CENTRO UNIVERSITÁRIO UNIVATES
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
CURSO DE ENGENHARIA DA COMPUTAÇÃO

FERNANDO JOSÉ BOHRER

SERVIÇO DE GEOLOCALIZAÇÃO PARA PLATAFORMA ANDROID

Lajeado
2011

FERNANDO JOSÉ BOHRER

SERVIÇO DE GEOLOCALIZAÇÃO PARA PLATAFORMA ANDROID

Trabalho de Conclusão de Curso apresentado ao Centro de Ciências Exatas e Tecnológicas do Centro Universitário UNIVATES, como parte dos requisitos para a obtenção do título de bacharel em Engenharia da Computação.
Área de concentração: Computação

ORIENTADOR: Fabrício Pretto

Lajeado

2011

FERNANDO JOSÉ BOHRER

SERVIÇO DE GEOLOCALIZAÇÃO PARA PLATAFORMA ANDROID

Este trabalho foi julgado adequado para a obtenção do título de bacharel em Engenharia da Computação do CETEC e aprovado em sua forma final pelo Orientador e pela Banca Examinadora.

Orientador: _____

Prof. Fabrício Pretto, UNIVATES

Mestre pela PUCRS – Porto Alegre, Brasil

Banca Examinadora:

Prof. Alexandre Stürmer Wolf, UNIVATES

Mestre pela PUC-Rio – Rio de Janeiro, Brasil

Prof. Maglan Cristiano Diemer, UNIVATES

Mestre pela UNISINOS – São Leopoldo, Brasil

Coordenador do curso de Engenharia da Computação: _____

Prof. Marcelo de Gomensoro Malheiros

Lajeado, julho de 2011.

"Mas se ainda hoje nossos semelhantes marcham por um pedaço de chão para sobreviver, e alienam seus instintos mais criativos em busca de algum reconhecimento dentro de uma esmagadora cultura de consumo auto destrutivo, nos deparamos com a questão: qual o papel que nós aqui já alimentados e abrigados temos em pensar numa soberania e transmissão de conhecimentos que buscam reverter esta pulsão auto destrutiva da humanidade?"
Movimento dos sem satélite

RESUMO

O mercado de telefonia móvel tem crescido de forma considerável nos últimos anos e a procura por dispositivos que agreguem cada vez maior número de funcionalidades tem feito com que a comercialização de *smartphones* aumente. De todas as plataformas de software executadas por *smartphones*, a que tem apresentado o crescimento de mercado mais significativo é a plataforma Android, uma plataforma construída a partir de um consórcio de empresas que criou a *Open Handset Alliance*. Com a crescente interação entre usuários e seus equipamentos e com a presença destes cada vez mais no cotidiano das pessoas, é cada vez maior o volume de informações que são armazenadas nestes dispositivos. A eventual perda de um dispositivo permite a terceiros terem acesso às informações armazenadas no aparelho e esse acesso pode trazer prejuízos pessoais e financeiros. O presente trabalho apresenta uma proposta de desenvolvimento de um sistema que permite geolocalizar e bloquear, remotamente, *smartphones* que executam a plataforma Android.

Palavras-chaves: Android, Aplicativos Móveis, Geolocalização, *Smartphone*.

ABSTRACT

The mobile market has grown considerably in recent years and demand for devices that add an increasing number of features has made the commercialization of smartphones increase. Among all software platforms running on smartphones, the one which has presented the most significant market growth is the Android platform, a platform built by a consortium of companies that created the Open Handset Alliance. With the increasing interaction between users and their equipments and the increasing presence of these devices in people's daily lives, the amount of information that is stored on these devices has considerably grown. The possible loss of a device allows third parties to gain access to information stored on this device and this access can bring personal and financial losses. This paper presents a proposal to develop a system that allows geotag and block, through a website, smart phones running the Android platform.

Keywords: Android, Mobile Applications, Geolocation, Smartphone.

SUMÁRIO

1 INTRODUÇÃO....12
1.1 Objetivos....13
1.2 Estrutura do trabalho....13
2 TELEFONIA MÓVEL....15
2.1 Principais plataformas de software de telefonia móvel....16
2.2 Cenário atual do mercado de dispositivos móveis de telefonia....17
3 PLATAFORMA ANDROID....20
3.1 Arquitetura da plataforma Android....20
3.1.1 Kernel do Linux....21
3.1.2 Bibliotecas....22
3.1.3 Android Runtime....22
3.1.4 Framework de aplicações....23
3.1.5 Aplicações....24
3.2 Componentes de uma aplicação Android....24
3.3 Ciclo de vida de Activities na plataforma Android....25
3.4 Ciclo de vida de Services na plataforma Android....27
3.5 Reutilização e substituição de aplicativos na plataforma Android....28
3.6 Gerenciamento de aplicações Android....30
3.7 Desenvolvimento de aplicativos para a plataforma Android....32
4 DESENVOLVIMENTO DO APLICATIVO BLOCK MY ANDROID....34
4.1 Funcionamento do Block my Android....34
4.2 Agendamento e execução do Service....35
4.3 Configuração do sistema BMA....40
4.4 Modo administrativo....42
4.5 Comunicação com o servidor da aplicação....46
4.6 Geolocalização....50
4.7 Troca de senha e bloqueio do dispositivo....53
4.8 Activity principal do sistema BMA....55
4.9 Trabalhos futuros....56
4.9.1 Implementação de solicitação de senha para alteração das configurações....56
4.9.2 Comunicação de dados entre o dispositivo móvel e o servidor da aplicação realizada de forma criptografada....56
4.9.3 Desenvolvimento de um portal web....56
4.9.4 Criação de contas através do próprio aplicativo....57
5 CONSIDERAÇÕES FINAIS....58

LISTA DE FIGURAS

Figura 1 Principais componentes do sistema operacional Android (ANDROID DOCUMENTATION, 2010).....21
Figura 2 Ciclo de vida de uma Activity na plataforma Android (MEIKE, 2009).....26
Figura 3: Interface da IDE Eclipse para adição de permissões em projetos Android.….36
Figura 4: Permissões do aplicativo BMA.....37
Figura 5: Activity de configuração do sistema BMA.....40
Figura 6: Dialog exibido enquanto a configuração do sistema BMA não for realizada…42
Figura 7: Dialog exibido enquanto o modo administrativo do sistema BMA não for habilitado.....43
Figura 8: Tela de confirmação da habilitação do modo administrativo.....44
Figura 9: Tela informando o insucesso na tentativa de remoção do BMA.....46
Figura 10: Notificação informando falha na tentativa de login.....47
Figura 11: Tela de autenticação para desbloqueio do dispositivo.....54
Figura 12: Activity principal do sistema BMA.....55

LISTA DE CÓDIGOS

Listagem 1 Solicitação das permissões através do arquivo AndroidManifest.xml.....36
Listagem 2 Mapeamento entre o final do processo de inicialização do dispositivo e a execução da classe BMAScheduleService.....38
Listagem 3 Classe BMAScheduleService instanciando o agendamento da execução do Service.....38
Listagem 4 Mapeamento entre o Intent "BLOCKMYANDROID" e a execução da classe BMAService.....39
Listagem 5 Reagendamento da execução do Service.....39
Listagem 6 Estrutura do arquivo xml com as permissões habilitadas pelo modo administrativo.....45
Listagem 7 Location Listener do serviço de localização geográfica do dispositivo.....51
Listagem 8 Método de obtenção da localização geográfica atual do dispositivo.....52
Listagem 9 Método de alteração da senha e bloqueio do dispositivo.....54

LISTA DE TABELAS

Tabela 1 Principais plataformas de telefonia móvel.....16
Tabela 2 Vendas mundiais de aparelhos celulares, em milhares de unidades, por fabricante, no terceiro trimestre de 2010.....18
Tabela 3 Vendas mundiais de smartphones, em milhares de unidades, por plataforma, no terceiro trimestre de 2010.....18
Tabela 4 Os quatro componentes básicos das aplicações Android.....24
Tabela 5 Descrição dos eventos utilizados no desenvolvimento de Activities.....27
Tabela 6 Descrição dos eventos utilizados no desenvolvimento de Services.....27
Tabela 7 Possíveis status do smartphone, seus códigos e as ações que estes desencadeiam. 48

LISTA DE ABREVIATURAS

2D: Duas dimensões

2G: Segunda geração de padrões e tecnologias de telefonia móvel

3D: Três dimensões

3G: Terceira geração de padrões e tecnologias de telefonia móvel

API: *Application Programming Interface*

BMA: *Block my Android*

CPU: *Central Processing Unit*

GPS: *Global Positioning System*

I/O: *Input/Output*

IDE: *Integrated Development Environment*

J2ME: *Java 2 Micro Edition*

J2SE: *Java 2 Standard Edition*

PC: *Personal Computer*

PIN: *Personal Identification Number*

RAM: *Random Access Memory*

SDK: *Software Development Kit*

SIM: *Subscriber Identity Module*

TCP: *Transmission Control Protocol*

URI: *Uniform Resource Identifier*

Wi-Fi: *Wireless Fidelity*

1 INTRODUÇÃO

O mercado de telefonia móvel não para de crescer. De acordo com TOTAL TELECOM (2011), em maio do ano de 2011 existiam no mundo um total de 5.6 bilhões de assinaturas a serviços móveis de comunicação. Esse montante é reflexo da disponibilidade de acesso às redes móveis de telefonia que, atualmente, atinge 90% da população mundial.

Considerando a estimativa do número de assinaturas a serviços móveis de comunicação existentes e a estimativa da população mundial que, segundo CIA (2011) era de 6.92 bilhões de pessoas em julho de 2011, é possível dizer que aproximadamente 81% da população mundial tem acesso à algum dispositivo móvel de telefonia.

Segundo ITU (2010), as pessoas estão migrando rapidamente da tecnologia 2G (segunda geração de padrões e tecnologias de telefonia móvel) para a 3G, tanto nos países desenvolvidos bem como nos considerados em desenvolvimento. Em 2010, 143 países ofereciam comercialmente serviços 3G contra 95 em 2007. Pelos números apresentados percebe-se a tendência cada vez maior da migração da tecnologia 2G para a 3G.

De acordo com STEVENS (2008), o número de PCs (*Personal Computer*) instalados no mundo atingiu 1 bilhão de unidades em 2008 e sua taxa de crescimento é de 12% ao ano. Com base nesses dados, é possível estimar que no ano de 2011 existam aproximadamente 1.40 bilhões de PCs em uso no mundo.

Considerando esses dados (aproximadamente 5.6 bilhões de dispositivos móveis de comunicação e aproximadamente 1.40 bilhões de PCs), pode se estimar que no ano de 2011 o mercado de dispositivos móveis de comunicação é 4 vezes maior do que o mercado de PCs.

De acordo com ZHENG (2006), a próxima geração da computação móvel promoverá a convergência da computação, comunicação e eletrônica que são respectivamente três indústrias tradicionalmente distintas e de interoperabilidade baixa. Como responsável por essa mudança estará um dispositivo inteligente que se tornará um terminal móvel universal.

Segundo LECHETA (2010) cada vez mais os usuários do sistema de telefonia móvel procuram por dispositivos que possuam recursos como suporte à áudio e vídeo, *bluetooth*, câmera, jogos, GPS (*Global Positioning System*) e acesso à Internet. Ainda segundo o mesmo autor, o crescimento do mercado corporativo tem feito com que empresas busquem incorporar aplicações móveis em seu cotidiano tendo como objetivo agilizar seus processos.

A procura cada vez maior por dispositivos mais robustos e com maior número de funcionalidades tem feito com que os *smartphones* sejam vistos como uma alternativa interessante e que promete mudar a forma como a computação é vista.

De acordo com PCMAGAZINE (2010), um *smartphone* é um telefone celular com aplicações integradas e acesso à Internet que, além das suas funcionalidades internas, pode executar múltiplas aplicações.

As facilidades propiciadas pela computação em nuvem fazem com que o acesso às informações fique cada vez mais fácil e em função disso dispositivos como *smartphones* se transformam em verdadeiros terminais de acesso através dos quais os usuários realizam suas operações.

Enquanto que por um lado se ganha com a maior mobilidade e flexibilidade do modelo de negócios, por outro, os problemas começam quando se fala em situações como perda e/ou roubo de dispositivos. Dispositivos perdidos e/ou roubados podem permitir o acesso, a pessoas não autorizadas, a informações armazenadas na nuvem e também às informações armazenadas no próprio dispositivo. Situações desse tipo podem trazer incontáveis prejuízos pessoais e financeiros.

Dessa forma, considerando o tamanho do mercado de dispositivos móveis de comunicação, o crescimento da procura por *smartphones*, as facilidades de acesso que estes propiciam e o risco eminente da perda de dados e do vazamento de informações, fica claro que um sistema que permita bloquear o acesso ao *smartphone* à distância tem grande potencial de diminuir a probabilidade das informações serem de fato acessadas e o vazamento das mesmas, de fato consumado.

1.1 Objetivos

Esse trabalho tem por objetivo desenvolver um aplicativo que permita o bloqueio e a geolocalização de *smartphones* baseados na plataforma Android.

A intenção é que o usuário, ao perder o acesso ao seu *smartphone*, possa autenticar em um *site* e através deste, enviar ao aparelho instruções que bloqueiem o dispositivo e que ativem serviços de geolocalização.

O desenvolvimento do aplicativo que será instalado nos *smartphones* é parte integrante do escopo desse trabalho. O desenvolvimento do sistema que executa no *site* não é parte integrante do escopo desse trabalho.

1.2 Estrutura do trabalho

O presente trabalho tem divisões estabelecidas na forma de capítulos, sendo, após essa introdução, apresentada a fundamentação teórica. O segundo capítulo expõe sobre a realidade

atual da telefonia móvel no mundo e o terceiro especificamente sobre a plataforma Android. Em ambos são definidos os principais conceitos referenciados durante o texto. No quarto capítulo está detalhado o desenvolvimento da aplicação e se pode identificar a forma através da qual foram atingidos os objetivos desse trabalho. No quinto e último capítulo estão detalhadas as considerações finais.

2 TELEFONIA MÓVEL

O mercado de telefonia móvel mudou consideravelmente nos últimos anos trazendo consigo uma nova realidade na forma através da qual as pessoas se comunicam. Se antes dispositivos móveis como telefones celulares serviam apenas para realizar ligações de voz para com outros telefones, hoje, cada vez mais através dos *smartphones*, o número de funcionalidades disponíveis em um único aparelho não param de crescer. Esse aumento do número de funcionalidades aliado às novas formas de comunicação fez com que a importância que os dispositivos móveis de comunicação têm hoje na vida cotidiana das pessoas seja muito maior do que foi no passado.

Com a importância desses dispositivos aumentando, a procura pelos mesmos cresceu, aquecendo o mercado. Enquanto que por um lado se beneficiam os consumidores que, ao adquirirem um *smartphone* passam a ter acesso às novas funcionalidades e formas de comunicação, por outro lado se beneficiam os fabricantes dos dispositivos que passam a comercializar mais unidades de seus produtos e a desenvolver novos modelos de equipamentos, agregando à estes, novas funcionalidades.

Cada fabricante dessa família de equipamentos desenvolveu o hardware de sua linha de produtos e a plataforma de software que é executada sobre esse hardware. Em função de estratégias de mercado, o modelo do desenvolvimento dos produtos normalmente é fechado fazendo com que os aplicativos desenvolvidos para uma determinada plataforma não sejam compatíveis com outra.

Os fabricantes de dispositivos móveis, ao entenderem que o mercado de aplicativos para estes aparelhos poderia ser expandido, optaram por incorporar em suas plataformas de software o suporte a J2ME (*Java 2 Micro Edition*). Essa incorporação permitiu que desenvolvedores externos às empresas fabricantes pudessem desenvolver aplicativos sem precisarem de acesso direto ao código fonte implementado pelos próprios fabricantes dos dispositivos. Assim, o código desenvolvido pelos fabricantes dos dispositivos pode continuar fechado porém desenvolvedores externos às empresas fabricantes puderam desenvolver aplicativos que eram compatíveis com os dispositivos desenvolvidos.

Assim como no caso do J2ME, o desenvolvimento de aplicativos para dispositivos móveis pode ser feito em outras linguagens de programação. Habitualmente C, C++ e Objective-C despontam como as principais alternativas.

Os fabricantes de dispositivos móveis de comunicação, ao defenderem suas estratégias de mercado, deram origem a diversas plataformas de software diferentes. Tipicamente cada

plataforma de software é desenvolvida pela mesma empresa que desenvolve o hardware dos equipamentos, assim, uma mesma empresa desenvolve o hardware dos dispositivos e o software que é executado sobre o hardware desenvolvido.

Em função de alianças desenvolvidas por empresas, existem também plataformas de software de cunho genérico e que podem ser executadas sob diferentes hardwares desenvolvidos inclusive por diferentes fabricantes. Para esse modelo de negócio, basta que o fabricante do hardware siga determinados padrões da plataforma. Uma vez seguidos os padrões, a plataforma de software será executada sem problemas no hardware em questão, independente de qual for a empresa que o desenvolver. Esse, por exemplo, é o modelo de negócio da plataforma Android.

Na subseção a seguir serão abordadas as principais plataformas de software de telefonia móvel e algumas de suas características.

2.1 Principais plataformas de software de telefonia móvel

As principais plataformas de software de telefonia móvel atualmente são: Android, BlackBerry, iOS, Symbian, Windows Mobile. Na tabela 1 estão descritos os detalhes de cada uma dessas plataformas.

Tabela 1 Principais plataformas de telefonia móvel

Plataforma	Descrição
Android	Plataforma desenvolvida por um consórcio de empresas que criou a *Open Handset Alliance*, é totalmente *opensource* e baseada no *kernel* do Linux. De propósito genérico, é executada em diversos modelos de dispositivos e ainda assim, oferece ao usuário uma experiência de uso singular. Tem mostrado um acentuado crescimento de mercado e conquistado muitos dos novos consumidores de *smartphones*.
BlackBerry	Plataforma desenvolvida pela empresa Research In Motion, é de código fonte proprietário. É executada apenas em dispositivos desenvolvidos pela própria empresa e tem seu foco no mercado corporativo.
iOS	De acordo com APPLE (2010), é a plataforma com maior número de aplicativos disponíveis para download. Desenvolvida pela fabricante de dispositivos móveis Apple, é de código fonte proprietário. Executada apenas em dispositivos desenvolvidos pela própria Apple, oferece ao usuário uma experiência de uso considerada agradável.
Symbian OS	De acordo com SYMBIAN (2010), foi a primeira plataforma desenvolvida para *smartphones*. Continua sendo desenvolvida e, atualmente, é a plataforma com maior número de dispositivos em uso do mercado. Inicialmente formada por um consórcio de empresas, atualmente a plataforma pertence à fabricante de dispositivos móveis Nokia. No início de seu desenvolvimento a plataforma

Plataforma	Descrição
	era de código fonte fechado porém desde fevereiro de 2010 a plataforma é totalmente *opensource*.
Windows Mobile	Plataforma desenvolvida pela empresa Microsoft, naturalmente, é de código fonte proprietário. A principal aposta da empresa é levar aos *smartphones* a experiência de uso dos PCs.

Fonte: próprio autor.

De acordo com EXAME (2010), a plataforma que tiver a comunidade de desenvolvedores mais ativa será a mais forte candidata a vencer a batalha da computação móvel. Para atingir esse objetivo, deve ser utilizada uma linguagem de programação que seja de rápido desenvolvimento e de fácil aprendizagem. Se a linguagem em questão já tiver uma base de desenvolvedores instalada, a tendência é que o desenvolvimento para a plataforma seja acelerado e a plataforma em si saia ganhando com um maior número de aplicativos desenvolvidos.

É exatamente em função dessa análise que a *Open Handset Alliance* optou por utilizar a linguagem de programação Java para o desenvolvimento de aplicativos para a plataforma Android. Como a base de desenvolvedores que conhecem a linguagem de programação Java é considerável, é mais fácil atrair novos desenvolvedores para a plataforma sem que estes precisem aprender uma nova linguagem de programação. Nesse caso o desenvolvimento de aplicativos em si é acelerado visto que a incorporação de novos desenvolvedores é mais rápida.

2.2 Cenário atual do mercado de dispositivos móveis de telefonia

De acordo com TUDOR (2010), as vendas mundiais de telefones móveis cresceram 35% no terceiro trimestre de 2010 e as vendas de *smartphones* aumentaram em 96% no mesmo período.

A tabela 2 informa sobre a realidade de mercado dos principais fabricantes de dispositivos de telefonia móvel.

Tabela 2 Vendas mundiais de aparelhos celulares, em milhares de unidades, por fabricante, no terceiro trimestre de 2010.

Fabricante	Vendas no 3º trimestre de 2010 (Milhares de Unidades)	Participação de mercado no 3º trimestre de 2010 (Percentual)	Vendas no 3º trimestre de 2009 (Milhares de Unidades)	Participação de mercado no 3º trimestre de 2009 (Percentual)
Nokia	117461	28,16%	113466,2	36,73%
Samsung	71671,8	17,18%	60627,7	19,63%
LG	27478,7	6,59%	31901,4	10,33%
Apple	13484,4	3,23%	7040,4	2,28%
BlackBerry	11908,3	2,86%	8522,7	2,76%
Sony Ericsson	10346,5	2,48%	13409,5	4,34%
Motorola	8961,4	2,15%	13912,8	4,50%
HTC	6494,3	1,56%	2659,5	0,86%
ZTE	6003,6	1,44%	4143,7	1,34%
Huawei Technologies	5478,1	1,31%	3339,7	1,08%
Others	137797,6	33,04%	49871,1	16,15%
Total	417085,7	100,00%	308894,7	100,00%

Fonte: TUDOR, 2010.

Conforme mencionado anteriormente, todos aparelhos celulares comercializados, independente de serem *smartphones* ou não, necessariamente são comercializados com uma plataforma de software que permite operar e interagir com o dispositivo. É essa plataforma que permite ao usuário vivenciar a experiência de uso do equipamento.

As plataformas anteriormente citadas disputam entre si, de forma acirrada, o domínio do mercado de dispositivos móveis. A atual líder em quantidade de dispositivos sendo utilizados no mundo é a plataforma Symbian.

Na tabela 3 são apresentados maiores detalhes da realidade de mercado das principais plataformas de software para telefonia móvel.

Tabela 3 Vendas mundiais de *smartphones*, em milhares de unidades, por plataforma, no terceiro trimestre de 2010.

Plataforma	Vendas no 3º trimestre de 2010 (Milhares de Unidades)	Participação de mercado no 3º trimestre de 2010 (Percentual)	Vendas no 3º trimestre de 2009 (Milhares de Unidades)	Participação de mercado no 3º trimestre de 2009 (Percentual)
Symbian	29480,1	36,61%	18314,8	44,57%

Plataforma	Vendas no 3º trimestre de 2010 (Milhares de Unidades)	Participação de mercado no 3º trimestre de 2010 (Percentual)	Vendas no 3º trimestre de 2009 (Milhares de Unidades)	Participação de mercado no 3º trimestre de 2009 (Percentual)
Android	20500	25,46%	1424,5	3,47%
iOS	13484,4	16,74%	7040,4	17,13%
BlackBerry	11908,3	14,79%	8522,7	20,74%
Microsoft Windows Mobile	2247,9	2,79%	3259,9	7,93%
Linux	1697,1	2,11%	1918,5	4,67%
Other OS	1214,8	1,51%	612,5	1,49%
Total	80532,6	100,00%	41093,3	100,00%

Fonte: TUDOR, 2010.

Ao analisar os dados apresentados na tabela 3 percebe-se o expressivo crescimento da plataforma Android que em um ano aumentou sua participação de mercado em 21,99%. Ao aprofundar ainda mais a análise, percebe-se que praticamente todas as demais plataformas objeto do estudo sofreram reduções de crescimento. A única exceção está em plataformas específicas com participação de mercado muito pequena e que, por isso na tabela, são apresentadas como *Other OS*.

É importante notar que a plataforma Android foi lançada em outubro de 2008 e portanto é uma plataforma nova no mercado de dispositivos móveis de comunicação. Ainda assim, com pouco tempo de mercado, já mostrou seu potencial ultrapassando plataformas que têm anos de mercado como iOS e BlackBerry.

Em função do acentuado crescimento da plataforma Android no mercado de telefonia móvel, estima-se que em breve a plataforma poderá inclusive tornar-se a mais utilizada mundialmente em *smartphones*, tirando assim, a atual posição da plataforma Symbian.

CANALYS (2010) defende que um dos motivos pelos quais a plataforma Android tem tido um crescimento de mercado acentuado é o fato de que existem múltiplas alianças com diversas empresas fabricantes de dispositivos. Fabricantes como Samsung, HTC, Motorola e Sony Ericson são exemplos de grandes empresas que comercializam dispositivos com a plataforma Android.

3 PLATAFORMA ANDROID

Android consiste em uma plataforma de software que foi desenvolvida para utilização no mercado de telefonia móvel. Nessa plataforma estão incluídos um sistema operacional, um *middleware*[1] e aplicativos diversos encontrados comumente em dispositivos móveis. O Android SDK (*Software Development Kit*) fornece as ferramentas e APIs (*Application Programming Interface*) necessárias para o desenvolvimento de aplicativos na plataforma Android utilizando a linguagem de programação Java (ANDROID DOCUMENTATION, 2010).

Apesar de o foco inicial e principal da plataforma ser o mercado de telefonia móvel, o Android já vem sendo utilizado com sucesso em projetos que vão além da telefonia. Com a crescente procura por dispositivos cada vez mais portáteis, empresas como a Samsung incorporaram a plataforma Android em *tablets*. A plataforma está também sendo incorporada no mercado de entretenimento através de um novo serviço do Google, o *Google TV*. Projetos de pesquisa vão ainda mais longe incorporando a plataforma em aparelhos de micro-ondas e até em máquinas de lavar roupas.

Apesar de todas essas possibilidades, no presente trabalho, a abordagem da plataforma se dará de forma direcionada ao mercado de telefonia móvel.

3.1 Arquitetura da plataforma Android

A figura 1 mostra os principais componentes do sistema operacional do Android. Cada seção é descrita em detalhes a seguir.

[1] Camada de software que conecta diferentes componentes de software entre si.

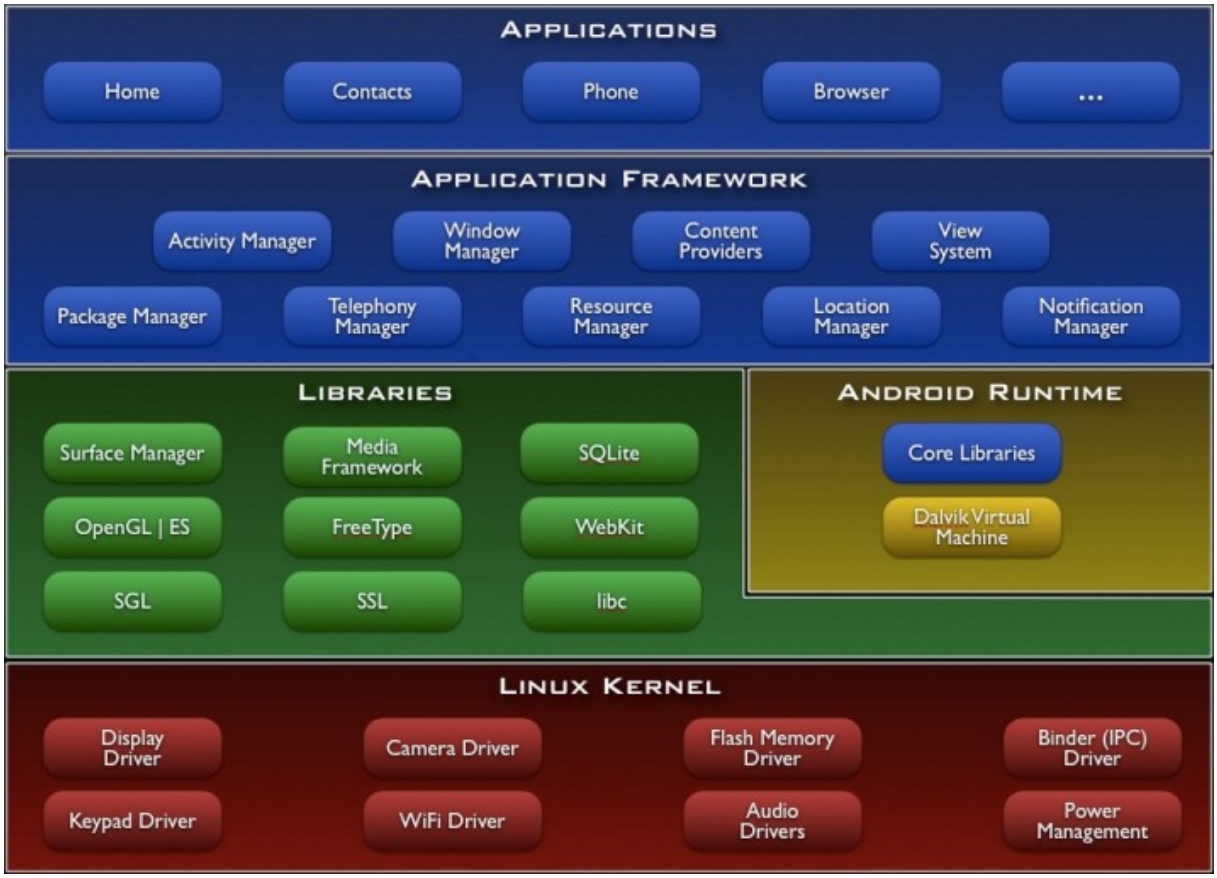

Figura 1 Principais componentes do sistema operacional Android (ANDROID DOCUMENTATION, 2010).

3.1.1 Kernel do Linux

De acordo com ANDROID DOCUMENTATION (2010), a arquitetura da plataforma Android está baseada no *kernel* versão 2.6 das distribuições Linux. O *kernel* é utilizado como uma camada de abstração do hardware.

Ainda segundo ANDROID DOCUMENTATION (2010), um dos motivos pelos quais a plataforma Android utiliza o *kernel* do Linux como base, tem relação com o fato de o *kernel* oferecer um modelo de gerenciamento de *drivers* comprovadamente funcional. O *kernel*, neste caso, também fornece à plataforma Android gerenciamento de memória, gerenciamento de processos, modelo de segurança, suporte a rede e uma série de características que são reconhecidamente robustas e comprovadamente funcionais.

Na figura 1, a camada do *kernel* do Linux (*Linux Kernel*) está representada através da cor vermelha.

3.1.2 Bibliotecas

Conforme ANDROID DOCUMENTATION (2010), a arquitetura Android inclui um conjunto de bibliotecas, desenvolvidas nas linguagens de programação C/C++, utilizadas por diversos componentes do sistema. Dentre as bibliotecas disponibilizadas, destacam-se as seguintes:

a) *Surface Manager* – biblioteca responsável por gerenciar a exibição de diferentes telas, que são geradas a partir de diferentes aplicativos, que executam em diferentes processos;

b) *OpenGL | ES* – biblioteca responsável pelo tratamento de imagens em 3D (três dimensões). Implementada em software, pode ser acelerada por hardware caso o dispositivo possua um *chip* para processamento gráfico em 3D;

c) *SGL* – biblioteca responsável pelo tratamento de imagens em 2D (duas dimensões) que são maioria nos aplicativos desenvolvidos para dispositivos móveis;

d) *Media Framework* – biblioteca que oferece suporte à gravação e execução de áudio e vídeo além de suportar a manipulação de imagens;

e) *FreeType* – biblioteca responsável por renderizar imagens de tipo *bitmap* utilizadas nos aplicativos;

f) *SQLite* – biblioteca que oferece suporte à armazenamento de dados em banco de dados SQL embutido na plataforma;

g) *WebKit* – motor do navegador Internet;

h) *libc* – uma implementação da biblioteca libc otimizada para dispositivos embarcados baseados em Linux;

Na figura 1, a camada das bibliotecas (*Libraries*) está representada através da cor verde indicando que todas as bibliotecas dessa camada foram desenvolvidas nas linguagens de programação C/C++.

3.1.3 Android Runtime

O *runtime* da plataforma foi desenvolvido de forma a permitir que as necessidades de execução de aplicativos em dispositivos embarcados, onde há limitações de bateria, RAM (*Random Access Memory*) e CPU (*Central Processing Unit*), fossem atendidas (ANDROID DOCUMENTATION, 2010).

A máquina virtual *Dalvik*, principal componente do *runtime* da plataforma, executa arquivos com a extensão .dex que são *bytecodes* resultantes da conversão, realizada em tempo

de compilação, de arquivos com as extensões .class e .jar. Os arquivos com a extensão .dex são transformados em *bytecodes* muitas vezes mais eficientes em relação aos originais e exatamente por isso podem ser executados com um grau satisfatório de desempenho em processadores de menor capacidade de processamento como os presentes em dispositivos móveis (ANDROID DOCUMENTATION, 2010).

O resultado final desta otimização é o que permite que se tenha múltiplas instâncias da máquina virtual *Dalvik* sendo executadas por diferentes aplicativos, no mesmo dispositivo ao mesmo tempo.

Ainda de acordo com ANDROID DOCUMENTATION (2010), na camada do *runtime* da plataforma, tem-se as *Core Libraries* que são bibliotecas desenvolvidas na linguagem de programação Java e que disponibilizam um conjunto de pacotes da linguagem de programação J2SE (*Java 2 Standard Edition*) versão 5.0 incluindo classes de coleções, utilitários, I/O (*Input/Output*), entre outros.

Na figura 1, a camada *Android Runtime* está representada através da cor amarela. As *Core Libraries* que, conforme mencionado anteriormente, foram desenvolvidas na linguagem de programação Java, aparecem representadas pela cor azul.

3.1.4 *Framework* de aplicações

Na plataforma Android a camada do *Framework* de aplicações (*Application Framework)* armazena o conjunto de ferramentas que são utilizadas pelos aplicativos instalados no dispositivo (ANDROID DOCUMENTATION, 2010).

Dentre as ferramentas disponibilizadas estão:

a) *Activity Manager* – ferramenta que gerencia o ciclo de vida dos aplicativos permitindo a integração entre estes;

b) *Package Manager* – ferramenta que gerencia os aplicativos que estão instalados no dispositivo identificando as funcionalidades de cada um destes;

c) *Window Manager* – ferramenta que gerencia a exibição das telas, é na verdade uma camada de abstração fornecida pela biblioteca *Surface Manager*;

d) *Telephony Manager* – ferramenta que contém as APIs necessárias para a utilização do dispositivo como telefone;

e) *Content Providers* – ferramentas que permitem que os aplicativos instalados no dispositivo compartilhem dados entre si;

f) *Resource Manager* – ferramenta que permite armazenar *strings*, imagens, informações de *layout* e tudo mais que não for código dos aplicativos instalados no dispositivo;

g) *View System* – ferramenta que disponibiliza acesso à botões, listas e demais componentes utilizados pela interface gráfica dos aplicativos. Gerencia também o disparo de eventos e *layout* dos aplicativos;

h) *Location Manager* – ferramenta responsável por obter a localização geográfica do dispositivo através de algum meio disponível de geolocalização. Dentre os meios disponíveis estão: GPS, caso o dispositivo ofereça suporte a essa tecnologia, triangulação de torres de telefonia celular e informações de geolocalização presentes em redes Wi-Fi (*Wireless Fidelity*);

i) *Notification Manager* – ferramenta responsável por oferecer aos aplicativos uma forma padronizada de notificar o usuário a respeito de informações relevantes;

Na figura 1, a camada *Application Framework* está representada através da cor azul indicando que todas as ferramentas dessa camada foram desenvolvidas na linguagem de programação Java.

3.1.5 Aplicações

A camada das aplicações (*Applications*) é a camada onde estão enquadrados os aplicativos que serão utilizados pelo usuário do dispositivo. É nesta camada que estão aplicativos diversos como gerenciadores de e-mail, de calendário, de agenda, navegador Internet, reprodutores de áudio/vídeo, entre outros.

Na figura 1, a camada *Applications* está representada através da cor azul indicando que os aplicativos dessa camada foram desenvolvidos na linguagem de programação Java.

3.2 Componentes de uma aplicação Android

De acordo com MEIKE (2009), todas as aplicações Android são construídas a partir de quatro tipos básicos de componentes que são definidos pela plataforma. A tabela 4 mostra maiores detalhes sobre cada componente.

Tabela 4 Os quatro componentes básicos das aplicações Android.

Componente	Descrição
Activities	Uma *Activity* é essencialmente uma parte da interface gráfica de um aplicativo sendo tipicamente composta por uma janela. Dessa forma, ao analisar um

Componente	Descrição
	aplicativo como um cliente de e-mail, ele poderá ser decomposto em aproximadamente três ou quatro *Activities* diferentes: uma *Activity* que lista as mensagens, outra que exibe as mensagens individualmente, uma terceira *Activity* que permite redigir e-mails e uma quarta que permite configurar contas, por exemplo. Quando não estiver em execução ativa, uma *Activity* pode ser eliminada pelo sistema operacional do Android para economizar RAM.
Services	*Services* são semelhantes aos serviços ou *daemons* nos sistemas operacionais de servidor ou *desktop*: são tarefas que geralmente não têm interface gráfica, que possuem tempo de execução longo e que são executadas em segundo plano. Como exemplo pode-se citar um software para reprodução de áudio. Ele é iniciado através de uma *Activity* porém deseja-se que o áudio não deixe de ser reproduzido quando o usuário finalizar a *Activity* que iniciou a reprodução do áudio. Dessa forma o código que mantém o áudio em execução é na verdade um *Service*, que executa em segundo plano. Não tem interface gráfica e permanece sendo executado, independente da *Activity* que o inicializou.
Broadcast Receivers e *Intent Receivers*	Um *Broadcast Receiver* responde a um aviso global de evento. Estes avisos podem ter origem no próprio Android (nível baixo de carga da bateria, por exemplo) ou em qualquer aplicativo em execução no sistema. Um *Intent Receiver* é um meio através do qual qualquer aplicativo pode registrar determinado trecho de código que não será executado até que um gatilho de um evento externo seja disparado por outra *Activity*. Dessa forma, pode-se criar aplicativos que executarão determinado processo sempre que o telefone tocar, por exemplo.
Content Providers	*Content Providers* permitem que se compartilhe informações entre *Activities* ou *Services*. Essa capacidade se dá através de um mecanismo que permite atender a solicitações de outras aplicações. Por exemplo, quando uma aplicação envia uma consulta de dados de um contato da agenda, ela direciona a consulta para um URI (*Uniform Resource Identifier*) na forma *content://contacts/people*. O sistema operacional do Android verifica quais aplicações se registraram como *Content Providers* para o determinado URI e envia a solicitação à aplicação adequada. Caso a aplicação não esteja iniciada, o sistema se encarrega de inicializá-la. Caso existam duas ou mais aplicações que respondam como *Content Providers* para o URI em questão, o sistema permitirá ao usuário escolher qual pretende utilizar.

Fonte: MEIKE, 2009.

3.3 Ciclo de vida de *Activities* na plataforma Android

A plataforma Android foi projetada em torno dos requisitos específicos às aplicações móveis. Isso significa que o projeto da plataforma leva em conta o ambiente no qual será executado. Nesse ambiente recursos como RAM e bateria são normalmente limitados e é necessário preservá-los (MEIKE, 2009).

Ainda segundo MEIKE (2009), o ciclo de vida das *Activities* na plataforma Android define os estados ou eventos pelos quais uma *Activity* passa desde o momento em que é criada até o término de sua execução.

A figura 2 mostra, na forma de esquema, o ciclo de vida de uma *Activity* na plataforma Android.

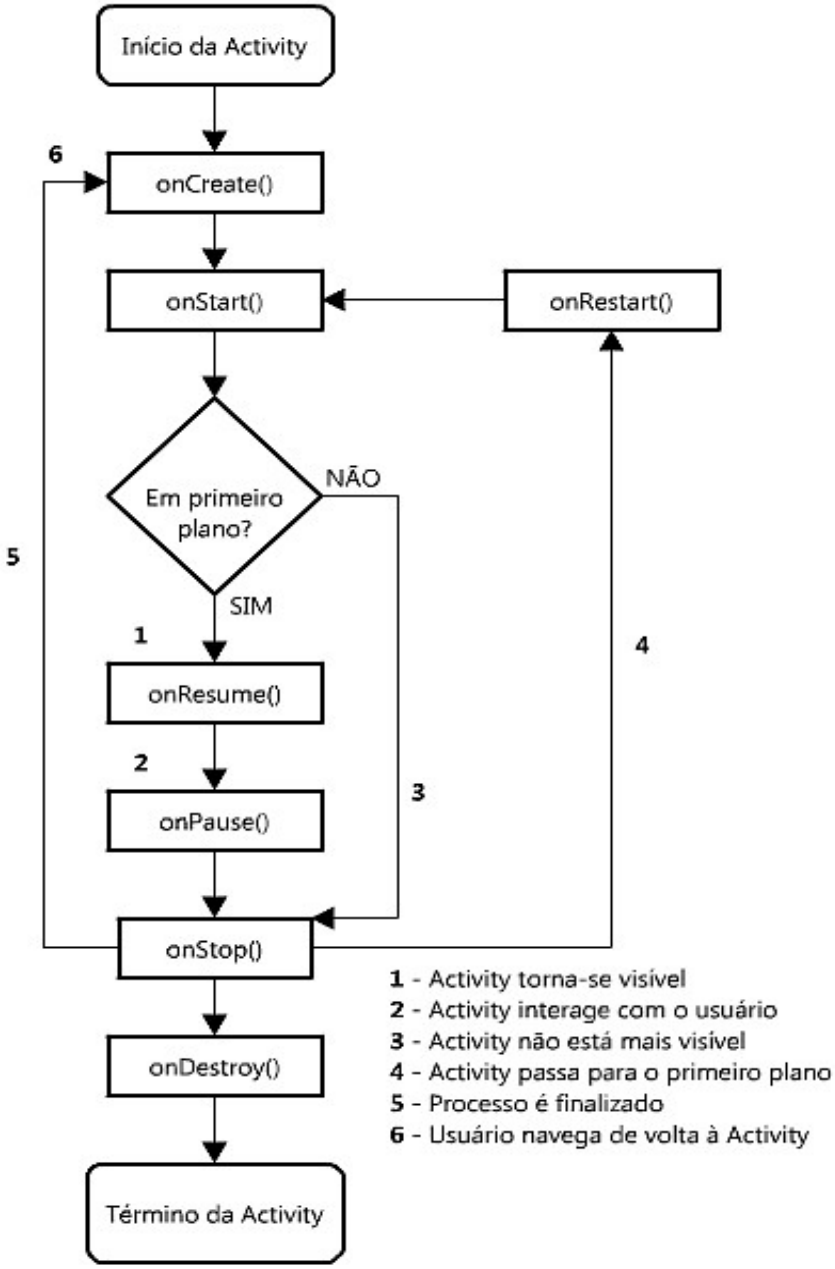

Figura 2 Ciclo de vida de uma *Activity* na plataforma Android (MEIKE, 2009).

A *Activity* monitora e reage aos eventos redefinindo os métodos da classe *Activity* para cada evento descrito na tabela 5.

Tabela 5 Descrição dos eventos utilizados no desenvolvimento de *Activities*.

Evento	Descrição
onCreate	Método chamado no momento em que a *Activity* é criada. É nesse estágio que são criadas as *Views*, são abertos quaisquer arquivos de dados que a *Activity* possa vir a utilizar e, em geral, a *Activity* é inicializada.
onStart	Método chamado antes de a *Activity* ficar visível na tela. Quando o método *onStart* for concluído e caso seja possível que a *Activity* se torne a *Activity* de primeiro plano na tela, o controle é transferido para o método *onResume*. Caso a *Activity* não possa ser a *Activity* de primeiro plano na tela, o controle é transferido para o método *onStop*.
onResume	Método chamado após o método *onStart* caso a *Activity* for a de primeiro plano em tela. Nesse ponto a *Activity* está em execução e interagindo com o usuário. O método *onResume* também é chamado nos casos em que a *Activity* atual é finalizada e é necessário exibir em tela a *Activity* anterior.
onPause	Método chamado quando o sistema operacional Android está prestes a retomar uma *Activity* diferente, concedendo a ela o primeiro plano. A *Activity* atual deve, idealmente, aproveitar esse método para armazenar qualquer estado que possa precisar futuramente quando obtiver novamente o primeiro plano.
onStop	Método chamado quando a *Activity* já não está visível em tela seja porque outra *Activity* obteve o primeiro plano ou porque a *Activity* está sendo finalizada.
onDestroy	Método chamado quando a *Activity* está sendo finalizada. Representa a última oportunidade para que a *Activity* efetue qualquer processamento antes de ser eliminada.

Fonte: MEIKE, 2009.

A correta utilização dos métodos anteriormente mencionados permite proporcionar ao usuário uma melhor experiência de uso do aplicativo desenvolvido.

3.4 Ciclo de vida de *Services* na plataforma Android

Conforme MEIKE (2009), assim como no caso das *Activities*, os *Services* da plataforma Android também possuem um ciclo de vida definido.

Na tabela 6 estão descritos os detalhes de cada evento utilizado no desenvolvimento de *Services*.

Tabela 6 Descrição dos eventos utilizados no desenvolvimento de *Services*.

Evento	Descrição
onCreate / *onStart*	*Services* podem ser iniciados quando um cliente chama o método *Context.startService(Intent)*. Caso o *Service* não esteja em execução, ele é inicializado pelo sistema operacional do Android através do método *onCreate* seguido do método *onStart*. Caso o *Service* já esteja em execução, o método

Evento	Descrição
	onStart é chamado novamente com o novo *Intent*. Assim, é bastante possível e comum que o método *onStart* de um *Service* seja chamado repetidas vezes em uma única execução do *Service*.
onResume / *onPause* / *onStop*	Conforme mencionado anteriormente, *Services* geralmente não têm interface gráfica e em função disso os métodos *onResume*, *onPause* e *onStop* não se aplicam. Sempre que um *Service* estiver em execução, ele estará em segundo plano.
onBind	Para os casos em que um cliente de um *Service* necessita uma conexão persistente com o *Service*, o cliente pode instanciar o método *Context.bindService*. A chamada desse método cria o *Service* caso este não esteja em execução e chama o método *onCreate* mas não o *onStart*. Ao invés disso, o método *onBind* é chamado com o *Intent* do cliente e este, por sua vez, retorna um objeto *IBind* que o cliente pode utilizar para efetuar futuras chamadas ao *Service*.
onDestroy	Exatamente como em uma *Activity*, o método *onDestroy* é chamado quando o *Service* estiver sendo finalizado. O sistema operacional do Android irá finalizar um *Service* quando não houver mais clientes inicializando-o ou caso não existam mais clientes a ele conectados. Ainda, analogamente ao que acontece nas *Activities*, o sistema operacional do Android poderá finalizar um *Service* quando a RAM do dispositivo estiver se esgotando. Para estes casos, o sistema tentará reinicializar o *Service* quando a escassez de RAM tiver passado.

Fonte: MEIKE, 2009.

3.5 Reutilização e substituição de aplicativos na plataforma Android

A plataforma Android foi projetada de forma a estimular a reutilização e a substituição de aplicativos (ANDROID DOCUMENTATION, 2010). Essa capacidade se dá através de um mecanismo interno da plataforma descrito a seguir.

Toda vez que um aplicativo é instalado na plataforma Android, através do *Package Manager*, o aplicativo se registra no sistema operacional como *Content Provider* para determinados tipos de requisições. Esse registro permite ao sistema operacional do Android identificar quais aplicativos manipulam os diferentes tipos de requisições que recebe.

Na plataforma Android, para que qualquer um dos aplicativos instalados no dispositivo possa ter acesso à funcionalidades de outros aplicativos, a primeira coisa que o aplicativo deve fazer é enviar uma requisição ao sistema operacional. Dessa forma, o aplicativo que deseja ter acesso a uma funcionalidade de outro aplicativo, realiza uma requisição para uma ação específica. A ação nada mais é do que a representação de uma classe formal da plataforma Android conhecida como *Intent*.

O sistema operacional, ao receber esta requisição, avalia todos os aplicativos instalados e analisa qual o aplicativo que melhor poderá atender a requisição. Ao encontrar o aplicativo adequado, o sistema operacional conecta o aplicativo solicitante da requisição (cliente) com o aplicativo que irá atendê-la (servidor) de forma a permitir que a tarefa seja concluída.

É importante perceber que neste caso o aplicativo cliente não tem construída em sua estrutura interna as funcionalidades necessárias para ter acesso direto ao objeto resultante da requisição. Ao invés do acesso direto, o aplicativo cliente envia a requisição e recebe como retorno o objeto enviado pelo aplicativo que atendeu a requisição enviada.

Esse comportamento torna a plataforma mais robusta uma vez que o aplicativo servidor da requisição não atende apenas um aplicativo cliente mas sim tantos quantos realizarem a requisição adequada. É dessa forma que a plataforma Android reutiliza componentes.

Por outro lado, analogamente aos diversos aplicativos clientes, pode-se ter diversos aplicativos servidores de requisições. Essa característica torna a plataforma mais flexível pois permite ao usuário optar por entre qual aplicativo servidor deseja completar a tarefa iniciada pelo aplicativo cliente da requisição.

A flexibilidade da plataforma nesse caso tem relação com o fato de que o usuário pode inclusive remover um aplicativo servidor de requisições e substitui-lo por outro. Caso o usuário opte em manter dois ou mais aplicativos que atendam ao mesmo tipo de requisição, a escolha do aplicativo que irá atender a requisição em questão é feita em tempo de execução no momento em que o sistema operacional solicita ao usuário qual aplicativo este pretende utilizar para realizar a tarefa pretendida.

É importante notar que este comportamento se estende a qualquer tipo de requisição visto que em última análise, qualquer tarefa passa, necessariamente, por um *Intent* (caso o usuário pretenda ir do ponto X para o ponto Y, há um *Intent* interligando os dois pontos). Cada um dos *Intents* que interligam quaisquer dois pontos distintos da plataforma representam uma oportunidade de reutilização e/ou substituição de um aplicativo servidor de requisições.

Assim, considerando a existência de *Intents* para envio de e-mail por exemplo, isso significa dizer que o usuário pode, caso queira, substituir o aplicativo que efetivamente envia a mensagem. Nesta mesma linha, considerando um *Intent* que seja responsável pela reprodução de áudio, significa dizer que o usuário pode, caso queira, substituir o aplicativo que executa o áudio.

3.6 Gerenciamento de aplicações Android

Conforme ANDROID DOCUMENTATION (2010), na plataforma Android, cada aplicativo é executado dentro de seu próprio processo. Esse modelo faz com que os aplicativos sejam executados isoladamente uns dos outros e isso garante que um aplicativo, ao utilizar intensamente a CPU do dispositivo, não irá bloquear atividades críticas como atender à uma ligação ou gerenciar o nível de carga da bateria, por exemplo.

É tarefa do sistema operacional do Android, gerenciar os processos necessários para cada aplicativo, iniciando processos de acordo com a demanda de uso e finalizando processos caso a demanda por recursos exista mas os recursos disponíveis no dispositivo estejam escassos (ANDROID DOCUMENTATION, 2010). Isso significa dizer que o sistema operacional do Android pode, em determinadas situações, eliminar determinados processos para liberar recursos de forma a permitir que novos processos sejam instanciados. A decisão sobre quais processos serão finalizados em detrimento dos novos é feita pelo sistema operacional do Android com base em critérios que serão detalhados a seguir.

Do ponto de vista do usuário, todas essas informações são irrelevantes visto que o objetivo é permitir que este realize a tarefa que desejar. O usuário não precisa conhecer detalhes internos da plataforma nem se preocupar com a disponibilidade de recursos no dispositivo para que seu objetivo seja atendido. Todo o gerenciamento que acontece para que que as tarefas sejam executadas é feito de forma transparente e realizado pelo sistema operacional do Android. A forma através da qual o sistema operacional do Android gerencia os recursos e interliga as aplicações será detalhada a seguir.

A forma pela qual as aplicações são interligadas na plataforma Android, está baseada no conceito de pilha. O primeiro processo empilhado na pilha de processos da plataforma Android é o processo *System Process*. O processo *System Process* contém o *Activity Manager* e este, por sua vez, é na verdade uma nova pilha que guarda as informações dos aplicativos que forem inicializados. É com base no *Activity Manager* que o sistema operacional do Android sabe qual aplicativo deverá ser exibido na tela quando receber um evento de tecla do tipo *Back* gerado pelo botão Voltar.

Sempre que um novo aplicativo for inicializado, o sistema operacional do Android salva o estado da *Activity* atual, do aplicativo atual, para que tenha condições de voltar à *Activity* caso o novo aplicativo apresente algum problema. As informações salvas são movidas para o *System Process*. Depois de salvo o estado da *Activity*, a informação da inicialização do novo aplicativo é inserida na pilha do *Activity Manager* e o processo do novo aplicativo é

criado e empilhado na pilha de processos da plataforma Android. Dentro do processo recém empilhado será criada a *Activity* do aplicativo inicializado e somente depois disso é que o aplicativo é exibido em tela. É importante notar que para os casos em que se tem mais de uma *Activity* por aplicativo, todas serão criadas dentro do mesmo processo.

As informações que são movidas para o *System Process* sempre que um novo aplicativo é inicializado não são informações internas do aplicativo que as gerou. Os dados salvos são na verdade metadados que permitem ao sistema operacional do Android reexibir a *Activity* do aplicativo da forma como estava quando o novo aplicativo foi inicializado.

Caso a pilha de processos da plataforma Android cresça de tal forma a atingir seu tamanho limite e caso ainda assim o sistema operacional do Android receba solicitações de inicialização de novos aplicativos, o sistema deverá encontrar algum processo que possa ser finalizado sem comprometer diretamente os aplicativos base da plataforma (*Home Application*, por exemplo) nem o aplicativo que atualmente está em utilização. Ao identificar algum processo que se encaixe nos parâmetros mencionados, o sistema operacional do Android finalizará a *Activity* (ou as *Activities*) do aplicativo e em seguida finalizará o processo do aplicativo. Dessa forma libera-se espaço na pilha de processos do Android e o novo aplicativo poderá ser inicializado.

É importante notar que foram finalizados a *Activity* (ou as *Activities*) e o processo do aplicativo, os metadados salvos no *System Process* permanecem inalterados. A causa pela qual estes dados permanecem armazenados no *System Process* tem relação com o tratamento de eventos do tipo *Back*. Quando o sistema operacional do Android recebe um evento do tipo *Back* o aplicativo atual deve ser finalizado de forma a permitir que volte para a tela do dispositivo o aplicativo que havia sido inicializado anteriormente.

Assumindo que o processo do aplicativo que se deseja voltar a exibir em tela ainda exista, a única coisa que precisa ser feita pelo sistema operacional do Android é resgatar os metadados que estão armazenados no *System Process* e exibir a *Activity* do aplicativo em questão da forma como indicam os metadados resgatados do *System Process*.

Caso o processo do aplicativo que se deseja voltar a exibir em tela tenha sido finalizado, o sistema operacional do Android deverá identificar um outro processo que possa ser finalizado (visto que a pilha de processos atingiu seu limite de tamanho e o número de processos ainda não diminuiu), finalizar o processo identificado e, no espaço da pilha de processos que foi liberado com a finalização do processo identificado, criar um novo processo no qual será criada uma nova *Activity*. Para que a nova *Activity* seja exibida da forma como

estava a *Activity* anterior, o sistema operacional do Android deverá resgatar os metadados salvos no *System Process* e com base nesses metadados, moldar a nova *Activity*.

Assim, somente depois de executados todos os passos citados é que o aplicativo será exibido em tela e irá então interagir com o usuário.

3.7 Desenvolvimento de aplicativos para a plataforma Android

Conforme mencionado anteriormente, o desenvolvimento de aplicativos para a plataforma Android é realizado através da linguagem de programação Java. Em ANDROID DOCUMENTATION 2 (2010), está disponível um manual para instalação do SDK da plataforma e posterior integração com a IDE (*Integrated Development Environment*) Eclipse.

É importante notar que o desenvolvimento de aplicativos para a plataforma não necessariamente precisa ser feito através da IDE Eclipse. Seu uso é recomendado em função da disponibilização, por parte do time de desenvolvedores da plataforma, de um *plug-in* que automatiza algumas das tarefas de desenvolvimento e facilita assim a tarefa do desenvolvedor.

Em ANDROID DOCUMENTATION (2010) estão descritos os detalhes de todos os componentes que podem ser utilizados para a criação de aplicativos. Estão descritos também detalhes sobre o desenvolvimento de aplicativos, publicação dos mesmos no *Android Market*, melhores práticas de desenvolvimento e demais informações úteis a quem pretende desenvolver aplicativos para a plataforma.

Uma característica importante para quem desenvolve aplicativos para a plataforma Android é o sistema de permissões. Através desse sistema o desenvolvedor especifica, durante o processo de codificação, as permissões que o aplicativo deverá ter para ser executado da forma adequada quando for instalado em algum dispositivo. Assim, supondo o caso de um aplicativo que dependa de alguma informação disponível na Internet, o aplicativo precisará da permissão de acesso à Internet para funcionar da forma adequada. As permissões definidas para cada aplicativo são exibidas ao usuário no ato da instalação. Caso o usuário não concorde com as permissões, a instalação do aplicativo é cancelada.

Depois de desenvolvidos, os aplicativos da plataforma Android devem ser exportados e assinados digitalmente para então serem disponibilizados para os usuários da plataforma. O processo de exportação e assinatura tem como resultado um arquivo único, de extensão .apk, que ao ser acessado pelos dispositivos que executam a plataforma Android, permite a instalação do software.

A disponibilização do software desenvolvido pode ser feita disponibilizando o arquivo de extensão .apk para o usuário final via algum *site* no qual o desenvolvedor tenha permissão para armazenar o arquivo ou então via *Android Market*.

Android Market é na verdade uma loja virtual voltada para a disponibilização de software desenvolvido para a plataforma Android. Para disponibilizar um aplicativo, o desenvolvedor deve registrar-se no *site* e após o registro, realizar o *upload* do arquivo de extensão .apk gerado através da exportação e assinatura do código fonte do projeto de desenvolvimento do software.

Apesar de se tratar de uma loja virtual, a *Android Market* disponibiliza também diversos aplicativos sem custo. O desenvolvedor do aplicativo é quem decide se irá ou não cobrar pelo software e, caso opte pela cobrança, o valor de venda do mesmo.

De acordo com ANDROID MARKET (2010), as transações financeiras, tanto de compra como de venda de aplicativos, são realizadas através do *Google Checkout* que é um serviço de processamento de compras do Google. No caso específico do Brasil e de países como Argentina, Israel, México, Rússia, Coreia do Sul e Taiwan, os valores gerados pela venda de aplicativos são creditados aos desenvolvedores através do *Google Adsense*, um serviço do Google que paga proprietários de *sites* por disponibilizarem, em seus *sites*, publicidade de clientes do Google.

Nesse capítulo foram apresentadas as principais características da plataforma Android. Percebe-se que a plataforma foi construída levando fortemente em conta o ambiente no qual é executada. É importante notar que o resultado desse cuidado é um dos principais responsáveis pela agradável experiência de uso que os dispositivos baseados na plataforma oferecem aos seus usuários e é exatamente essa experiência de uso que tem levado a plataforma Android para além do mercado de telefonia.

4 DESENVOLVIMENTO DO APLICATIVO BLOCK MY ANDROID

O sistema desenvolvido através do presente trabalho consiste em um aplicativo para a plataforma de telefonia móvel Android cujas principais funcionalidades são o bloqueio remoto de *smartphones* e a geolocalização dos mesmos através de informações de GPS, caso o dispositivo ofereça suporte a essa tecnologia, triangulação de torres de telefonia celular e através de informações de geolocalização presentes em redes Wi-Fi.

O projeto está dividido em duas frentes distintas que são a criação do aplicativo para a plataforma Android e a criação de um portal de administração através do qual poderão ser enviados aos *smartphones* instruções de bloqueio e de geolocalização. A criação do aplicativo que será executado nos *smartphones* é parte integrante do escopo desse trabalho. A criação do portal de administração não é parte integrante do escopo desse trabalho.

O aplicativo desenvolvido será instalado em *smartphones* da plataforma Android versão 2.2 ou superior e, através de um *Service* inicializado no momento em que o dispositivo é ligado, realizará constantes checagens ao portal de administração. Dessa forma, o usuário do *smartphone*, ao deixar de ter acesso ao dispositivo, poderá acessar o portal de administração e atualizar o *status* do aparelho para a realidade adequada, podendo esta ser *Lost* ou *Stolen*. No momento em que o usuário voltar a ter acesso ao dispositivo, deverá acessar novamente o portal de administração e alterar o *status* do aparelho para *OK*. As ações que cada uma dessas opções de *status* desencadeiam estão detalhadas a seguir.

O *Service* que será executado no *smartphone*, ao sincronizar seus dados com os do portal de administração, moldará o comportamento do aplicativo de acordo com o *status* do dispositivo configurado no portal.

Em função das características do sistema, uma conexão de dados entre o dispositivo no qual o aplicativo final for instalado e o servidor da aplicação é necessária. A conexão em questão pode ser realizada tanto através de redes Wi-Fi bem como através de conexões móveis disponibilizadas pelas operadoras de telefonia.

O aplicativo desenvolvido recebeu o nome de *Block my Android* e nas seções seguintes serão demonstrados em detalhes os recursos desenvolvidos, a metodologia e as ferramentas utilizadas.

4.1 Funcionamento do *Block my Android*

O BMA (*Block my Android)* é composto por duas *Activities*, por um *Service*, por dois *Broadcast Receivers* além de classes auxiliares.

As *Activities*, como mencionado anteriormente, são essencialmente partes da interface gráfica do aplicativo em questão sendo tipicamente compostas por uma única janela. Através das *Activities* é que é realizada a configuração de parâmetros de uso do aplicativo e podem ser checados dados como data e hora da última sincronização, data e hora da próxima sincronização e o *status* atual do *smartphone*, retornado pelo servidor.

Services, como mencionado anteriormente, são tarefas que não têm interface gráfica, que possuem tempo de execução longo e que são executados em segundo plano. O *Service* desenvolvido é responsável por autenticar no portal de administração, checar o *status* do dispositivo e atualizar localmente essas informações. De acordo com o retorno recebido pelo *Service*, é disparado um gatilho e, através dos *Broadcast Receivers*, o comportamento da aplicação é moldado.

4.2 Agendamento e execução do *Service*

Através do arquivo *AndroidManifest.xml*, que é um arquivo de existência obrigatória em todas aplicações desenvolvidas para a plataforma de telefonia móvel Android e que apresenta informações essenciais para a execução destas aplicações, o final do processo de inicialização do dispositivo é detectado e ações específicas são disparadas afim de inicializar o *Service* do BMA.

O arquivo *AndroidManifest.xml* gerencia uma série de informações necessárias para o funcionamento da aplicação desenvolvida. É nele que estão configuradas informações como local de instalação do aplicativo, que pode ser na memória interna do dispositivo ou em memórias externas, os nodos da aplicação que mapeiam as classes em função de suas funcionalidades como *Activities*, *Broadcast Receivers* e *Services*, e as permissões que o aplicativo desenvolvido precisa ter para funcionar da forma como foi projetado.

As permissões determinam funcionalidades e informações as quais o BMA deve ter acesso para operar conforme o projeto. No caso específico do BMA, as permissões de acesso necessárias são "*RECEIVE BOOT COMPLETED*" que permite ao aplicativo receber uma mensagem gerada pelo sistema operacional do Android informando que o processo de inicialização do dispositivo chegou ao fim, "*INTERNET*" que permite ao aplicativo se conectar através de interfaces de comunicação de dados, sejam estas disponibilizadas pela operadora através de planos de dados ou então através de conexões a redes Wi-Fi, "*VIBRATE*" que permite o aplicativo disparar ações que façam o telefone vibrar, que no caso do aplicativo desenvolvido são utilizadas no momento de gerar notificações ao usuário, e

"*ACCESS FINE LOCATION*" que permite ao aplicativo obter dados de geolocalização através de GPS, quando disponível, ou então através de triangulação de torres de telefonia celular e através de informações de geolocalização presentes em redes Wi-Fi.

É importante notar que as permissões mencionadas anteriormente são de aceitação obrigatória por parte do usuário. A aceitação é solicitada no momento da instalação do aplicativo. O usuário não pode aceitar algumas das permissões e negar outras, ou aceita todas ou simplesmente a instalação do aplicativo é abortada.

Na listagem 1 podem ser vistas as entradas do arquivo *AndroidManifest.xml* que solicitam o acesso às permissões mencionadas anteriormente.

```
1 <uses-permission
  android:name="android.permission.RECEIVE_BOOT_COMPLETED"></uses-
  permission>
2 <uses-permission  android:name="android.permission.INTERNET"></uses-
  permission>
3 <uses-permission   android:name="android.permission.VIBRATE"></uses-
  permission>
4 <uses-permission
  android:name="android.permission.ACCESS_FINE_LOCATION"></uses-
  permission>
```

Listagem 1 Solicitação das permissões através do arquivo *AndroidManifest.xml*.

É importante notar que as solicitações de permissão podem ser adicionadas diretamente através de entradas como as demonstradas na listagem 1 ou, caso o desenvolvedor não queira editar o arquivo manualmente, existe a opção de utilizar a interface específica da IDE Eclipse para esse fim, conforme demonstrado na figura 3.

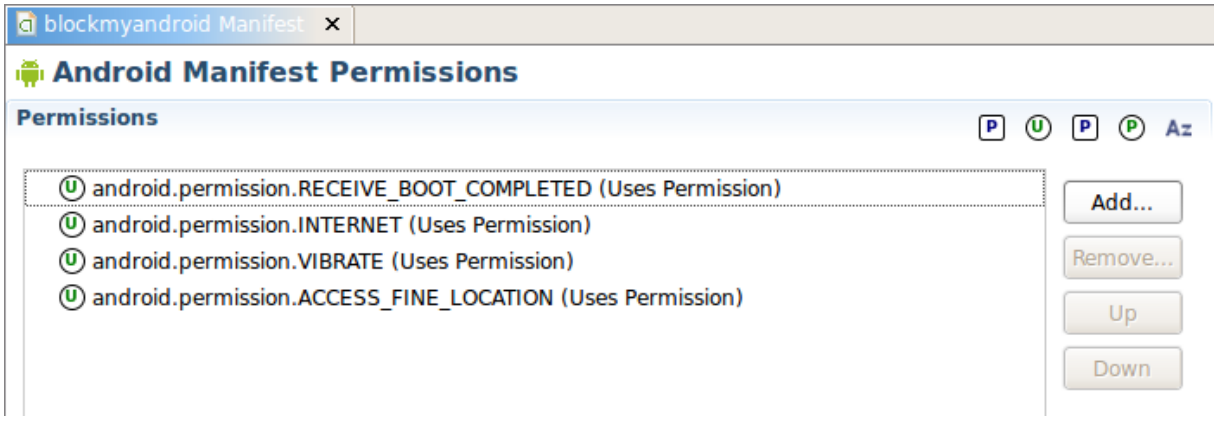

Figura 3: Interface da IDE Eclipse para adição de permissões em projetos Android.

As permissões solicitadas no momento do desenvolvimento do projeto se traduzem em informações exibidas ao usuário, nos detalhes do aplicativo, quando este estiver instalado no dispositivo. A figura 4 mostra em detalhes como essas informações são exibidas ao usuário.

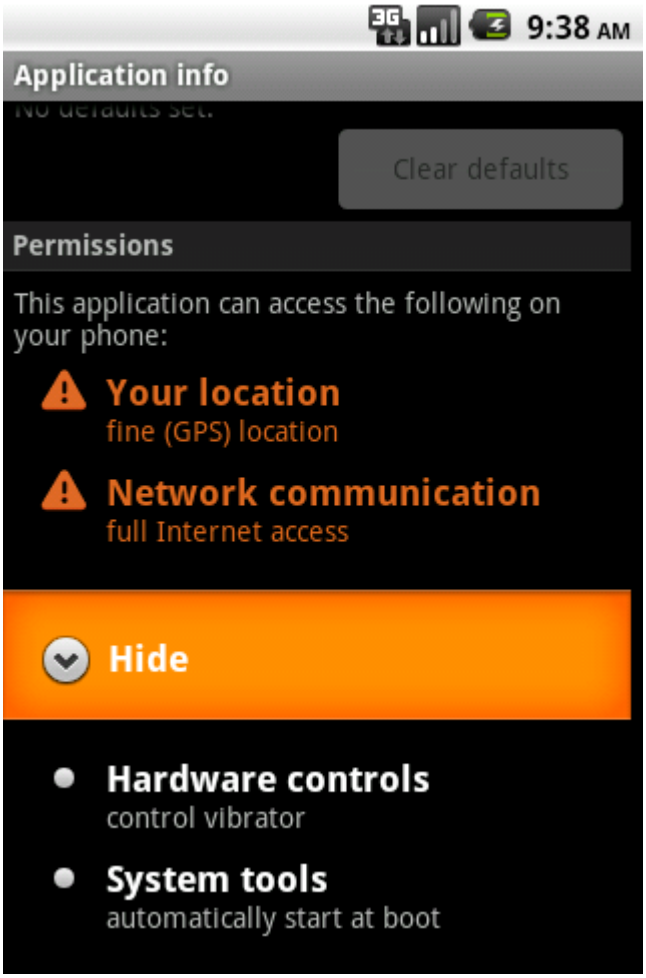

Figura 4: Permissões do aplicativo BMA.

No caso do BMA, é em função da permissão "*RECEIVE BOOT COMPLETED*" que o agendamento do *Service* acontece. No arquivo *AndroidManifest.xml* consta uma entrada que informa ao sistema que, assim que for detectado o final do processo de inicialização do dispositivo, a classe *BMAScheduleService* deve ser executada. A classe *BMAScheduleService* na verdade é um *Broadcast Receiver* que, ao receber a notificação do sistema operacional do Android sobre o final do processo de inicialização do dispositivo, executa o método *schedule* da própria classe, passando por parâmetro o tempo e sua grandeza, expressa em minutos ou segundos, para o agendamento da execução do *Service*.

Na listagem 2 podem ser vistas as entradas do arquivo *AndroidManifest.xml* que mapeiam a execução da classe *BMAScheduleService* quando for detectado o final do processo de inicialização do dispositivo e na listagem 3, a forma através da qual a classe *BMAScheduleService* agenda a execução do *Service*.

```
1 <receiver android:name="BMAScheduleService">
2     <intent-filter>
3           <action
  android:name="android.intent.action.BOOT_COMPLETED" />
4           <action android:name="android.intent.action.DEFAULT" />
5     </intent-filter>
6 </receiver>
```

Listagem 2 Mapeamento entre o final do processo de inicialização do dispositivo e a execução da classe *BMAScheduleService*.

```
1 public class BMAScheduleService extends BroadcastReceiver {
2     public void onReceive(Context context, Intent intent) {
3           Log.i(LABEL, "scheduling the execution of service..");
4           Log.i(LABEL, " ");
5           schedule(context, 15, "seconds");
6     }
```

Listagem 3 Classe *BMAScheduleService* instanciando o agendamento da execução do *Service*.

O tempo passado por parâmetro para o método *schedule* no momento em que o dispositivo é inicializado é de quinze segundos. Esse tempo foi adotado para que o dispositivo tenha tempo de se conectar à Internet através das interfaces de comunicação de dados disponíveis, sejam estas disponibilizadas pela operadora de telefonia ou através de conexões à redes Wi-Fi. Depois de executado pela primeira vez, o *Service* agenda a sua próxima execução com base no tempo configurado pelo usuário na *Activity* de configuração.

O método *schedule*, quando executado, gera um *Intent* personalizado que, no caso do aplicativo desenvolvido, é o *Intent* "*BLOCKMYANDROID*". Depois de gerar o *Intent*, o método agenda sua execução para o tempo atual mais o tempo recebido por parâmetro no momento da chamada do método.

O *Intent* "*BLOCKMYANDROID*", por sua vez, está mapeado no arquivo *AndroidManifest.xml* de forma que, quando o sistema receber a notificação do *Intent*, é chamada a execução da classe *BMAService*. A classe *BMAService* é na verdade o *Service* em si, que ao ser instanciado, executa o método *service* da própria classe.

Na listagem 4 pode se ver a forma através da qual é feito o mapeamento entre o *Intent* "*BLOCKMYANDROID*" e a execução da classe *BMAService*.

```
1 <service android:name="BMAService">
2     <intent-filter>
3           <action android:name="BLOCKMYANDROID" />
4           <category
  android:name="android.intent.category.DEFAULT" />
5     </intent-filter>
6 </service>
```

Listagem 4 Mapeamento entre o *Intent "BLOCKMYANDROID"* e a execução da classe *BMAService*.

O método *service*, por sua vez, depois de executar todas as rotinas desenvolvidas, checa a configuração de intervalo de sincronização definida pelo usuário e executa novamente o método *schedule* da classe *BMAScheduleService* passando por parâmetro o tempo definido pelo usuário através da interface de configuração do sistema. É dessa forma que acontece o agendamento e a execução do *Service* dentro de determinados intervalos de tempo.

É importante notar que toda parte de agendamento e execução do *Service* é feita sem nenhum tipo de intervenção do usuário. Isso garante a execução do *Service* mesmo em eventuais casos nos quais o usuário queira, por algum motivo, que o *Service* não seja executado.

Caso o *Service* seja finalizado pelo sistema operacional do Android, por aplicativos terceiros ou pelo próprio usuário, através do submenu de configuração "*Running services*", única forma pela qual o usuário pode manualmente finalizar quaisquer serviços, através do método *onDestroy* da classe *Service*, que é o método instanciado exatamente no último momento de execução do *Service*, um novo agendamento do *Service* é realizado através do método *schedule* da classe *BMAScheduleService* para o tempo da finalização do *Service* mais um segundo.

Na listagem 5 é possível observar a forma através da qual o novo agendamento é realizado através do método *schedule* da classe *BMAScheduleService*.

```
1 public void onDestroy() {
2     super.onDestroy();
3     Log.i(LABEL, "BMAService onDestroy()");
4     Log.i(LABEL, " ");
5     bmaScheduleService.schedule(getBaseContext(), 1, "second");
6 }
```

Listagem 5 Reagendamento da execução do *Service*.

Na prática isso significa dizer que, caso o usuário finalize o *Service* às 21 horas, 50 minutos e 05 segundos, às 21 horas, 50 minutos e 06 segundos o *Service* será novamente inicializado.

Esse processo torna virtualmente impossível finalizar o *Service* de forma definitiva. Virtualmente porque de fato o *Service* foi finalizado e ficou parado durante um segundo mas, após passado esse tempo, ele volta a sua execução normal.

4.3 Configuração do sistema BMA

A configuração do sistema BMA se dá toda através de uma *Activity* específica para esse fim. É através dessa *Activity* que são armazenadas as credenciais de acesso compostas por usuário e senha, o intervalo de tempo entre as sincronizações e é habilitado o modo administrativo do dispositivo. A figura 5 mostra a *Activity* de configuração do sistema BMA.

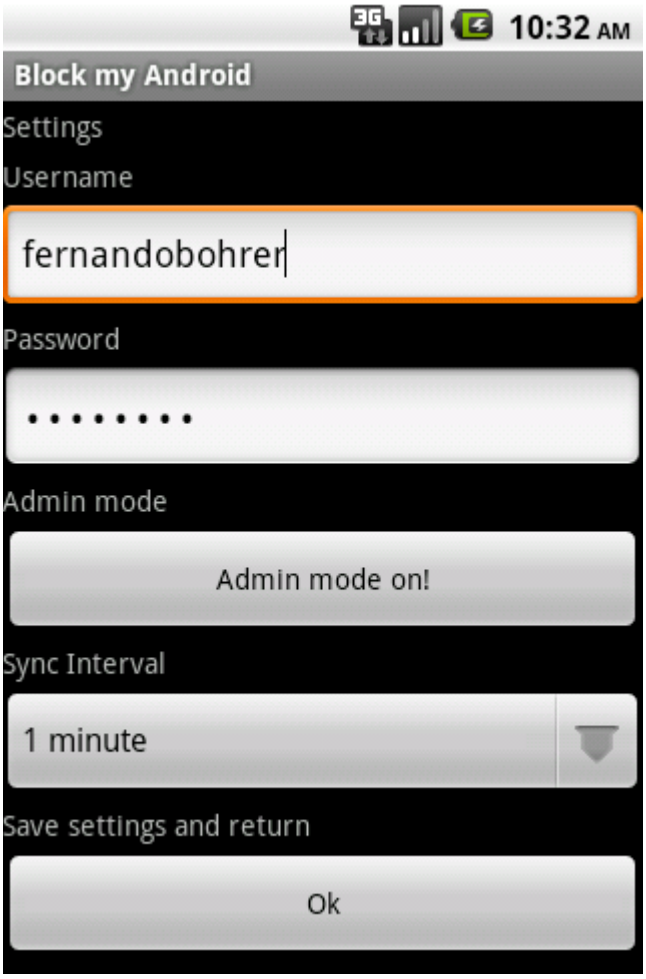

Figura 5: *Activity* de configuração do sistema BMA

Os dados locais do dispositivo são armazenados através de uma classe da plataforma Android chamada *SharedPreferences* que viabiliza o armazenamento de informações da

aplicação em tempo de execução. Os dados salvos são de tipos básicos como inteiros, *strings* e booleanos, e na prática se traduzem em informações como nome de usuário, senha (armazenada na forma de texto claro para utilização local e de forma cifrada através da função de *hash* SHA-512 para utilização via Internet), dados de geolocalização, estados intermediários da aplicação, entre outros.

O armazenamento das informações através da classe *SharedPreferences* não representa nenhum risco à segurança da aplicação visto que na plataforma Android, cada aplicativo, ao ser instalado, recebe um código identificador e as *SharedPreferences* de uma aplicação estão isoladas dos outros aplicativos exatamente em função desse código identificador. Na prática isso significa dizer que as informações armazenadas através da classe *SharedPreferences* estão acessíveis apenas para a aplicação que as salvou.

Existem *flags* específicas que, ativadas no momento do armazenamento das informações, permitem o compartilhamento das informações salvas entre aplicações diferentes mas no sistema desenvolvido, essa prática não foi adotada.

Afim de forçar a configuração do aplicativo, quando a *Activity* principal é iniciada pela primeira vez, é realizada uma checagem que indica se o sistema já foi configurado. Caso o sistema ainda não tenha sido configurado, um *Dialog* informando a situação é exibido e é solicitado ao usuário se este deseja realizar a configuração naquele momento. Por questões de projeto, apenas um botão de confirmação integra o *Dialog*, o que força o usuário a realizar a configuração. A figura 6 mostra o *Dialog* solicitando ao usuário que realize a configuração do sistema BMA.

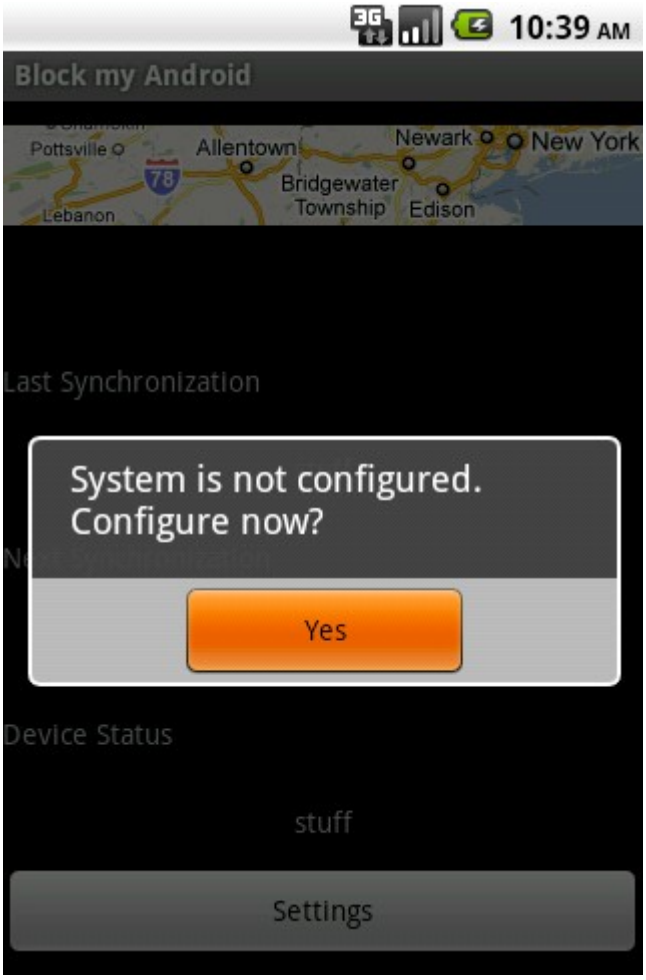

Figura 6: *Dialog* exibido enquanto a configuração do sistema BMA não for realizada.

Ao fechar a *Activity* de configuração, seja através de um evento de tecla do tipo *Back* gerado pelo teclado do dispositivo ou através do botão *Ok* da própria *Activity*, as configurações são automaticamente salvas e a *flag configured* recebe um booleano de valor "*true*" indicando que a configuração foi realizada. Dessa forma, na próxima vez em que a *Activity* principal for iniciada, o *Dialog* solicitando a configuração do sistema não será exibido.

4.4 Modo administrativo

O modo administrativo é na verdade uma API disponibilizada a partir do Android versão 2.2 e que viabiliza o acesso a uma série de funcionalidades necessárias para o correto funcionamento da aplicação conforme ela foi projetada.

Entre as funcionalidades disponibilizadas estão a capacidade de apagar por completo todos os dados do dispositivo sem nenhum tipo de confirmação por parte do usuário, alterar

remotamente a senha de desbloqueio do dispositivo de forma que só poderá desbloquear o aparelho quem conhecer a senha, configurar os tipos de senha que se utiliza no dispositivo, as quais podem ser senhas alfanuméricas, PINs (*Personal Identification Numbers*) ou padrões de conexão entre pontos, monitorar as tentativas de *login* de forma que se possa tomar ações de acordo com o número de falhas e bloquear o dispositivo forçando a quem o detenha, a inserir novamente a senha de desbloqueio para voltar a usar o aparelho.

Por questões de projeto do Android em si não é possível habilitar o modo administrativo sem a expressa aceitação do usuário. Como e quando a confirmação é solicitada depende da implementação do aplicativo. No BMA, sempre que a *Activity* principal é inicializada, é realizada uma checagem que indica se o modo administrativo está habilitado ou não. Caso o modo administrativo não esteja habilitado, um *Dialog* informando a situação é exibido e é solicitado ao usuário se este deseja habilitar o modo naquele momento. Por questões de projeto, apenas um botão de confirmação integra o *Dialog*, o que força o usuário a habilitar o modo administrativo. A figura 7 mostra o *Dialog* solicitando ao usuário que habilite o modo administrativo do sistema BMA.

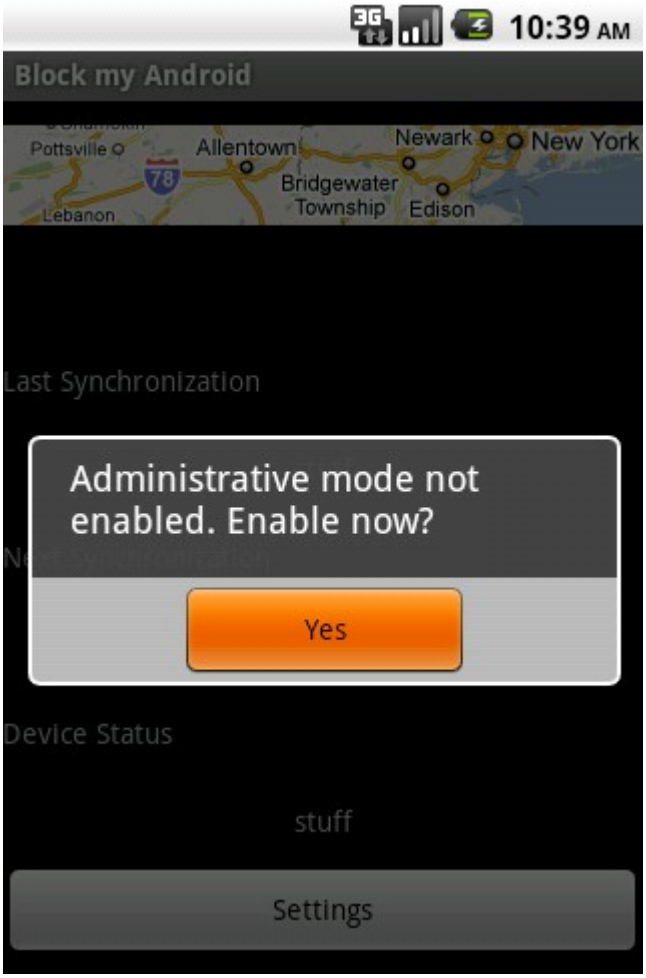

Figura 7: *Dialog* exibido enquanto o modo administrativo do sistema BMA não for habilitado.

No momento em que o usuário clica no botão "*Yes*" da figura 7, ele é direcionado para a tela de confirmação da habilitação do modo administrativo que exibe informações sobre as permissões extras que o sistema passará a ter. A figura 8 mostra a tela de confirmação de habilitação do modo administrativo.

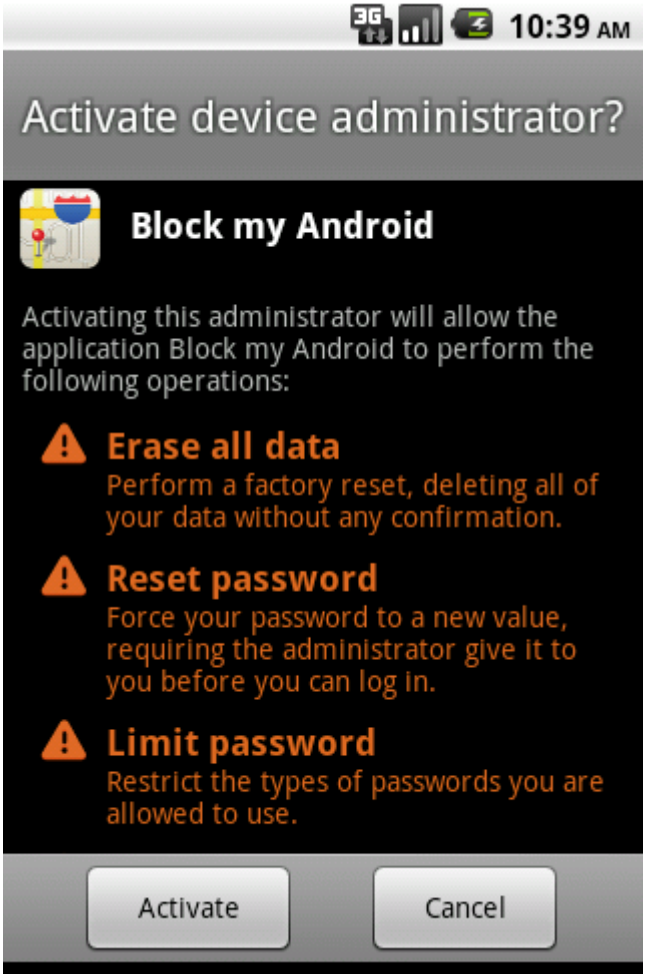

Figura 8: Tela de confirmação da habilitação do modo administrativo.

É importante notar que as permissões extras habilitadas pelo modo administrativo são permissões diferentes das solicitadas no momento da instalação do aplicativo. Como a habilitação do modo administrativo normalmente é facultativa e, no caso do BMA, é forçada através do *Dialog* demonstrado anteriormente, as permissões do modo administrativo não são inseridas no arquivo *AndroidManifest.xml* mas sim em um arquivo xml diferente que, no momento da habilitação do modo, é referenciado e determina a quais novas permissões o sistema passará a ter acesso.

A estrutura do arquivo xml com as permissões extras habilitadas pelo modo administrativo pode ser observada na listagem 6.

```
1 <device-admin
  xmlns:android="http://schemas.android.com/apk/res/android">
2     <uses-policies>
3           <limit-password />
4           <watch-login />
5           <reset-password />
6           <force-lock />
7           <wipe-data />
8     </uses-policies>
9 </device-admin>
```

Listagem 6 Estrutura do arquivo xml com as permissões habilitadas pelo modo administrativo.

Uma vez habilitado o modo administrativo, o aplicativo passa a ter pleno acesso às funcionalidades anteriormente citadas podendo então bloquear a tela do dispositivo, alterar senhas de acesso e outras funcionalidades mais que compõem o escopo do projeto.

O modo administrativo, enquanto estiver habilitado, impossibilita a remoção do aplicativo responsável pela sua habilitação. Para desinstalar aplicativos que habilitaram o modo administrativo, é preciso primeiro desabilitar o modo administrativo daquele aplicativo para depois proceder com a remoção.

Afim de garantir que o BMA não possa ser desinstalado dos dispositivos nos quais foi instalado, não há no programa, nenhuma forma de desabilitar o modo administrativo. Assim, o usuário ao instalar o aplicativo em seu dispositivo é forçado a habilitar o modo administrativo em função dos *Dialogs* que constantemente são exibidos na tela principal do programa e, uma vez esse modo habilitado, não há nenhuma forma de desabilitá-lo, o que passa a impedir, em caráter definitivo a remoção do aplicativo.

Caso o usuário tente remover o BMA, a figura 9 é exibida, informando o insucesso na tentativa de desinstalação do aplicativo.

É importante notar que com o modo administrativo habilitado, atualizações do aplicativo podem ser recebidas e instaladas sem problemas, apenas a remoção é que deixa de ser possível.

A única forma de remover o aplicativo do dispositivo nesse cenário é através de um "*Factory Reset*" do aparelho que nada mais é do que um processo que restaura as configurações do aparelho para o mesmo estado em que estavam quando o dispositivo foi integrado pelo seu fabricante. Essa forma de remoção do aplicativo não chega a representar uma ameaça ao modelo de segurança implementado visto que o processo de restauração das

configurações originais do aparelho só é possível mediante autenticação com senha que, em condições normais de uso, só é conhecida pelo proprietário do dispositivo.

Figura 9: Tela informando o insucesso na tentativa de remoção do BMA.

4.5 Comunicação com o servidor da aplicação

A comunicação entre o aplicativo desenvolvido e o servidor da aplicação é toda realizada através da classe *BMANetworkController* que, na prática, implementa as conexões através de *sockets*.

Ao ser criado, pelo método *service* da classe *BMAService*, o objeto da classe *BMANetworkController* realiza a conexão com o servidor através da porta estabelecida. Por questões de projeto as informações de servidor de conexão e porta do serviço não podem ser configuradas através do aplicativo e são estabelecidas no código fonte do projeto. No BMA, o servidor da aplicação tem endereço "*http://fernandobohrer.com*" e a porta na qual a aplicação ouve, no lado do servidor, é a porta 7777 do protocolo TCP (*Transmission Control Protocol*).

O *Service* do aplicativo desenvolvido, ao ser executado, conecta ao servidor e, após a conexão, através do método *connect*, envia as credenciais de acesso compostas por nome de usuário e senha, que é enviada de forma cifrada.

O servidor, por sua vez, ao receber os dados procede com a análise das informações, comparando os valores recebidos com os valores armazenados localmente. Caso as credenciais de acesso não estejam corretas, o servidor retorna um booleano de valor "*false*". Caso as credenciais de acesso estejam corretas, o servidor retorna um booleano de valor "*true*". Caso o *Service* não consiga conectar com o servidor da aplicação por algum motivo qualquer, uma nova tentativa de conexão é agendada para dentro de um minuto.

Em função do retorno do servidor é que o *Service* molda o seu comportamento. Caso o retorno do servidor seja um booleano de valor "*false*", uma notificação é gerada na barra de notificações do sistema informando ao usuário que, ou o nome de usuário ou a senha foram informados de forma incorreta.

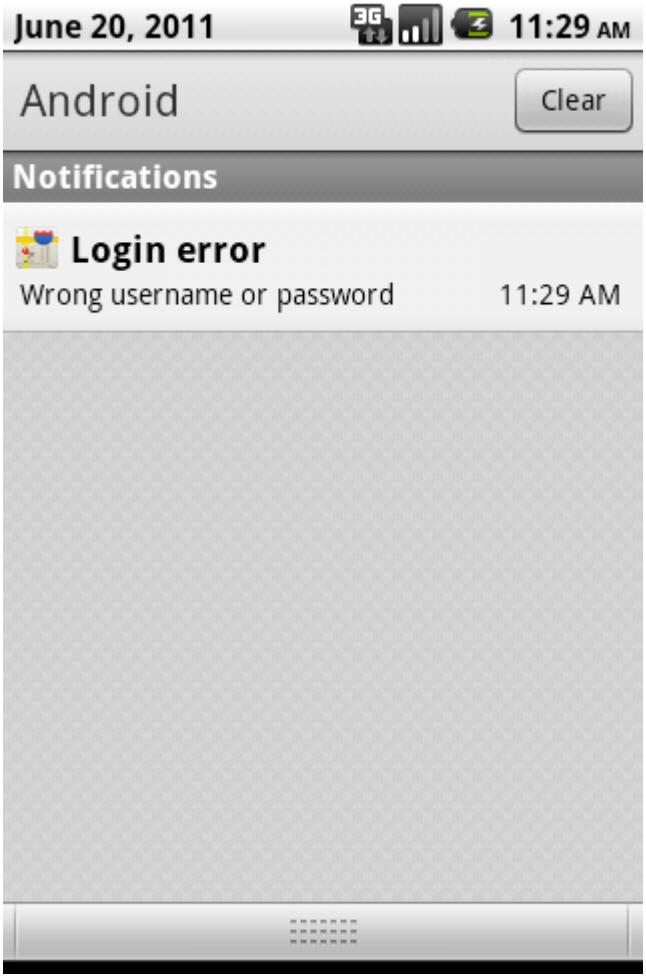

Figura 10: Notificação informando falha na tentativa de *login*.

Caso o retorno do servidor seja um booleano de valor "*true*", o protocolo de checagem continua.

Considerando a possibilidade de o mesmo usuário utilizar o aplicativo em mais de um dispositivo, o protocolo de checagem mapeia a verificação do dispositivo em função de um código gerado aleatoriamente para cada dispositivo no qual o aplicativo é instalado. Dessa forma, a próxima troca de informações entre o aplicativo instalado no dispositivo e o servidor acontece no momento em que o aplicativo envia para o servidor, o nome de usuário configurado e o código identificador do dispositivo.

Com base nesses dois valores enviados pelo aplicativo, o servidor analisa o *status* do dispositivo, retornando para o aplicativo um valor inteiro que representa o status configurado no servidor.

Os possíveis *status* do dispositivo, os códigos de retorno e as respectivas ações desencadeadas estão descritos na tabela 7.

Tabela 7 Possíveis *status* do *smartphone*, seus códigos e as ações que estes desencadeiam.

Status	Código	Ação
OK	*0*	Dados locais são atualizados.
Lost	*1*	Dados locais são atualizados e serviços de geolocalização são ativados.
Stolen	*2*	Dados locais são atualizados e serviços de geolocalização e de bloqueio do dispositivo são ativados.

Com base no valor inteiro retornado pelo servidor, que representa o *status* do dispositivo, é que a aplicação molda o seu comportamento.

Caso o *status* retornando pelo servidor seja de código 0, o método local da classe *BMAService*, *updateData* é executado. O método *updateData* trabalha em conjunto com dois outros métodos da mesma classe, o *getMinutes* e o *getDateTime*.

O método *getMinutes* na verdade retorna o valor, em minutos, para cada opção de seleção do *Spinner* da *Activity* de configuração. No *Spinner* é que se determina o tempo, em minutos ou horas, do intervalo entre as sincronizações. A única maneira de obter e salvar a escolha do usuário, se dá através do método *getSelectedItemPosition* que retorna um inteiro relativo à opção escolhida pelo usuário. Dessa forma, se faz necessário mapear o valor, em minutos, da opção escolhida pelo usuário e é exatamente isso que o método *getMinutes* faz.

O método *getDateTime* é responsável por formatar a data e hora com base na *string* de formatação e, com base no valor recebido por parâmetro, retornar a data e hora atuais ou, data

e horas atuais acrescidas do valor, em minutos, do intervalo de tempo para a próxima sincronização.

Com base nesses dois métodos o método *updateData* salva as informações de data e hora da última sincronização, data e hora da próxima sincronização e *status* do dispositivo. O armazenamento das informações se dá através da classe *SharedPreferences* no momento em que o método *updateData* instancia um novo editor e, com base nesse editor, adiciona as informações. Como os valores salvos nesse caso são apenas *strings* de texto, o método *putString* é utilizado para adicionar as informações. No final do processo, quando todas informações foram adicionadas, o método *commit* é instanciado de forma a salvar em definitivo os dados.

Depois de salvos localmente os dados, o método *closeConnections* da classe *BMANetworkController* é instanciado de forma a finalizar e fechar a conexão com o servidor.

Ainda, no final do processo, o método *schedule* da classe *BMAScheduleService* é instanciado de forma a agendar a próxima execução do *Service*. Nesse caso, o tempo passado como parâmetro para o agendamento da próxima execução do *Service* é o tempo configurado pelo usuário via interface de configuração, transformado para o equivalente tempo em minutos através do método *getMinutes*.

Assim, ao final do processo, caso o retorno do servidor seja de código 0, os dados locais são atualizados, as conexões de dados finalizadas e é agendada a próxima execução do *Service*.

Caso o retorno do servidor seja de código 1, os processos executados são exatamente os mesmos do código de retorno 0, com o detalhe de que antes da execução do método *updateData*, são inicializados os serviços de geolocalização do dispositivo e é enviada para o servidor a localização atual composta por latitude e longitude. Detalhes do funcionamento dos serviços de geolocalização serão vistos na seção 4.6 - Geolocalização.

Ainda, caso o retorno do servidor seja de código 2, os processos executados são exatamente os mesmos do código 1, com o detalhe de que antes da execução dos serviços de geolocalização, a senha de acesso é alterada conforme senha enviada pelo servidor e a tela do dispositivo é bloqueada. Dessa forma, só poderá desbloquear o dispositivo quem conhecer a senha enviada pelo servidor que, antes de ser enviada, deve ter sido previamente cadastrada no portal de administração do sistema. Detalhes do funcionamento dos serviços de troca de senha e bloqueio do dispositivo serão vistos na seção 4.7 - Bloqueio da interface e troca de senha.

4.6 Geolocalização

Caso o status retornado pelo servidor seja de código 1, é preciso localizar geograficamente o dispositivo.

Ao desenvolver aplicativos de geolocalização para a plataforma Android, o desenvolvedor pode optar por entre qual meio deseja obter as informações de localização geográfica. O desenvolvedor pode utilizar GPS, caso o dispositivo ofereça suporte a essa tecnologia, triangulação de torres de telefonia celular e informações de geolocalização presentes em redes Wi-Fi.

Embora a geolocalização através de GPS seja muito precisa e com uma margem de erro de apenas alguns poucos metros, ela só funciona em ambientes ao ar livre, consome muita energia da bateria e tem um tempo de retorno da localização demorado visto que é preciso primeiro sincronizar o dispositivo com os satélites do serviço de GPS.

A geolocalização através da triangulação de torres de telefonia celular e redes Wi-Fi funciona tanto em ambientes ao ar livre bem como em ambientes fechados, responde de forma mais rápida e usa menos energia da bateria para obter as informações de posicionamento. A única contrapartida nesse caso é a precisão que, através desse método, tem uma margem de erro maior do que a obtida através da geolocalização realizada por meio do GPS.

No caso específico do BMA, o modelo de geolocalização foi desenvolvido de forma que, caso o dispositivo conheça sua localização atual com base nas informações obtidas através do GPS, essa será a forma de geolocalização utilizada. Caso o dispositivo não conheça sua localização com base nas informações de GPS, a geolocalização se dará através da triangulação das torres de telefonia celular e através de redes Wi-Fi.

Para solicitar ao dispositivo a sua localização geográfica, a primeira coisa feita foi instanciar um objeto de tipo "*LocationManager*" que nada mais é do que uma referência para o sistema de geolocalização da plataforma Android.

Depois de instanciado o objeto de tipo "*LocationManager*", o próximo passo foi instanciar um objeto de tipo "*LocationListener*" que nada mais é do que uma estrutura através da qual atualizações da localização geográfica são tratadas.

Depois de instanciados os dois objetos mencionados, "*LocationManager*" e "*LocationListener*", foi solicitado à plataforma que esta se localize geograficamente através de um dos provedores de geolocalização. As opções nesse caso são "*GPS_PROVIDER*" que, como o nome da a entender, utiliza como base o sistema de GPS e "*NETWORK_PROVIDER*" que, seguindo a mesma lógica, utiliza como base as conexões de dados, sejam estas

estabelecidas por meio de redes Wi-Fi ou através de conexões de dados disponibilizadas pelas operadoras de telefonia.

É importante notar que o provedor "*NETWORK_PROVIDER*" gerencia automaticamente a localização geográfica do dispositivo seja ela obtida com base nas conexões de dados disponibilizadas pelas operadoras de telefonia ou através de redes Wi-Fi.

No caso do BMA, se obteve atualizações dos dois provedores, "*NETWORK_PROVIDER*" e "*GPS_PROVIDER*", e para isso bastou instanciar o método *requestLocationUpdates* duas vezes, uma para cada provedor. Maiores detalhes sobre o funcionamento do método de localização geográfica do dispositivo podem ser vistos na listagem 7.

```
1 public void defineLocationListener() {
2     LocationManager     locationManager     =     (LocationManager)
  this.getSystemService(Context.LOCATION_SERVICE);
3     LocationListener locationListener = new LocationListener() {
4         //...
5     };
6
  locationManager.requestLocationUpdates(LocationManager.NETWORK_PROVI
  DER, 0, 0, locationListener);
7
  locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
  0, 0, locationListener);
8 }
```

Listagem 7 *Location Listener* do serviço de localização geográfica do dispositivo.

É importante notar o segundo parâmetro enviado para o método *requestLocationUpdates*. Esse parâmetro permite especificar o intervalo de tempo entre as atualizações das informações de localização. O valor zero, passado por parâmetro nesse caso, faz com que o sistema cheque constantemente a sua localização geográfica.

Depois de instanciados os métodos *requestLocationUpdates* para cada um dos provedores, o próximo passo é obter a localização atual do dispositivo.

Como a checagem da localização geográfica é feita de forma constante em função do parâmetro zero passado ao método *requestLocationUpdates*, não é necessário solicitar a localização e aguardar o cálculo da localização atual do dispositivo. Esse procedimento seria necessário caso o intervalo de tempo entre as checagens fosse maior do que zero mas, como no caso do BMA a checagem é constante, basta utilizar o método *getLastKnowLocation*.

O método *getLastKnowLocation* irá retornar a última localização geográfica conhecida do dispositivo e, através do retorno desse método, é possível obter os valores da latitude e longitude nos quais o dispositivo está geograficamente localizado.

Como o BMA solicita atualizações de dois provedores diferentes, "*GPS_PROVIDER*" e "*NETWORK_PROVIDER*", no momento em que é solicitada a última localização conhecida através do método *getLastKnowLocation* isso é levado em conta. Dessa forma, afim de otimizar as informações de localização, primeiro o sistema tenta obter a sua localização geográfica com base no provedor "*GPS_PROVIDER*". A decisão de tentar obter a localização primeiro com base no provedor "*GPS_PROVIDER*" se deve ao fato de que, se disponível, essa informação tende a ser mais precisa do que se comparada à precisão obtida através do provedor "*NETWORK_PROVIDER*".

Caso o sistema não consiga determinar a sua localização geográfica com base nas informações de GPS é acionada a segunda opção que, no caso, é a obtenção das informações de localização geográfica com base nas informações de alguma rede de dados disponível através do provedor "*NETWORK_PROVIDER*".

Detalhes do processo de obtenção das informações de geolocalização podem ser vistos na listagem 8.

```
1  public Location getLocation() {
2      LocationManager     locationManager     =     (LocationManager)
   this.getSystemService(Context.LOCATION_SERVICE);
3      Location                        location                        =
   locationManager.getLastKnownLocation(LocationManager.GPS_PROVIDER);
4      if (location == null) {
5            location                                                  =
   locationManager.getLastKnownLocation(LocationManager.NETWORK_PROVIDE
   R);
6            if (location == null) {
7                  return null;
8            }
9      }
10     return location;
11 }
```

Listagem 8 Método de obtenção da localização geográfica atual do dispositivo.

Uma vez obtida a localização atual do dispositivo, basta utilizar os métodos *getLatitude* e *getLongitude* para obter, respectivamente, as informações de latitude e longitude.

O retorno dos métodos *getLatitude* e *getLongitude* são valores, expressos na forma de ponto flutuante que, depois de convertidos para *strings*, são enviados para o servidor da aplicação informando onde o dispositivo está localizado geograficamente naquele instante de tempo.

4.7 Troca de senha e bloqueio do dispositivo

Caso o status retornado pelo servidor seja de código 2, além de geolocalizar o dispositivo e enviar as informações de localização geográfica para o servidor, é preciso bloquear e alterar a senha local do aparelho, utilizada para seu desbloqueio.

Nesse caso, no momento em que as informações de localização geográfica compostas por latitude e longitude são enviadas para o servidor, o retorno do servidor é uma *string* de texto que é, na verdade, a nova senha para o desbloqueio do aparelho.

A alteração da senha de desbloqueio e o bloqueio do dispositivo são possíveis exatamente em função de o aplicativo já contar com o modo administrativo habilitado.

Assim, no momento em que o dispositivo receber do servidor a nova senha para seu desbloqueio, basta instanciar o método *resetPassword* da classe *DevicePolicyManager* passando por parâmetro a senha para desbloqueio e a *flag* 0. A *flag* 0, nesse caso, é a *flag* padrão para a operação de troca de senha.

Depois de alterada a senha de desbloqueio, o método *lockNow* da classe *DevicePolicyManager* é instanciado, bloqueando de forma definitiva o dispositivo.

A partir do momento em que o processo acima descrito é executado, o dispositivo fica permanentemente bloqueado sendo necessário conhecer a senha enviada pelo servidor para proceder com o desbloqueio.

Maiores detalhes a respeito da forma como são realizados o bloqueio do dispositivo e a alteração da senha de desbloqueio podem ser vistos na listagem 9.

```
1 devicePolicyManager = (DevicePolicyManager)
  getSystemService(Context.DEVICE_POLICY_SERVICE);
2 componentName = new ComponentName(BMAService.this,
  BMAService.class);
3
```

```
4 devicePolicyManager.resetPassword(password, 0);
5 devicePolicyManager.lockNow();
```

Listagem 9 Método de alteração da senha e bloqueio do dispositivo.

É importante notar que, para os casos onde o usuário já utiliza algum método de desbloqueio do dispositivo seja este composto por PIN ou por um padrão de conexão entre pontos, no momento em que o método *resetPassword* é instanciado, automaticamente o processo de autenticação passa a ser realizado através de senha alfanumérica. Essa configuração poderá ser alterada posteriormente pelo usuário mas, caso o método *resetPassword* seja novamente instanciado, a configuração de desbloqueio voltará a ser setada para desbloqueio via senha alfanumérica.

A figura 11 mostra em detalhes a tela de autenticação na qual deverá ser informada a senha retornada pelo servidor afim de desbloquear o dispositivo.

Figura 11: Tela de autenticação para desbloqueio do dispositivo.

Caso o usuário informe de forma incorreta a senha por 5 vezes consecutivas, um contador de 30 segundos é inicializado, impedindo a digitação da senha durante esse tempo.

Vale mencionar ainda que a partir do momento em que o dispositivo se encontra bloqueado, a única forma de realizar o desbloqueio é através da senha enviada pelo servidor.

É importante mencionar também que desligar e ligar o aparelho via remoção da bateria, única opção para desligamento nesse caso, não altera em nada o status de bloqueado do dispositivo.

4.8 *Activity* principal do sistema BMA

A *Activity* principal do aplicativo permite ao usuário obter informações básicas do sistema como data e hora da última sincronização, data e hora da próxima sincronização e o *status* atual do *smartphone*, retornado pelo servidor. Todos os dados apresentados na *Activity* principal são carregados em tempo de execução através da classe *SharedPreferences*.

Além disso, na tela principal existe um botão "*Settings*" que inicializa a *Activity* de configuração do sistema.

A figura 12 demonstra em detalhes a *Activity* principal do sistema.

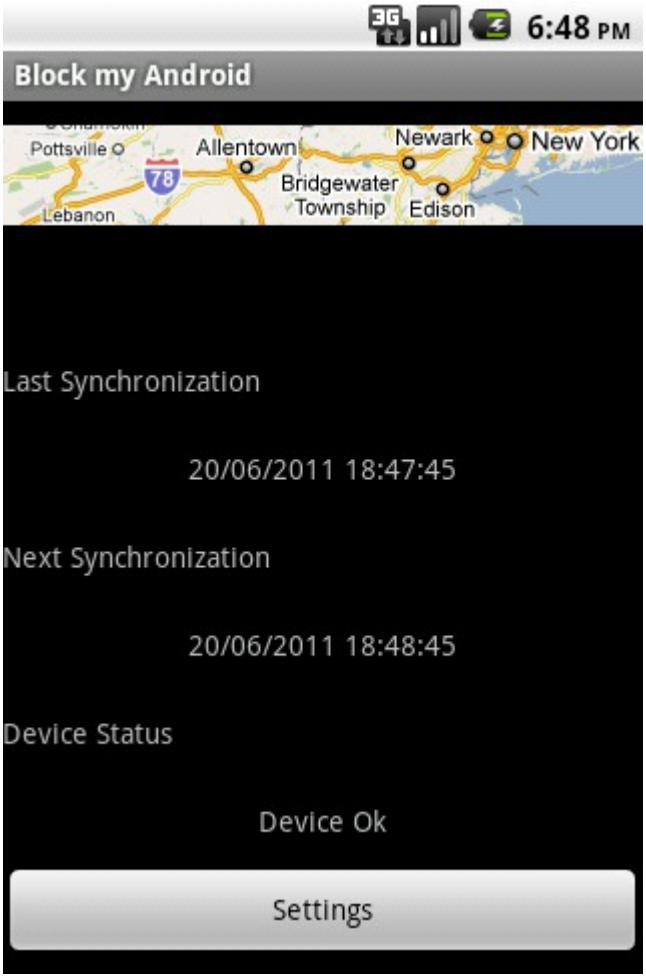

Figura 12: *Activity* principal do sistema BMA.

4.9 Trabalhos futuros

Como forma de dar continuidade ao presente trabalho, algumas funcionalidades que poderão ser implementadas futuramente estão detalhadas a seguir.

4.9.1 Implementação de solicitação de senha para alteração das configurações

Da forma como o sistema está desenvolvido até o momento, o usuário pode configurar o intervalo entre as sincronizações para um dos seguintes tempos: 1 minuto, 5 minutos, 10 minutos, 20 minutos, 30 minutos, 1 hora, 3 horas, 6 horas, 12 horas e 24 horas.

Quem determina o intervalo entre as sincronizações é o próprio usuário ao realizar a configuração através da *Activity* desenvolvida para esse fim.

Uma ideia que pode ser posta em prática através de trabalhos futuros é a implementação de um sistema de autenticação que só permita alterar as configurações do sistema no dispositivo, quem conhecer a senha criada especificamente para esse fim.

4.9.2 Comunicação de dados entre o dispositivo móvel e o servidor da aplicação realizada de forma criptografada

Da forma como o sistema está desenvolvido até o momento, a comunicação realizada entre o servidor da aplicação e os dispositivos móveis é realizada através de *sockets*.

Uma ideia que pode ser posta em prática através de trabalhos futuros é a implementação de um novo modelo de comunicação onde todos os dados trafegados, transitem de forma criptografada entre o dispositivo móvel e o servidor através de conexões de dados seguras.

4.9.3 Desenvolvimento de um portal web

Da forma como o sistema está desenvolvido até o momento, o servidor da aplicação envia as informações de status do dispositivo para o aplicativo e este molda o seu comportamento com base nas informações recebidas.

No entanto, na prática, não há um portal web onde os usuários alteram os status dos dispositivos. Os testes e validações do sistema foram realizados através de edição manual de arquivos de texto onde constam as informações analisadas e enviadas pelo servidor para o aplicativo.

Uma ideia que pode ser posta em prática através de trabalhos futuros é o desenvolvimento de um portal web que permita gerenciar os dispositivos, oferecendo aos usuários do sistema uma experiência de uso agradável e permitindo que estes rastreiem a localização do seu dispositivo em função do tempo.

4.9.4 Criação de contas através do próprio aplicativo

Da forma como o sistema está desenvolvido, a criação de contas de usuário precisa, necessariamente, ser realizada no servidor da aplicação.

Uma ideia que pode ser posta em prática através de trabalhos futuros é o desenvolvimento da funcionalidade que permite criar as contas através dos dispositivos móveis, checando a existência da conta no servidor e criando, caso não exista.

5 CONSIDERAÇÕES FINAIS

Como pode ser observado ao longo desse trabalho, o mercado de telefonia não para de crescer. A cada ano são atingidos novos recordes de vendas de dispositivos móveis de comunicação e estes, por sua vez acabam por facilitar cada vez mais a vida cotidiana das pessoas. É uma tendência global que permite que cada vez mais, as pessoas possam se comunicar entre si.

Crescente também é a procura por dispositivos que agregam diversas funcionalidades em um único aparelho fazendo com que os *smartphones* passem a chegar a um, cada vez maior, número de consumidores.

Dentre as plataformas de software que são executadas nesses *smartphones*, a plataforma Android tem sido a que tem atraído a atenção do público consumidor. Experiência agradável de uso e facilidades disponíveis têm sido os principais motivos para que isso aconteça.

A procura cada vez maior por esse tipo de dispositivo e a crescente demanda por comunicação rápida, instantânea e em qualquer lugar, tem feito com que as pessoas armazenem, cada vez mais, informações importantes em seus *smartphones*. A perda ou roubo de tais dispositivos pode permitir que outras pessoas tenham acesso às informações armazenadas. Situações como essas geralmente culminam em prejuízos financeiros e/ou pessoais.

Em virtude dessa nova realidade, do risco associado à perda ou roubo de um dispositivo como um *smartphone* e do potencial vazamento de informações que essa perda ou roubo pode trazer é que o presente trabalho foi elaborado.

O presente trabalho procurou demonstrar a forma através da qual se desenvolve aplicativos para a plataforma de telefonia móvel Android e, dessa maneira, pode ser utilizado como referência para o desenvolvimento de novos softwares para a plataforma.

REFERÊNCIAS

ANDROID DOCUMENTATION. **Android Documentation: The Developer's Guide.** Disponível em: <http://developer.android.com/guide/index.html>. Acesso em 22 set. 2010.

ANDROID DOCUMENTATION 2. **Android Documentation: Installing the SDK.** Disponível em: <http://developer.android.com/sdk/installing.html>. Acesso em 16 nov. 2010.

ANDROID MARKET. **Android Market: Processing orders and receiving payouts.** Disponível em: <http://market.android.com/support/bin/answer.py?hl=en&answer=137997>. Acesso em 16 nov. 2010.

APPLE. **iOS 4**. Disponível em: <http://www.apple.com/iphone/ios4/>. Acesso em 14 nov. 2010.

CANALYS. **Apple takes the lead in the US smart phone market with a 26% share.** With a 33% share, Nokia is still the leading vendor worldwide, but Android-based smart phones grab a quarter of the market. Disponível em: <http://www.canalys.com/pr/2010/r2010111.html>. Acesso em 16 nov. 2010.

CIA. **The World Factbook**. Disponível em: <https://www.cia.gov/library/publications/the-world-factbook/geos/xx.html>. Acesso em 19 jul. 2011.

EXAME. **A próxima fronteira do software.** Apple, Google e Microsoft se armam para uma nova grande batalha: o controle dos telefones celulares. Disponível em: <http://app.exame.abril.com.br/revista-exame/edicoes/0914/tecnologia/noticias/a-proxima-fronteira-do-software-m0154707>. Acesso em 14 nov. 2010.

ITU. **The World in 2010: ICT facts and figures.** Disponível em: <http://www.itu.int/ITU-D/ict/material/FactsFigures2010.pdf>. Acesso em 07 nov. 2010.

LECHETA, R. R. **Google android**: aprenda a criar aplicações para dispositivos móveis com o Android SDK. São Paulo: Novatec, 2010. 608 p.

MEIKE, B. et al. **Desenvolvimento de aplicações android**. São Paulo: Novatec, 2009. 376 p.

PCMAGAZINE. **Smartphone definition from PC Magazine Encyclopedia.** Disponível em: <http://www.pcmag.com/encyclopedia_term/0,2542,t=Smartphone&i=51537,00.asp>. Acesso em 07 nov. 2010.

STEVENS, H.; PETTEY, C. **Gartner Says More than 1 Billion PCs In Use Worldwide and Headed to 2 Billion Units by 2014.** Analysts to provide more detailed analysis on the state of the PC industry at the Gartner Hardware Briefing, June 25, in Egham, U.K. STAMFORD, Conn., June 23, 2008. Disponível em: <http://www.gartner.com/it/page.jsp?id=703807>. Acesso em 07 nov. 2010.

SYMBIAN. **The Symbian Platform.** Disponível em: <http://www.symbian.org/symbian-platform>. Acesso em 14 nov. 2010.

TOTAL TELECOM. **Number of phones exceeds population of world.** Disponível em: <http://www.totaltele.com/view.aspx?ID=464922>. Acesso em 19 jul. 2011.

TUDOR, B.; PETTEY, C. **Gartner Says Worldwide Mobile Phone Sales Grew 35 Percent in Third Quarter 2010; Smartphone Sales Increased 96 Percent.** Disponível em: <http://www.gartner.com/it/page.jsp?id=1466313>. Acesso em 16 nov. 2010.

ZHENG, P.; NI, L. M. **Smart phone and next generation mobile computing.** San Francisco: Elsevier, 2006. 551 p.