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

JAIME CATUSSO

UTILIZANDO ANDROID SDK NO DESENVOLVIMENTO DE APLICAÇÕES PARA DISPOSITIVOS MÓVEIS

TRABALHO DE DIPLOMAÇÃO

MEDIANEIRA
2011

JAIME CATUSSO

UTILIZANDO ANDROID SDK NO DESENVOLVIMENTO DE APLICAÇÕES PARA DISPOSITIVOS MÓVEIS

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: Prof. M. Sc Fernando Schütz.

MEDIANEIRA

2011

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

TERMO DE APROVAÇÃO

UTILIZANDO ANDROID SDK NO DESENVOLVIMENTO DE APLICAÇÕES PARA DISPOSITIVOS MÓVEIS

Por

JAIME CATUSSO

Este Trabalho de Diplomação (TD) foi apresentado às 14:00 h do dia 14 de junho de 2011 como requisito parcial para a obtenção do título de Tecnólogo no Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, 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. M. Sc. Fernando Schütz
UTFPR – *Campus* Medianeira
(Orientador)

Prof. M. Sc. Allan Gavioli
UTFPR – *Campus* Medianeira
(Convidado)

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

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

A folha de aprovação assinada encontra-se na Coordenação do Curso.

“O único lugar aonde o sucesso vem antes do trabalho é no dicionário.”

Albert Einstein

RESUMO

CATUSSO, Jaime**.** Utilizando Android SDK no desenvolvimento de aplicações para dispositivos móveis. Trabalho de Conclusão do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas. Universidade Tecnológica Federal do Paraná. Medianeira, 2011.

Com o aumento de dispositivos móveis que fazem uso do sistema operacional *Android*, também aumentou a necessidade de aplicativos que atendam os usuários desse mercado. Por possuir uma plataforma de desenvolvimento aberta, o *Android* oferece aos desenvolvedores a capacidade e recursos necessários para desenvolver aplicações inovadoras. Através de recursos que a API do *Android* oferece, é possível integrar as aplicações entre diversas linguagens de programação. A comunicação com a *Web* é de grande utilidade para o desenvolvimento de aplicativos que tragam informações em tempo real ao usuário, dessa forma poupando memória de armazenamento do dispositivo. Segundo pesquisas, o mercado de aplicativos para celulares que faz uso de *Android* cresce muito rapidamente, devido à demanda de usuários que aumenta a cada ano. Este trabalho apresenta um estudo sobre desenvolvimento de aplicações para dispositivos móveis utilizando *Android SDK* e suas APIs.

Palavras-chaves: *Android SDK*, Dispositivos Móveis, API, *Web services.*

ABSTRACT

CATUSSO, Jaime. Using Android SDK to develop applications for mobile devices. Trabalho de Conclusão do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas. Universidade Tecnológica Federal do Paraná. Medianeira, 2011.

With the increase of mobile devices that use the operating system Android also increased the need for applications that meet the Users of this market. By owning a development platform open, Android offers developers the ability and resources necessary to develop innovative applications. By resource Android's API to offer, you can integrate applications from various programming languages. Communication with the Web is of great useful for developing applications that bring information real-time user, thus saving memory storage device. According to surveys, the market for mobile applications that makes use of Android is growing very rapidly due to demand users increases every year. This paper presents a study on development of mobile applications using Android SDK and its APIs

Keywords: *Android SDK*, Mobile Devices, API, *Web services.*

LISTAS DE SIGLAS

ADT	*Android Development Tools*
API	*Application Programming Interface*
ASCII	*American Standard Code for Information Interchange*
EUA	*United States of America*
GPS	*Global Positioning System*
HTC	*High Tech Computer Corporation*
HTTP	*Hypertext Transport Protocol*
HTTPS	*Hypertext Transfer Protocol Secure*
ID	Chave de Identificação
IDE	*Integrated Development Environment*
MER	Modelo De Entidades E Relacionamento
PDF	*Portable Document Format*
SDK	*Software Development Kit*
SMS	*Short Message Service*
SO	*Sistema Operacional*
SOAP	*Simple Object Access*
SQL	*Structured Query Language*
UDDI	*Universal Discovery, Description, and Integration*
UML	*Unified Modeling Language*
URL	*Uniform Resource Locator*
UTFPR	Universidade Tecnológica Federal do Paraná
W3C	*World Wide Web Consortium*
WSDL	*Web Service Description Lenguage*
XML	*Extensible Markup Language*

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

Segundo UOL ECONOMIA (2011) o sistema operacional *Android*, criado pelo Google para dispositivos móveis, se transformou no SO líder para *smartphones* nos EUA em 2011, na frente dos desenvolvidos por *Apple* e *Research in Motion* (RIM). A popularidade do *Android* foi possível por se tratar de um *software* que o Google coloca à disposição de todas as empresas de telefonia, por isso é vendido através de distintas marcas como Motorola, Samsung e HTC. O *Android* vem ganhando força não só como sistema operacional, mas também no número de aplicativos disponibilizados.

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*.

Assim como aplicações *Web* e *Desktop* interagem entre si através de *Web services*, é possível fazer com que um dispositivo *Android*, através de uma aplicação interaja com um banco de dados de uma empresa por exemplo. A integração entre essas duas tecnologias permite atender necessidades tanto do usuário como de empresas.

Através de um *Web service* integrado com uma aplicação *Android*, um usuário pode consultar informações como a previsão do tempo por exemplo. Do mesmo modo, um funcionário de uma empresa pode através de um *Web service* integrado com uma aplicação no dispositivo *Android*, poderá enviar dados de uma venda diretamente para o banco de dados da empresa em que trabalha.

1.1 OBJETIVO GERAL

Esse trabalho tem por objetivo projetar e desenvolver um aplicativo para dispositivos móveis, utilizando tecnologias do sistema operacional *Android* integrada com um serviço *Web*. Este aplicativo trata-se de uma lista telefônica com números de médicos e farmácias. O aplicativo *Android* realizará consultas de números de telefone fixo e celular a um *Web service* que contém um banco de dados MYSQL com dados de médicos e farmácias, essa consulta será realizada através de conexão com a internet.

1.2 OBJETIVOS ESPECÍFICOS

Como objetivos específicos do projeto têm-se:

- Elaborar um referencial teórico sobre as tecnologias as serem utilizadas.
- Desenvolver análise e projeto do sistema proposto como estudo de caso.
- Codificar um aplicativo de exemplo utilizando técnicas apropriadas ao sistema *Android*. Este aplicativo será desenvolvido na plataforma *Android SDK*. Trata-se de uma lista telefônica para celular com números de médicos e farmácias, fazendo uso de conexão com *Internet* para acesso a um serviço *Web*.
- Testar o aplicativo.

Durante e após o processo de desenvolvimento serão realizado testes de funcionamento do aplicativo.

1.3 JUSTIFICATIVA

Segundo pesquisas realizadas, constatou-se que o *Android* vem crescendo no mercado móvel, assim como os aplicativos para esse sistema operacional.

Segundo o site IDG NOW (2011), um recente estudo elaborado pela agência de pesquisas *Market Force,* informa que a intenção de compra de aparelhos com o sistema da Google supera as intenções de adquirir um *iPhone*. Em dezembro de 2010, outro levantamento, realizado entre 5.600 americanos, dava conta de os *smartphones* já estarem nos bolsos de 51% da população. Entre os 49% restantes, 34% disseram planejar a compra de um *smartphone* com *Android*, contra 21% que avaliam a compra de um *iPhone*. Para 12%, o aparelho de escolha é o *BlackBerry* e outros 25% ainda estavam indecisos. (IDG NOW, 2011)

"Um fator que pesa razoavelmente na decisão por uma ou outra plataforma é a quantidade de aplicativos disponíveis nos mercados da *internet*", diz o estudo da *Market Force*. O estudo informa que tal fator era previsível, mas que ninguém suspeitava que o segmento fosse crescer tanto. Em se tratando de *Android*, esse crescimento é de fato assustador. Enquanto a Apple registrava um aumento de 132% entre os anos 2009 e 2010, o *Android Market* explodiu com 861.5% no mesmo período. (IDG NOW, 2011)

O *Gartner* divulgou uma pesquisa realizada em 16 de Fevereiro de 2011 mostrando que o mercado de sistemas operacionais para *smartphones* que fazem uso de *Android* cresceu 888,8% em 2010 e passou para a 2ª posição do *ranking*.

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

A estrutura geral do trabalho de conclusão de curso é composta por cinco capítulos constituídos:

O Capítulo um apresenta a introdução sobre o assunto abordado com os objetivos gerais e específicos, e uma justificativa sobre o tema de estudo.

O Capítulo dois apresenta o referencial teórico do trabalho. Além do sistema operacional *Android*, o portal *Android Market* e as vantagens que proporcionam ao desenvolvedor. Também descreve como se pode organizar uma estrutura de um projeto *Android*, além das principais *widget* disponíveis no *Android SDK*. Descreve o conceito básico de *Web service*, assim como suas principais tecnologias.

O Capítulo três aborda os materiais e métodos utilizados no desenvolvimento do trabalho de diplomação.

O Capítulo quatro aborda o desenvolvimento de um aplicativo para dispositivos móveis integrado com um *Web service* utilizando o *Android SDK*.

O Capítulo cinco apresenta as considerações finais sobre o estudo realizado.

2 REFERENCIAL TEÓRICO

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

2.1 O SISTEMA OPERACIONAL ANDROID

Os sistemas operacionais móveis estão presentes na grande maioria dos dispositivos que utilizamos todos os dias, como por exemplo, seu aparelho celular, *notebook*, GPS e até mesmo seu carro.

Android é um sistema operacional móvel desenvolvido pela Google que roda sobre um núcleo *Linux*. O *Android SDK* fornece as ferramentas e *APIs* necessárias para começar o desenvolvimento de aplicativos na plataforma *Android* usando a linguagem de programação Java. (ANDROID DEVELOPER, 2011)

Por se tratar de uma plataforma para desenvolvimento gratuita, o *Android* oferece aos desenvolvedores a capacidade de criar aplicações extremamente inovadoras. Os desenvolvedores podem aproveitar o *hardware* do dispositivo, assim como informações de localização de acesso, definir alarmes, notificações para adicionar a barra de status, entre outras funcionalidades.

Os desenvolvedores têm acesso total às APIs de desenvolvimento, a arquitetura do aplicativo é projetada para simplificar a reutilização dos componentes.

O *Android* inclui um conjunto de bibliotecas que fornece a maioria das funcionalidades disponíveis nas principais bibliotecas da linguagem de programação Java.

O Google fornece várias classes para interfaces gráficas, comunicação entre processos, gerenciamento de banco de dados, gerenciamento de arquivos, gráficos e até vocalização de texto. Como desenvolvedor, é possível criar aplicativos com o SDK do *Android*, testá-los em um emulador ou em um dispositivo real, empacotá-los e liberá-los para o mundo (Linux Magazine, 2011, p 39).

O *Android SDK* conta com um emulador e várias ferramenta ótimas para o desenvolvedor assinar seus aplicativos, depurar seu código e desenhar interfaces gráficas.

2.2 ANDROID MARKET

Quando um aplicativo desenvolvido com *Android SDK* estiver pronto para ser divulgado para o mundo, é possível distribuí-lo gratuitamente, ou de forma paga através do *Android Market*, loja fornecida pela Google.

O *Android Market* é apenas uma das várias formas de disponibilizar um aplicativo a usuários e clientes. É útil por apresentar aplicativos somente para dispositivos capazes de executá-los. Por exemplo, se um aplicativo requer uma câmera, dispositivos sem câmeras nem ao menos poderão ver que este aplicativo existe no *Android Market*. Como outro exemplo, se um aplicativo precisa de *Android* 2.1 ou posterior para correto funcionamento, os dispositivos que possuem as versões mais antigas não conseguirão vê-lo na lista de aplicativos (Linux Magazine, 2011, p. 44).

Segundo dados apurados pelo site *androidz.com.br* através das estatísticas do *androlib.com,* o *Android Market* triplica o numero de aplicativos e cresce em ritmo acelerado ao redor do mundo. Este crescimento deve-se ao fato do *Android* ser uma plataforma O*pen Source*, ou seja, fabricantes de todo mundo podem alterar a plataforma ao seu gosto. Essa "adaptabilidade" do sistema favoreceu e muito a disseminação entre os *smartphones* (AndroidZ, 2011).

Figura 1 - Estatísticas de aplicativos *Android Market*.

Fonte: AndroidZ (2011)

A Figura 1 mostra o crescimento do *Android* até o ano 2011. Em Março de 2011, 34349 novas aplicações foram adicionadas ao *Android Market*, o que revela um aumento de 364% em relação o mesmo período do ano passado, quando 9421 aplicações foram adicionadas. A quantidade de aplicativos e jogos disponíveis no *Android Market* triplicou no último ano, garantindo cada vez mais interesse dos desenvolvedores, e claro, dos usuários (AndroidZ, 2011).

Como desenvolvedor também é possível controlar no *Android Market* em quais países um aplicativo será disponibilizado, e em quais operadoras, além disso, é possível ter gráficos estatísticos para o controle de *downloads* do aplicativo.

A forma como o *Android Market* apresenta estatísticas de *downloads* de um aplicativo para o desenvolvedor pode ser vista em gráficos no painel de desenvolvedores:

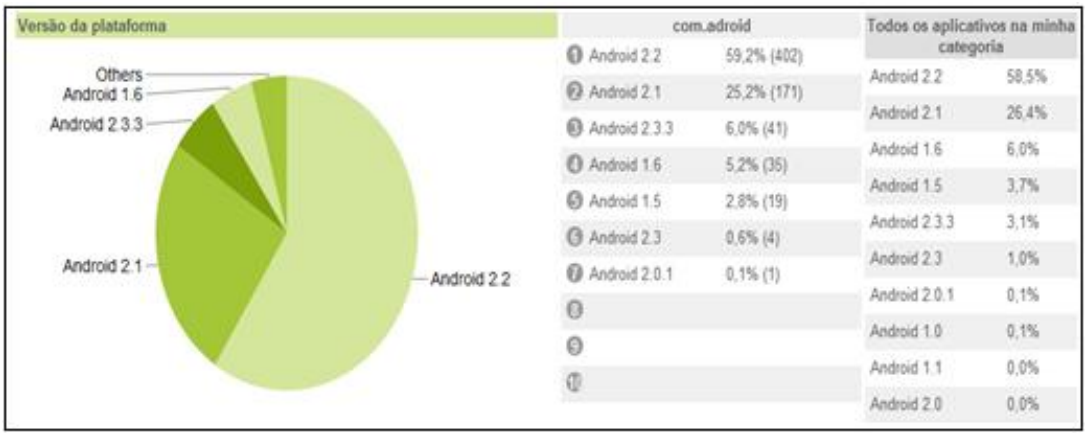
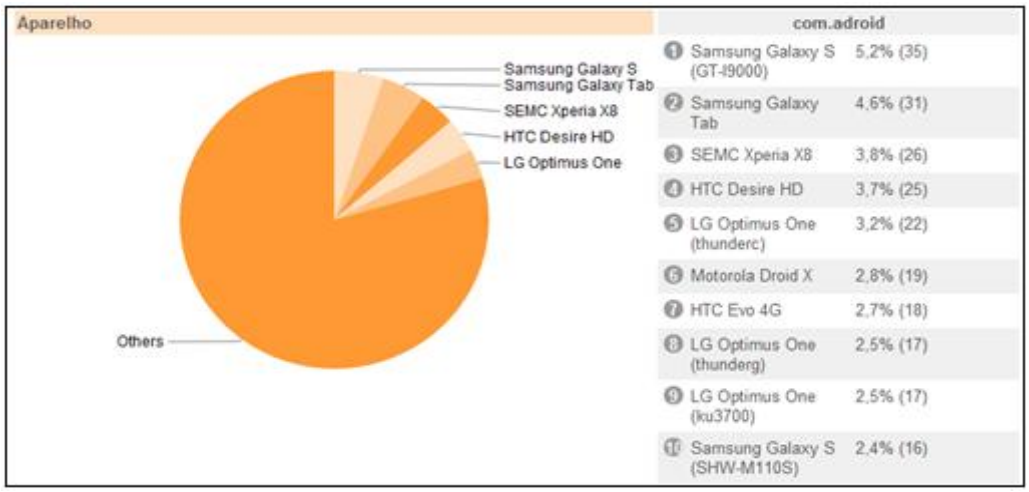


Figura 2 – Relatório por versão da plataforma.

Fonte: Android Market (2011)

A Figura 2 mostra estatísticas de *downloads* por versão da plataforma, apresentadas no painel para desenvolvedores do *Android Market*.

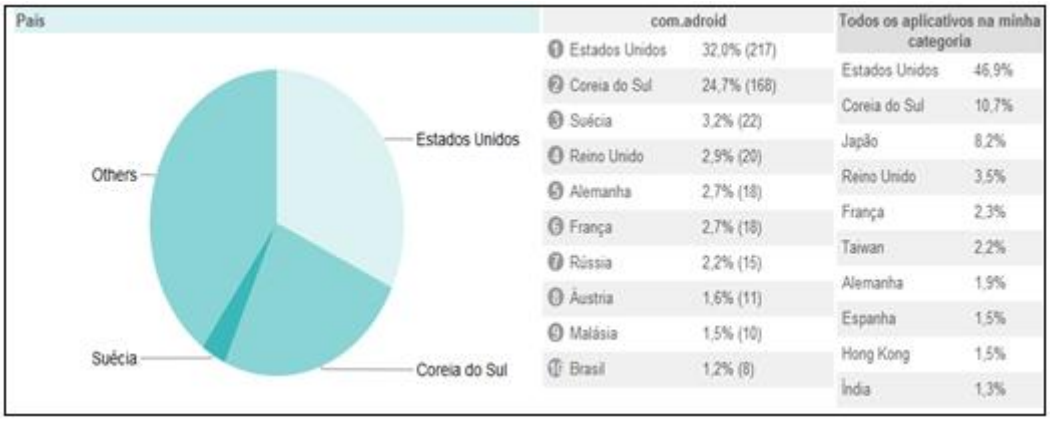


	com.adroid	
1	Samsung Galaxy S (GT-I9000)	5,2% (35)
2	Samsung Galaxy Tab	4,6% (31)
3	SEMC Xperia X8	3,8% (26)
4	HTC Desire HD	3,7% (25)
5	LG Optimus One (thunderc)	3,2% (22)
6	Motorola Droid X	2,8% (19)
7	HTC Evo 4G	2,7% (18)
8	LG Optimus One (thunderg)	2,5% (17)
9	LG Optimus One (lku3700)	2,5% (17)
10	Samsung Galaxy S (SHW-M110S)	2,4% (16)

Figura 3 - Relatório por aparelhos.

Fonte: Android Market (2011)

A Figura 3 mostra estatísticas de *downloads* por aparelho, apresentadas no painel de desenvolvedores do *Android Market*.



	com.adroid		Todos os aplicativos na minha categoria	
1	Estados Unidos	32,0% (217)	Estados Unidos	46,9%
2	Coreia do Sul	24,7% (168)	Coreia do Sul	10,7%
3	Suécia	3,2% (22)	Japão	8,2%
4	Reino Unido	2,9% (20)	Reino Unido	3,5%
5	Alemanha	2,7% (18)	França	2,3%
6	França	2,7% (18)	Taiwan	2,2%
7	Rússia	2,2% (15)	Alemanha	1,9%
8	Áustria	1,6% (11)	Espanha	1,5%
9	Malásia	1,5% (10)	Hong Kong	1,5%
10	Brasil	1,2% (8)	Índia	1,3%

Figura 4 - Relatório por País.

Fonte: Android Market (2011)

A Figura 4 mostra estatísticas de *downloads* por País, apresentadas no painel de desenvolvedores do *Android Market*.

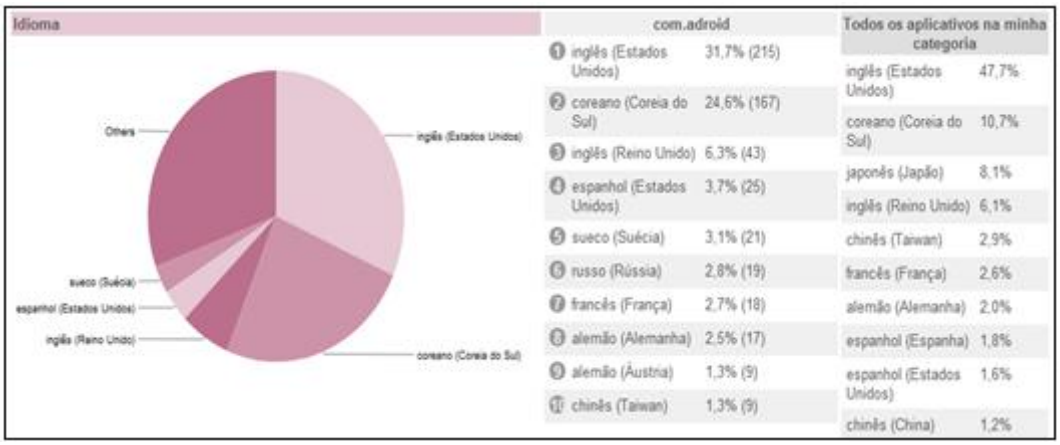


Figura 5 - Relatório por idioma.

Fonte: Android Market (2011)

A Figura 5 mostra estatísticas por idioma, apresentadas no painel de desenvolvedores do *Android Market*.

2.3 ESTRUTURA DE UM PROJETO ANDROID

Para desenvolver um projeto utilizando *Android SDK*, o desenvolvedor deve efetuar o *download* do SDK no site do *Android Developers*, e fazer a instalação do mesmo. Além disso, é necessário que haja uma ferramenta de desenvolvimento. Para este trabalho de diplomação foi utilizado o *Eclipse SDK*, e foi instalado o *ADT Plugin* específico para *Android*, também encontrado no site do *Android Developers*. O site do *Android* fornece as informações passo a passo para o preparo do ambiente de desenvolvimento.

Com o ambiente de desenvolvimento configurado, para criar um projeto o desenvolvedor deve abrir o *Eclipse* e clicar em *File/New/AndroidProject*. Assim será aberta uma tela onde se descrevem os dados do projeto.

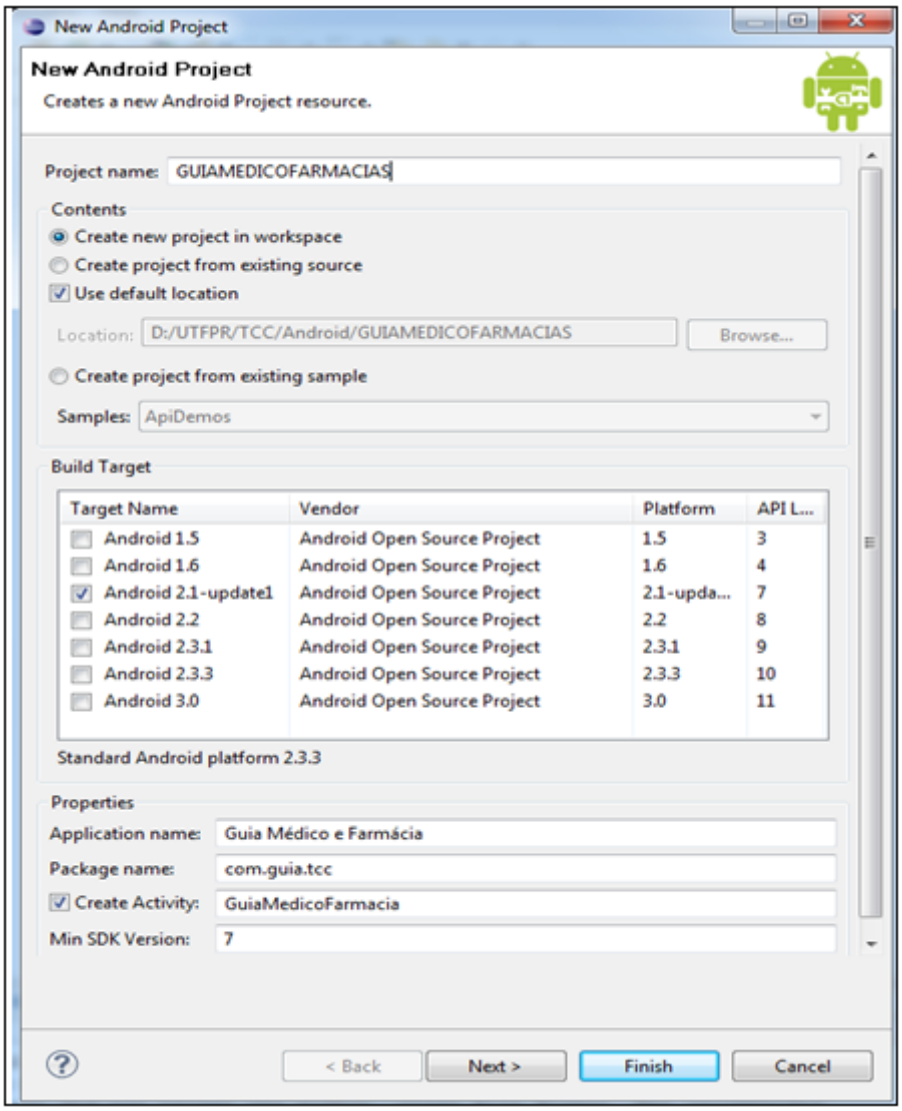


Figura 6 – *Screenshot* do formulário *New Android Project* no *Eclipse*.

A Figura 6 mostra a tela onde é definido o nome do projeto *Android*, onde o desenvolvedor pode descrever qual versão da plataforma o aplicativo criado será executado. O nome da aplicação e o pacote também podem ser descritos nessa tela.

Quando o desenvolvedor marca a opção *Create Activity* o *Eclipse* gera a classe *Java* que estende de *Activity*, essa classe é a classe principal do projeto, quando executado ou instalado no dispositivo *Android*, será a primeira tela a ser apresentada para o usuário. Ainda é possível definir qual é o nível mínimo da API exigido pelo dispositivo. Sendo assim, quando clicado em *Finish*, é gerada a estrutura do projeto.

Para esclarecimento, uma *Activity* (atividade) é uma única ação interagindo com uma interface de usuário que realiza uma função especifica.

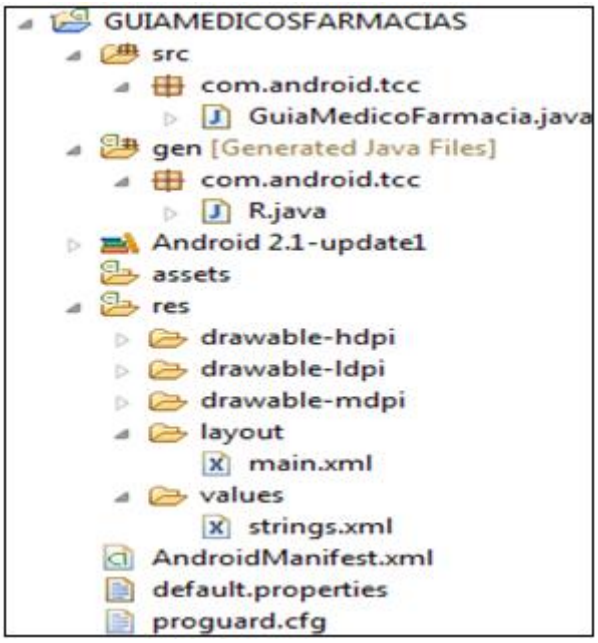


Figura 7 - Estrutura do projeto *Android.*

A Figura 7 mostra a estrutura de um projeto *Android* criado no *Eclipse*, note que foi gerado o pacote e a classe *Java* principal.

```
package com.android.tcc;

import android.app.Activity;

public class GuiaMedicoFarmacia extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

Figura 8 - Classe GuiaMedicoFarmacia.java.

A Figura 8 mostra a classe *Java* principal que estende de *Activity*, essa classe contém um método chamado *onCreate*, esse método é responsável pelo momento em que a classe é chamada no dispositivo, essa classe faz chamada através do método *setContentView* a um arquivo XML chamado *main*, esse arquivo é responsável pelo *layout* da tela.

Na pasta `gen/` existe a classe R.java que é gerada automaticamente, essa classe é utilizada para facilitar a obtenção dos componentes ou recursos apenas informando o seu nome, não é recomendado alterar essa classe manualmente.

A pasta `res/` contém os recursos que serão utilizados pela aplicação. Estes recursos são organizados em pastas de acordo com o seu propósito. O diretório `res/` possui três subpastas:

- ***drawable***: esta pasta contendo somente a imagem icon.png, deve conter todos os recursos de imagens utilizados na aplicação.

- ***layout***: contém as definições dos *layouts* utilizados na aplicação que são arquivos no formato XML.

- ***values***: contêm valores estáticos, *strings* por exemplo, que podem ser carregados de um arquivo XML.

No projeto o *Eclipse* é gerado um arquivo chamado "*AndroidManifest.xml"*, esse arquivo é obrigatório e é nele que devem serem feitas as configurações de todos os recursos utilizados pela aplicação, como *Activity*, componentes gráficos, *layouts*, imagens, recursos de *hardware* e etc.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.android.tcc"
      android:versionCode="1"
      android:versionName="1.0">
    <uses-sdk android:minSdkVersion="7" />

    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".GuiaMedicoFarmacia"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

    </application>
</manifest>
```

Figura 9 - Arquivo *AndroidManifest.xml*.

A Figura 9 mostra a estrutura do arquivo "*AndroidManifest.xml"*, note que ele contém dados como pacote, versão do aplicativo, nome da aplicação, e nele é definido a *Activity* principal, referenciada a classe *Java*. Através do *intent-filter* deve ser definido que essa classe será carregada no momento em que a aplicação é executada.

Toda a aplicação criada pode ser testada através do *Android* Emulador que vem junto com o *Android SDK*.

Para executar a aplicação o desenvolvedor deve clicar com o botão direito encima do projeto e em *Run/Run As/Android Application*. Após esta etapa, é possível verificar o funcionamento do aplicativo no emulador. Quando o projeto é exportado, é necessário assinar o aplicativo, para isso o desenvolvedor deve informar uma chave de segurança *KeyStore*, essa chave pode ser gerada pelo próprio *Eclipse SDK*, e quando finalizado todas as etapas de exportação o SDK gera um arquivo com extensão .apk, esse arquivo pode ser distribuído e instalado em dispositivos que rodam *Android*.

2.4 WIDGETS ANDROID SDK

Uma *widget* é um componente de uma interface gráfica do usuário, o que inclui janelas, botões, menus, ícones, barras de rolagem, etc. De uma forma mais técnica, algumas *widget* têm como objetivo receber dados do usuário e com isso gerar algum tipo de registro.

No *Android SDK* uma *widget* é definida e configurada em um arquivo XML, podendo ser acessada por uma classe *Java*.

Para o desenvolvimento deste trabalho de diplomação foram utilizadas algumas *widgets* que o *Android SDK* oferece, estas *widgets* serão mostradas ao decorrer desta sessão.

2.4.1 WIDGET LINEARLAYOUT

A *widget LinearLayout* é utilizada para organização dos componentes na tela, através dela é possível flutuar outras *widgets* para o centro da tela, organizar de forma vertical ou horizontal e etc.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:id="@+id/layout_root"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:padding="2dp"
  >

</LinearLayout>
```

Figura 10 - A *Widget LinearLayout.*

A Figura 10 mostra como é declarado um *LinearLayout* em um arquivo XML, na figura através da propriedade *orientation* é possível orientar outras *widgets* em forma vertical. Na propriedade *layout_width* com o valor *fill_parent* é possível que as *widgets* ocupem o tamanho da direita até a esquerda na tela. As *widget*s que farão parte do *LinearLayout* devem estar declaradas antes da tag </*LinearLayout*>.

2.4.2 WIDGET TEXTVIEW

A *widget TextView* funciona como se fosse uma *Label* (rótulo), onde nela poderá ser mostrado alguma informação, mensagem de texto e etc.

```
<TextView
android:text="Escolha uma opção:"
android:id="@+id/textView1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:textColor="#FFF"
android:textSize="17dp"
android:background="@drawable/backgroundtext"
android:padding="5px"
android:textStyle="bold"
/>
```

Figura 11 - A *Widget TextView*.

A Figura 11 mostra como e declarado uma *TextView* em um arquivo XML, tem-se a propriedade ID como base para acesso a *widget* de uma classe *Java*.

2.4.3 WIDGET SPINNER

A *widget Spinner* é um componente do tipo caixa de combinação (*ComboBox*) onde nele são armazenado vários itens a serem selecionados. Para que um componente possa ser selecionado, o usuário deve clicar na seta exibida na tela do dispositivo, para que os itens possam ser apresentados, e serem selecionados.

```
<Spinner
android:id="@+id/sp_cidades"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
/>
```

Figura 12 - A *Widget Spinner.*

A Figura 12 mostra como é declarado um *Spinner* em um arquivo XML, para popular um *Spinner* com dados é necessário acessá-lo através de uma classe *Java*, passando uma Lista de objetos.

2.4.4 WIDGET EDITTEXT

A *widget EditText* funciona como se fosse uma caixa de texto onde é possível digitar dados do teclado do dispositivo.

```
<EditText
android:id="@+id/edt_nomemedico"
android:layout_width="fill_parent"
android:text=""
android:layout_height="wrap_content"
android:enabled="false"
android:textColor="#000"
/>
```

Figura 13 - A *Widget EditTest.*

A Figura 13 mostra como é declarado um *EditText* no arquivo XML, através da propriedade ID, é possível acessá-lo de uma classe *Java* para recuperar ou até mesmo adicionar de dados.

2.4.5 WIDGET RADIOGROUP

A *widget RadioGroup* é um componente muito utilizado em opções de múltipla escolha, onde somente uma única opção pode ser selecionada.

```
<RadioGroup android:id="@+id/radioGroupFarmacias"
android:layout_width="wrap_content"
android:layout_height="wrap_content" >
    <RadioButton android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Buscar todas"
    android:id="@+id/Farradio0"
    android:checked="true"
    android:onClick="bt_radioGroupFarmaciasClick"
    android:textStyle="bold" android:textColor="#000"
    />
    <RadioButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Buscar por nome"
    android:id="@+id/Farradio1"
    android:onClick="bt_radioGroupFarmaciasClick"
    android:textStyle="bold"
    android:textColor="#000"
    />
</RadioGroup>
```

Figura 14 - A *Widget RadioGroup.*

A Figura 14 mostra como é declarado um *RadioGroup* no arquivo XML, note que foram declaradas duas *widget RadioButton*, cada uma representa uma opção de escolha para seleção.

2.4.6 WIDGET BUTTON

A *widget Button* é um botão de comando, que quando clicado, dispara uma ação ou um evento.

```
<Button
android:text="Buscar médico"
android:id="@+id/button1"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:onClick="bt_buscarMedicos"
/>
```

Figura 15 - A *Widget Button.*

A Figura 15 mostra como é declarado um *Button* no arquivo XML, onde possui a propriedade *onClick*, através dessa propriedade atribuí-se o nome do método que fará o

evento do clique do botão, esse método deve ser criado com o mesmo nome junto à classe *Java* que faz chamada a *widget Button*.

2.5 WEB SERVICES

Para realizar o acesso a base de dados no aplicativo *Android* GUIA MÉDICO E FARMÁCIA deste trabalho, foi feito uso de *Web services*. Esta sessão irá apresentar o conceito de *Web services*.

Um *Web service* é uma aplicação de *software* que pode ser acessada remotamente usando diferentes linguagens baseadas em XML. Normalmente, um *Web service* é identificado por um URL, exatamente como qualquer outro site *Web*. O que torna os *Web services* diferentes dos sites *Web* comuns é o tipo de informação que podem fornecer. (POTTS, Kopack, 2003, p.3)

A maioria dos sites *Web* é projetada para fornecer uma resposta a uma requisição de uma pessoa. Essa requisição assume a forma de um documento de texto contendo algumas instruções bastante simples para o servidor. Essas instruções limitam-se ao nome de um documento a ser retornado ou a uma chamada a um programa do lado servidor, juntamente com alguns parâmetros. (POTTS, Kopack, 2003, p.4)

Um *Web service* é semelhante na medida em que é acessado através de um URL. A diferença está no conteúdo do que é enviado na requisição do cliente para o servidor. Os clientes de Web service enviam um documento XML formatados de uma maneira especial de acordo com as regras de especificação SOAP. (POTTS, Kopack, 2003, p.4)

Uma mensagem SOAP pode conter uma chamada a um método juntamente com quaisquer parâmetros que poderiam ser necessários. Além disso, a mensagem pode conter diversos itens de cabeçalho que especificam melhor o propósito do cliente. Os itens de cabeçalho podem designar o que os *Web services* farão esse método chamar após o serviço atual terminar seu trabalho ou podem conter informações de segurança. (POTTS, Kopack, 2003, p.4)

A maioria do entusiasmo em torno dos *Web services* é baseada na promessa de interoperabilidade. A arquitetura dos *Web services* se baseia no envio de mensagens XML em um formato SOAP específico. A XML pode ser representada como caracteres ASCII comuns, que podem ser transferidos facilmente de um computador para outro. (POTTS, Kopack, 2003, p.4)

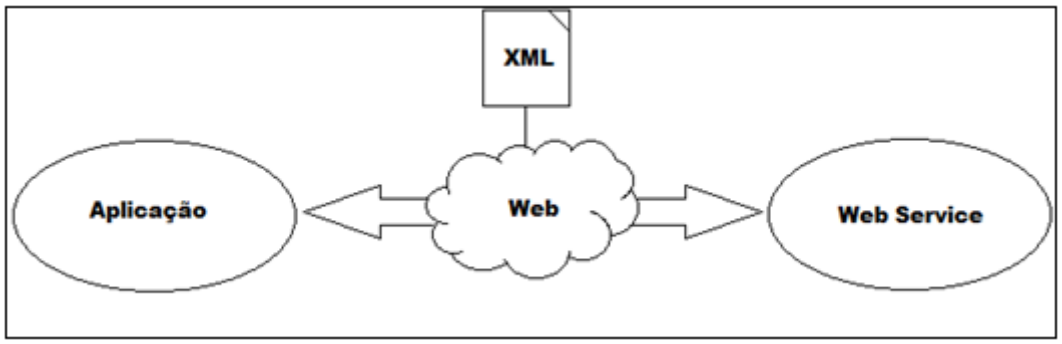


Figura 16 - Aplicação interagindo com *Web service*.

Com base no texto, a Figura 16 foi criada para mostrar uma aplicação que se comunica com um *Web service* através de um documento XML.

As bases para a construção de um *Web service* são os padrões XML e SOAP. O transporte dos dados é realizado normalmente via protocolo HTTP ou HTTPS para conexões seguras. Os dados são transferidos no formato XML, encapsulados pelo protocolo SOAP.

As questões mais relevantes na segurança de *Web services* são:

- Autenticidade - ter a certeza que uma transação do *Web service* ocorreu entre o servidor e seu cliente.
- Privacidade - todas as mensagens trocadas entre o servidor e o cliente não são interceptadas por uma pessoa não autorizada.
- Integridade - as mensagens enviadas tanto pelo servidor ao cliente, como o contrário, devem permanecer inalteradas.

Os principais padrões de tecnologias dos *Web services* são:

- SOAP (*Simple Object Access*) – O SOAP é uma especificação que define uma gramática XML para o envio e a resposta de mensagens que você recebe de outras partes. Seu objetivo é descrever um formato de mensagem que não esteja vinculado a qualquer arquitetura de *hardware* ou *software*, mas que transmita uma mensagem de qualquer plataforma para qualquer outra plataforma de uma maneira inequívoca. O padrão SOAP contém duas partes: o cabeçalho que conduz as instruções de processamento, e o corpo, que contem o *payload*. O *payload* contém as informações que você deseja enviar. Os dois tipos de mensagens SOAP são os documentos e as *Remote Procedure Calls* (RPCs). O *payload* de uma mensagem de documento é qualquer documento XML que você queira mover de um computador para outro. Uma RPC é uma chamada de método que deve ser executada no computador do *Web service*. A mensagem de RPC realiza a mesma função de uma chamada

de método comum a uma linguagem de programação comum. A diferença é que a chamada pode ocorrer pela *internet*. (POTTS, Kopack, 2003, p.6)

- XML (*Extensible Markup Language*) – A XML é uma linguagem sobre a qual é construído todos os *Web services*. É uma linguagem de marcação recomendada pela W3C para a criação de documentos com dados organizados hierarquicamente, tais como textos, banco de dados ou desenhos vetoriais. A linguagem XML é classificada como extensível porque permite definir os elementos de marcação. (TEC MUNDO, 2011) Responsável por criar gramáticas que são descritas em esquemas XML, SOAP, WSDL e UDDI são exemplos de gramáticas baseadas em XML.
- HTTP (*Hypertext Transport Protocol*) – O HTTP é um padrão desenvolvido para facilitar a transferência de requisições de um navegador par um servidor *Web*. A Web service faz uso desse protocolo para transportar mensagens SOAP e documentos WSDL de um computador para outro.
- WSDL (*Web Service Description Lenguage*) – WSDL é uma linguagem baseada em XML utilizada para descrever *Web services* funcionando como um contrato do serviço. Trata-se de um documento escrito em XML que além de descrever o serviço, especifica como acessá-lo e quais as operações ou métodos disponíveis.
- UDDI (*Universal Discovery, Description, and Integration*) – A especificação UDDI descreve como um cliente potencial de um *Web service* poderia aprender sobre suas capacidades e obter informações básicas necessárias para fazer o contato inicial com o site. Normalmente, esse contato inclui o *download* da WSDL. (POTTS, Kopack, 2003, p.7)

Através do uso desses padrões de tecnologias descritos, os *Web services* fazem a comunicação de dados e informações na *Web*.

2.6 DIAGRAMAS UML

A UML (*Unified Modeling Language*) é uma linguagem padrão para a elaboração da estrutura de projetos de *software*. A UML poderá ser empregada para a visualização, a especificação, a construção e a documentação de artefatos que façam uso de sistemas complexos de *software*. (BOOCH, Grady, 2000, p.13)

Um diagrama UML segundo BOOCH (2000) é a apresentação gráfica de um conjunto de elementos, geralmente representadas como gráficos de vértices (itens) e arcos (relacionamentos). São desenhados para permitir a visualização de um sistema sob diferentes perspectivas; nesse sentido, um diagrama constitui uma projeção de um determinado sistema. (BOOCH, Grady, 2000, p.25)

O Diagrama de Caso de Uso pode descrever a funcionalidade proposta para um novo sistema, que será projetado.

Um Caso de Uso é um documento que descreve a seqüência de eventos de um ator que usa um sistema para completar um processo, também pode representar uma unidade discreta da interação entre um usuário e o sistema. (MACORATTI, 2011)

O objetivo dos diagramas de seqüência é descrever as comunicações necessárias entre objetos para a realização dos processos em um sistema. Os diagramas de seqüência têm este nome porque descrevem ao longo de uma linha de tempo a seqüência de comunicações entre objetos. (MACORATTI, 2011)

Este trabalho de diplomação fez uso de diagramas UML para uma documentação resumida do Aplicativo *Android*.

3 MATERIAIS E MÉTODOS

A partir deste capítulo serão apresentados os materiais e métodos usados neste trabalho de diplomação.

Para o desenvolvimento de um exemplo prático utilizando *Android SDK*, será desenvolvida uma aplicação *Android* integrada com um servidor *Web service*. Para uma documentação resumida do sistema, foi feito uso de diagramas UML, sendo estes: diagrama de casos de uso, diagrama de classes, diagrama de seqüência.

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

3.1 GUIA MÉDICO E FARMÁCIAS ANDROID

Ao decorrer desta sessão será apresentada a análise do projeto da aplicação GUIA MÉDICO E FARMÁCIA para *Android*.

3.1.1 DESCRICÃO DO PROJETO

O usuário tem a necessidade de consultar um número telefônico de um médico ou uma farmácia para compras ou marcar consultas, ou em casos de extrema urgência. Para fornecer um meio rápido de consulta, foi desenvolvido um aplicativo para celular capaz de se conectar com uma base de dados remota e realizar consultas através de uma conexão com a *Internet.* Sendo assim o usuário não terá necessidade de ter uma lista telefônica com ele, apenas o aplicativo no seu aparelho celular, podendo realizar consultas de qualquer lugar, desde que o aparelho possa se conectar com a *Web*.

3.1.2 DIAGRAMA DE CASOS DE USO

A Figura 17 apresenta o diagrama de casos de uso do sistema GUIA MÉDICO E FARMÁCIA para *Android*:

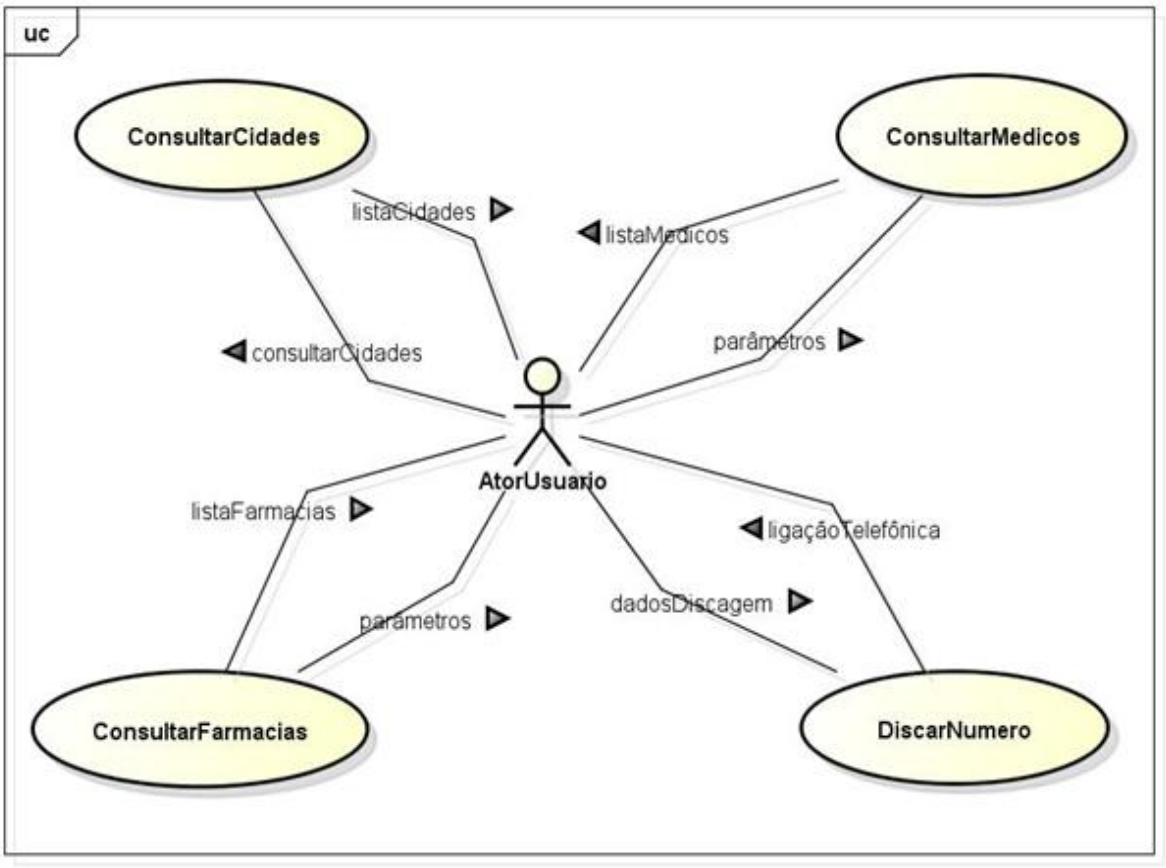


Figura 17 - Diagrama de casos de uso.

Como visto no diagrama da Figura 17, o modelo de caso de uso, trata do Ator Usuário, que será o principal ator do sistema.

3.1.3 DESCRÇÃO DOS CASOS DE USO

Para cada caso de uso, que representa a interação do ator com o sistema, foram representados os seus relacionamentos com os atores e uma descrição do seu comportamento. Ao decorrer deste titulo, será mostrado à descrição dos casos de uso por ator do aplicativo GUIA MÉDICO E FARMÁCIA para *Android*.

Número: 01

Caso de Uso: ConsultarCidade

Descrição: Este caso de uso trata da consulta de cidade.

Ator: AtorUsuario

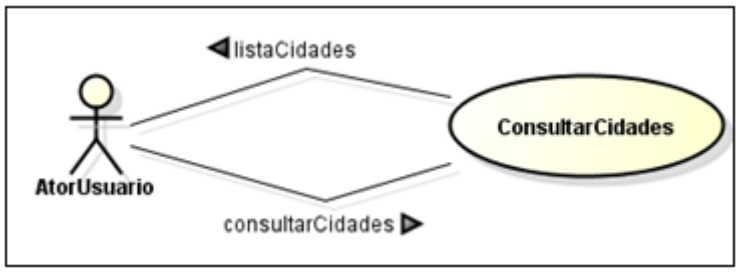


Figura 18 - Caso de Uso ConsultarCidades.

Curso Normal:

1. Usuário solicita consulta de cidades.
2. Sistema busca lista de cidades.
3. Aplicativo mostra as cidades disponíveis.
4. Usuário seleciona uma cidade.
5. Sistema retorna dados da cidade.

Cursos Alternativos:

3. Sistema não encontra cidades cadastradas.
3.1 Sistema retorna objeto nulo.
3.2 Encerra caso de uso.

Número: 02

Caso de Uso: ConsultarMedicos

Descrição: Este caso de uso trata da consulta de médicos.

Ator: AtorUsuario

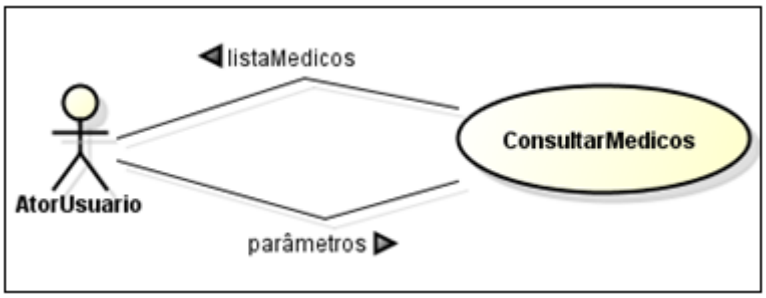


Figura 19 - Caso de Uso ConsultarMedicos.

Curso Normal:

1. Usuário seleciona a categoria médicos no aplicativo.
2. Sistema busca lista de médicos.
3. Aplicativo exibe ao usuário a lista de números de médicos com opção de discagem.
4. Usuário seleciona um número.
5. Sistema retorna dados de discagem.

Cursos Alternativos:

3. Sistema não encontra médicos.
3.1 Sistema retorna objeto nulo.
3.2 Encerra caso de uso.

Número: 03

Caso de Uso: ConsultarFarmacias

Descrição: Este caso de uso trata da consulta de farmácias.

Ator: AtorUsuario

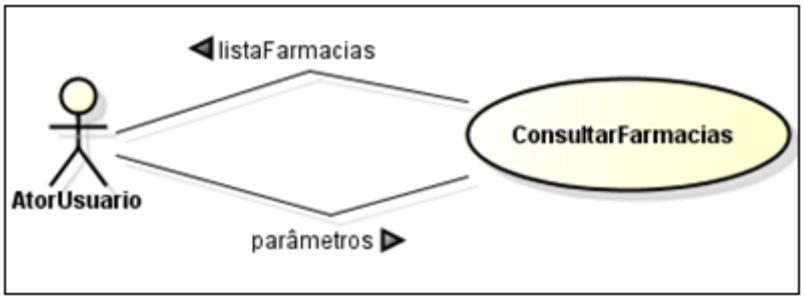


Figura 20 - Caso de Uso ConsultarFarmacias.

Curso Normal:

1. Usuário seleciona a categoria farmácias no aplicativo.
2. Sistema busca lista de farmácias.
3. Aplicativo exibe ao usuário a lista de números de farmácias com opção de discagem.
4. Usuário seleciona um numero.
5. Sistema retorna dados de discagem.

Cursos Alternativos:

3. Sistema não encontra farmácias.
3.1. Sistema retorna objeto nulo.

N**úmero:** 04

Caso de Uso: DiscarNumero

Descrição: Este caso de uso trata da discagem de número.

Ator: Ator Usuário

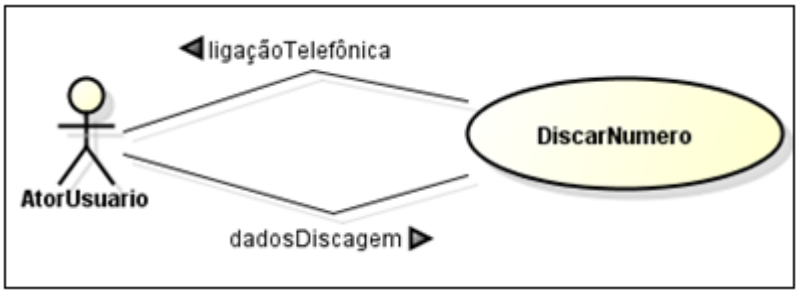


Figura 21 - Caso de Uso DiscarNumero.

Curso Normal:

1. Usuário seleciona um telefone no aplicativo.
2. Sistema disca o número selecionado.
3. Aplicativo retorna ligação para o usuário.

A descrição dos casos de uso assim como o curso normal e alternativo é de grande utilidade para o programador no desenvolvimento de um sistema.

3.1.4 DIAGRAMA DE CLASSES

O diagrama de classe da Figura 22 representa as classes relacionadas às tabelas do banco de dados, essas classes encontram-se no pacote “com.guia.tcc.pojo” no projeto *Android*.

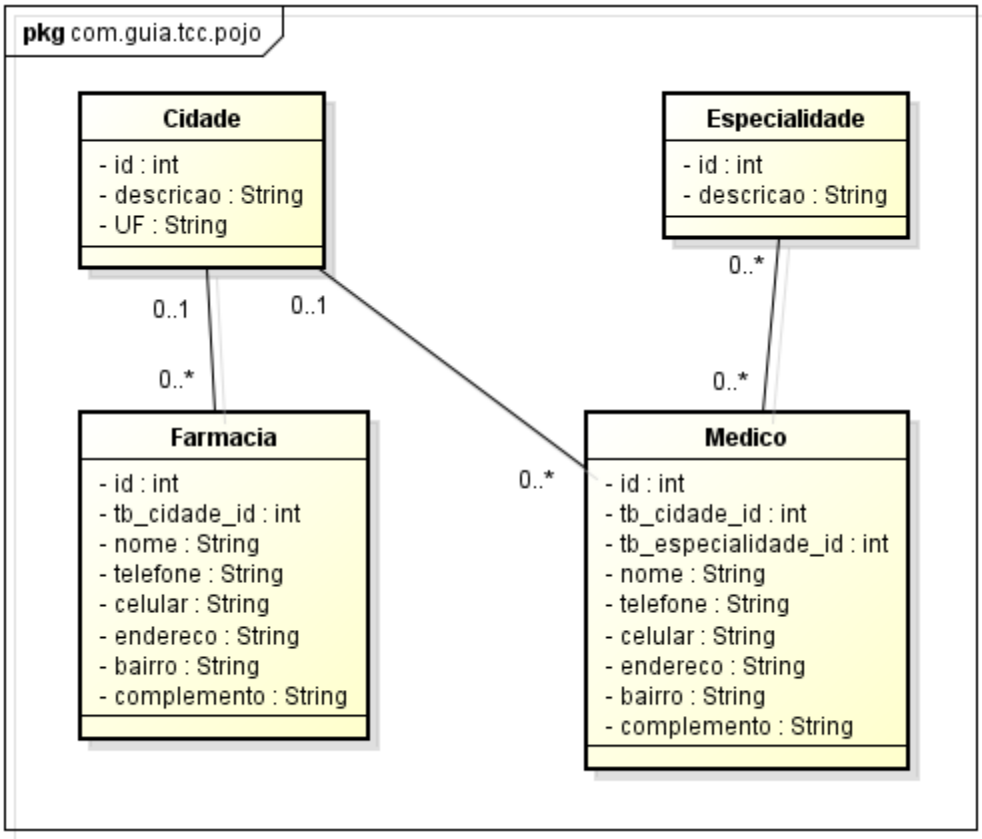


Figura 22 - Diagrama de Classes.

Para o digrama de classes das telas do sistema, foi desenvolvido um diagrama separado.

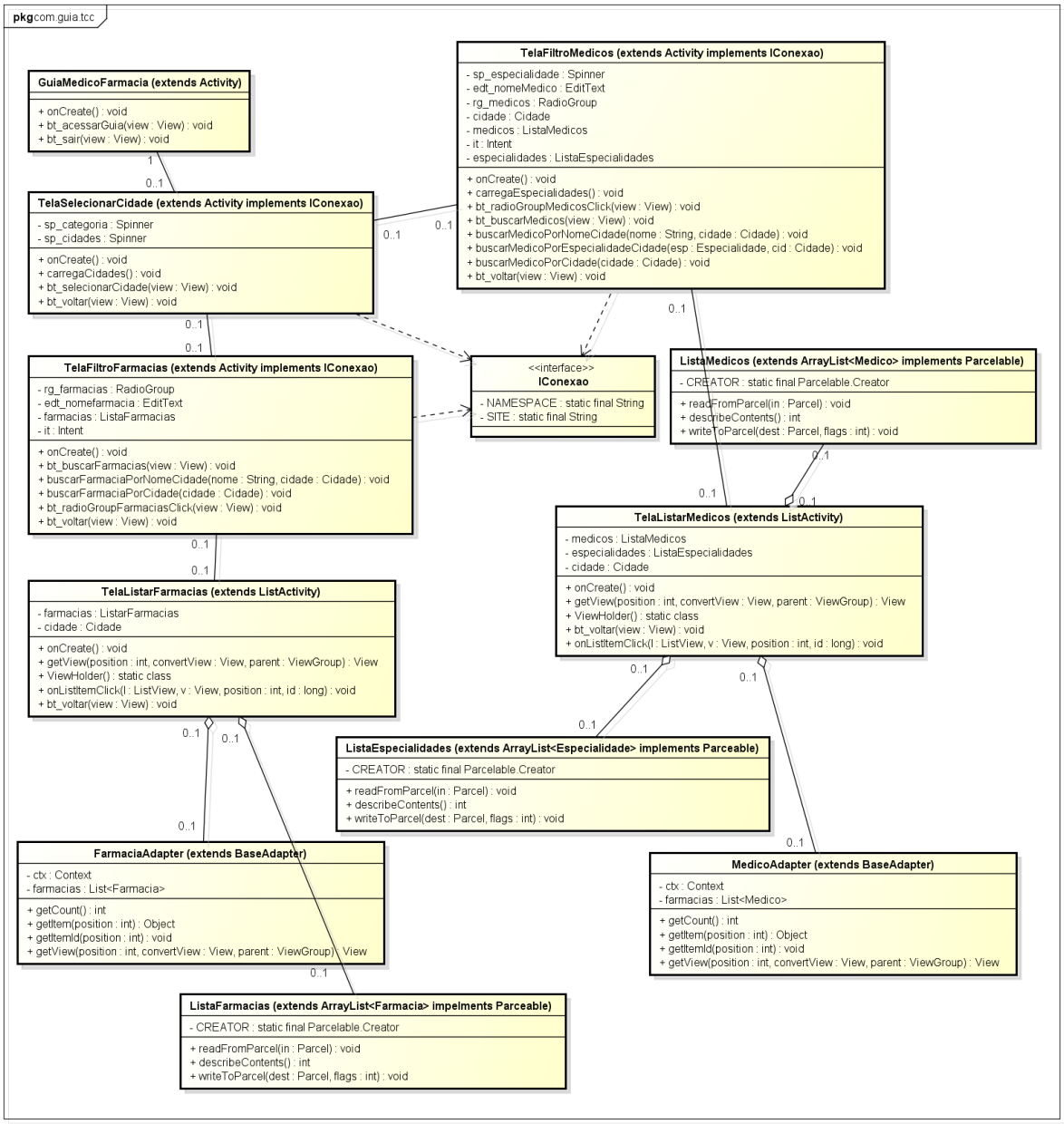


Figura 23 - Diagrama de Classe das Telas do Aplicativo Android.

A Figura 23 mostra o diagrama de classe das telas do aplicativo *Android*, onde se tem as classes que apresentam o aplicativo ao usuário.

3.1.5 DIAGRAMAS DE SEQÜÊNCIAS

Esse título aborda os diagramas de seqüências referentes aos casos de uso do sistema.

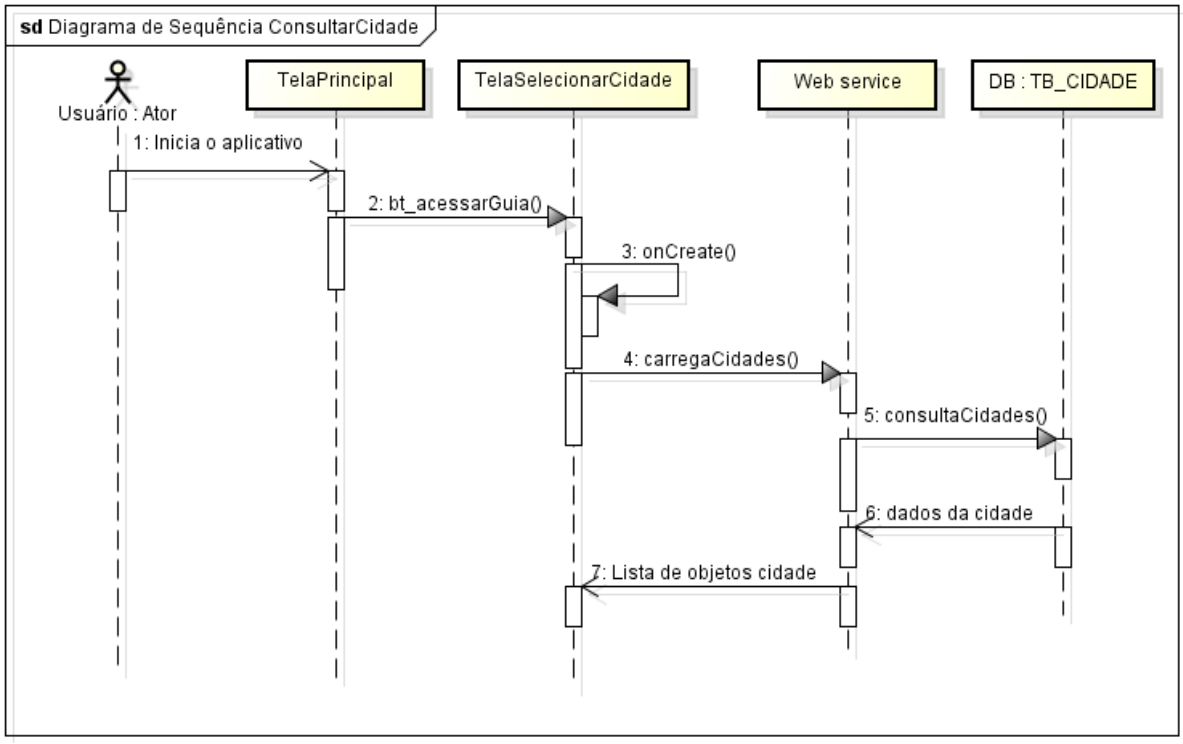


Figura 24 - Diagrama de Seqüência ConsultarCidade.

A Figura 24 mostra o diagrama de seqüência ConsultarCidade, apresentando os métodos de acesso a telas assim como o método de consulta ao *Web service*.

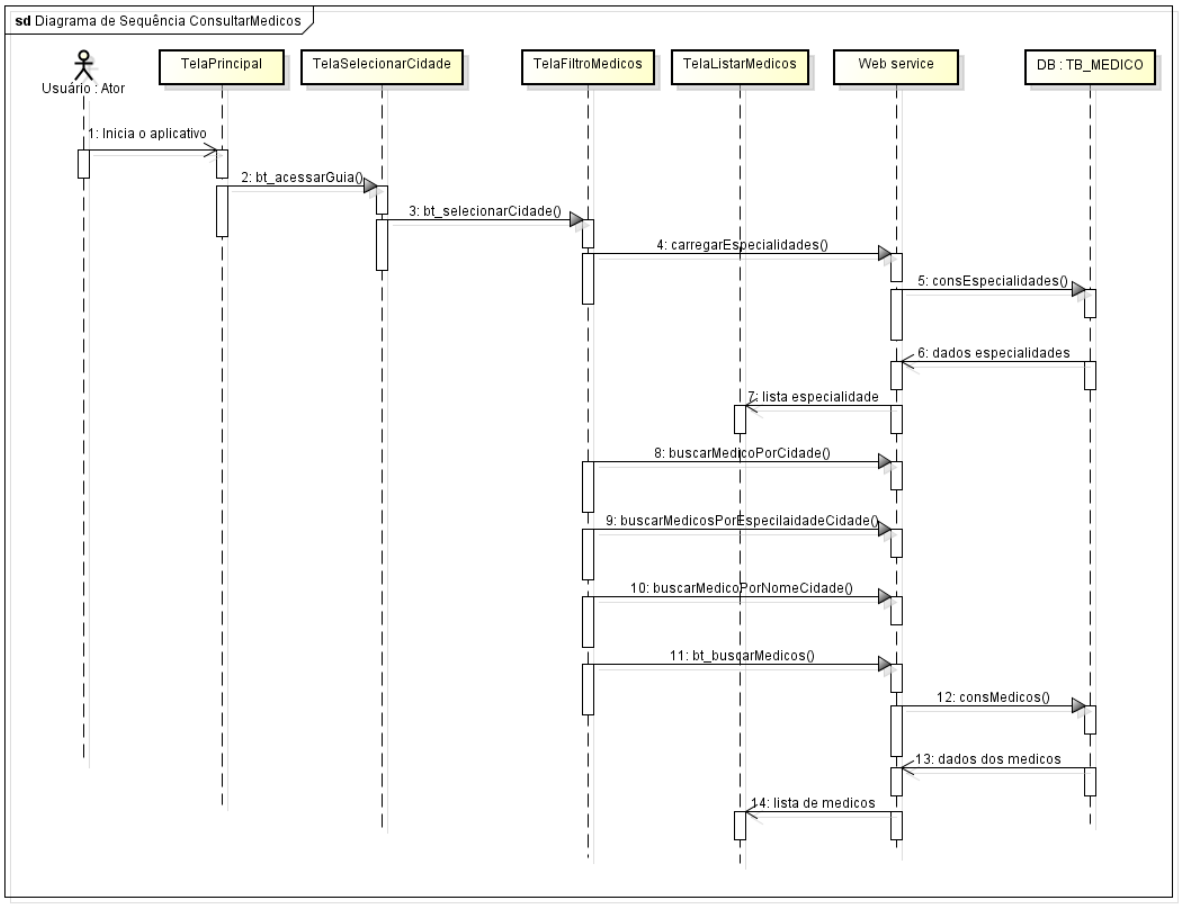


Figura 25 - Diagrama de Seqüência ConsultarMedicos.

A Figura 25 mostra o diagrama de seqüência ConsultarMedicos, apresentando os métodos de consultas por parâmetros, assim como os métodos de acesso a telas.

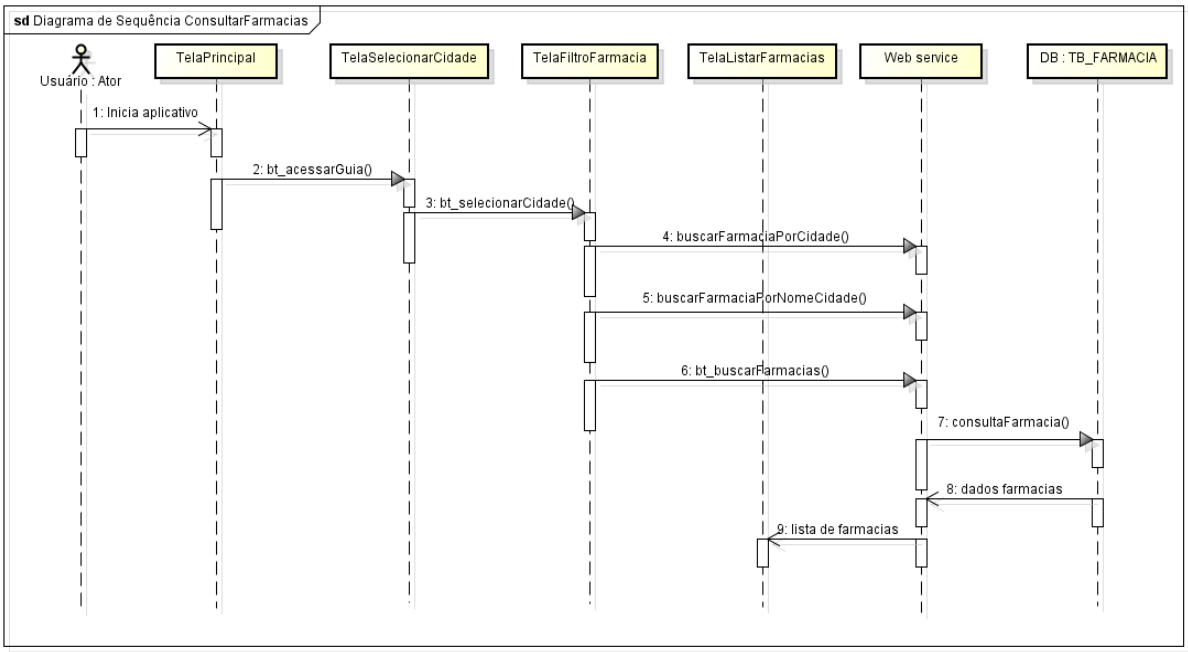


Figura 26 - Diagrama de Seqüência ConsultarFarmacias.

A Figura 26 mostra o diagrama de seqüência ConsultarFarmacia, apresentando os métodos de consultas, assim como métodos de acesso a telas.

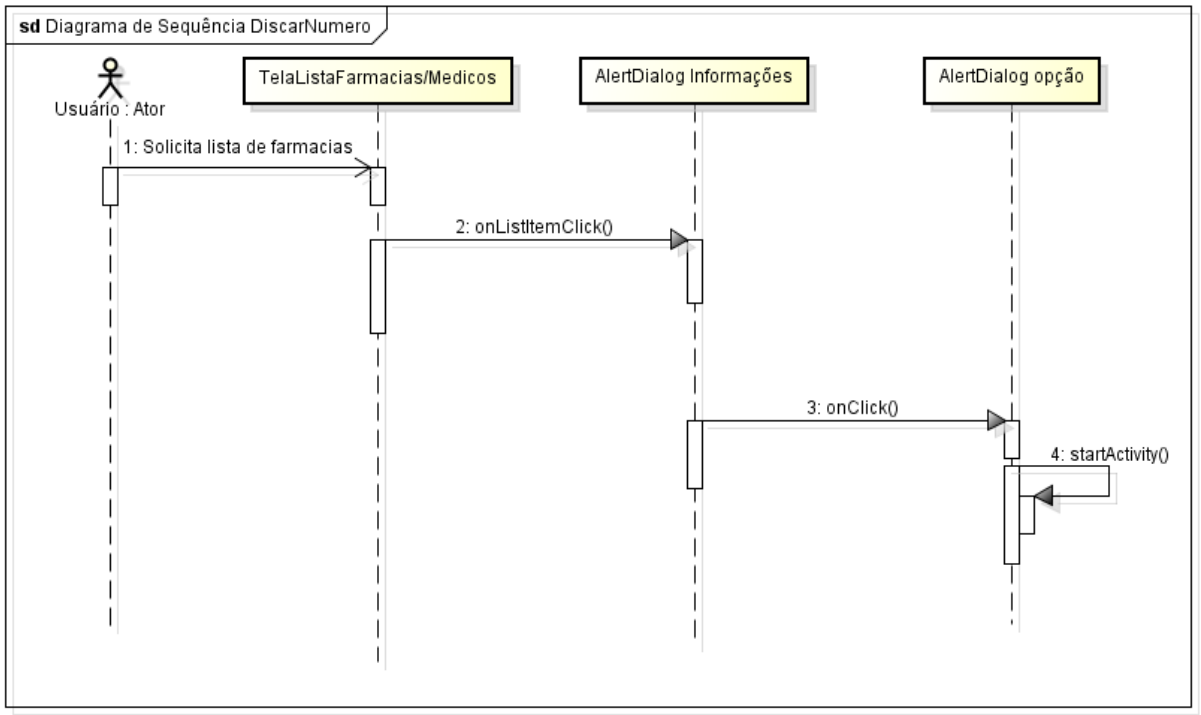


Figura 27 - Diagrama de Seqüência DiscarNumero.

A Figura 27 mostra o diagrama de seqüência DiscarNumero, apresentando os métodos de acesso a telas, assim como os métodos responsáveis pela discagem, este diagrama foi

desenvolvido para apresentar a lógica de discagem quando o usuário solicita discar um número de telefone.

3.2 WEB SERVICE GUIA MÉDICO E FARMÁCIA

Para que o aplicativo GUIA MÉDICO E FARMÁCIA *Android* tenha acesso a um banco de dados disponível na *Web* para consultas, foi desenvolvido um *Web service* contendo operações para o acesso a esses dados. Este *Web service* não contém os métodos de cadastros de dados, para inserção de cidades, especialidades, farmácias e médicos foram utilizados *scripts SQL*. Ao decorrer desta sessão será apresentado este *Web* service.

3.2.1 MODELO DE ENTIDADE E RELACIONAMENTO (M.E.R)

O Modelo de Entidade Relacionamento (MER) é um modelo abstrato cuja finalidade é descrever, de maneira conceitual, os dados a serem utilizados em um sistema. A principal ferramenta do modelo é sua representação gráfica, o diagrama entidade relacionamento. Normalmente o modelo e o diagrama são conhecidos por suas siglas: MER e DER.

Para o desenvolvimento do MER utilizou-se o *Power Architect*, ferramenta livre da SQL *Power Software*, muito utilizada para construção de *Data Warehouse*, possui diversas funcionalidades para desenvolvimento do MER e geração de SQL, torna fácil as ligações entre entidades relacionadas, e de atributos, possuindo opção para exportar o MER em PDF.

O Banco de Dados desenvolvido para o *Web service* conta com apenas cinco tabelas, sendo elas:

- TB_CIDADE – responsável por armazenar o nome das cidades que ficarão disponíveis para consultas. Essa tabela terá ligação com médicos e farmácias.
- TB_ESPECIALIDADES – responsável por armazenas as especialidades médicas disponíveis para consulta.
- TB_MEDICO – responsável por armazenar os dados do médico que será consultado através do sistema *Android*.
- TB_ESP_MED – responsável pela relação entre médico e especialidade, pois um médico pode ter uma ou muitas especialidades, e uma especialidade pode ter vários médicos.
- TB_FARMACIA – responsável por armazenar os dados da farmácia que será consultada através do sistema *Android*.

O MER do Banco de Dados GUIA MÉDICO E FARMÁCIA pode ser visto na figura 28.

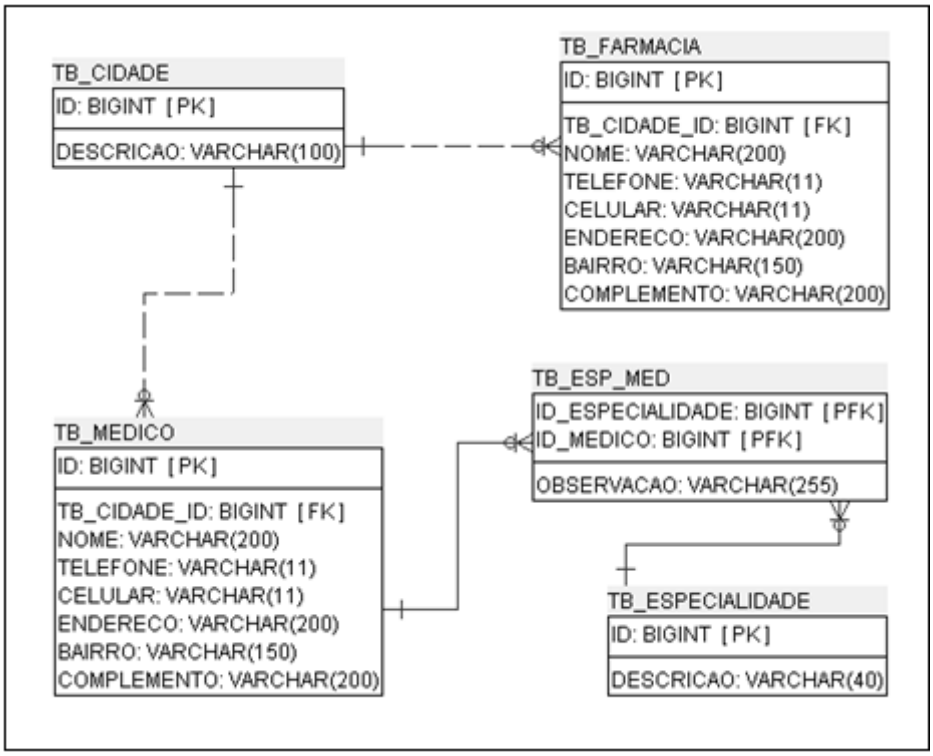


Figura 28 – M.E.R GUIA MÉDICO E FARMÁCIA.

A Figura 28 mostra o MER do banco de dados desenvolvido em MYSQL que é um sistema gerenciador de banco de dados *Open Source*.

4 RESULTADOS E DISCUSSÕES

A partir deste capítulo serão apresentados os resultados e discussões deste trabalho de diplomação.

4.1.1 ESTRUTURA DO PROJETO WEB SERVICE

Para o desenvolvimento do *Web service* GUIA MÉDICO E FARMÁCIA, utilizou-se a IDE *NetBeans* 6.9.1. O *NetBeans* conta com recursos para criação de um *Web service* de forma fácil e rápida.

O primeiro passo foi criar uma aplicação *Web* para rodar em um servidor Tomcat 6.0, o segundo passo foi gerar um serviço *Web* para essa aplicação.

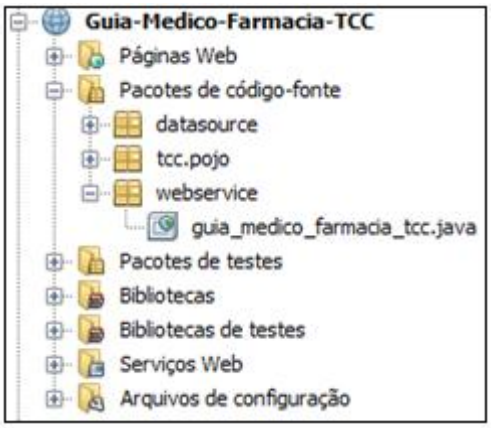


Figura 29 - Estrutura do projeto Web service.

A Figura 29 apresenta a estrutura do projeto *Web service* GUIA MÉDICO E FARMÁCIA criado no *NetBeans*, a classe *Java* "guia_medico_farmacia_tcc.java" é gerada quando é criado o serviço *Web*, essa classe é quem deve conter todas as operações do *Web service*. O pacote `tcc.pojo` contém todas as classe referente a cada tabela do banco de dados. Já o pacote `datasource` contém as classes criadas que são responsáveis por realizar a conexão com o banco de dados MYSQL. Para que isso ocorra foi necessário inserir nas bibliotecas do projeto um arquivo chamado "*mysql-connector-java-5.1.6-bin.jar*" que contém o *drive* de conexão com o banco.

Para ser criado o banco de dados, foi necessário configurar a guia Ferramentas/Servidores para a conexão MYSQL. Após essa guia configurada, foi criado um banco de dados chamado "guia_medico_farmacia" no MYSQL, e foi executado o *script* SQL para gerar as tabelas e inserir dados referente ás tabelas.

Os métodos criados para o acesso aos dados na classe *Java* “guia_medico_farmacia.java” são:

- **listarCidades** – responsável pela listagem de resultado com todas as cidades cadastradas no banco.
- **listarEspecialidades** – responsável pela listagem de resultado com todas as especialidades medicas cadastradas no banco.
- **listarFarmaciasPorCidade** – responsável pela listagem de todas as farmácias de uma determinada cidade, nesse método e passado o ID da cidade como referencia.
- **listarFarmaciasPorNomeCidade** – responsável pela listagem de todas as farmácias de uma determinada cidade com um determinado nome, nesse método e passado o ID da cidade e nome como referência.
- **listarMedicosPorCidade** – responsável pela listagem de todos os médicos de uma determinada cidade, nesse método é passado o ID da cidade como referência.
- **listarMedicosPorNomeCidade** – responsável pela listagem de todos os médicos de uma determinada cidade com um determinado nome, nesse método e passado o ID da cidade e nome como referência.
- **listarMedicosPorEspecialidadeCidade** – responsável pela listagem de todos os médicos de uma determinada cidade com uma determinada especialidade, nesse método e passado o ID da cidade e o ID da especialidade como referência.

Esses métodos ficam disponível na classe Java que é o *Web service.*

```
@WebService()
public class guia_medico_farmacia_tcc {

    static DataSource ds = DataSource.getInstance();
    PreparedStatement pst = null;
    /**
     * Operação de serviço web
     */
    @WebMethod(operationName = "listarCidades")
    public List<Cidade> listarCidades() {
        List<Cidade> cidades = new ArrayList<Cidade>();
        try {
            pst = ds.getConnection().prepareStatement("select * from tb_cidade "
                    + "order by descricao asc");
            ResultSet rs = pst.executeQuery();
            while (rs.next()) {
                if (!rs.getString("descricao").equalsIgnoreCase("")) {
                    Cidade cidade = new Cidade();
                    cidade.setId(rs.getInt("id"));
                    cidade.setDescricao(rs.getString("descricao"));
                    cidades.add(cidade);
                }
            }
            rs.close();
            pst.close();
            return cidades;
        } catch (SQLException e) {
            throw new RuntimeException(e);
        }
    }
```

Figura 30 - Classe guia_medico_farmacia.java.

A Figura 30 mostra a classe *Java* “guia_medico_farmacia.java”, essa classe é anotada com *@WebService*, esta anotação indica que a classe é o *Web service*, a partir dela é gerado o um arquivo WSDL que será o meio de troca de dados entre a aplicação *Android* e o servidor *Web*. Essa classe possui o método chamado listarCidades que vai retornar uma lista de cidades. Esse método é anotado com *@WebMethod* e é passado o nome da operação que será chamado através da WSDL. Através do *DataSource* é executado a consulta ao banco e atribuído os dados retornados a uma lista que será devolvida pelo *Web service* ao cliente que solicitou a consulta.

Para passar parâmetros em uma operação na *Web service* é necessário que o método possua a anotação *@WebParam* passando-se o nome do parâmetro.

```
@WebMethod(operationName = "buscaCidadePorID")
public Cidade buscaCidadePorID(@WebParam(name = "idcidade")
int idcidade) {
    Cidade cidade = new Cidade();
    try {
        pst = ds.getConnection().prepareStatement("select * from tb_cidade "
                + "where id = "+idcidade+"order by descricao asc");
        ResultSet rs = pst.executeQuery();

        while (rs.next()) {
            if (!rs.getString("descricao").equalsIgnoreCase("")) {
                cidade.setId(rs.getInt("id"));
                cidade.setDescricao(rs.getString("descricao"));
            }
        }
        rs.close();
        pst.close();
        return cidade;
    } catch (SQLException e) {
        throw new RuntimeException(e);

    }
}
```

Figura 31 - Método buscarCidadePorID.

A Figura 31 mostra um método que contém como parâmetro o ID de uma cidade, o parâmetro recebido pelo *Web service* é passado a uma variável do tipo inteira. Esse método realiza a consulta de uma determinada cidade e devolve ao cliente um objeto do tipo cidade.

Para que o serviço *Web* fique disponível, é preciso implantá-lo no servidor, dessa forma torna-se possível acessá-lo no dispositivo *Android* através de uma URL.

4.2 APLICATIVO ANDROID GUIA MÉDICO E FARMÁCIA

Ao decorrer desta sessão será apresentado o aplicativo desenvolvido com *Android SDK* no *Eclipse SDK*.

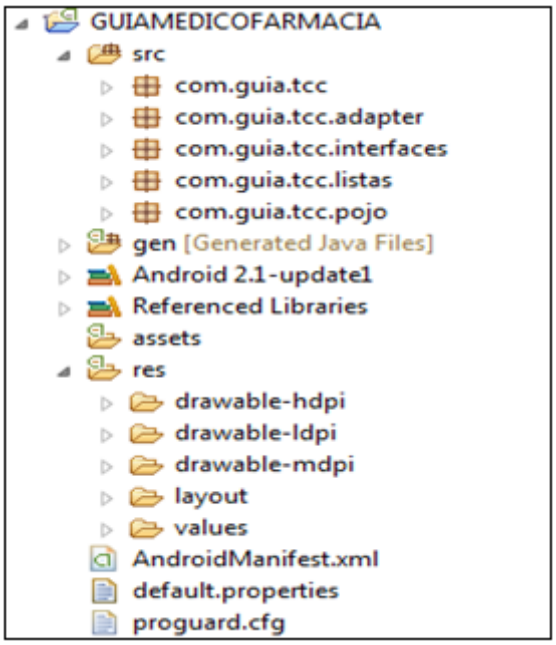


Figura 32 - Estrutura do projeto GUIA MÉDICO E FARMÁCIA Eclipse.

A Figura 32 mostra a estrutura do projeto GUIA MÉDICO E FARMÁCIA para *Android*, a pasta `src/` possui um conjunto de pacotes sendo eles:

- **com.guia.tcc** - pacote que contém as classes *Java* que tratam das telas do aplicativo, essas classes estendem da classe *Activity*.
- **com.guia.tcc.adapter** – pacote que contém as classes *Java* que estendem de *BaseAdapter*, essas classe são responsáveis pelas telas que exibem uma lista de dados no aplicativo.
- **com.guia.tcc.interfaces** – pacote que contém a interface *Java* responsável pela conexão com o *Web service*, todas a *Activity* que possuem métodos de conexão com o *Web service*, implementam essa interface.
- **com.guia.tcc.listas** – pacote que contem as classes *Java* que estendem de um *ArrayList* de objetos, e que implementam *Parcelable*, há telas em que é necessário passar uma lista de objetos para outra tela, nesse caso usa-se essas classes.
- **com.guia.tcc.pojo** – pacote que contem as classes *Java* que representam as tabelas do banco de dados.

A pasta `res/` possui algumas subpasta, sendo elas:

- ***drawable*** – pasta que contem todas as imagens usadas no projeto.
- ***layout*** – pasta que contem todos os arquivos XML responsáveis pelo *layout* das telas do aplicativo, esses arquivos contem as *widgets* utilizadas como: *TexView*, *LynearLayout*, *Button* e etc.

- ***values*** – pasta que contém valores estáticos, *strings* por exemplo, que podem ser carregados de um arquivo XML. Nesse projeto foram utilizados valores estáticos para um *Spinner*.

Para que o sistema tenha métodos e classes responsáveis pela conexão com o *Web service*, foi necessário adicionar as bibliotecas do projeto um arquivo chamado "*ksoap2-android-assembly-2.5.4-jar-with-dependencies.jar*", esse arquivo contém pacotes com um conjunto de classe que fazem todo o trabalho de requisição e resposta ao *Web service*, no decorrer deste trabalho serão apresentado esses métodos.

4.2.1 TELA PRINCIPAL

A tela principal é carregada no momento em que se inicia o aplicativo, esta tela além de ser uma tela de boas vindas é responsável por informar o usuário que o aplicativo fará uso de conexão com a I*nternet* para consulta de dados.

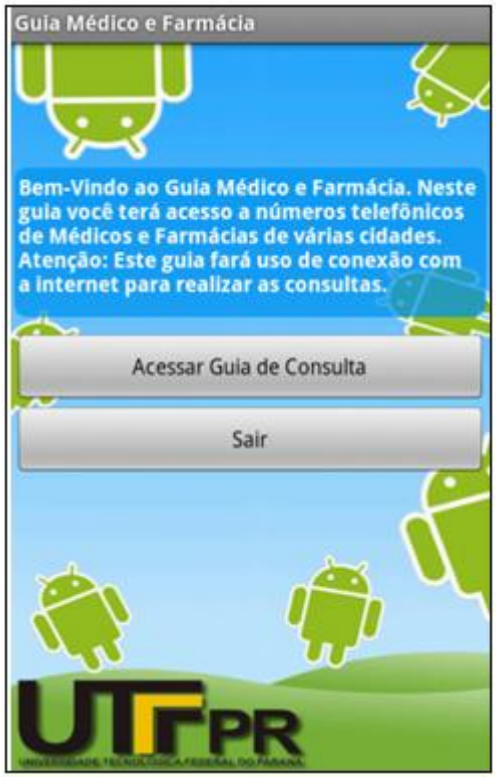


Figura 33 - Tela Principal do aplicativo Android.

A Figura 33 mostra a tela principal do aplicativo GUIA MÉDICO E FARMÁCIA para *Android*. No momento em que o usuário clica no botão "Acessar Guia de Consulta", o aplicativo carrega a tela de seleção de cidades.

4.2.2 TELA SELECIONAR CIDADE

No momento em que a tela “Selecionar Cidade” é criada no aplicativo, ela executa o método *onCreate*, nesse método é feito a chamada ao arquivo XML responsável pelo *layout* da tela, e é recuperado as *widget* desse arquivo.

```
public class TelaSelecionarCidade extends Activity implements IConexao{
    Spinner sp_categorias;
    Spinner sp_cidades;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.tela_selecionar_cidades);
        sp_categorias = (Spinner) findViewById(R.id.sp_categoria);
        sp_cidades = (Spinner)findViewById(R.id.sp_cidades);
        ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(
                this, R.array.categoriaCidades_array,
                android.R.layout.simple_spinner_item);

        adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
        sp_categorias.setAdapter(adapter);
        carregaCidades();
    }
```

Figura 34 - Classe TelaSelecionarCidade.java.

Na Figura 34 foi declarado dois *Spinner*, um para categoria e outro para cidade, já no método *onCreate* foi usado o método *setContentView* para chamar o arquivo XML referente a essa classe. Logo após foram recuperados pelo ID as duas *widget Spinner* do XML e atribuído a os *Spinner* declarados anteriormente.

No código da figura 34 instanciou-se um *ArrayAdapter* onde foi atribuído um *array* estático criado na pasta `values` do projeto no arquivo “*strings.xml*” que contém os itens Farmácias e Médicos.

Logo após foi atribuído o objeto *adapter* ao *Spinner* categoria, e é chamado o método carregarCidades.

O método carregar cidade é responsável por fazer uma chamada a *Web service*, e recuperar todas as cidades cadastradas no banco de dados, e atribuí-las no *Spinner* cidade.

```
public void carregaCidades(){
    try {
        SoapObject envio = new SoapObject(NAMESPACE, "listarCidades");
        SoapSerializationEnvelope envelope = new SoapSerializationEnvelope(
                SoapEnvelope.VER11);
        envelope.setOutputSoapObject(envio);
        HttpTransportSE androidHttpTransport = new HttpTransportSE(SITE);
        androidHttpTransport.call("listarCidades", envelope);
        SoapObject ks =  (SoapObject) envelope.bodyIn;
        ArrayAdapter <Cidade> adapterCidade =
              new ArrayAdapter <Cidade> (this, android.R.layout.simple_spinner_item );
            adapterCidade.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
        for(int i = 0; i<ks.getPropertyCount(); i++){
            SoapObject ob = (SoapObject) ks.getProperty(i);
            Cidade c = new Cidade();
            c.setId(Integer.parseInt(ob.getProperty("id").toString()));
            c.setDescricao(ob.getProperty("descricao").toString());
            adapterCidade.add(c);
        }
        sp_cidades.setAdapter(adapterCidade);
    } catch (Exception e) {
        Log.e("log", e.getMessage());

    }
}
```

Figura 35 - Método carregarCidades.

A Figura 35 mostra o método carregarCidades, onde foi instanciado um objeto *SoapObject* chamado “envio”, e é passado ao construtor o *NAMESPACE* do *Web service*, e o nome da operação que deve ser executada, o *NAMESPACE* tem relação com o nome do pacote criado no projeto *Web service*. Logo após foi instanciado um envelope de transporte chamado “envelope” e foi chamado o método *setOutputSoapObject*, passando-se o objeto ”envio” como parâmetro. Feito isso, foi instanciado um objeto de transporte (*HttpTransportSE*) chamado “*androidHttpTransport”* onde foi passado ao construtor uma variável chamada SITE que é o URL do WDSL do *Web service*. Após isso, foi chamado o método *call* onde é novamente passada a operação e o envelope serializado. Esse envelope vai conter o retorno do *Web service*.

Após a chamada, o retorno que está no envelope *Soap* é atribuído a outro objeto do tipo *SoapObject*.

Para que seja possível exibir esse retorno no *Spinner*, foi necessário instanciar um *ArrayAdapter* do tipo da classe Cidade, e executar um laço de repetição para recuperar cada objeto Cidade e adicioná-lo no *ArrayAdapter*.

Figura 36 - Tela Selecionar Cidade.

A Figura 36 mostra o resultado da tela “Selecionar Cidade”, onde possui um *Spinner* com todas as cidades recuperadas do *Web service*, e possui um *Spinner* com duas categoria estática.

O usuário seleciona uma cidade, e seleciona se a consulta será referente a médicos ou farmácias, dependendo da escolha, no momento em que clicar no botão selecionar, o aplicativo apresentara a tela de filtro de consulta de acordo com a categoria selecionada.

No momento em que o usuário clica em “Selecionar“ é chamado o método de o botão selecionar, esse método tem o nome bt_selecionarCidade.

```
public void bt_selecionarCidade(View view){
    Intent it;
    Cidade cidade = (Cidade) sp_cidades.getSelectedItem();

    if(sp_categorias.getSelectedItemPosition() == 0){
        it = new Intent(this, TelaFiltroFarmacias.class);
    }else{
        it = new Intent(this, TelaFiltroMedicos.class);
    }
    it.putExtra("cidade", cidade);
    startActivity(it);
}
```

Figura 37 - Método do botão bt_selecionarCidade.

A Figura 37 mostra o código do botão bt_selecionarCidade, à *Activity* reconhece esse método como referente ao botão, pois no arquivo XML da tela selecionar cidade é definido na propriedade *android:onClick* o nome do método.

Nesse método foi criada uma variável do tipo *Intent* e foi recuperado um objeto Cidade do *Spinner* sp_cidades, então é testado qual categoria foi selecionada e instancia-se a intenção referente à tela desejada.

O contexto atual foi passado através do "*this*" e o nome da classe *Activity*, logo após é atribuído o objeto cidade a *Intent* através do método putExtra, pois é necessário recuperá-lo na tela seguinte. E por fim foi iniciado a *Activity* passando a *Intent* que contém a classe *Activity* no método *startActivity*.

Uma *Intent* nada mais é do que um tratamento de requisições *Android*, podendo ser uma chamada telefônica, um envio de SMS, uma chamada a uma *Activity*, etc.

4.2.3 TELA DE FILTRO DE FARMÁCIA

Quando o usuário escolhe consultar farmácias em determinada cidade, o aplicativo exibe a tela de "Filtro de Farmácias".

Figura 38 - Tela Filtro de Farmácias.

A Figura 38 mostra a tela de “Filtro de Farmácias”, nessa tela o usuário escolhe se deseja buscar todas as farmácias da cidade selecionada, ou se deseja buscar por nome.

Como visto anteriormente foi utilizado um objeto do tipo *SoapObject* e alguns métodos necessários para passar a informação da operação para o *Web service*, mas no método anterior quando foram buscadas todas as cidades não era passado parâmetro, já nesta tela foram passados dois métodos um para buscar as farmácias da cidade escolhida, e outro para buscar as farmácias por nome e cidade, nesse caso foi necessário também informar ao *SoapObject* os parâmetros da operação.

```
SoapObject envio = new SoapObject(NAMESPACE, "listarFarmaciasPorNomeCidade");
SoapSerializationEnvelope envelope = new SoapSerializationEnvelope(
        SoapEnvelope.VER11);
envio.addProperty("nome", edt_nomefarmacia.getText().toString());
envio.addProperty("idcidade", cidade.getId());
envelope.setOutputSoapObject(envio);
HttpTransportSE androidHttpTransport = new HttpTransportSE(SITE);
androidHttpTransport.call("listarFarmaciasPorNomeCidade", envelope);
SoapObject ks =  (SoapObject) envelope.bodyIn;
```

Figura 39 - Código de chamada a um Web service.

A Figura 39 mostra o código para chamar a operação do *Web service*, note que foram passado dois parâmetros através do método *addProperty* onde é passado o nome do parâmetro, e o seu valor.

Quando realizado a consulta, os dados retornados são atribuídos a uma lista de farmácias, com essa lista formada, então é passado através da *Intent* essa lista para a tela que exibira o resultado, também é necessário passar a cidade através da *Intent*.

4.2.4 TELA FILTRO DE MÉDICO

Quando o usuário escolhe consultar médicos em determinada cidade, o sistema exibe a tela de “Filtro de Médicos”.

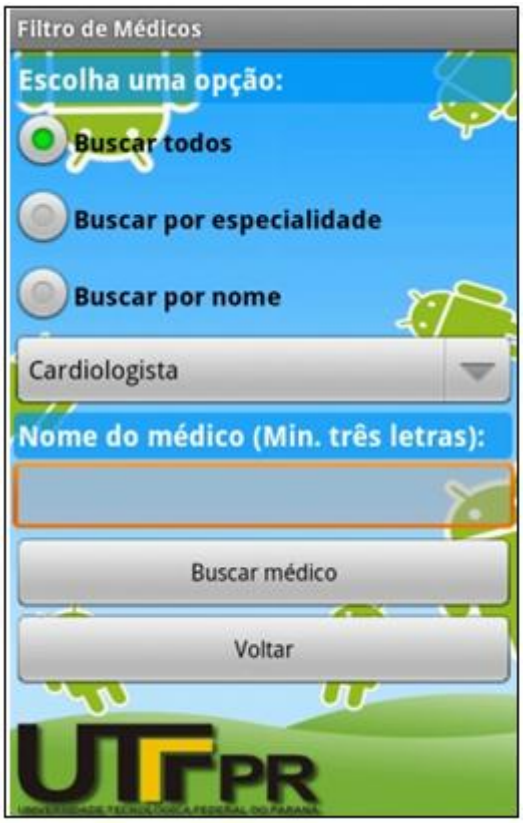


Figura 40 - Tela Filtro de Médicos.

A Figura 40 mostra a tela “Filtro de Médicos”, nessa tela o usuário escolhe se deseja filtrar os médicos da cidade selecionada, podendo buscar todos, buscar por especialidades, ou buscar por nome. No momento em que o aplicativo exibe a tela de filtro é executado um método que traz todas as especialidades médicas cadastrada e o resultado é adicionado no *Spinner*. Os demais métodos seguem o modelo da tela de ”Filtro de Farmácias”, sendo eles; buscarMedicoPorCidade, buscarMedicoPorEspecialida, buscarMedicoPorNomeCidade.

4.2.5 TELA LISTAR FARMÁCIAS

Quando o usuário seleciona o botão “Buscar farmácias” na tela de “Filtro de Farmácias” é exibido à tela “Listar Farmácias”, essa tela apresenta a lista de farmácias que foi passada pela *Intent* da tela “Filtro de Farmácias”.

Figura 41 - Tela Listar Farmácias.

A Figura 41 mostra a tela “Listar Farmácias”, exibindo nome da farmácia e telefone.

4.2.6 TELA LISTAR MÉDICOS

Quando o usuário seleciona o botão “Buscar médicos” na tela de “Filtro de Médicos” é exibido à tela “Listar Médicos”, essa tela apresenta a lista de médicos passada pela *Intent* da tela “Filtro de Médicos”.

Figura 42 - Tela Listar Médicos.

A Figura 42 mostra a tela que “Listar Médicos”, exibindo nome do médico e telefone.

4.2.7 ALERTDIALOG INFORMAÇÕES DAS FARMÁCIAS

Quando o usuário seleciona uma das farmácias listadas na tela “Listar Farmácias”, é exibido um *AlertDialog* com todas as informações referentes à farmácia selecionada, oferecendo a opção de discagem através de um botão chamado “Discar”.

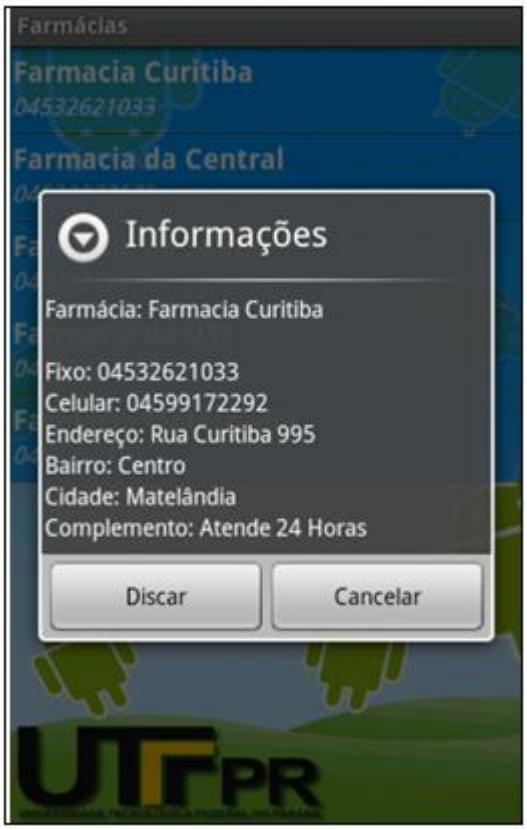


Figura 43 - AlertDialog Informações de Farmácia.

A Figura 43 mostra as informações da farmácia selecionada, oferecendo opção de discar ou cancelar.

4.2.8 ALERTDIALOG INFORMAÇÕES DOS MÉDICOS

Quando o usuário seleciona um dos médicos listados na tela “Listar Médicos”, é exibido um *AlertDialog* com todas as informações referentes ao médico selecionado, oferecendo a opção de discagem através de um botão chamado “Discar”.

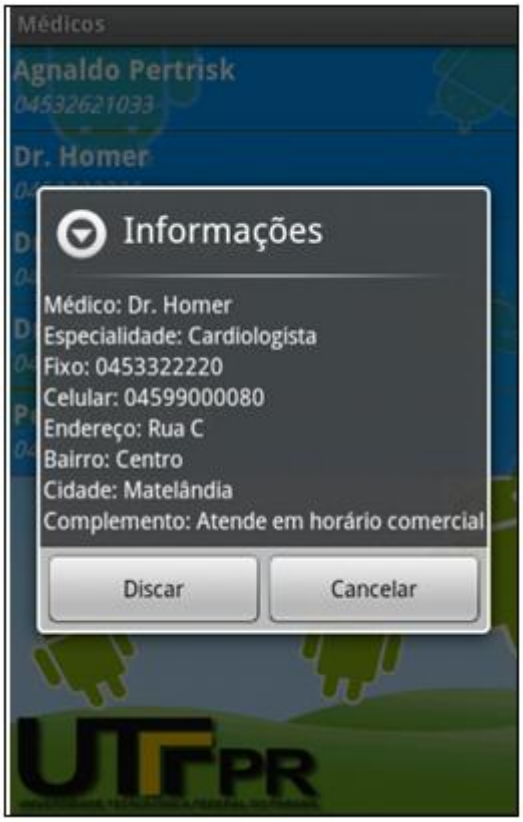


Figura 44 - AlertDialog Informações de Médico.

A Figura 44 mostra as informações do médico selecionado, oferecendo opção de discar ou cancelar.

4.2.9 ALERTDIALOG OPÇÃO DE DISCAGEM

Quando o usuário seleciona “Discar”, sendo na tela das farmácias ou dos médicos, o aplicativo exibe um *AlertDialog* pedindo para selecionar se o usuário deseja discar para o telefone fixo ou para o celular. Pois há dois telefones cadastrados.

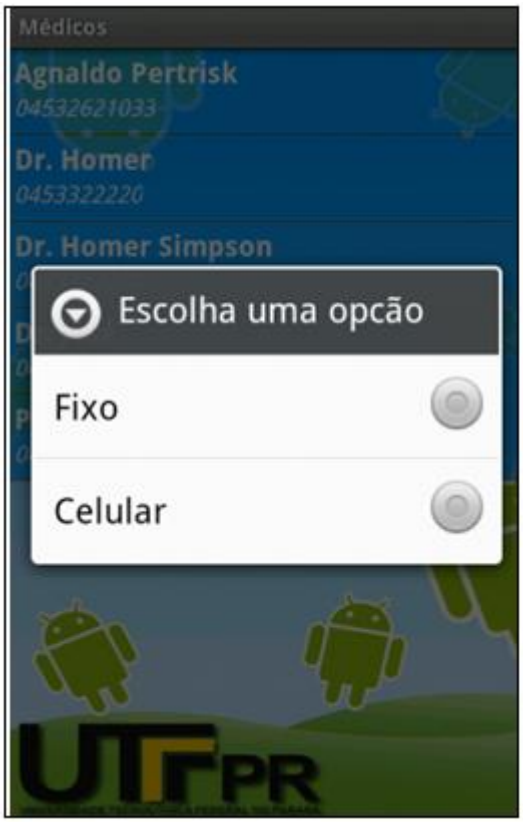


Figura 45 - AlertDialog Opção de Discagem.

A Figura 45 mostra a tela de opções de discagem, quando o usuário seleciona uma das opções é iniciada uma chamada telefônica para o telefone selecionado.

```
Intent dial = null;
if (items[item] == "Fone") {
    dial = new Intent(
            android.content.Intent.ACTION_CALL,
            Uri.parse("tel: "
                    + farmacia
                            .getTelefone()));
} else {
    dial = new Intent(
            android.content.Intent.ACTION_CALL,
            Uri.parse("tel: "
                    + farmacia
                            .getCelular()));
}
startActivity(Intent.createChooser(
        dial, "Discando...."));
```

Figura 46 - Código do evento onClick do AlertDialog Opção de Discagem.

A Figura 46 mostra o código do evento *onClick* do *AlertDialog* de opção de discagem, foi declarado uma *Intent* e testado qual das opções é selecionada, em seguida instancia-se a *Intent* passando ao construtor uma *ACTION_CALL* que representa uma chamada telefônica e uma classe chamada Uri com o método *parse* onde é passado o número desejado. Em seguida inicia-se a *Intent* através do método *startActivity*.

4.2.10 ANDROID MANIFEST

Para que o dispositivo tenha permissão de acesso à *Internet*, chamadas telefônicas, e até mesmo as *Activity* do sistema, é necessário configurar algumas linhas no arquivo "*AndroidManifest.xml*".

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.guia.tcc"
      android:versionCode="1"
      android:versionName="1.0">
    <uses-sdk android:minSdkVersion="7" />
    <uses-permission android:name="android.permission.INTERNET"></uses-permission>
    <uses-permission android:name="android.permission.CALL_PHONE"></uses-permission>
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".GuiaMedicoFarmacia"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".TelaSelecionarCidade" android:label="Selecionar Cidade" />
        <activity android:name=".TelaFiltroFarmacias" android:label="Filtro de Farmácias" />
        <activity android:name=".TelaFiltroMedicos" android:label="Filtro de Médicos" />
        <activity android:name=".TelaListarFarmacias" android:label="Farmácias" />
        <activity android:name=".TelaListarMedicos" android:label="Médicos" />
    </application>
</manifest>
```

Figura 47 - Arquivo AndroidManifest.xml.

A Figura 47 mostra o arquivo "*AndroidManifest.xml*", nele são configurado tags de permissões a *Internet*, e chamadas telefônicas, assim como as *Activity* do projeto.

5 CONSIDERAÇÕES FINAIS

Ao decorrer deste capítulo serão abordadas as considerações finais deste Trabalho de Diplomação.

5.1 CONCLUSÃO

Através das pesquisas realizadas e o desenvolvimento de um exemplo prático de uso de integração entre dispositivos *Android* e *Web services*, foi possível concluir que ambos oferecem suporte para desenvolvimento de novas aplicações para o mercado de usuários, trazendo mais praticidade.

O *Android SDK* por ser uma ferramenta livre e oferecer vários recursos ao desenvolvedor vem tornando mais prático e ágil o desenvolvimento de aplicativos para dispositivos móveis. Além disso, o desenvolvedor pode contar com o *Android Market*, onde poderá publicar suas aplicações para o mundo, de uma maneira rápida, segura e eficiente.

A integração entre *Android* e *Web service* traz agilidade ao processo de comunicação entre um dispositivo móvel e um servidor *Web*. Com isso é possível atender tanto as necessidades de um usuário comum, como é possível desenvolver aplicativos corporativos onde há necessidade de interagir um aplicativo móvel com o banco de dados de uma empresa.

Por fim, o mercado de aplicativos para dispositivos móveis que fazem uso do sistema operacional *Android* é grande e promissor, e essa possibilidade de integração com um serviço *Web* surge para ajudar nos aspectos práticos inovadores.

5.2 TRABALHOS FUTUROS/CONTINUAÇÃO DO TRABALHO

Essa monografia sugere uma pesquisa mais aprofundada nos recursos do *Android SDK*, podendo ser desenvolvida uma nova versão do aplicativo de estudo de caso. Além disso, o mercado de trabalho nessa área de desenvolvimento é muito promissor.

REFERÊNCIAS BIBLIOGRÁFICAS

ANDROID DEVELOPER. **Package Widget.** Disponível em <http://developer.android.com/reference/android/widget/package-summary.html>. Acesso em 19/05/2011.

ANDROID DEVELOPER. **What is Android?**.Disponível em <http://developer.android.com/guide/index.html>. Acesso em 10/03/2011.

ANDROID MARKET. **Publish**. Disponível em <https://market.android.com/publish>. Acessado em 19/05/2011.

ANDROIDZ. **Android Market cresce 364% em um ano, veja as estatísticas**. Disponível em <http://www.androidz.com.br/portal/index.php?option=com_content&view=article&id=251:android-market-cresce-364-em-um-ano&catid=38:destaques&Itemid=101>. Acesso em 23/05/2011.

ASTAH. **Astah Community - FREE UML Modeling Tool.** Disponível em <http://astah.change-vision.com/en/product/astah-community.html>. Acesso em 25/05/2011.

EUANDROID. **Criando Widget Spinner.** 2011. Disponível em. <http://www.euandroid.com.br/tutoriais/tutorial-dev/2010/11/criando-widget-spinner/>. Acesso em 25/05/2011.

FELIPE SILVEIRA. **Criando uma Activity secundária.** Disponível em <http://www.felipesilveira.com.br/2010/05/criando-uma-activity-secundaria/>. Acesso em 25/05/2011.

GOOGLE Code. **Ksoap2 Android.** Disponível em <http://code.google.com/p/ksoap2-android/>. Acesso em 05/05/2011.

IDG NOW, NOYES KATHERINE. **Android cresce com velocidade maior que o imaginado, aponta pesquisas, 2011**. Disponível em

<http://idgnow.uol.com.br/mercado/2011/03/01/android-cresce-com-velocidade-maior-que-o-imaginado-apontam-pesquisas/>. Acesso em 10/03/2011.

IT WEB. **Android cresce mais de 880%.** Disponível em

<http://www.itweb.com.br/noticias/index.asp?cod=75925>. Acesso em 10/03/2011.

MACLEAN, Dave. **Programação para Android**. Linux Magazine. v.75. p.38 - 44, Fevereiro de 2011.

MACORATTI. **UML - Conceitos Básicos II.** Disponível em

<http://www.macoratti.net/vb_uml2.htm>. Acesso em 31/05/2011.

MEUPOST. **Visão Como conectar o Java ao MySQL usando o NetBeans**. Disponível em <http://www.meupost.com/2008/11/25/como-conectar-o-java-ao-mysql-usando-o-netbeans/>. Acesso em 20/05/2011.

MYSQL. **Visão Geral do Sistema de Gerenciamento de Banco de Dados MySQL.** Disponível em <http://dev.mysql.com/doc/refman/4.1/pt/what-is.html>. Acesso em 20/05/2011.

NETBEANS. **Conexão a um banco de dados MySQL**. Disponível em <http://netbeans.org/kb/docs/ide/mysql_pt_BR.html>. Acesso em 20/04/2011.

NGLAUBER. **Adapter Eficiente no Android.** Disponível em

<http://nglauber.blogspot.com/2011/03/adapter-eficiente-no-android.html>. Acesso em 10/03/2011.

POTTS, Kopack, Mike. **Aprenda Web Services em 24 horas**. Tradução de Marcos Vieira. Ed. Campus, 2003.

TEC MUNDO. **O que é XML?.** Disponível em <http://www.tecmundo.com.br/1762-o-que-e-xml-.htm>. Acesso em 17/05/2011.

BOOCH, Grady, RUMBAUGH, James, IVAR, Jacobson . **UML, guia do usuário**. Tradução de Fabio Freitas da Silva. Ed. Campus, 2000.

UOL ECONOMIA. **Android é o sistema operacional para smartphones mais popular nos EUA**. Disponível em <http://economia.uol.com.br/ultimas-noticias/efe/2011/03/03/android-e-o-sistema-operacional-para-smartphones-mais-popular-nos-eua.jhtm>. Acesso em 02/04/2011.

WIKIPÉDIA. **Android**. Disponível em <http://pt.wikipedia.org/wiki/Android>. Acesso em 10/03/2011.

ZEBRA. **Consumindo Web Service em aplicações Android**. Disponível em <http://zbra.com.br/2011/03/30/consumindo-web-service-em-aplicacoes-android/>. Acesso em 10/03/2011.