura 14 - Diagrama de seqüência AdicionarContato.

A Figura 14 apresenta o diagrama de seqüência do caso de uso "*AdicionarContato*", apresentando os métodos de acesso e de persistência na base de dados.

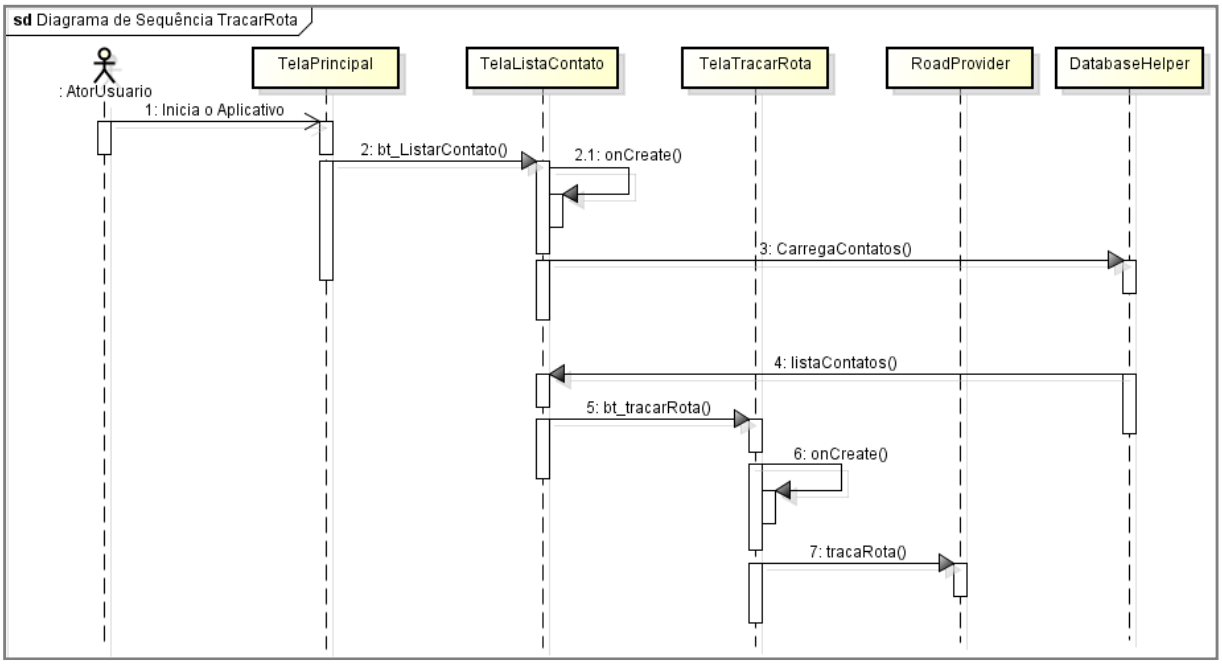

Figura 15 - Diagrama de seqüência TracarRota.

A Figura 15 trata do caso de uso “*TracarRota*”, que é responsável por carregar os pontos de origem e destino através do Google Maps e traçar a rota na tela do aplicativo.

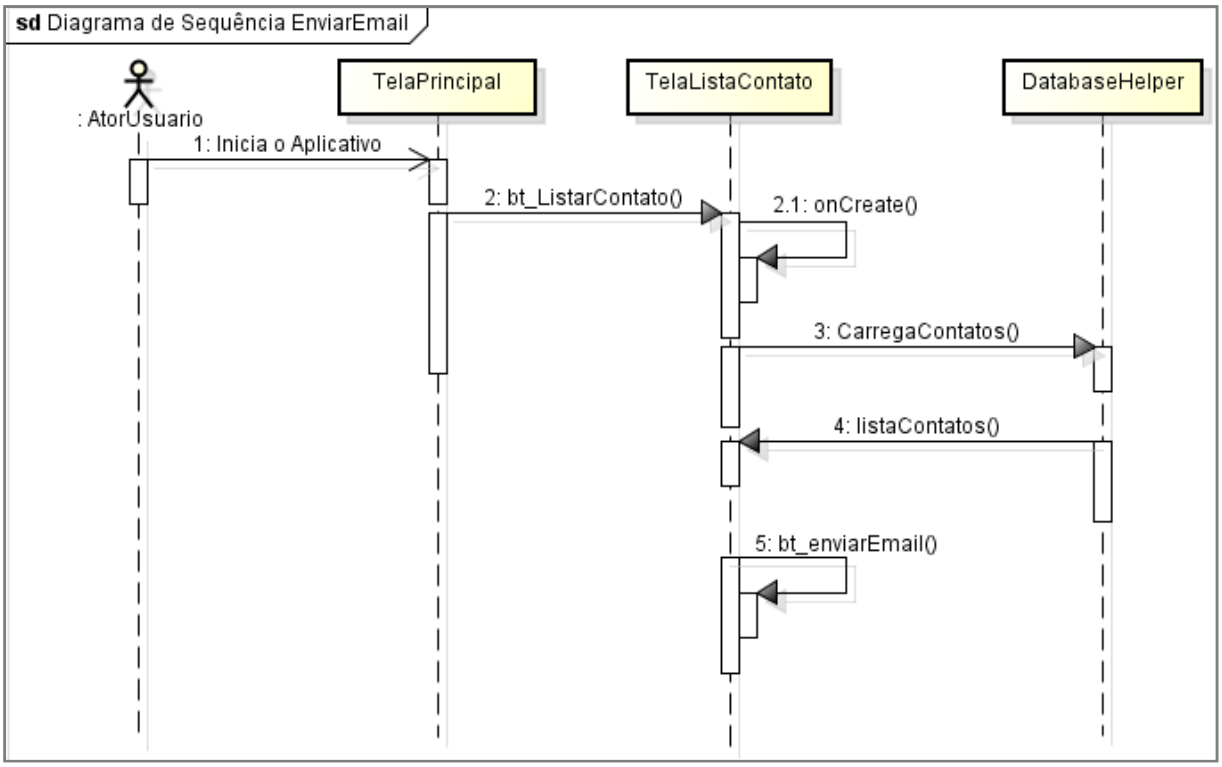

Figura 16 - Diagrama de seqüência EnviarEmail.

A Figura 16 mostra o diagrama de seqüência para o caso de uso “*EnviarEmail*”, apresentando os métodos para recuperação da lista de contatos até o método de enviar o e-mail para o contato selecionado.

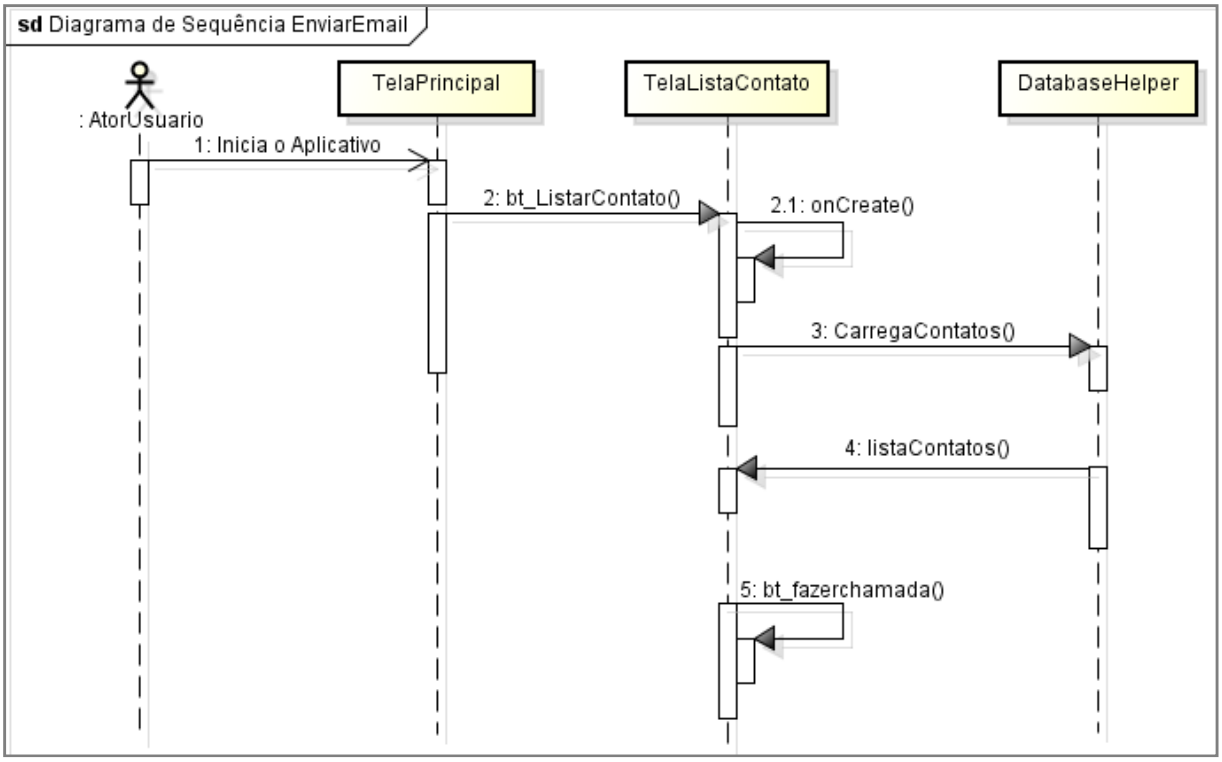

Figura 17 - Diagrama de seqüência FazerChamada.

A Figura 17 apresenta o diagrama de seqüência para o caso de uso "*FazerChamada*", demonstrando os métodos para a realização de uma ligação telefônica a partir de dados do banco.

3.6 MODELO DE ENTIDADE E RELACIONAMENTO

O modelo de entidade e relacionamento (MER) tem por finalidade descrever de maneira conceitual os dados a serem utilizados no sistema. O banco de dados desenvolvido para a aplicação conta com duas tabelas, são elas:

- GRUPO: Responsável por armazenar grupos para cadastro de contatos.
- CONTATO: Responsável por armazenar informações do contato para ligações, envio de email e o endereço para que possa ser traçada a rota no Google Maps.

A Figura 18 apresenta o MER do sistema iCADASTRO.

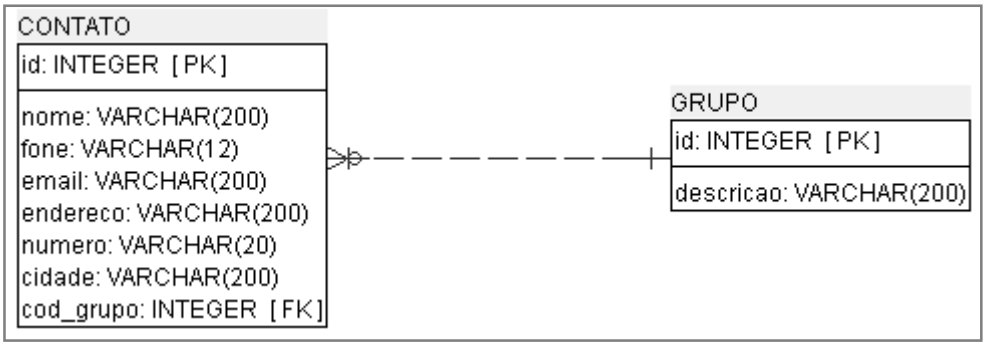

Figura 18 - MER sistema iCADASTRO.

Como observado na Figura 18, o sistema conta com duas tabelas e também com a associação entre a tabela GRUPO, que pode ser adicionado ao contato para organização destes dados.

4 RESULTADOS E DISCUSSÕES

O desenvolvimento do estudo experimental de agenda telefônica utilizou tecnologias do Sistema Operacional *Android*, Google Maps e SQLite. Assim, o primeiro passo é a obtenção da chave para utilização da API, na sequência, foi configurado o ambiente de desenvolvidas as classes, telas e bancos de dados para o sistema. Este capítulo apresenta detalhadamente a utilização destas técnicas para o desenvolvimento e os resultados obtidos com os testes efetuados.

4.1 OBTER CHAVE GOOGLE MAPS

Para poder utilizar o Google Maps API em uma aplicação Android, é necessário a obtenção da chave de registro. Ao executar uma aplicação no emulador do Android, o Eclipse compila seu projeto e assina o `.apk` com este registro de debug e somente ai envia para instalação. Desta forma, quando se exporta uma aplicação que utiliza o Google Maps, ela deve contar a assinatura do desenvolvedor, que serve como um identificador.

Para que o processo tenha êxito, o sistema operacional que esta sendo utilizado para desenvolver o sistema deve estar configurado com linguagem padrão “Inglês (Estados Unidos)”.

O processo de compilação pode ser feito através do *prompt* de comando, no caso do S.O do Windows. O comando para execuçao é `keytool -list -alias androiddebugkey -keystore debug.keystore -storepass android -keypass android`. Tal comando deve ser executando dentro do diretório raiz do Android. Observe o resultado deste comando e o local onde deve ser executado o comando na Figura 19.

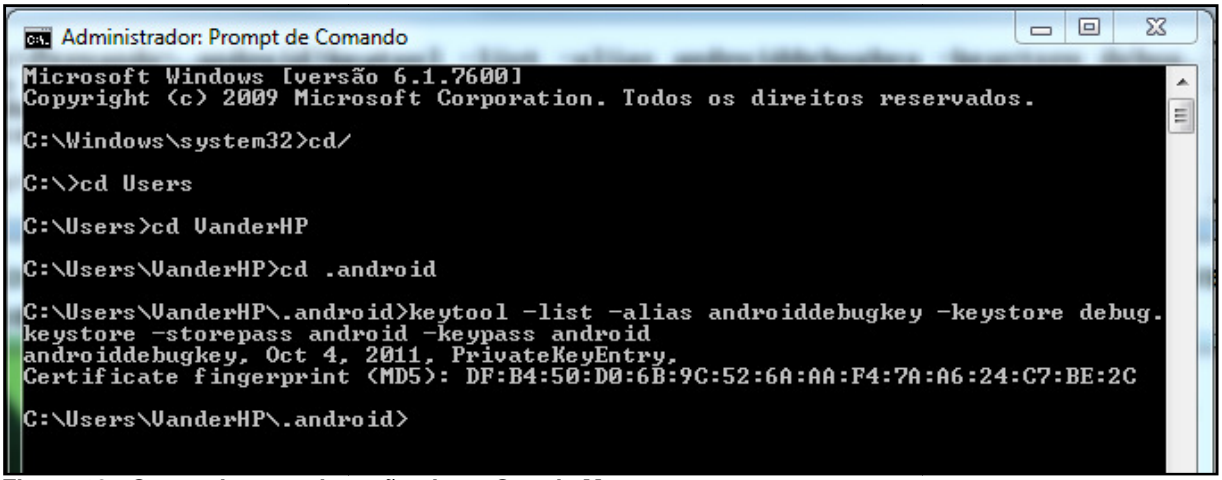

Figura 19 - Comando para obtenção chave Google Maps.

O código exibido na linha "*Certificate fingerprint* (MD5)" corresponde à chave gerada pelo Android SDK. Neste momento é necessário acessar o endereço http://code.google.com/intl/pt-BR/android/add-ons/google-apis/maps-api-signup.html para fornecer a chave obtida e conseguir uma assinatura para incluir nos projetos. A Figura 20 apresenta a página onde deve ser inserida esta chave.

Figura 20 - Solicitação de assinatura para utilizar o Google Maps API.

Para obter uma chave, é necessário que o usuário possua uma conta no Google. Adicionando a chave obtida pelo comando no campo "*My Certificate's MD5 fingerprint*" e concordando com o contrato a assinatura será fornecida pelo Google. Uma assinatura semelhante a `0cw2TCfABDxEAAgT5FksHtO3IpCSbVkQB0mwWTw` será exibida.

Com a chave obtida, pode-se dar inicio ao desenvolvimento da aplicação que fará a integração com o Google Maps API.

4.2 APLICATIVO iCADASTRO

No decorrer desta seção será apresentado o desenvolvimento do aplicativo Android integrado com Google Maps API.

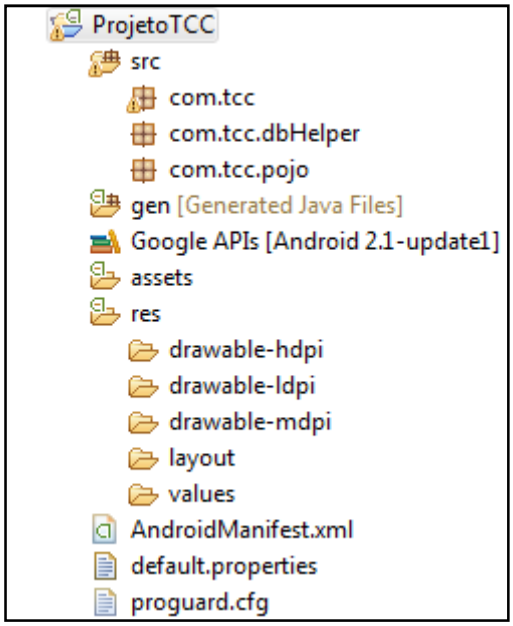

Figura 21 - Estrutura do projeto no Eclipse.

A Figura 21 apresenta a estrutura do projeto na plataforma Eclipse Helios, a pasta `src/` apresenta um conjunto de pacotes sendo eles:

- `com.tcc`: Este pacote contem as classes Java que tratam das telas do aplicativo e são responsáveis pelo processo de geolocalização no projeto.

- `com.tcc.dbHelper`: Este pacote contém uma classe Java responsável pela criação e manipulação da base de dados.
- `com.tcc.pojo`: Este pacote contém as classes Java que representam as tabelas do banco de dados.

A pasta `res/` contém:

- *`drawable`*: contém todos as imagens utilizadas no aplicativo.
- *`layout`:* contém os arquivos XML responsáveis pelo layout das telas.
- *`Values`:* Este pacote contém valores estáticos para serem carregados em um XML, como por exemplo um *String*. Neste projeto não foram utilizados valores estáticos.

Para acesso aos atributos do banco de dados de forma externa, foram criados *pojos*. Duas classes que contém os atributos de cada classe utilizada neste sistema, Grupo e Contato.

```
3  public class Contato {
4      private int id;
5      private String nome;
6      private String fone;
7      private String email;
8      private String endereco;
9      private String numero;
10     private String cidade;
11     private int cod_grupo;
```

Figura 22 – Classe Contato.java.

```
3  public class Grupo {
4      private int id;
5      private String descricao;
```

Figura 23 – Classe Grupo.java.

Foram atribuídos métodos *`getters`* (responsáveis pela busca de valores de determinado atributo de um objeto) e *`setters`* (responsáveis pela atribuição de valores para determinado atributo de um objeto) para cada atributo.

Estas classes são utilizadas para recuperação de contatos e grupos cadastrados agindo em conjunto com a classe *`DatabaseHelper.java`*.

A aplicação iCADASTRO conta com um banco de dados em SQLite. Esta base de dados começa a ser desenvolvida a partir da classe *DatabaseHelper.java*. Esta classe estende de `SQLiteOpenHelper`, que é uma classe nativa do Android para manipulação da base de dados. Ela contém métodos para a criação e a atualização do banco de dados. A declaração da classe, do nome da base de dados que utilizou-se neste sistema, e também a declaração das tabelas estão contidas na Figura 24.

```
13 public class DatabaseHelper extends SQLiteOpenHelper {
14     public static final String NOME_BD = "base_diplomacao4";
15
16     public static final String TB_GRUPO = "tb_grupo";
17     public static final String GRU_ID = "id";
18     public static final String DESCRICAO = "descricao";
19
20     public static final String TB_CONTATO = "tb_contato";
21     public static final String CON_ID = "id";
22     public static final String NOME = "nome";
23     public static final String FONE = "fone";
24     public static final String EMAIL = "email";
25     public static final String ENDERECO = "endereco";
26     public static final String NUMERO = "numero";
27     public static final String CIDADE = "cidade";
28     public static final String COD_GRUPO = "cod_grupo";
29
30     public DatabaseHelper(Context context){
31         super(context, NOME_BD, null, 20);
32     }
```

Figura 24 - Classe DatabaseHelper.java.

Utilizou-se campos estáticos do tipo `String` para manipular de forma mais fácil a criação e o acesso aos atributos nos métodos da classe. E também criou-se um construtor para receber o contexto da aplicação, quando for necessário instanciar um novo atributo do tipo *DatabaseHelper*, passa-se o contexto, que é o local onde esta sendo invocado. Este construtor carrega o contexto, a base de dados e a versão atual do sistema.

Na Figura 25 os métodos para criação e de atualização das tabelas na base de dados estão descritos.

```
39      @Override
40      public void onCreate(SQLiteDatabase db) {
41          db.execSQL("CREATE TABLE "+TB_GRUPO+" ("+GRU_ID+ " INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT, "+
42                  DESCRICAO+ " TEXT)");
43
44          db.execSQL("CREATE TABLE "+TB_CONTATO+" ("
45                  +CON_ID+" INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT, "+
46                  NOME+" TEXT, "+
47                  FONE+" TEXT, "+
48                  EMAIL+" TEXT,"+
49                  ENDERECO+" TEXT NOT NULL, "+
50                  NUMERO+" TEXT NOT NULL, "+
51                  CIDADE+" TEXT NOT NULL, "+
52                  COD_GRUPO+" INTEGER NOT NULL);" +
53                   "FOREIGN KEY ("+COD_GRUPO+") REFERENCES "+TB_GRUPO+" ("+GRU_ID+");");
54      }
55      @Override
56      public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
57          db.execSQL("DROP TABLE IF EXISTS "+ TB_GRUPO);
58          db.execSQL("DROP TABLE IF EXISTS "+ TB_CONTATO);
59      }
```

Figura 25 - Metodos onCreate e onUpgrade classe DatabaseHelper.java.

O método `onCreate(SQLiteDatabase)` faz a criação das tabelas no banco. O comando `execSQL` é nativo da classe `SQLiteDatabase`, permitindo a inserção de tabelas no banco de dados. Os dados utilizados foram do tipo “TEXT” e `INTEGER`. Houve um caso de `Foreign Key` que é a chave estrangeira do grupo, que vai à tabela contato como mostra a linha 55. O método `onUpgrade()` ele também faz utilização da classe `SQLiteDatabase` e trata da atualização das tabelas, podendo-se versioná-las ou excluí-las e criá-las novamente a cada quando necessário.

Dentro desta classe foram executadas todas as tarefas de inserção, exclusão, pesquisa e *update* dos dados. Na Figura 26 os métodos referentes aos grupos.

```
56      public void AdicionarGrupo(Grupo g){
57          SQLiteDatabase db = this.getWritableDatabase();
58          ContentValues cv = new ContentValues();
59          cv.put(DESCRICAO, g.getDescricao());
60          db.insert(TB_GRUPO, DESCRICAO, cv);
61          db.close();
62      }
63      public Cursor getAllGrupos(){
64           SQLiteDatabase db=this.getWritableDatabase();
65           Cursor cur=db.rawQuery("SELECT "+GRU_ID+" as _id, "+DESCRICAO+
66                   " from "+TB_GRUPO,new String [] {});
67           return cur;
68      }
69      public int UpdateGrupo(Grupo g){
70           SQLiteDatabase db=this.getWritableDatabase();
71           ContentValues cv=new ContentValues();
72           cv.put(DESCRICAO, g.getDescricao());
73           return db.update(TB_GRUPO, cv, GRU_ID+"=?",
74                   new String []{String.valueOf(g.getId())});
75      }
76
77      public void DeleteGrupo(Grupo g){
78           SQLiteDatabase db=this.getWritableDatabase();
79           db.delete(TB_GRUPO,GRU_ID+"=?",
80                   new String [] {String.valueOf(g.getId())});
81           db.close();
82      }
```

Figura 26 - Métodos para os grupos de contatos.

Dentro de cada método, foi criado um atributo do tipo `SQLiteDatabase`. O método de adição armazena um conjunto de valores e depois faz a inserção através do comando `db.insert()` que contem o nome da tabela, um atributo que pode ser nulo, e o conjunto de valores. O método responsável por atualizar o grupo, `UpdateGrupo` recebe por parâmetro o grupo a ser alterado, faz a *update* dos valores de acordo com o id do objeto passado por parâmetro. O método `DeleteGrupo` faz a exclusão do grupo passado por parâmetro. O método `getAllGrupos()` faz uma instrução `SELECT` na base de dados retornando os grupos cadastrados. Todos os métodos do tipo `getWritableDatabase()` que possibilitam a busca e escrita dos dados na base de dados.

Para a tabela contato foram métodos semelhantes. A Figura 27 traz os métodos relacionados a tabela contato.

```
93      public void AdicionarContato(Contato c){
94          SQLiteDatabase db = this.getWritableDatabase();
95          ContentValues cv = new ContentValues();
96          cv.put(NOME, c.getNome());
97          cv.put(FONE, c.getFone());
98          cv.put(EMAIL, c.getEmail());
99          cv.put(ENDERECO, c.getEndereco());
100         cv.put(NUMERO,c.getNumero());
101         cv.put(CIDADE, c.getCidade());
102         cv.put(COD_GRUPO, c.getCod_grupo());
103         db.insert(TB_CONTATO, NOME, cv);
104         db.close();
105     }
106     public Cursor getTodosContatos(){
107         SQLiteDatabase db = this.getReadableDatabase();
108         Cursor cur = db.rawQuery("SELECT "+CON_ID+" as _id, "+NOME+", "+FONE+
109                 ", "+EMAIL+", "+ENDERECO+", "+NUMERO+", "+CIDADE+", "+COD_GRUPO+
110                 " FROM "+TB_CONTATO,new String[]{});
111         return cur;
112     }
113     public int UpdateContato(Contato c){
114         SQLiteDatabase db=this.getWritableDatabase();
115         ContentValues cv=new ContentValues();
116         cv.put(NOME, c.getNome());
117         cv.put(FONE, c.getFone());
118         cv.put(EMAIL, c.getEmail());
119         cv.put(ENDERECO, c.getEndereco());
120         cv.put(NUMERO,c.getNumero());
121         cv.put(CIDADE, c.getCidade());
122         cv.put(COD_GRUPO, c.getCod_grupo());
123         return db.update(TB_CONTATO, cv, CON_ID+"=?", new String []{String.valueOf(c.getId())});
124     }
125     public void DeleteContato(Contato c){
126         SQLiteDatabase db=this.getWritableDatabase();
127         db.delete(TB_CONTATO,CON_ID+"=?", new String [] {String.valueOf(c.getId())});
128         db.close();
129      }
```

Figura 27 – Métodos para contatos.

Observa-se que os métodos de inserção, exclusão, recuperação e de *update* são muito semelhante aos utilizados na área de grupos de contatos.

Para o desenvolvimento das telas utilizou-se arquivos XML contendo os widgets (componentes de interface gráfica do usuário, que inclui janelas,botões, menus, ícones e etc). A tela principal do sistema é a primeira a ser executada ao iniciar o aplicativo. Ela traz em seu conteúdo as opções para o usuário fazer utilização do sistema. A tela principal é apresentada na Figura 28.

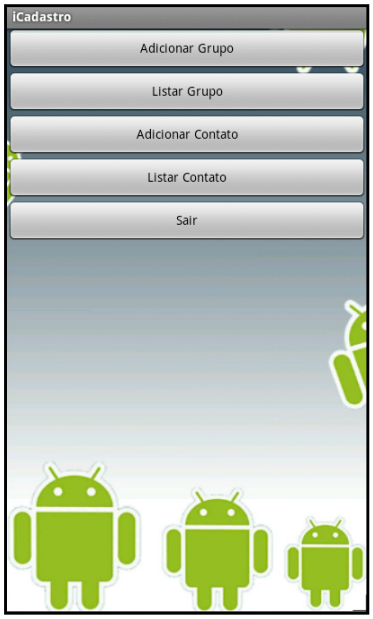

Figura 28 - Tela Principal.

A Figura 28 apresenta as opções de adição e busca de grupos e contatos. Todas as telas do sistema iCADASTRO, assim como qualquer outro sistema Android, são criadas à partir de arquivos XML que são armazenados na pasta "*layouts*" que já é criada juntamente com o projeto Android. A Figura 29 apresenta o esquema XML para apresentação da tela principal:

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      android:orientation="vertical"
4      android:layout_width="fill_parent"
5      android:background="@drawable/background"
6      android:layout_height="fill_parent"
7      >
8      <Button android:text="Adicionar Grupo"
9      android:id="@+id/btAddGrupo"
10     android:layout_width="fill_parent"
11     android:onClick="bt_addGrupo"
12     android:layout_height="wrap_content"></Button>
13
14     <Button android:text="Listar Grupo"
15     android:id="@+id/btListarGrupo"
16     android:layout_width="fill_parent"
17     android:onClick="bt_ListarGrupo"
18     android:layout_height="wrap_content"></Button>
19
20     <Button android:text="Adicionar Contato"
21     android:id="@+id/btAddContato"
22     android:layout_width="fill_parent"
23     android:onClick="bt_addContato"
24     android:layout_height="wrap_content"></Button>
25
26     <Button android:text="Listar Contato"
27     android:id="@+id/btListarContato"
28     android:layout_width="fill_parent"
29     android:onClick="bt_ListarContato"
30     android:layout_height="wrap_content"></Button>
31
32     <Button android:text="Sair"
33     android:id="@+id/btSair"
34     android:layout_width="fill_parent"
35     android:onClick="bt_Sair"
36     android:layout_height="wrap_content"></Button>
37 </LinearLayout>
```

Figura 29 - Arquivo XML para Criação tela Principal.

Algumas considerações sobre a Figura 29 que serão comuns entre todas as telas:

- Linha 1 <?xml version=”1.0” encoding=”utf-8”?>: Declaração do arquivo XML;
- Linha 2 <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android">: Definicao do tipo de container que suportará a Tela Principal, neste caso o LinearLayout, que adiciona os componentes um após o outro, de acordo com sua orientação que pode ser vertical ou horizontal.
- Linha 3 <android:orientation=”vertical”>: Define que os elementos serão dispostos um abaixo do outro, conforme foram criados no documento XML.
- Linha 4 <android:layout_width="fill_parent">: Define que o estrutura de layout da tela ocupará a largura total da tela do dispositivo móvel.
- Linha 5 <android:background="@drawable/background">: Atribui uma imagem de fundo à Tela Principal. O Android contém alguns diretórios *defaults*, como o *drawable*, que por definição armazenará as imagens do sistema. Neste caso, a imagem a ser utilizada denomina-se “*background*”.
- Linha 6 <android:layout_height="fill_parent">: Define que a estrutura de layout ocupara a altura total da tela do aplicativo.
- Linha 7 <Button android:text="Adicionar Grupo">: Insere um botão à tela principal.
- Linha 8 <android:id="@+id/btAddGrupo">: Adiciona um *id* ao botão criado.
- Linha 9 <android:layout_width="fill_parent">: Define que o botão ocupara a largura toda do layout onde esta inserido. Este atributo podia ser do tipo “*wrap_content*” que faz com que o botão tenha o tamanho ideal para que fique bem expresso seu texto. Sem ocupar a largura total da tela.

- Linha 10 <android:onClick="bt_addGrupo">: Define um evento *onClick* que será executado dentro da classe que carrega esta tela.
- Linha 11 <android:layout_height="wrap_content">: Define que o botão ocupara apenas a largura necessária para sua existência, uma largura padrão sem ocupar a tela toda.

Definiu-se estes detalhes acima, pois são comuns a várias telas do aplicativo. Estas definições foram bastante empregadas no iCADASTRO.

Este arquivo XML foi carregado pela classe `Principal.java`. Esta classe estende de `Activity`. Declarou-se da seguinte forma `public class Principal extends Activity`. Esta extensão corresponde ao aplicativo como uma atividade que pode ser chamada a qualquer momento, e deve ser declarada no arquivo `AndroidManifest.xml`, que é um arquivo de mapeamento e responsável por carregar as atividades do sistema.

Uma classe carrega um XML no momento de sua criação, através de um método chamado `onCreate()`. A Figura 30 apresenta este método na Tela Principal:

```
13    @Override
14    public void onCreate(Bundle savedInstanceState) {
15        super.onCreate(savedInstanceState);
16        setContentView(R.layout.main);
17    }
```

Figura 30 - Método onCreate() Tela Principal.

Onde se observa que é carregado o arquivo `main` que contem os atributos anteriormente citados.

A Tela de cadastro de grupo trata da inserção de grupos na base de dados. Os grupos foram criados para separar os contatos conforme divisão firmada pelo usuário. Estes grupos serão carregados na tela de cadastro de contatos. A tela de cadastro de grupos esta na Figura 31.

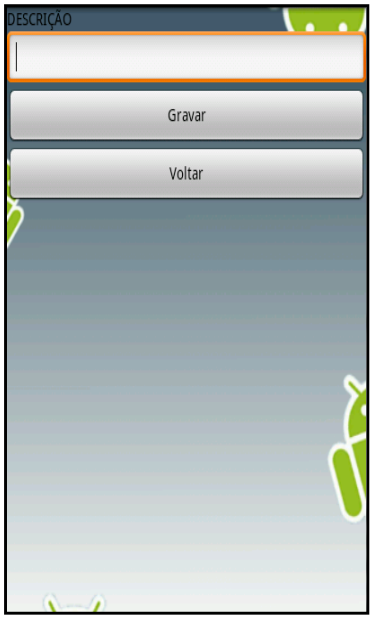

Figura 31 - Tela Cadastrar Grupo.

A tela de cadastro de contatos faz a inserção de contatos na base de dados, validando campos em branco. Os grupos são adicionados aos contatos através da *widget spinner* (*combobox*) que traz uma lista de Strings com a descrição dos grupos para seleção no momento do cadastro do contato.

A Figura 32 apresenta a tela de cadastro de contatos:

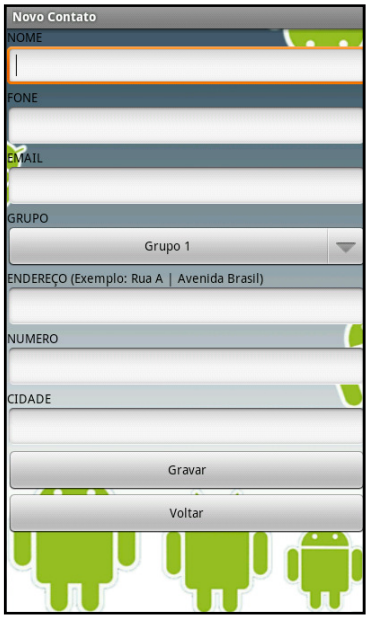

Figura 32 - Tela Cadastro de Contatos.

Os campos de endereçamento foram assim divididos para que exijam o seu preenchimento, para traçar uma rota é necessário o endereço completo, contendo a rua, o numero e a cidade de destino.

As Figuras 33 e 34 apresentam as telas de listagem de grupos e edição de grupos:

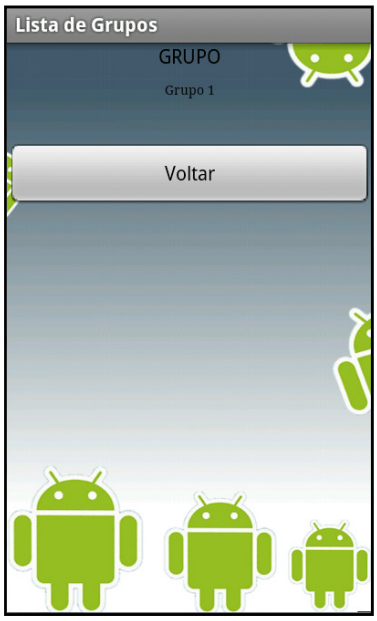

Figura 33 - Tela Lista de Grupos.

Figura 34 - *AlertDialog* Editar Grupo.

Como apresenta a Figura 33, a tela apresenta uma lista de grupos já adicionados permitindo que, quando selecionados, abra um `AlertDialog` (alerta exibido na tela do aplicativo demonstrando uma mensagem ou esperando uma ação do usuário) de edição, contendo opções de atualização e exclusão do grupo como mostra a Figura 34.

Neste `Dialog` selecionando a opção “Gravar” a classe que carrega o XML e estende de `Activity` invoca os métodos `updateGrupo()` da classe `DatabaseHelper.java` e a opção “Delete” o método `DeleteGrupo()`, citados anteriormente.

As Figuras 35 e 36 apresentam as telas de listagem de contatos e opções:

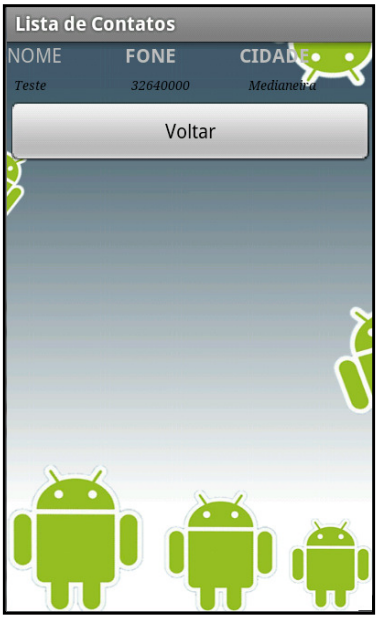

Figura 35 - Tela Lista de Contatos.

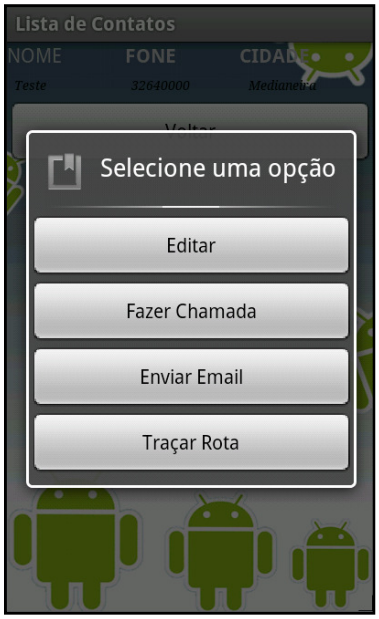

Figura 36 - AlertDialog opções Contato.

Como na tela de listagem de grupos, a listagem de contatos funciona da mesma forma. Quando um contato for selecionado, abre o `AlertDialog` de opções contendo diversas tarefas que podem ser efetuadas.

Quando o selecionada a opção "*Editar*", abre um `AlertDialog` de edição semelhante ao de grupos, porem com todos os dados do contato selecionado.

Ao selecionar a opção "Fazer Chamada" o sistema busca o numero do contato selecionado através do método apresentado na Figura 37:

```
111    public void bt_fazerchamada(View v){
112        Cursor c = dbhHelper.recuperaEmailChamada(id1);
113        startManagingCursor(c);
114        fone = c.getString(c.getColumnIndex(DatabaseHelper.FONE));
115        Intent dial = new Intent(android.content.Intent.ACTION_CALL,Uri.parse("tel: "+fone.toString()));
116        ContatoListar.this.startActivity(Intent.createChooser(dial,"Chamando... "));
117    }
```

Figura 37 - Método para Fazer Chamada.

Este método instancia um `Cursor`, que recupera o valor da coluna da tabela na base de dados. Neste caso, a String "fone" na linha 114, recebe o valor contido no *Cursor* que "aponta" para a coluna "FONE" de acordo com o *id* passado por parâmetro. O método `recuperaEmailChamada()` é um método em comum para as ações de enviar email e fazer chamada. Este método está apresentado na Figura 38.

```
139     public Cursor recuperaEmailChamada(int id){
140         SQLiteDatabase db = this.getReadableDatabase();
141         String[] params = new String[]{String.valueOf(id)};
142         Cursor c = db.rawQuery("SELECT * FROM "+TB_CONTATO+" WHERE "+CON_ID+"=?",params);
143         c.moveToFirst();
144         return c;
145     }
```

Figura 38 - Método para recuperar email para envio e número para chamada.

Na Figura 38 pode-se observar que a instância de `SQLiteDatabase` é do tipo `getReadableDatabase()`, ou seja, somente leitura. O `Cursor` recebe o `SELECT` (linha 160) da tabela contato onde o *id* for igual ao recebido por parâmetro e a ação do `Cursor` na linha 161 `c.moveToFirst()` traz o valor encontrado para a primeira posição do `Cursor`. Para haver possibilidade de realizar uma chamada através de uma aplicação que não é nativa do dispositivo móvel, é necessário incluir a linha de permissão `<uses-permission android:name=" android. permission. CALL_PHONE"/>` no `AndroidManifest.xml`.

Selecionando a opção “Enviar Email”, o usuário pode enviar um email direto de sua conta pré cadastrada em seu *smartphone* Android para o contato cadastrado. Utilizou-se o mesmo método `recuperaEmailChamada`() pois a busca era baseada no mesmo *id*. O método de envio de email segue na Figura 39:

```
119     public void bt_enviarEmail(View v){
120         Cursor c = dbhHelper.recuperaEmailChamada(id1);
121         startManagingCursor(c);
122         email = c.getString(c.getColumnIndex(DatabaseHelper.EMAIL));
123         Intent enviaremail = new Intent(android.content.Intent.ACTION_SEND);
124
125         enviaremail.setType("plain/text");
126         enviaremail.putExtra(android.content.Intent.EXTRA_EMAIL, new String[]{email.toString()});
127         enviaremail.putExtra(android.content.Intent.EXTRA_SUBJECT, "");
128         enviaremail.putExtra(android.content.Intent.EXTRA_TEXT, "");
129         ContatoListar.this.startActivity(Intent.createChooser(enviaremail, "Enviar Email"));
130     }
```

Figura 39 - Método Enviar Email.

Selecionando a opção “Traçar Rota” o usuário pode traçar uma rota no Google Maps a partir da sua localização atual. Foi instanciada uma nova classe chamada `TraçarRota.java` que estende, por sua vez, de `MapActivity`, para que possa trabalhar com alguns recursos geográficos. O método para instanciar esta nova tela segue na Figura 40.

```
132     public void bt_tracarRota(View v){
133         Cursor c =dbhHelper.recuperaEndereco(id1);
134         TracarRota tc = new TracarRota();
135         String endereco_ = c.getString(c.getColumnIndex(DatabaseHelper.ENDERECO));
136         String cidade_ = c.getString(c.getColumnIndex(DatabaseHelper.CIDADE));
137         String numero_ = c.getString(c.getColumnIndex(DatabaseHelper.NUMERO));
138         tc.passaDadosContato(endereco_, cidade_,numero_);
139
140         startActivity(new Intent(this, TracarRota.class));
141     }
```

Figura 40 - Método chamada da tela TracarRota.java.

Na Figura 40 instanciou-se uma um objeto do tipo "`TracarRota` onde através dele, foi acessado o método `passaDadosContato()`, que contém dados do tipo `String` que servirão de parâmetros para a rota no mapa. Logo após enviar os dados, o iCADASTRO instancia uma nova `Intent`, que nada mais é que uma intenção de começar uma nova atividade.

O Google Maps trabalha sobre coordenadas geográficas, então foi necessário fazer a tradução da localização do contato cadastrado, que na base de dados está como String, em coordenadas. Para isto, utilizou-se um processo chamado *Geocodificação*, o qual trata os endereços de em formato String para coordenadas. A Figura 41 apresenta o método utilizado para este processo.

```
115     public void buscarCoordenadasEndereco(String destino) throws IOException {
116         Geocoder geoCoder = new Geocoder(this);
117         List<Address> adresses = null;
118         adresses = geoCoder.getFromLocationName(destino, 1);
119         toLat = adresses.get(0).getLatitude();
120         toLon = adresses.get(0).getLongitude();
121     }
```

Figura 41 - Processo de Geocodificação.

O método apresentado na Figura 41 recebe o endereço de destino em formato String e através da classe `Geocoder` que armazena em uma lista de endereços, através do método `getFromLocationName()`, as coordenadas geográficas de forma precisa e as armazena em duas variáveis estáticas `toLat` e `toLon` nas linhas 127 e 128.

Para recuperar a coordenada atual do dispositivo móvel, utilizou-se recursos no momento de criação da tela, ou seja, no método `onCreate()`. O método para essa verificação segue na Figura 42.

```
71          LocationManager locManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
72          locManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 1000L,
73                  500.0f, locationListener);
74          Location location = locManager
75                  .getLastKnownLocation(LocationManager.GPS_PROVIDER);
76          if (location != null) {
77              fromLat = location.getLatitude();
78              fromLon = location.getLongitude();
79              Toast.makeText(this, "MINHA LOCALIZACAO " + fromLat + fromLon,
80                      Toast.LENGTH_SHORT).show();
81          }
```

Figura 42 - Verifição posicionamento atual dispositivo móvel.

Este método listado na Figura 42 trata da posição atual, a classe `LocationManager` pega o serviço disponível. O método `requestLocationUpdates()` trata de como e em quanto tempo será atualizado as coordenadas. Neste caso, utiliza o `GPS_PROVIDER`, que recupera a posição atual via satélite, atribui a atualização a cada 1000 milissegundos ou a cada 500 metros do último local, e chama o método `locationlistener` que será descrito em breve. Foi criada uma instancia de `Location`, que armazena as últimas coordenadas do dispositivo móvel utilizadas pelo provedor através do método `getLastKnowLocation()`. Foi realizado um teste para apresentar a coordenada atual do dispositivo móvel.

```
 99    private final LocationListener locationListener = new LocationListener() {
100        public void onLocationChanged(Location location) {
101            updateWithNewLocation(location);
102        }
103        public void onProviderDisabled(String provider) {
104            updateWithNewLocation(null);
105        }
106        public void onProviderEnabled(String provider) {
107        }
108        public void onStatusChanged(String provider, int status, Bundle extras) {
109        }
110    };
```

Figura 43 - Método *LocationListener*.

A Figura 43 descreve o método que instancia a classe `LocationListener`, que é responsável por tratar da posição atual do dispositivo móvel. Esta classe possui quatro métodos implementados. O `onLocationChanged()` detecta se o dispositivo mudou de localização, o método `onProviderDisabled()` trata se o provedor de localização, que neste caso é o sinal via satélite está desativado, o método `onProviderEnabled()` trata da detecção de sinal via satélite e o método `onStatusChanged()` pode tratar caso o provedor estiver desligado, ligado ou outra

forma, exibindo mensagens na tela. Se o dispositivo mudar de localização, o método `upradeWithNewLocation()` trata da nova posição nova do dispositivo. A Figura 44 apresenta este método:

```
84    private void updateWithNewLocation(Location location) {
85        if (location != null) {
86            fromLat = location.getLatitude();
87            fromLon = location.getLongitude();
88        } else {
89            Toast.makeText(this,"LOCALIZAÇÃO NÃO IDENTIFICADA ",
90                                        Toast.LENGTH_SHORT).show();
91        }
92        tracaRota();
93    }
```

Figura 44 - Método de atualização do local do dispositivo.

Este método é invocado somente na classe `LocationListener`, onde lhe é passada a nova localização quando há mudança de local. Este faz a atualização das variáveis responsáveis por armazenar as coordenadas locais e então, com a geocodificacao do endereço do destino e possuindo as coordenadas do endereço de origem (posição atual do dispositivo móvel) chama-se o método `tracaRota()` que irá desenhar a rota no Google Maps.

```
128    public void tracaRota() {
129        new Thread() {
130            @Override
131            public void run() {
132                String url = RoadProvider
133                        .getUrl(fromLat, fromLon, toLat, toLon);
134                InputStream is = getConnection(url);
135                mRoad = RoadProvider.getRoute(is);
136                mHandler.sendEmptyMessage(0);
137            }
138        }.start();
139    }
```

Figura 45 - Método tracarRota().

Este método apresentado na Figura 45 instancia uma nova `Thread()` que contém um método `run()`. Este método chama a classe `RoadProvider()`. A String "*url*" criada recebe o retorno da montagem do endereço para solicitar as informações. O pacote `com.google.maps` contém um serviço de localização de rota através de KML, que é um arquivo desenvolvido especialmente para exibir

dados geográficos em um navegador. Para obter este KML são necessárias as coordenadas de origem e destino. O método que monta a solicitação segue na Figura 46.

```
34      public static String getUrl(double fromLat, double fromLon, double toLat,
35              double toLon) {
36      StringBuffer urlString = new StringBuffer();
37      urlString.append("http://maps.google.com/maps?f=d&hl=en");
38      urlString.append("&saddr=");// from
39      urlString.append(Double.toString(fromLat));
40      urlString.append(",");
41      urlString.append(Double.toString(fromLon));
42      urlString.append("&daddr=");// to
43      urlString.append(Double.toString(toLat));
44      urlString.append(",");
45      urlString.append(Double.toString(toLon));
46      urlString.append("&ie=UTF8&0&om=0&output=kml");
47      return urlString.toString();
48  }
```

Figura 46 - Passagem de coordenadas para o Google Maps.

Após a invocação do método da Figura 46, o sistema faz a montagem da url para requisição via `URLConnection` para obter informações do arquivo KML. A classe que abre a conexão segue na Figura 47.

```
153     private InputStream getConnection(String url) {
154         InputStream is = null;
155         try {
156             URLConnection conn = new URL(url).openConnection();
157             is = conn.getInputStream();
158         } catch (MalformedURLException e) {
159             e.printStackTrace();
160         } catch (IOException e) {
161             e.printStackTrace();
162         }
163         return is;
164     }
```

Figura 47 - Classe que faz a conexão com o serviço do Google Maps.

Após o recebimento das informações, o método `getRoute()` da classe `RoadProvider.java` é requisitado passando esta conexão. Como mostra a linha 135 da Figura 45. Este método está descrito na Figura 48.

```
18 public class RoadProvider {
19     public static Road getRoute(InputStream is) {
20         KMLHandler handler = new KMLHandler();
21         try {
22                 SAXParser parser = SAXParserFactory.newInstance().newSAXParser();
23                 parser.parse(is, handler);
24         } catch (ParserConfigurationException e) {
25                 e.printStackTrace();
26         } catch (SAXException e) {
27                 e.printStackTrace();
28         } catch (IOException e) {
29                 e.printStackTrace();
30         }
31         return handler.mRoad;
32 }
```

Figura 48 - Método getRoute(). Fonte: autoria própria.

Este método instancia a classe `KMLHandler`, que esta inclusa dentro da classe `RoadProvider.java`, e é responsável abrir o arquivo KML e armazenar os pontos necessários para traçar a rota. Armazena os valores em duas classes especificas como demonstradas nas Figuras 49 e 50.

```
3 public class Point {
4     String mName;
5     String mDescription;
6     String mIconUrl;
7     double mLatitude;
8     double mLongitude;
9 }
```

Figura 49 - Classe que armazena os pontos da rota recebidos do arquivo KML.

```
6 public class Road {
7     public String mName;
8     public String mDescription;
9     public int mColor;
10    public int mWidth;
11    public double[][] mRoute = new double[][] {};
12    public Point[] mPoints = new Point[] {};
13 }
```

Figura 50 - Classe que armazena dados do caminho a ser traçado.

A classe Java `Point.java` armazena o ponto e a classe `Road.java` armazena a coleção destes pontos para depois traçar a rota no mapa.

A classe `KMLHandler` esta demonstrada na Figura 51:

```
48 class KMLHandler extends DefaultHandler {
49     Road mRoad;
50     boolean isPlacemark;
51     boolean isRoute;
52     boolean isItemIcon;
53     private Stack mCurrentElement = new Stack();
54     private String mString;
55 
56     public KMLHandler() {
57         mRoad = new Road();
58     }
59 
60 public void startElement(String uri, String localName, String name,
61         Attributes attributes) throws SAXException {
62     mCurrentElement.push(localName);
63     if (localName.equalsIgnoreCase("Placemark")) {
64         isPlacemark = true;
65         mRoad.mPoints = addPoint(mRoad.mPoints);
66     } else if (localName.equalsIgnoreCase("ItemIcon")) {
67         if (isPlacemark)
68             isItemIcon = true;
69     }
70     mString = new String();
71 }
72 public void characters(char[] ch, int start, int length)
73         throws SAXException {
74     String chars = new String(ch, start, length).trim();
75     mString = mString.concat(chars);
76 }
77 public void endElement(String uri, String localName, String name)
78         throws SAXException {
79 if (mString.length() > 0) {
80     if (localName.equalsIgnoreCase("name")) {
```

Figura 51 - Classe KMLHandler (a)

```
81          if (isPlacemark) {
82              isRoute = mString.equalsIgnoreCase("Route");
83              if (!isRoute) {
84                  mRoad.mPoints[mRoad.mPoints.length - 1].mName = mString;
85              }
86          } else {
87              mRoad.mName = mString;
88          }
89      } else if (localName.equalsIgnoreCase("color") && !isPlacemark) {
90          mRoad.mColor = Integer.parseInt(mString, 16);
91      } else if (localName.equalsIgnoreCase("width") && !isPlacemark) {
92          mRoad.mWidth = Integer.parseInt(mString);
93      } else if (localName.equalsIgnoreCase("description")) {
94          if (isPlacemark) {
95              String description = cleanup(mString);
96              if (!isRoute)
97                  mRoad.mPoints[mRoad.mPoints.length - 1].mDescription = description;
98              else
99                  mRoad.mDescription = description;
100         }
101     } else if (localName.equalsIgnoreCase("href")) {
102         if (isItemIcon) {
103             mRoad.mPoints[mRoad.mPoints.length - 1].mIconUrl = mString;
104         }
105     } else if (localName.equalsIgnoreCase("coordinates")) {
106         if (isPlacemark) {
107             if (!isRoute) {
108                 String[] xyParsed = split(mString, ",");
109                 double lon = Double.parseDouble(xyParsed[0]);
110                 double lat = Double.parseDouble(xyParsed[1]);
111                 mRoad.mPoints[mRoad.mPoints.length - 1].mLatitude = lat;
112                 mRoad.mPoints[mRoad.mPoints.length - 1].mLongitude = lon;
113             } else {
```

Figura 52 - Classe KMLHandler (b)

```
114                 String[] coodrinatesParsed = split(mString, " ");
115                 mRoad.mRoute = new double[coodrinatesParsed.length][2];
116                 for (int i = 0; i < coodrinatesParsed.length; i++) {
117                     String[] xyParsed = split(coodrinatesParsed[i], ",");
118                     for (int j = 0; j < 2 && j < xyParsed.length; j++)
119                         mRoad.mRoute[i][j] = Double
120                                 .parseDouble(xyParsed[j]);
121                 }
122             }
123         }
124     }
125 }
126     mCurrentElement.pop();
127     if (localName.equalsIgnoreCase("Placemark")) {
128         isPlacemark = false;
129         if (isRoute)
130             isRoute = false;
131     } else if (localName.equalsIgnoreCase("ItemIcon")) {
132         if (isItemIcon)
133             isItemIcon = false;
134     }
135 }
136 private String cleanup(String value) {
137     String remove = "<br/>";
138     int index = value.indexOf(remove);
139     if (index != -1)
140     value = value.substring(0, index);
141     remove = " ";
142     index = value.indexOf(remove);
143     int len = remove.length();
144     while (index != -1) {
145         value = value.substring(0, index).concat(
146                 value.substring(index + len, value.length()));
```

Figura 53 - Classe KMLHandler (c)

```
147             index = value.indexOf(remove);
148         }
149         return value;
150         }
151
152     public Point[] addPoint(Point[] points) {
153         Point[] result = new Point[points.length + 1];
154         for (int i = 0; i < points.length; i++)
155             result[i] = points[i];
156         result[points.length] = new Point();
157         return result;
158     }
159     private static String[] split(String strString, String strDelimiter) {
160         String[] strArray;
161         int iOccurrences = 0;
162         int iIndexOfInnerString = 0;
163         int iIndexOfDelimiter = 0;
164         int iCounter = 0;
165         if (strString == null) {
166             throw new IllegalArgumentException("Não pode ser nulo.");
167         }
168         if (strDelimiter.length() <= 0 || strDelimiter == null) {
169             throw new IllegalArgumentException("Delimitador não pode ser nulo.");
170         }
171         if (strString.startsWith(strDelimiter)) {
172             strString = strString.substring(strDelimiter.length());
173         }
174         if (!strString.endsWith(strDelimiter)) {
175             strString += strDelimiter;
176         }
177         while ((iIndexOfDelimiter = strString.indexOf(strDelimiter,
178                 iIndexOfInnerString)) != -1) {
179             iOccurrences += 1;
180             iIndexOfInnerString = iIndexOfDelimiter + strDelimiter.length();
181         }
182         strArray = new String[iOccurrences];
183         iIndexOfInnerString = 0;
184         iIndexOfDelimiter = 0;
185         while ((iIndexOfDelimiter = strString.indexOf(strDelimiter,
186                 iIndexOfInnerString)) != -1) {
187             strArray[iCounter] = strString.substring(iIndexOfInnerString,
188                     iIndexOfDelimiter);
189             iIndexOfInnerString = iIndexOfDelimiter + strDelimiter.length();
190             iCounter += 1;
191         }
192         return strArray;
193         }
194 }
```

Figura 54 - Classe KMLHandler (d)

Esta classe faz a leitura do arquivo KML que contém as informações dos locais e pontos para traçar a rota no mapa. À medida que a classe vai sendo executada, estes pontos vão sendo adicionadas na coleção do tipo *Point*. Este arquivo KML calcula a distância entre os dois pontos de partida e chegada, e também traz uma estimativa de tempo para percorrer esta distância.

Após a execução do aplicativo pode-se comprovar que o serviço de localização atual do dispositivo móvel é atualizada via GPS, demorando alguns segundos e o processo de geocodificação do endereço de destino é realizado e a rota é traçada em questão de minutos. Comprovando-se assim que é possível a integração entre as tecnologias propostas neste trabalho de diplomação.

5. CONSIDERAÇÕES FINAIS

Ao decorrer deste capitulo serão abordadas as considerações finais deste trabalho de diplomação.

5.1 CONCLUSÃO

O desenvolvimento de aplicações para dispositivos móveis através do Sistema Operacional Android esta cada dia mais presente. Com os recursos encontrados através de pesquisas e a possibilidade de desenvolvimento com ferramentas *Open Source* facilita e estimula os analistas e programadores a sempre se atualizar com as tendências do mercado móvel.

O Android SDK traz um amplo conjunto de recursos, fazendo assim com que o desenvolvimento de aplicações seja prático e eficaz. Além da possibilidade de publicação dos aplicativos na Internet através do maior site de vendas desenvolvido pelo próprio Google, o Android Market.

O Google Maps, que é amplamente utilizado em versões desenvolvidas para web, também torna-se acessível ao desenvolvedor de aplicativos Android, trazendo a possibilidade de obtenção de chave de licença gratuita. Possibilita o desenvolvimento de aplicações cada vez mais ricas no ambiente móvel.

Na base de dados SQLite a persistência de dados é de forma clara e objetiva. As criações das tabelas e as consultas SQL de forma extremamente fácil e compacta.

A integração entre estas tecnologias trouxe resultados positivos no sentido de que é possível sim criar cada vez mais aplicações versáteis. A resposta da base de dados para busca de informações, o Google Maps API atuando como um provedor de localização e a diversidade de recursos disponibilizados pelo Android possibilitaram a execução da rota dentro do aplicativo.

Por fim, a mobilidade, que é o que o usuário busca a cada dia mais, faz com que aumente o mercado permitindo assim desenvolver aplicações cada vez mais

completas.

5.2 TRABALHOS FUTUROS

Com a evolução gradativa das tecnologias é possível vislumbrar a continuação do estudo sobre esta integração entre a tecnologia móvel e o processo de georeferenciamento. Assim, continuando a realidade do amplo mercado de aplicativos para dispositivos móveis torna-se cada vez mais necessário o estudo aprofundado para obter ao máximo no momento das integrações entre tecnologias.

REFERÊNCIAS BIBLIOGRÁFICAS

ANDROID BRASIL. **O que é Android?.** Disponível em <http://www.androidbrasil.com/noticias/android/166-o-que-e-android>. Acesso em 19/09/2011.

ANDROID DEVELOPERS. **What is Android?**. Disponível em <http://developer.android.com/guide/basics/what-is-android.html> Acesso em 19/09/2011.

CELESTINO, André L.. **A evolução dos celulares. 2010**. Disponível em <http://www.onucleo.com/index.php/geral-tech/257-a-evolucao-dos-celulares>. Acesso em 20/09/2011.

COMSCORE. **Android Captures #2 Ranking Among Smartphone Platforms in EU5**. Disponível em <http://www.comscore.com/Press_Events/Press_Releases/2011/9/Android_Captures_number_2_Ranking_Among_Smartphone_Platforms_in_EU5>. Acesso em 22/09/2011.

CONSTANTINI, Ulisses. **Aplicação Móvel de Auto-localização Baseada na Ferramenta Google Maps**. Disponível em <http://www.jornaljava.com/wp-content/uploads/2011/02/artigo-ulisses-2009.pdf>. Acesso em 22/09/2011.

DEVMEDIA. **SQLite no Android Trabalhando com persistência de dados no Android**. Disponível em <http://www.devmedia.com.br/post-19201-SQLite-no-Android.html>. Acesso em 25/09/2011.

GILSOGAMO, Ana P. **O acesso a internet pelo celular deverá superar o acesso pelo computador**. Disponível em <http://www.mobilepedia.com.br/noticias/o-acesso-a-internet-pelo-celular-devera-superar-o-acesso-pelo-computador>. Acesso em 19/09/2011.

GLOBO. **Acesso à Internet via celular 3G cresce 90% no semestre. 14 de Julho de 2011**. Disponível em <http://g1.globo.com/tecnologia/noticia/2011/07/acesso-internet-3g-cresce-90-no-semestre.html>. Acesso em 19/09/2011.

GONÇALVES, Eduardo Corrêa. **SQLite, Muito Prazer!**. Disponível em <http://www.devmedia.com.br/post-7100-SQLite-Muito-Prazer.html>. Acesso em 26/09/2011.

GOOGLE. Google Code. **Projetos do Google para o Android**. Disponível em: <http://code.google.com/intl/pt-BR/android/>. Acesso em 21/09/2011.

IG TECNOLOGIA. **Conheça todas as versões do Android já lançadas. 2011**. Disponível em <http://tecnologia.ig.com.br/noticia/2011/04/18/conheca+todas+as+versoes+do+android+ja+lancadas+10403329.html>. Acesso em 22/09/2011.

JORDÃO, Fábio. **História: a evolução do celular. 2009**. Disponível em <http://www.tecmundo.com.br/2140-historia-a-evolucao-do-celular.htm>. Acesso em 21/09/2011

MEEHAN, Daniel. **By the end of 2011 Android Market will have as many users as Apple's App Store**. 15/09/2011. Disponível em <http://www.research2guidance.com/by-the-end-of-2011-android-market-will-have-as-many-users-as-apple%E2%80%99s-app-store/>. Acesso em 26/09/2011.

MIKALAJUNAITE, Egle. **Weather category apps are the most profitable in Android Market**. 12/09/2011. Disponível em <http://www.research2guidance.com/weather-category-apps-are-the-most-profitable-in-android-market/>. Acesso em 26/09/2011.

MIRANDA, Joarez. **Android lidera crescimento em mercado europeu**. 13/09/2011. Disponível em

<http://www.techtudo.com.br/artigos/noticia/2011/09/android-lidera-crescimento-em-mercado-europeua.html>. Acesso em 22/09/2011

MOBILEIN. **Conceitos chave do Android – parte 1 – Camadas**. 2010. Disponível em <http://mobilein.com.br/?p=55>. Acesso em 22/09/2011.

MOTOROLA. **Motorola Dyna TAC 8000X, 1983**. Disponível em <http://mediacenter.motorola.com/imagelibrary/default.aspx?SubjectID=191>. Acesso em 25/10/2011.

NOKIA DEVELOPERS. **Symbian OS – Português**. Disponível em < http://www.developer.nokia.com/Community/Wiki/Symbian_OS_-_Portugu%C3%AAs>. Acesso em 2/09/2011.

OLHAR DIGITAL. **Android apresenta maior crescimento entre plataformas móveis**. Disponível em <http://olhardigital.uol.com.br/produtos/mobilidade/android-apresenta-maior-crescimento-entre-plataformas-moveis>. Acesso em 25/09/2011.

OLIVEIRA, Rômulo Silva de; CARISSIMI, Alexandre da Silva; TOSCANI, Simão Sirineo. **Sistemas Operacionais**. Volume III. 03/12/2001. Disponível em <http://www.das.ufsc.br/~romulo/artigos/Romulo-Carissimi-Simao-Rita2001.pdf>. Acesso em 26/10/2011.

PALMBRASIL. **Conheça o Windows Mobile**. Disponível em <http://www.palmbrasil.com.br/windows-mobile/informacoes-windows-mobile/conheca-windows-mobile>. Acesso em 26/09/2011.

PRADO, Sérgio. **Introdução ao funcionamento interno do Android**. 15/08/2011. Disponível em <http://www.sergioprado.org/2011/08/15/introducao-ao-funcionamento-interno-do-android/>. Acesso em 22/09/2011.

RODRIGUES, Francisco. **Proposta arquitetural para uma aplicação em Android**. 03/04/2011. Disponível em <http://www.thecodebakers.org/2011/04/proposta-arquitetural-para-uma.html>. Acesso em 26/10/2011.

SOARES, Diego. **Versões Android – Conhecendo cada uma delas. 2011**. Disponível em <http://www.mestreandroid.com.br/versoes-android-conhecendo-cada-uma-delas/>. Acesso em 22/09/2011.

SQLITE. **About SQLite**. Disponível em <http://www.sqlite.org/about.html>. Acesso em 19/09/20112011.

STRICKLAND, Jonathan. **Como funciona o Android (Google Phone)**. 16/09/2009. Disponível em <http://informatica.hsw.uol.com.br/google-phone2.htm>. Acesso em 22/09/2011.

TECHTUDO. **iOS**. Disponível em <http://www.techtudo.com.br/tudo-sobre/ios.html>. Acesso em 24/09/2011.

TOZZETO, Claudia. **Venda de celulares básicos começa a cair no Brasil em 2011**. Disponível em <http://tecnologia.ig.com.br/venda-de-celulares-basicos-comeca-a-cair-no-brasil-em-2011/n1597254559575.html>. Acesso em 25/10/2011

VOGEL, Lars. **Android SQLite Database - Tutorial**. 2011. Disponível em <http://www.vogella.de/articles/AndroidSQLite/article.html>. Acesso em 26/09/2011.