UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO – BACHARELADO

ANDROID MAPBR: UM ESTUDO SOBRE A PLATAFORMA ANDROID COM FOCO NA MANIPULAÇÃO DE MAPAS USANDO INTERFACES DE PROGRAMAÇÃO DE APLICATIVOS DO GOOGLE

MARCO AURÉLIO DE OLIVEIRA WEISS

BLUMENAU
2008

2008/2-15

MARCO AURÉLIO DE OLIVEIRA WEISS

ANDROID MAPBR: UM ESTUDO SOBRE A PLATAFORMA ANDROID COM FOCO NA MANIPULAÇÃO DE MAPAS USANDO INTERFACES DE PROGAMAÇÃO DE APLICATIVOS DO GOOGLE

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

Prof. Dalton Solano dos Reis, M.Sc. - Orientador

BLUMENAU
2008

2008/2-15

ANDROID MAPBR: UM ESTUDO SOBRE A PLATAFORMA ANDROID COM FOCO NA MANIPULAÇÃO DE MAPAS E ROTOS USANDO INTERFACES DE PROGRAMAÇÃO DE APLICATIVOS DO GOOGLE

Por

MARCO AURÉLIO DE OLIVEIRA WEISS

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

Presidente:	Prof. Dalton Solano dos Reis, M.Sc. – Orientador, FURB
Membro:	Prof. Paulo César Rodacki Gomes – FURB
Membro:	Prof. Mauro Marcelo Mattos – FURB

Blumenau, 16 de Dezembro de 2008

AGRADECIMENTOS

A Deus, por agraciar-me com serenidade na busca e conclusão deste objetivo.

À minha família, que diretamente ou indiretamente me guiou nesta caminhada.

À minha namorada, pela compreensão em relação a minha ausência para realização do trabalho e pelo apoio e incentivo ao mesmo.

Aos meus amigos, também pelo apoio e incentivo para conclusão do trabalho.

Ao meu orientador, Dalton Solano dos Reis, por ter acreditado na conclusão deste trabalho e por ter se dedicado para possibilitar isso.

Os bons livros fazem “sacar” para fora o que a pessoa tem de melhor dentro dela.

Lina Sotis Francesco Moratti

RESUMO

No presente trabalho são descritas a fundamentação teórica, a especificação e a implementação de uma aplicação desenvolvida em Java que contempla a utilização de classes, conceitos e recursos da plataforma Android. Os *layouts* das classes de atividade estão descritos na sintaxe XML e as ações disponibilizadas pela aplicação seguem o paradigma de intenções proposto em Android. O trabalho também faz uso de interfaces de programação de aplicativos disponibilizadas pela Google, que incorporadas à aplicação buscam portar recursos relacionados à visualização de mapas e manipulação de marcadores para o ambiente móvel do sistema operacional Android. A aplicação foi desenvolvida usando a API do Google Maps e dessa forma, comporta-se de maneira similar ao mapa do Google.

Palavras-chave: Android. APIs Google. Logística. Mobilidade.

ABSTRACT

In this present work are described the theoretical fundamentation basis, the especification and implementation of a software developed in java which encompasses the use of classes, concepts and resources of android platform. The activity class's layouts are described in xml syntax and the actions which result from this tool (software) follow the paradigm if intentions proposed by android. This work also makes use of programming interfaces of softwares offered by google, which once built into operational system of android platform, enable the user to have all resources related to map visualization and mark manipulation in his cell phone or mobile. The map application has been developed using the Google Maps API and therefore behaves just a like a Google Map.

Key-words: Android. Google’s APIs. Logistic. Mobility.

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

ADB - *Android Debug Bridge*

ADT - Android *Development Tools*

ADT - *Android Development Tools*

ALP - *Access Linux Platform*

APIs - *Application Programming Interfaces*

BSD - *Berkeley Software Distribution*

GPS - *Global Position System*

OMA - *Open Mobile Alliance*

PC - *Personal Computer*

PDAs - *Personal Digital Assistants*

SDK - *Software Development Kit*

SMS - *Short Message Service*

SO - Sistema Operacional

W3C - *World Wide Web Consortium*

Wi-Fi - *Wireless Fidelity*

XML - *eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

As ferramentas para dispositivos móveis trazem significativas mudanças ao dia-a-dia da população, uma vez que incorporadas a estes conseguem disponibilizar a qualquer momento as informações que o usuário deseja. Dessa forma, ao passo que os sistemas evoluem e facilitam a execução das tarefas diárias, pode-se verificar também que as inovações tecnológicas possibilitam que novas soluções sejam desenvolvidas.

Segundo Neiva (2007), as tecnologias móveis representam uma área de mercado, pesquisa e desenvolvimento que cresce a cada ano como resultado de grandes investimentos. Constata-se também, acima de tudo, que esta ascensão se deve em grande parte ao fato das novas tendências terem sido aceitas sem traumas, como promessa de revolucionar, inicialmente, o dia a dia empresarial e hoje, a sociedade como um todo. Este consentimento em prol dos dispositivos móveis se dá, sobretudo, pelo fato de que as inovações, dentre as quais se destacam as relacionadas à telefonia celular, terem sido incorporadas ao cenário do cotidiano mundial de forma transparente e gradual, ao mesmo tempo em que criaram anseios e necessidades cada vez maiores de novos recursos e soluções.

Para atender a essas novas necessidades e ao número de usuários que cresce de forma acentuada no mundo todo, uma incrível diversidade de aparelhos e programas é lançada a cada dia, juntamente com novos conceitos relacionados a dispositivos portáteis.

Entre os portáteis têm-se também os *Smartphones* e *Personal Digital Assistants* (PDAs). Estes dispositivos estão entre as novidades presentes nos dias atuais e segundo Tecsinapse (2006), surgiram com o intuito de tornar acessível algumas das ferramentas e recursos presentes apenas em um computador pessoal, no ambiente móvel, suprindo necessidades relacionadas ao lazer e ao trabalho. Dentre essas necessidades pode-se citar o acesso ao serviço de caixa postal, conteúdo multimídia e a sítios da Web de forma geral (*blogs*, fóruns, notícias, etc.), jogos, agenda, mapas, consulta e edição de documentos, entre outros. Para suportar todos esses serviços e recursos existentes foi preciso fundamentar a plataforma móvel em um dos pilares mais antigos da computação, o Sistema Operacional[1] (SO).

Diante do exposto, o presente trabalho pretende estudar e pesquisar a plataforma Android, que incorpora o sistema operacional da Google para dispositivos móveis, buscando

reunir as especificações, padrões e recursos que fazem parte do pacote de desenvolvimento de software[2] fornecido com a plataforma. Do mesmo modo, interfaces de programação de aplicativos[3] disponibilizadas pela Google e incorporadas ao Android serão empregadas para portar ao ambiente móvel, através da pesquisa, especificação, modelagem e desenvolvimento de uma aplicação, algumas das funcionalidades existentes no Google Maps, tal como a visualização de mapas.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é disponibilizar uma biblioteca que implemente a visualização de mapas em Android.

Os objetivos específicos do trabalho são:

a) elucidar a plataforma Android frente as demais plataformas móveis existentes;
b) incorporar interfaces de programação de aplicativos disponibilizadas pela Google na ferramenta a ser desenvolvida em Android, buscando portar a visualização e manipulação do movimento, posicionamento e zoom em mapas para o ambiente móvel.

1.2 ESTRUTURA DO TRABALHO

O presente trabalho está estruturado em quatro capítulos. Nesse contexto, o segundo capítulo apresenta a fundamentação teórica necessária para o desenvolvimento do trabalho. Nele são expostos detalhes sobre alguns dos sistemas operacionais disponíveis no mercado, partindo em seguida para a plataforma Android, onde é definida a estrutura do sistema operacional Android, o ciclo de vida e a estrutura das aplicações, a classe de atividade, seu ciclo de vida e métodos, entre outros. O capítulo também mostra características de alguns

---

[1] Sistema operacional é um programa ou um conjunto de programas com a função de servir de interface para que o usuário e outras aplicações tenham acesso aos recursos que o dispositivo oferece.

[2] Pacote de desenvolvimento de software é proveniente da tradução de *Software Development Kit* (SDK).

[3] Interface de programação de aplicativos é a tradução literal de *Application Programming Interfaces* (APIs).

trabalhos correlatos.

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

Finalizando, no quarto capítulo são apresentadas as conclusões e a lista de tarefas para continuidade do estudo sobre a plataforma Android.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados alguns aspectos teóricos relacionados ao trabalho. É exposta a estrutura do sistema operacional Android, o ciclo de vida e a estrutura das aplicações, a classe de atividade, seu ciclo de vida e métodos. Na continuação são apresentados métodos para a obtenção de resultados a partir de atividades, persistência de dados e também é onde está descrita a correlação entre as atividades e o ciclo de vida de um processo. Na última seção são examinadas características dos trabalhos correlatos que serviram de base para o desenvolvimento da aplicação proposta.

2.1 SISTEMAS OPERACIONAIS MÓVEIS

Segundo Mendonça (2007), o sistema operacional móvel PalmOS possui mais de 10 anos de história e pode ser considerado o primeiro SO para dispositivos móveis, chegando a absorver quase 90% do mercado da área de *palmtops* quando de sua ascensão. Hoje, no entanto, mesmo com uma gama de aplicações bastante grande (mais de 30.000), esse sistema vem perdendo espaço no mercado (MENDONÇA, 2007). Nesse contexto, o lançamento do sistema operacional Palm OS Cobalt foi importante para trazer um novo destaque e condições que garantissem a presença da marca Palm de forma competitiva por mais alguns anos no mercado, uma vez que o sistema era considerado deficiente nos quesitos relacionados à multitarefa e recursos avançados de segurança.

Este sistema conservava a simplicidade e a flexibilidade das aplicações já existentes pelo fato de manter a compatibilidade com as versões anteriores, além de trazer algumas inovações. Dentre elas, destaca-se um sistema totalmente 32 bits, de arquitetura escalar, modular e padronizada, com suporte a *multithreading* e *multitasking*, arquitetura de comunicação unificada e expansível a diversos padrões de tela, além de um sistema de segurança totalmente reescrito (PALMBRASIL, 2004). No cenário atual, a empresa Palm Inc projeta um novo sistema, conhecido como Palm OS II, enquanto a empresa Access mantém a última versão do sistema operacional Palm, chamada de GarnetOS. Em paralelo, esta última empresa aprimora seu novo projeto, chamado de *Access Linux Platform* (ALP). Estas novas propostas, Palm OS II e ALP, apostam em uma plataforma aberta Linux para recuperar o

mercado e a importância da linha de produtos Palm. Em específico para o sistema ALP, um SDK é distribuído, visando popularizar o desenvolvimento neste ambiente (PALMBRASIL, 2008).

Ainda neste contexto, outro sistema de uma considerável importância e com crescimento constante no cenário móvel é o Windows Mobile, que, como o nome sugere, é a versão do sistema operacional Windows para *smartphones* e PDAs. Este sistema foi considerado inicialmente pesado, pois sua desenvolvedora, a Microsoft, reproduziu na plataforma móvel, grande parte de recursos do ambiente para computadores pessoais[4], o que exigia dispositivos com um poder de processamento até então não existentes (PALMBRASIL, 2007). O grande diferencial desse sistema é justamente o fato de portar recursos, integrar e seguir os padrões do ambiente *desktop* do PC, evitando a adaptação por parte do usuário, que se identifica de forma intuitiva e rápida com o que já conhece e faz uso no cotidiano. Essas características, aliadas ao fato do sistema geralmente estar presente em aparelhos com consideráveis recursos de hardware, tal como *Wireless Fidelity* (Wi-Fi), *bluetooth*, infravermelho, *Global Position System* (GPS), *touchscreen*, entre outros, proporcionam uma infra-estrutura adequada para o ambiente corporativo e de entretenimento ao qual o Windows Mobile prontifica-se a atender.

Como resultado dos esforços da *Open Mobile Alliance*[5] (OMA), Symbian é o sistema operacional móvel de maior participação no mercado, com seu maior enfoque na Europa. Concebido desde o princípio para dispositivos móveis, além de seguir padrões de comunicação de dados, esse sistema tem como principais características o baixo consumo de bateria e memória, preservando seu adjetivo de sistema leve, ao mesmo tempo em que demonstra robustez ao apresentar suporte a multitarefa. Seus diferenciais residem no fato de ser uma plataforma aberta, o que possibilita o desenvolvimento de aplicações em diferentes linguagens, dentre as quais, algumas mais conhecidas como Java, Perl, Python e Symbian C/C++. Outra importante peculiaridade do sistema, inerente ao conceito de *Open Standard Operating System* (Sistema Operacional Padrão Aberto) de qual faz parte, se traduz na possibilidade de customização e personalização do sistema e de seus aplicativos por parte dos

---

[4] Computador pessoal é a tradução do inglês *Personal Computer* (PC), sendo uma sigla bastante comum nos meios de comunicação.

[5] *Open Mobile Alliance* é uma organização formada pela iniciativa de alguns fóruns e empresas relacionados à mobilidade, entre os quais participam as principais operadoras de telefonia, fabricantes de redes e terminais, empresas de tecnologia de informação, desenvolvedores e provedores de conteúdo, além de grandes nomes em dispositivos móveis. Seu intuito é facilitar a adoção de serviços de dados móveis, assegurando a interoperabilidade entre terminais, países, provedores de aplicativos, operadores e redes, por meio da criação e discussão de especificações e padrões abertos para dispositivos móveis (OPEN MOBILE ALLIANCE, 2008).

desenvolvedores, da operadora ou do próprio usuário (AZAMBUJA, 2007).

Desenvolvido pela Apple, o sistema operacional Mac OS X é parte integrante do dispositivo móvel iPhone. Por meio deste sistema uma interface intuitiva, de grande interatividade é disponibilizada, provendo funcionalidades e recursos embasados em tecnologias avançadas, como acelerômetro, GPS, toque e multitoque. Em termos técnicos a arquitetura do iPhone não é inovadora, sendo muitas vezes inferior a de outros dispositivos móveis. Seu diferencial reside no fato do uso dos recursos de hardware de forma unificada, simples e intuitiva ao usuário leigo. A disponibilidade de um ambiente aberto à programação incentiva o desenvolvimento e a aceitação da plataforma, ao passo que o App Store, um centro para a disponibilização de aplicações, facilita o crescimento do dispositivo, uma vez que garante o suprimento de aplicações que atendam as necessidades dos usuários.

Lançada em 2005, a plataforma Maemo é desenvolvida pela Nokia. Baseando-se no sistema operacional Linux visa o desenvolvimento voltado à Internet Tablets, ou seja, aparelhos intermediários entre um Palm ou Pocket PC e um Notebook ou Tablet PC. A plataforma Maemo utiliza um sistema operacional Linux com *desktop* GNOME e interface Hildon, tendo como intuito levar ao ambiente móvel as aplicações *desktop* atuais que já rodam sobre o sistema operacional de código aberto de qual faz uso (NOKIABR, 2007).

2.2 ANDROID

Desenvolvido pela *Open Handset Alliance*[6], Android é, segundo GOOGLE (2008p), um pacote de software para dispositivos móveis que inclui um sistema operacional, um *middleware*[7] e aplicativos de características essenciais, dentre os quais se encontram o emulador e um *debbuger*, juntamente com toda a biblioteca de desenvolvimento. Além disso, é considerada uma plataforma móvel completa, livre e aberta (GOOGLE, 2008g), apresentando características incorporadas de outros sistemas, além de inúmeros conceitos inovadores para o segmento móvel.

---

[6] *Open Handset Alliance* é um grupo de mais de 30 empresas de tecnologias móveis que se uniram para acelerar a inovação, fornecendo para o projeto, conhecimento tecnológico baseado na licença Apache v2 Open Source, além de dispositivos com suporte a plataforma Android (OPEN HANDSET ALLIANCE, 2007).

[7] *Middleware* é um software responsável por realizar a mediação entre outros softwares, abstraindo as diferenças existentes em termos de protocolos de comunicação, plataformas ou sistemas operacionais.

2.2.1 Estrutura do sistema operacional

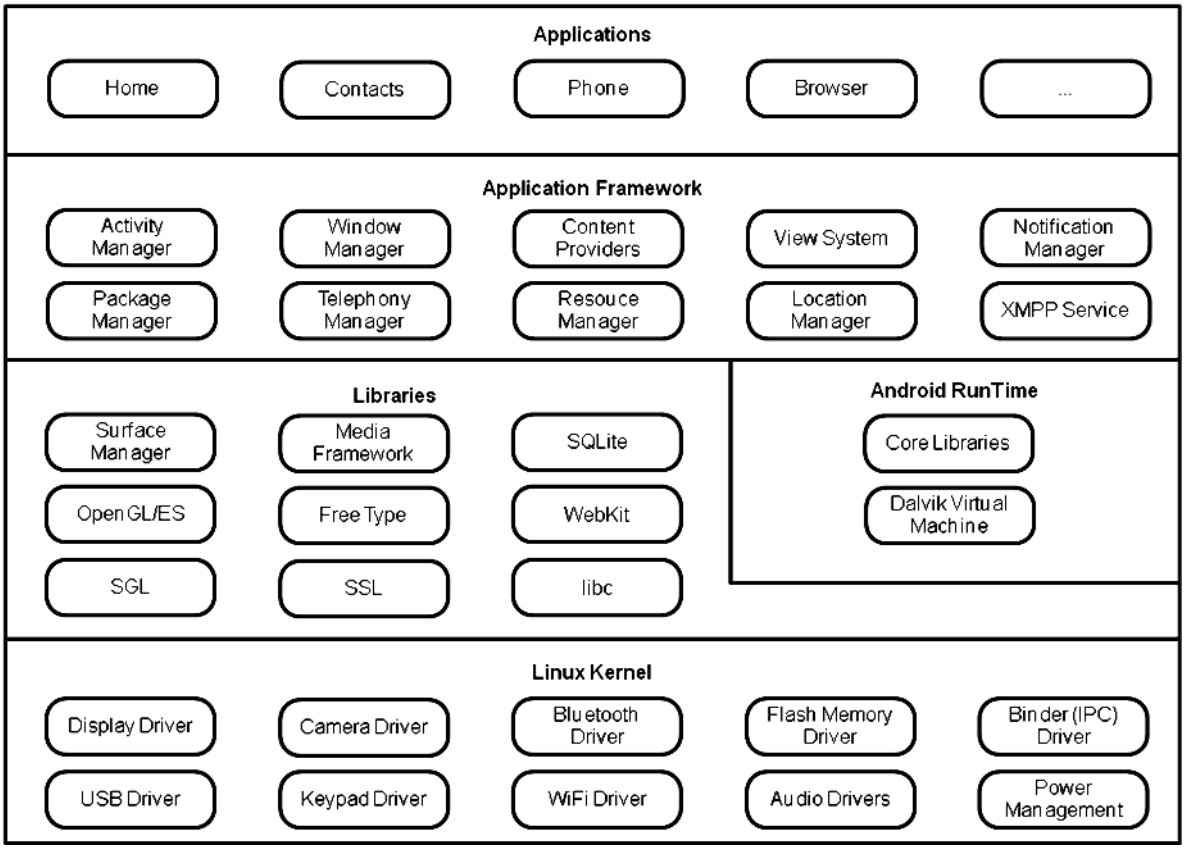
A estrutura do sistema operacional Android é dividida em basicamente quatro camadas, cada qual com módulos internos especializados na resolução de alguma tarefa em particular, conforme ilustra a Figura 1.

Conforme especificado pela Google (2008p), no nível mais baixo de toda estrutura encontra-se o *kernel* do Linux 2.6, no qual a arquitetura do sistema é baseada. Essa camada tem como objetivos o controle dos serviços relacionados à segurança, gerência de memória, gerência de processos, pilha de rede e a manutenção dos modelos de *drivers*[8]. De uma forma geral, esse nível serve de abstração para a comunicação entre o hardware e o software.

Escritas na linguagem de programação C++, as *libraries* (bibliotecas) podem ser acessadas apenas através da camada de nível três e contemplam módulos que servem como base para as aplicações. Seu objetivo é fornecer recursos para o gerenciamento do acesso ao subsistema de visualização, bem como um sistema gráfico de aceleração 2D e/ou 3D, uma base para a criação e reprodução de vários formatos de mídias, um renderizador de fontes *Bitmap* ou vetoriais, um sistema de armazenamento de dados, um motor para navegação Web e as bibliotecas padrões C/C++ derivadas do *Berkeley Software Distribution*[9] (BSD), largamente usadas para dispositivos baseados em Linux (GOOGLE, 2008p).

[8] *Drivers* são programas que atuam como tradutores entre o sistema operacional ou aplicações do usuário, para que essas possam controlar e interagir com o hardware ao qual o *driver* foi concebido.

[9] *Berkeley Software Distribution* é um sistema operacional pertencente a uma larga família de sistemas derivados do UNIX, um sistema operacional portátil, multitarefa e multiusuário.



Fonte: adaptado de Google (2008p).

Figura 1 – Estrutura do sistema operacional Android

Google (2008p) afirma que, paralelamente às bibliotecas encontra-se a máquina virtual Android, desenvolvida para atender as necessidades de um sistema que rode com limitações de bateria, memória e processamento. Nesse contexto, se fazem presentes tanto a máquina virtual Dalvik[10] como as principais bibliotecas Java para entrada e saída de dados, coleção de classes, entre outras.

De acordo com Google (2008p), o terceiro nível foi completamente escrito em Java e constitui a base para os aplicativos, com a qual é possível fazer uso da reutilização e substituição de componentes. Nesse nível os desenvolvedores têm acesso aos mesmos recursos utilizados pelas aplicações do sistema, podendo fazer uso, mediante algumas restrições de segurança, de recursos dispostos por outras aplicações.

Por fim, a camada de aplicativos é o ambiente de contato com o usuário e contempla aplicações como cliente de *e-mail*, programa de envio de mensagens pelo *Short Message*

[10] Dalvik é parte da plataforma Android e pode ser definida como uma máquina virtual baseada em registradores, projetada e escrita com contribuições de engenheiros do Google. Concebida de forma a requerer pouca memória e projetada para permitir que múltiplas instâncias da máquina virtual rodem ao mesmo tempo, ela entrega ao sistema operacional a responsabilidade sobre o isolamento de processos, gerenciamento de memória e o suporte a *threading* (GOOGLE, 2008c).

Service[11] (SMS), calendários, mapas, navegador Web, contatos, entre outros, todos desenvolvidos em Java (GOOGLE, 2008p).

2.2.2 Ciclo de vida das aplicações

De uma forma geral, cada aplicação em Android roda em seu próprio processo Linux. Esses processos são criados sempre que alguma parte do código da aplicação é requisitada, mantendo-se em execução até que não seja mais necessário e o sistema operacional solicite sua área de memória para o uso de outra aplicação (GOOGLE, 2008n).

Percebe-se que o ciclo de vida de uma aplicação não é diretamente controlado por seu código, mas que esse paradigma de abordagem é importante, uma vez que é o sistema operacional que reconhece a real necessidade da execução de uma aplicação para o usuário através de fatores como a avaliação do conjunto das partes em execução da própria aplicação e o quanto de memória global disponível no sistema (GOOGLE, 2008n).

É fundamentalmente importante que os desenvolvedores entendam como diferentes componentes de aplicação, em particular atividades (`activities`), serviços (`services`) e receptores de intenções (`BroadcastReceiver`), impactam no ciclo de vida de um processo. O uso incorreto desses componentes pode resultar no encerramento do processo da aplicação mesmo quando este estiver executando um trabalho importante.

Dessa forma, Google (2008n) afirma que, para determinar qual processo necessita ser encerrado, de modo a garantir a disponibilidade de memória, o sistema operacional Android classifica os processos dentro de uma hierarquia de cinco níveis de importância, a qual é baseada nos componentes em execução usados em processo e no estado de cada um desses componentes.

O primeiro nível da hierarquia contempla processos necessários para a atividade que o usuário está executando. Vários componentes de aplicação podem de diferentes formas fazer com que um processo seja considerado um processo de primeiro plano. De forma geral, um processo será considerado um processo de primeiro plano nas seguintes situações:

a) se estiver executando uma atividade no topo da tela, com o usuário interagindo com esta atividade (o método `onResume()` foi chamado);

---

[11] *Short Message Service* é um serviço disponível para telefones celulares que permite o envio de texto em

b) se contiver um `BroadcastReceiver` rodando (seu método `Broadcast.onReceive()` foi chamado);

c) se contiver um serviço que está executando algum processo em suas chamadas de retorno (`Service.onCreate()`, `Service.onStart()` ou `Service.onDestroy()`).

Além disso, processos em primeiro plano, existirão em menor número durante a execução do sistema. São considerados de grande importância e assim, devem ser encerrados apenas em última instância, quando não houver mais memória para a execução deste nível de processos (GOOGLE, 2008n).

Processos visíveis também possuem uma atividade na tela, são visíveis ao usuário, mas não se encontram em primeiro plano (o método onPause() foi chamado), por isso, serão encerrados antes dos processos de primeiro nível, caso haja necessidade de memória por parte do sistema (GOOGLE, 2008n).

Para Google (2008n), embora os processos de serviços não sejam visíveis ao usuário, esses executam ações em tempo real, sendo muitas delas de grande importância, o que leva ao sistema encerrá-los apenas antes dos processos acima citados.

Na quarta posição hierárquica encontram-se processos de fundo (seu método onStop() foi chamado), que possuem atividades não visíveis ao usuário no momento e desta forma, não acarretam impacto direto, se for necessário finalizar sua execução, que é feita baseada em uma lista de processos de fundo, onde o mais recente processo visualizado pelo usuário será o último a ser encerrado (GOOGLE, 2008n).

Por fim, um processo é classificado como vazio quando mesmo que não possua componentes de aplicação ativos, é mantido em memória com o único intuído de tornar a carga da aplicação mais rápida quando essa necessitar ser iniciada. Esses processos serão retirados da memória em primeira instância (GOOGLE, 2008n).

Quando o sistema estiver decidindo qual processo será retirado de execução, sua decisão será baseada nos componentes de maior importância que estiverem ativos dentro dos processos analisados. Ainda nesse contexto, a prioridade de um processo pode ser elevada em função da dependência de que outros processos tenham com relação a ele.

---

mensagens curtas para tele móveis ou mesmo entre telefones de linha fixa.

2.2.3 Estrutura das aplicações

No Android é possível utilizar-se de quatro blocos distintos para a construção de aplicações. Não se faz necessário, no entanto, que as aplicações façam uso de todos os quatro, mas sim, de uma combinação desses blocos, que devem estar listados em um arquivo gerado na *eXtensible Markup Language*[12] (XML), onde são declarados todos os componentes da aplicação, juntamente com suas capacidades e requisitos (GOOGLE, 2008a). Os blocos de uma aplicação Android são: *Activity*, *Intent Receiver*, *Service* e *Content Provider*.

Segundo Google (2008a), o bloco *Activity* (Atividade) é considerado o bloco mais comum de qualquer projeto e geralmente representa uma tela da aplicação. Seguindo esse conceito, cada atividade deve ser desenvolvida como uma única classe, que estende a classe de base `Activity` e exibe uma interface com o usuário, respondendo a eventos. A atividade também é composta por componentes, que em sua maioria têm como base, a classe `View`.

Outro bloco de importância vital nas aplicações desenvolvidas na plataforma Android é proposto sob o inovador conceito de intenções, representado pela classe `Intent`, que descreve o que uma aplicação quer que seja realizado. Nesse contexto, para Google (2008a), uma `Intent` é composta por duas partes fundamentais, de modo que uma delas informa a ação, enquanto a segunda parte especifica detalhes, informações sobre as quais a ação deve ser executada. Relacionado à intenção, a classe `IntentFilter` tem como intuito descriminar quais intenções uma atividade, ou seja, uma `IntentReceiver` é capaz de tratar. Em outras palavras, usa-se o bloco *Intent Receiver* para publicar a ação, ou `Intent` em Android, que deve ser executada em reação a ocorrência de um evento externo, descrito e publicado pela classe `IntentFilter`.

Similar a uma atividade, de acordo com Google (2008a), o bloco *service* (serviço) não apresenta interface com o usuário e especifica uma ação, no entanto, é capaz de continuar a ser executado mesmo quando em segundo plano. Esse bloco necessita de uma interface exposta para disponibilizar comunicação e controle ao ambiente externo.

Por fim, Google (2008a) conclui que o bloco *Content Provider* (provedor de conteúdo) apresenta os métodos necessários para compartilhar dados com outra aplicação, obter a

[12] *eXtreme Markup Language* é uma linguagem de marcação recomendada pelo *World Wide Web Consortium* (W3C), simples e flexível, capaz de descrever tipos de dados diversos, com o objetivo de facilitar o compartilhamento de informações através da Internet. O W3C é um consórcio que desenvolve padrões para a criação e a interpretação de conteúdos para a Web.

informação de que tipo de dado é tratado pelo provedor, ou para possibilitar que uma aplicação guarde informações em algum meio, como por exemplo, arquivos ou em um banco de dados SQLite.

2.2.4 A classe de atividade

Segundo Google (2008c), a classe de atividade representa uma ação que o usuário pode realizar. Quase todas as atividades interagem com o usuário e dessa forma, a classe atividade torna-se a responsável por criar a janela na qual será inserida a interface com o usuário, o que pode ser feito, por exemplo, através do método `setContentView(View)`.

De forma geral, uma atividade é geralmente executada em modo *fullscreen* (tela cheia), mas nada impede que ela seja apresentada de outra maneira, como em janelas flutuantes. As atividades são iniciadas através do método `startActivity(Intent)`. Para que se possa fazer uso deste método é necessário que as atividades estejam declaradas no arquivo `AndroidManifest.xml`[13]. Da mesma forma, ao programar uma atividade deve-se obrigatoriamente desenvolver dois métodos distintos. O primeiro, `onCreate(Bundle)` é onde se dá o início da atividade e onde define-se a interface com o usuário. Neste método também é possível obter *widgets*[14] que serão necessários dentro da atividade. A obtenção destes é feita através de métodos como `findViewById(int)`. O segundo método obrigatório de uma atividade é o método `onPause()`. Esse método é executado quando o usuário interrompe a execução da aplicação e seu ponto de chamada é estratégico de forma a possibilitar que todas as informações necessárias possam ser persistidas (GOOGLE, 2008c).

2.2.4.1 Ciclo de vida de uma atividade

Google (2008c) elucida que perante o sistema as atividades são gerenciadas em forma

---

[13] AndroidManifest é um arquivo de existência obrigatória, localizado na pasta raiz da aplicação, que descreve valores globais para o pacote da aplicação, incluindo componentes (atividades, serviços, etc.) que o pacote disponibiliza e as classes de implementação de cada componente, bem como o tipo de dado com que trabalham e a indicação de onde podem ser usados (GOOGLE, 2008g).

[14] Widgets são elementos de interface com o usuário contidos no pacote widget da plataforma Android, geralmente visuais, que podem ser criados e/ou usados pelo desenvolvedor em sua aplicação (GOOGLE, 2008h).

de pilha. Quando uma nova atividade é iniciada ela é alocada no topo da pilha e torna-se uma atividade em execução. Em contrapartida, a atividade anterior sempre se mantém abaixo na pilha e não virá para o topo novamente enquanto a nova atividade existir.

Uma atividade tem essencialmente quatro estados (Google, 2008c). O primeiro estado descreve atividades ativas (*active*) ou em execução (*running*). Essas atividades estão no topo da pilha de execução, pois aparecem na tela sobre todas as demais janelas existentes.

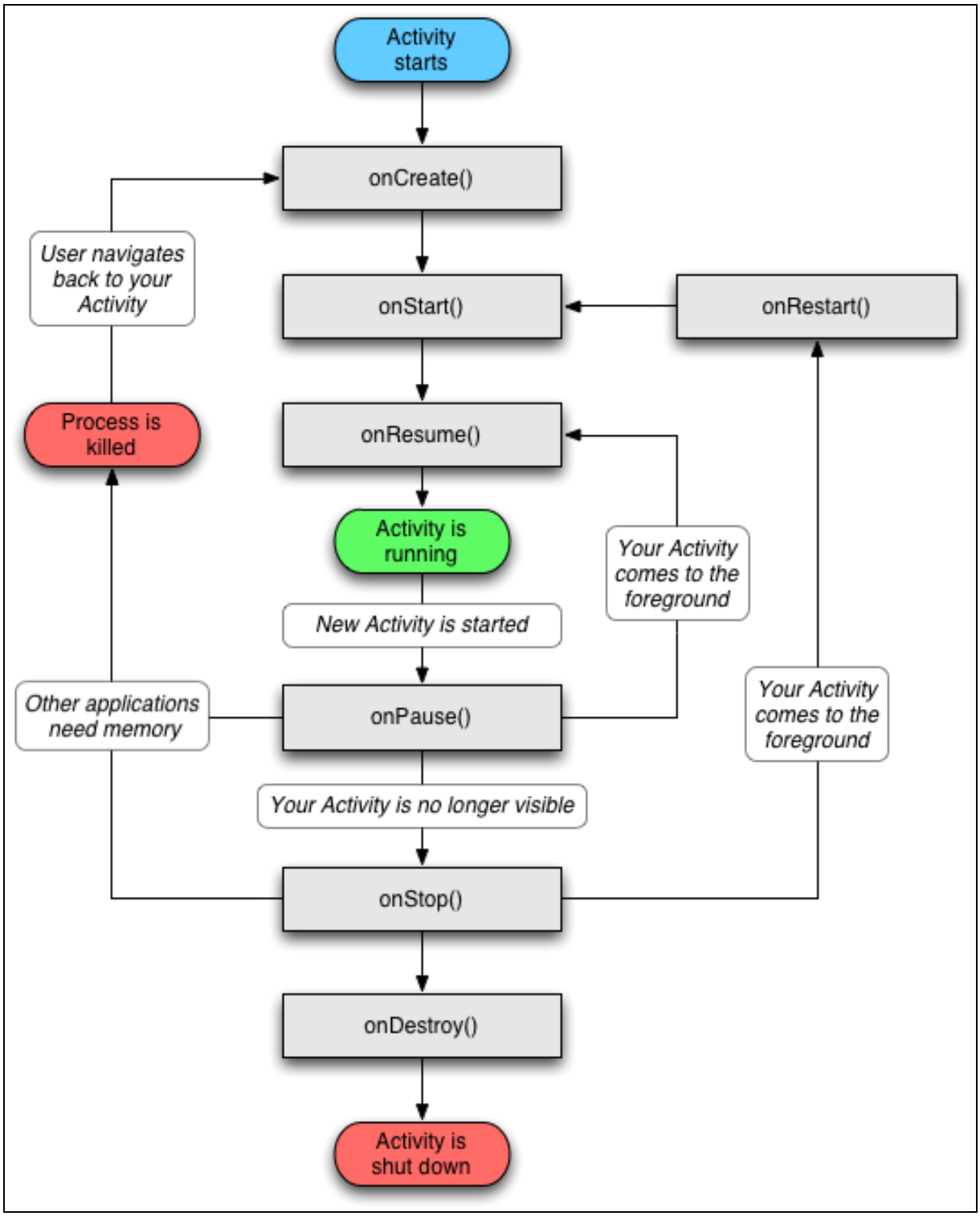
Caso a atividade tenha perdido o foco, mas de alguma forma continua visível, o que pode acontecer quando uma nova atividade é iniciada, mas não ocupa a tela inteira ou se a nova atividade é transparente, então a atividade que perdeu o foco é pausada (*paused*). Uma atividade pausada está completamente viva, mantendo todos os estados e informações, bem como sua referência no gerenciador de janela. Essas atividades podem ser encerradas pelo sistema em situações extremas, quando a memória for muito baixa.

Quando a atividade não está visível ela está parada (*stopped*). Todos os estados e informações são mantidos, mas como a atividade não está visível ela será encerrada pelo sistema quando alguma memória for requerida.

Ainda segundo Google (2008c), uma atividade pausada ou parada pode ser retirada da memória quando o sistema solicita que esta seja encerrada ou mesmo, que o processo seja terminado. Em outro momento, quando a atividade for novamente exibida ao usuário, essa será iniciada e deve ter seu estado restaurado por completo.

A Figura 2 ilustra os estados importantes de uma atividade que são representados pelos ovais, enquanto os retângulos representam métodos onde é possível programar ações que serão realizadas na troca de estados.

Conforme Google (2008c) existe também três ciclos distintos para uma atividade. O primeiro é descrito como o ciclo completo e contempla os demais ciclos. Esse acontece entre primeira a chamada ao método `onCreate(Bundle)` e a única chamada ao método `onDestroy()`. No contexto deste ciclo uma atividade vai realizar toda a configuração do seu estado global no método `onCreate()` e liberar todos os recursos no método `onDestroy()`.



Fonte: adaptado de Google (2008c).

Figura 2 – Os estados importantes e métodos transitórios de uma atividade

O segundo ciclo de vida é o ciclo visível de uma atividade e acontece entre a chamada ao método `onStart()` e uma chamada correspondente ao método `onStop()`. Durante esse tempo a atividade estará disponível em tela, mesmo que não esteja no topo e interagindo com o usuário. Os métodos delimitadores descritos podem ser chamados várias vezes, uma vez que a atividade pode se tornar visível e invisível ao usuário em função de motivos diversos,

inerentes a regra de negócio programada. Entre esses métodos podem ser mantidos os recursos necessários para exibir a atividade ao usuário.

Por fim, a partir de uma chamada ao método `onResume()` até uma chamada correspondente ao método `onPause()` uma atividade está no topo da execução, interagindo com o usuário. Durante a execução dos métodos deste ciclo os códigos devem ser leves, uma vez que algumas atividades trocam de estado com grande freqüência.

2.2.4.2 Métodos de uma atividade

Os métodos de uma atividade podem ser sobrescritos pelo programador de forma a possibilitar a execução de um comportamento em específico quando ocorre uma mudança de estado na atividade. Ainda sobre os métodos, sem exceção, todas as atividades necessitam implementar `onCreate(Bundle)` para realizar sua configuração inicial. Para outras atividades o desenvolvimento de cada método deve ser analisado de acordo com as necessidades, no entanto, sempre que algum dos métodos for programado deve ser chamado o método correspondente da superclasse, sob a penalidade do lançamento de uma exceção se este não for referenciado (GOOGLE, 2008c). Baseado em Google (2008c), alguns dos principais métodos existentes para uma atividade são descritos abaixo:

a) `onCreate()`: chamado quando a atividade é criada pela primeira vez, quando estiver sendo iniciada. Neste método podem ser efetuadas as inicializações necessárias para a execução da atividade. Ainda neste método é possível, caso o `Bundle`[15] não for nulo, recuperar o estado anterior da atividade. O método `onCreate()` é sempre seguido do método `onStart()`;
b) `onRestart()`: chamado antes da atividade ser novamente iniciada, toda vez que a atividade teve a sua execução interrompida. É seguido pelo método `onStart()`;
c) `onStart()`: chamado quando uma atividade se torna visível ao usuário. É sucedido por um `onResume()` se a atividade fica em segundo plano, ou pelo método `onStop()` se a atividade ficar escondida;
d) `onResume()`: chamado quando a atividade começa a interagir com o usuário. Quando ocorre a chamada deste método a atividade encontra-se no topo da pilha

de atividades, mas não necessariamente está visível ao usuário. É sempre seguido pelo método `onPause()`;

e) `onPause()`: chamado quando uma atividade estiver sendo atribuída para o segundo plano, mas ainda não foi encerrada. Este método é geralmente utilizado para salvar alterações, o contexto de execução da atividade, ou para encerrar a execução de códigos que possam estar consumindo memória, processamento, etc. Recomenda-se que este método tenha uma execução rápida, uma vez que a próxima atividade não será resumida até que o método `onPause()` em execução seja finalizado. É seguido por um `onResume()` se a atividade retorna para o topo da execução ou pelo método `onStop()` se a atividade torna-se invisível ao usuário;

f) `onStop()`: chamado quando a atividade não está mais visível ao usuário devido ao fato de que outra atividade foi resumida e está sobre esta. A chamada deste método pode acontecer porque uma nova atividade está sendo iniciada, uma atividade existente está sendo recebendo o foco de execução ou mesmo quando a atividade está sendo destruída. É seguido pelo método `onRestart()` se a atividade for novamente interagir com o usuário ou pelo método `onDestroy()` se a atividade estiver sendo encerrada;

g) `onDestroy()`: este método corresponde a uma chamada final que a atividade recebe antes de ser efetivamente destruída. A chamada também pode acontecer porque a atividade está encerrando. As duas situações são podem acontecer quando o método `finish()` foi chamado ou porque o sistema está destruindo temporariamente esta instância da atividade para liberar recursos.

Alguns dos métodos acima descritos podem ser encerrados, ou seja, após a efetivação desses métodos é possível que a execução do processo que mantinha a atividade seja encerrada pelo sistema sem qualquer intervenção do usuário. Para garantir a guarda dos dados recomenda-se, quando for necessário, o uso do método `onPause()` para a persistência de dados. Além deste, pode-se também utilizar-se do método `onSaveInstanceState(Bundle)`, cuja chamada precede o encerramento forçado de uma atividade. Este método possibilita que o contexto de execução da atividade seja salvo na `Bundle` que estará disponível para o método `onCreate(Bundle)`. Nesse método, com base na `Bundle`, a atividade pode ser corretamente restaurada para o seu estado anterior, recuperando todo o contexto se vier a ser

[15] Bundle é uma classe de mapeamento de valores para todos os tipos de dados disponíveis em Android, que podem ser alterados, consultados, inseridos ou removidos por meio de uma chave em forma de cadeia de

recriada.

O Quadro 1 ilustra os métodos de uma atividade. Para aqueles que não estão assinalados como encerráveis o processo da atividade não será finalizado pelo sistema enquanto o método não retornar, ou seja, uma atividade está em um estado onde pode ser encerrada, por exemplo, apenas entre o período de tempo depois do método `onPause()` e antes do método `onResume()`.

ORDEM	MÉTODO	Encerrável	Próximo
1	`onCreate()`	não	`onStart()`
1.1	`onRestart()`	não	`onStart()`
1.2	`onStart()`	não	`onResume()` ou `onStop()`
1.2.1	`onResume()`	não	`onPause()`
1.2.2	`onPause()`	sim	`onResume()` ou `onStop()`
1.3	`onStop()`	sim	`onRestart()` ou `onDestroy()`
2	`onDestroy()`	sim	nenhum

Fonte: adaptado de Google (2008c).

Quadro 1 – Métodos de uma atividade

2.2.4.3 Iniciando atividades e obtendo resultados

O método `startActivity(Intent)` é utilizado para iniciar uma nova atividade, a qual será alocada no topo da pilha de execução. Ele requer apenas um argumento, uma `Intent`, que descreve qual atividade deve ser executada (GOOGLE, 2008c).

Algumas vezes, quando a atividade é encerrada, pode ser necessário obter um resultado como retorno. Para tanto, pode-se fazer uso do método `startActivityForResult(Intent, int)`, onde o segundo parâmetro, um inteiro, identifica a origem da chamada. Após a execução, quando a atividade encerra, o método `onActivityResult(int, int, Intent)` é invocado. Seus parâmetros correspondem ao identificador de chamada da origem e ao resultado de execução, além de uma intenção genérica onde é possível retornar dados a origem da chamada.

Da mesma forma que são passados parâmetros à uma nova atividade, quando esta é encerrada também é possível retornar parâmetros à sua origem, o que pode ser feito por meio do método `setResult(int)`. Nesse contexto uma atividade sempre deve fornecer um código de resultado, o qual pode ser `RESULT_CANCELED`, `RESULT_OK` ou qualquer valor customizado iniciando em `RESULT_FIRST_USER`. Além disso, a atividade pode opcionalmente retornar uma

---

caracteres.

`Intent` contendo qualquer informação adicional necessária. Toda essa informação aparece novamente na origem, em função do método `Activity.onActivityResult()`, juntamente com o identificador de chamada que foi originalmente fornecido (GOOGLE, 2008c).

Caso a atividade filha falhe por alguma razão, a atividade pai receberá como resultado o código referente a constante `RESULT_CANCELED`.

2.2.4.4 Salvando o estado persistente

Há geralmente dois tipos de estados persistentes com que uma atividade irá trabalhar. Conforme Google (2008c), o primeiro deles é reconhecido por viabilizar informações compartilhadas, tipicamente armazenadas em uma base SQLite utilizando um *content provider*. O segundo estado é considerado um estado de armazenamento interno, como por exemplo, das preferências de um usuário.

Para dados de um *content provider*, sugere-se que as atividades usem um modelo de usuário *edit in place*. Nesse modelo toda edição que um usuário faz deve ser imediatamente efetivada, sem exigir qualquer confirmação. Para contemplar o uso do modelo apresentado sugere-se seguir apenas duas regras:

a) ao criar um novo documento, a base de dados ou o arquivo de entrada deve ser criado imediatamente. Por exemplo, caso o usuário de um sistema de correio eletrônico escolher por redigir um novo e-mail, uma nova entrada para o e-mail deverá criada logo que começar a inserção dos dados, de forma que caso o usuário vá para outra atividade, o e-mail aparecerá na lista de rascunhos;
b) quando o método `onPause()` de uma atividade for chamado, este deve submeter ao *content provider* ou a um arquivo quaisquer alterações que o usuário tenha feito. Dessa forma, garante-se que essas mudanças serão vistas por qualquer outra atividade que for executada a partir de então. Ainda é aconselhável também, verificar a necessidade de submeter dados à persistência em momentos chave durante o ciclo de vida de uma atividade, como por exemplo, antes de iniciar uma nova atividade, antes de concluir a atividade em execução ou quando o usuário alterna entre os campos de entrada, etc.

Este modelo tem por objetivo prevenir a perda de dados quando um usuário estiver navegando entre as atividades. Da mesma forma, permite que o sistema encerre uma atividade com segurança a qualquer momento, mesmo quando recursos de sistema forem necessários

para outro processo e a atividade em execução for derrubada para liberá-los.

Em contrapartida notam-se algumas implicações relacionadas ao modelo, como o fato de que ao pressionar *BACK* em uma atividade, a ação a ser executada não será um cancelamento, mas sim um abandono da atividade, preservando o estado atual em um meio de persistência. Nesse contexto, o cancelamento de edições em uma atividade deve ser fornecido através de outros mecanismos, tais como uma explícita opção para reverter ou desfazer uma determinada ação.

A classe Atividade também fornece uma API para a gestão interna dos estados persistentes associados a uma atividade. Esse recurso pode ser usado, por exemplo, para carregar qual a forma de exibição de um calendário o usuário prefere; visualizar por dia ou por semana; ou a página inicial padrão do usuário em um navegador web.

O estado persistente de uma atividade é gerido com o método `getPreferences(int)`, que lhe permite copiar e modificar um conjunto de pares de nome/valor associados à atividade. Para usar preferências que são partilhadas através de vários componentes de aplicação, tais como atividades, receptores (*receivers*), serviços (*services*) ou memso fornecedores (*providers*), pode-se usar o método `Context.getSharedPreferences()` para recuperar um objeto de preferências armazenado sob um nome específico. No entanto, não é possível compartilhar dados através de pacotes de aplicação, sendo necessário para tanto um provedor de conteúdo (*content provider*) (GOOGLE, 2008c).

2.2.4.5 Atividades e o ciclo de vida de um processo

De acordo com Google (2008c), o sistema Android procura manter em execução os processos de aplicações pelo maior tempo possível, no entanto, eventualmente será necessário remover processos antigos quando os níveis de memória estiverem baixos. Conforme elucidado por Google, durante o ciclo de vida de uma atividade a decisão de qual processo o sistema deve remover está diretamente relacionada ao estado de interação com o usuário com o qual o processo é encontrado. Dessa forma, considerando as atividades em execução, torna-se comum classificar em quatro os estados de execução de um processo. É com base nesses estados que o sistema tomará a decisão de qual processo será encerrado por primeiro, sempre optando primeiramente pelos processos de menor importância, para em últimos casos encerrar os processos de maior importância. As atividades que caracterizam a relevância de um processo são descritos a seguir:

a) atividade em primeiro plano: são atividades encontradas no topo da tela, com as quais o usuário está interagindo. Estas atividades são consideradas as mais importantes, de modo que processos que contemplem atividades com execução em primeiro plano serão derrubados apenas como ultimo recurso, caso estiverem usando mais memória do que existe disponível no dispositivo. Nesses casos o dispositivo está geralmente em um estado de paginação de memória e faz-se necessário requisitar a memória do processo para manter a interface com o usuário respondendo;
b) atividade visível: caracteriza uma atividade que é visível para o usuário mas não está em primeiro plano. É considerada extremamente importante e dessa forma, processos que contemplem atividades visíveis serão derrubados apenas quando forem necessários recursos para manter atividades em primeiro plano rodando;
c) atividade em segundo plano: são atividades que não estão visíveis para o usuário e que encontram-se pausadas. O sistema poderá encerrar estas atividades de forma segura para disponibilizar memória a processos em primeiro plano ou a processos visíveis. Caso um processo em segundo plano teve sua execução encerrada e o usuário retornar sua execução, tornando-o novamente visível em tela, o método `onCreate(Bundle savedInstanceState)` será chamado. Com base no parâmetro `savedInstanceState` que poder ser inicializado pelo método `onSaveInstanceState(Bundle)`, o contexto de execução anterior ao enceramento da atividade poderá ser retornado e o processo exibido novamente com o mesmo estado em que usuário havia o deixado;
d) processo vazio: um processo vazio não apresenta atividade alguma, bem como também não faz uso de qualquer componente de aplicação. Esses processos são encerrados rapidamente pelo sistema quando a memória estiver baixa e por esta razão, qualquer operação de segundo plano que seja feita fora de uma atividade deve ser executada no contexto de uma atividade receptora de requisições ou como um serviço, para que o sistema entenda que é necessário mantê-la em execução.

Ainda segundo Google (2008c), às vezes é necessário que uma atividade mantenha uma operação em execução sem vínculo com o seu ciclo de vida, ou seja, é preciso que seja executa determinada operação em paralelo, de forma exclusiva, independentemente do estado da atividade ser pausado, parado ou mesmo encerrado. Para situações desta natureza deve-se iniciar um serviço para tratar especificamente a execução da rotina desejada.

2.3 TRABALHOS CORRELATOS

Com relação a trabalhos correlatos, são descritos sistemas que apresentam algumas das funcionalidades que serão apresentadas no projeto. Neste contexto, o Google Maps pode ser considerado o serviço de pesquisa e visualização de mapas e imagens de satélite de maior popularidade, sendo disponibilizado pela Google gratuitamente na Web, bem como para plataformas móveis. Este serviço proporciona também uma infinidade de outras funcionalidades, dentre as quais, segundo Google (2007), se fazem presentes recursos para:

a) pesquisar endereços;
b) localizar serviços e empresas, obtendo endereços, números de telefones e rotas;
c) traçar rotas;
d) criar e compartilhar mapas personalizados;
e) planejar um trajeto com múltiplas paradas, com a possibilidade de arrastar e soltar pontos para personalizar uma rota;
f) marcar lugares favoritos em um mapa personalizado;
g) desenhar linhas e contornos para destacar caminhos e áreas;
h) acrescentar textos, fotos e vídeos ao mapa;
i) compartilhar mapas;
j) ler e escrever resenhas;
k) cadastrar empresas no mapa de modo que outras pessoas possam encontrá-las.

O sítio Apontador é uma referência em localização no Brasil, por ser um sistema de caráter nacional, proposto para facilitar a busca de endereços e locais, além da visualização de rotas traçadas pelo sistema. Também é possível a criação de locais personalizados, com informações detalhadas como descrição, telefone, horários, fotos e vídeos. O sistema conta inclusive com uma versão móvel, que favorece a agilidade do serviço em função de um menor tráfego de dados (APONTADOR, 2008).

3 DESENVOLVIMENTO

O desenvolvimento da aplicação Android MapBr envolveu as fases de instalação e configuração do ambiente de desenvolvimento, introdução ao desenvolvimento e depuração de aplicações Android, levantamento de requisitos, especificação e posterior implementação da aplicação, bem como a documentação das operacionalidades com o intuito de orientar o usuário a como fazer uso das funcionalidades que foram disponibilizadas. Neste capítulo, além do detalhamento das fases acima, são apresentados os resultados obtidos.

3.1 DESENVOLVIMENTO EM ANDROID

Detalhes sobre a instalação do pacote de desenvolvimento de softwares Android, bem como informações necessárias para configurar o ambiente Eclipse[16] a fazer uso das bibliotecas e dos recursos integrados ao pacote podem ser obtidos na referência Google (2008m). Da mesma forma, o conteúdo introdutório sobre a criação de um projeto, desenvolvimento e depuração de aplicações em Android pode ser encontrado na referência Google (2008k).

3.2 REQUISITOS DA APLICAÇÃO ANDROID MAPBR

Nesta seção são apresentados os requisitos funcionais (Quadro 2) e não funcionais (Quadro 3) propostos para a aplicação Android MapBr.

[16] Eclipse consiste em uma comunidade de código aberto cujos projetos são focados na construção de uma plataforma de desenvolvimento composta por ambientes de trabalho extensíveis e ferramentas para a construção, desenvolvimento e manipulação de programas durante todo o seu ciclo de vida (ECLIPSE, 2008). Eclipse disponibiliza um dos ambientes de programação Java de maior popularidade, o Eclipse IDE.

REQUISITOS FUNCIONAIS	CASO DE USO
RF01: Ser desenvolvido usando a plataforma Android.	
RF02: Permitir ao usuário visualizar o mapa global disponibilizado pelo serviço Google *Maps* sob o ponto de vista do modo desenho e do modo satélite.	UC01
RF03: Permitir ao usuário realizar movimento por todo o mapa.	UC01
RF04: Permitir ao usuário aplicar níveis de zoom ao mapa.	UC01
RF05: Permitir ao usuário posicionar o mapa na localização GPS do dispositivo móvel.	UC01
RF06: Permitir ao usuário visualizar, adicionar, selecionar, editar, remover, ocultar, listar e salvar marcadores na aplicação.	UC02, UC03 e UC04
RF07: Permitir ao usuário pesquisar endereços ou locais, bem como listar os resultados da pesquisa.	UC05 e UC06

Quadro 2 – Requisitos funcionais

REQUISITOS NÃO FUNCIONAIS
RNF01: Ser desenvolvido usando a análise orientada a objetos.
RNF02: Ser desenvolvido usando o paradigma de intenções proposto pelo Android.
RNF03: Ser desenvolvido de modo a fazer uso de um tipo de provedor de conteúdo, seja por meio da gravação em arquivo ou através de banco de dados.

Quadro 3 – Requisitos não funcionais

3.3 ESPECIFICAÇÃO DA APLICAÇÃO ANDROID MAPBR

A aplicação Android MapBr foi especificada através da ferramenta Enterprise Architect (SPARX SYSTEMS, 2007), utilizando os conceitos de orientação a objetos e baseando-se nos diagramas da UML, gerando como produtos os diagramas de caso de uso, de classes e de seqüência apresentados nas seções seguintes.

3.3.1 Casos de uso

A aplicação Android MapBr possui seis casos de uso (Figura 3), sendo que segundo uma seqüência lógica, o primeiro caso de uso deve ser obrigatoriamente executado pelo usuário, enquanto a execução de cada um dos demais casos de uso é facultativa. Caso o usuário resolva executar os casos de uso opcionais, esta execução será feita sempre a partir do primeiro caso de uso, ou seja, com este caso de uso em execução, sobreposto pelo novo caso de uso. Ao final da execução a aplicação deve retornar ao caso de uso `Explorar mapa`, o caso

de uso principal. Todos os casos de uso são executados pelo ator `Usuário`, que representa a pessoa que faz uso da aplicação Android MapBr.

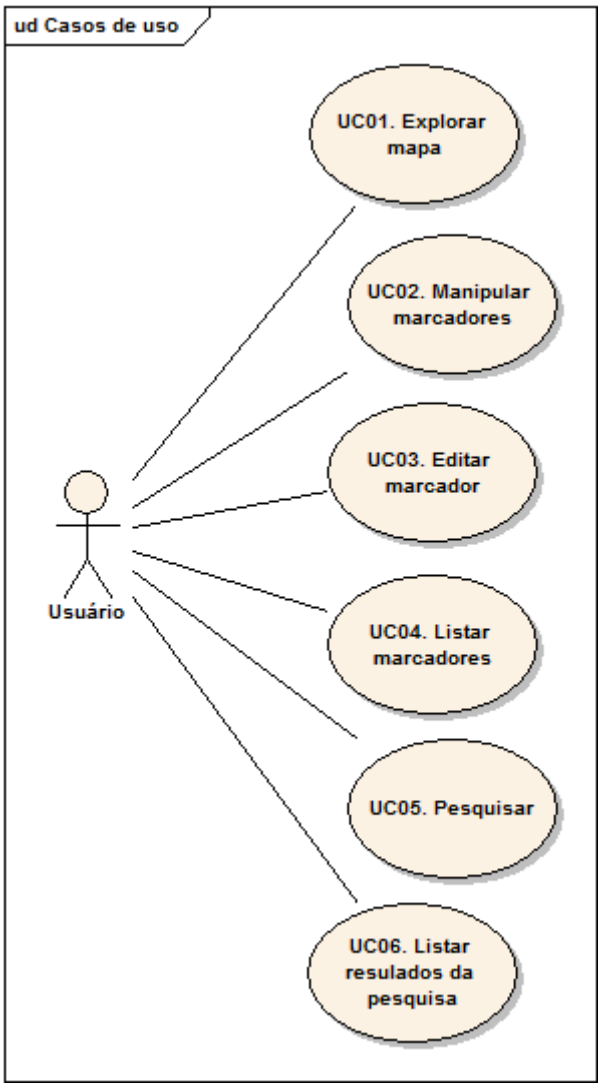


Figura 3 – Diagrama de casos de uso

3.3.1.1 Explorar mapa

O primeiro caso de uso (Quadro 4), designado `Explorar mapa`, descreve como o usuário pode explorar os recursos visuais da aplicação Android MapBr. Além do cenário principal o caso de uso possui quatro cenários alternativos que permitem centralizar o mapa em posicionamento global, definir o modo de visualização ou o nível de zoom, bem como reposicionar o mapa na localização GPS do dispositivo móvel. O caso de uso não possui cenários de exceção.

UC01 – Explorar mapa: possibilita ao usuário uma forma de centralizar o mapa em posicionamento global, definir o modo de visualização ou o nível de zoom, bem como reposicionar o mapa na localização GPS do dispositivo móvel.	
Requisitos atendidos	RF02, RF03 e RF04.
Pré-condições	Não possui.
Cenário principal	1) O usuário visualiza o mapa em posicionamento global com zoom em nível mínimo, sob o ponto de vista do modo de desenho, padrão da aplicação. São exibidos os marcadores de acordo com seu estado de visualização individual e/ou o estado conjunto de visualização de marcadores da aplicação. 2) O usuário realiza a movimentação pelo mapa.
Fluxo alternativo 01	Alterar modo de visualização: 1) No passo 2 do cenário principal, caso o usuário desejar alterar o modo de visualização do mapa, este deve optar entre os dois modos visualização existentes para o item de menu `Visualização` do menu principal, ou seja, entre o modo `Desenho` e o modo `Satélite`. 2) A aplicação aplica o modo de visualização escolhido pelo usuário. 3) Voltar ao passo 2 do cenário principal.
Fluxo alternativo 02	Alterar o nível de zoom: 1) No passo 2 do cenário principal, caso o usuário desejar alterar o nível de zoom aplicado ao mapa, este deve optar por realizar mais ou menos zoom utilizando os controles na base do mapa. 2) A aplicação aplica o nível de zoom escolhido pelo usuário. 3) Voltar ao passo 2 do cenário principal.
Fluxo alternativo 03	Posicionar mapa na localização GPS do dispositivo móvel: 1) No passo 2 do cenário principal, caso o usuário desejar posicionar o mapa na localização GPS do dispositivo móvel, este deve optar pelo item de menu `GPS` do menu principal. 2) A aplicação posiciona o mapa na localização GPS do dispositivo móvel com o nível máximo de zoom. 3) Voltar ao passo 2 do cenário principal.
Fluxo alternativo 04	Centralizar mapa: 1) No passo 2 do cenário principal, caso o usuário desejar centralizar novamente o mapa em posicionamento global com zoom em nível mínimo, sob o ponto de vista do modo de desenho, este deve optar pelo item de menu `Centralizar` do menu principal. 2) A aplicação reposiciona o mapa em posicionamento global, com zoom em nível mínimo, sob o ponto de vista do modo de desenho. 3) Voltar ao passo 2 do cenário principal.
Pós-condições	O mapa deve ser exibido com o modo de visualização padrão ou com o modo de visualização optado pelo usuário, na área centralizada pelo movimento do usuário, ou em posicionamento global ou na localização GPS do dispositivo, com o nível de zoom definido pelo usuário ou no nível mínimo do início da aplicação.

Quadro 4 – Caso de uso `UC01`

3.3.1.2 Manipular marcadores

O segundo caso de uso (Quadro 5), designado `Manipular marcadores`, trata da visualização, inclusão, seleção, edição e remoção dos marcadores manipulados pela aplicação.

Ele possui o cenário principal e seis cenários alternativos. Não existem cenários de exceção para este caso de uso.

UC02 – Manipular marcadores: possibilita ao usuário uma forma de adicionar, selecionar, editar e remover marcadores no mapa, bem como definir a visualização dos marcadores de forma individual ou conjunta.	
Requisitos atendidos	RF06.
Pré-condições	A aplicação deve estar executando o caso de uso `UC01`.
Cenário principal	1) O usuário visualiza o mapa onde são exibidos os marcadores de acordo com seu estado de visualização individual e/ou o estado conjunto de visualização de marcadores da aplicação.
Fluxo alternativo 01	Adicionar marcador: 1) Após o passo 1 do cenário principal, caso o usuário desejar adicionar um marcador, este deve pressionar no mapa durante quatro segundos o ponto onde deseja adicionar o marcador. No entanto, caso existir um resultado da pesquisa posicionado e o usuário desejar adicionar um marcador ao resultado, este deve optar pelo sub-item `Adicionar` do item de menu `Marcadores` do menu principal. 2) Ver caso de uso `UC03`. 3) Voltar ao passo 1 do cenário principal.
Fluxo alternativo 02	Selecionar marcador: 1) Após o passo 1 do cenário principal, caso o usuário desejar selecionar um marcador, este deve pressionar o marcador desejado no mapa. 2) A aplicação seleciona e exibe o título do marcador. 3) Voltar ao passo 1 do cenário principal.
Fluxo alternativo 03	Editar marcador: 1) Após o passo 1 do cenário principal, caso o usuário desejar editar um marcador, este deve pressionar durante quatro segundos o marcador desejado e optar pela opção `editar` do menu *popup*. No entanto, caso existir um marcador selecionado e o usuário desejar editar o marcador, este deve optar pelo sub-item `Editar` do item de menu `Marcadores` do menu principal. 2) Ver caso de uso `UC03`. 3) Voltar ao passo 1 do cenário principal.
Fluxo alternativo 04	Remover marcador: 1) Após o passo 1 do cenário principal, caso o usuário desejar remover um marcador, este deve pressionar durante quatro segundos o marcador desejado e optar pela opção `remover` do menu *popup*. No entanto, caso existir um marcador selecionado e o usuário desejar remover o marcador, este deve optar pelo sub-item `Remover` do item de menu `Marcadores` do menu principal. 2) A aplicação solicita a confirmação do usuário, que se for positiva, permite que a aplicação remova o marcador do mapa e delete o marcador do banco de dados. 3) Voltar ao passo 1 do cenário principal.
Fluxo alternativo 05	Remover todos os marcadores: 1) Após o passo 1 do cenário principal, caso o usuário desejar remover todos os marcadores, este deve optar pelo sub-item Remover Todos do item de menu Marcadores do menu principal. 2) A aplicação solicita a confirmação do usuário, que se for positiva, permite que a aplicação remova todos os marcadores do mapa e delete todos os marcadores do banco de dados. 3) Voltar ao passo 1 do cenário principal.

Fluxo alternativo 06	Alterar estado de visualização conjunta 1) Após o passo 1 do cenário principal, caso o usuário desejar altera o estado de visualização conjunto dos marcadores, este deve optar por alterar o estado do sub-item `Exibir` do item de menu `Marcadores` do menu principal. 2) A aplicação atribui o novo estado de visualização conjunto, exibindo ou ocultando todos os marcadores do mapa. 3) Voltar ao passo 1 do cenário principal.
Pós-condições	Os marcadores manipulados pela aplicação estarão salvos em banco de dados e serão exibidos de acordo com seu estado de visualização individual e/ou o estado conjunto de visualização de marcadores da aplicação.

Quadro 5 – Caso de uso `UC02`

3.3.1.3 Editar marcador

O terceiro caso de uso (Quadro 6), designado `Editar marcador`, trata da inclusão e alteração dos marcadores manipulados pela aplicação. O caso de uso possui o cenário principal, um cenário alternativo e dois cenários de exceção.

UC03 – Editar marcador: possibilita ao usuário uma forma de adicionar ou alterar um marcador no mapa.	
Requisitos atendidos	RF06.
Pré-condições	A aplicação estava executando o caso de uso `Manipular marcadores` (`UC02`) e o usuário optou por executar o fluxo alternativo 01 (Adicionar marcador) ou o fluxo alternativo 03 (Editar marcador). Também é possível que a aplicação estivesse executando o caso de uso `Listar marcadores` (`UC04`) e o usuário tenha optado pelo fluxo alternativo 02 (`Editar marcador`).
Cenário principal	1) A aplicação exibe a tela de edição de marcador. Os dados do marcador estarão preenchidos caso o usuário estiver alterando um marcador já existente. Sempre serão exibidos valores para a latitude e a longitude, seja em uma inclusão ou em uma alteração. 2) O usuário informa os dados de marcador. 3) O usuário opta por salvar o marcador fazendo uso do botão `Salvar` da tela de edição de marcador. 4) Caso o marcador estiver sendo incluído, a aplicação adiciona o marcador no mapa e insere o marcador em banco de dados. Caso o marcador estiver sendo alterado, a aplicação altera os dados do marcador no mapa e atualiza o marcador em banco de dados.
Fluxo alternativo 01	Cancelar a adição ou a alteração de marcador: 1) Após o passo 1 ou no passo 3 do cenário principal, caso o usuário desejar cancelar a adição ou a alteração do marcador, este deve optar pela opção `Cancelar` da tela de edição de marcador.
Exceção 01	Título não informado: Após o passo 3 do cenário principal, caso o usuário não tenha informado um título para o marcador, uma exceção é gerada pela aplicação.
Exceção 02	Detalhe não informado: Após o passo 3 do cenário principal, caso o usuário não tenha informado um detalhe para o marcador, uma exceção é gerada pela aplicação.

Pós-condições	Um novo marcador será adicionado no mapa e no banco de dados ou um marcador existente será alterado no mapa e no banco de dados. Observação: Os marcadores manipulados pela aplicação serão exibidos no mapa de acordo com seu estado de visualização individual e o estado conjunto de visualização de marcadores da aplicação.

Quadro 6 – Caso de uso `UC03`

3.3.1.4 Listar marcadores

O quarto caso de uso (Quadro 7), designado `Listar marcadores`, trata da exibição, edição e remoção na lista de marcadores, bem como da definição do estado de visualização individual dos marcadores manipulados pela aplicação. O caso de uso possui o cenário principal, três cenários alternativos, mas nenhum cenário de exceção.

UC04 – Listar marcadores: possibilita ao usuário uma forma de listar os marcadores do mapa, bem como editar, remover ou definir o estado de visualização individual de cada marcador manipulado pela aplicação.	
Requisitos atendidos	RF06.
Pré-condições	A aplicação deve estar executando o caso de uso `UC01`.
Cenário principal	1) O usuário visualiza o mapa onde são exibidos os marcadores de acordo com seu estado de visualização individual e/ou o estado conjunto de visualização de marcadores da aplicação. 2) O usuário opta por listar os marcadores fazendo uso do sub-menu `Listar` do item de menu `Marcadores` do menu principal. 3) A aplicação exibe a tela listando todos os marcadores manipulados pela aplicação. Os marcadores visíveis são exibidos com um indicativo de seleção checado, enquanto os marcadores ocultos são exibidos com o mesmo indicativo não checado. 4) O usuário visualiza a lista de marcadores manipulados pela aplicação.
Fluxo alternativo 01	Visualizar marcador: 1) Após o passo 4 do cenário principal, caso o usuário desejar visualizar um marcador, este deve pressionar o marcador desejado. 2) A aplicação exibe o marcador no mapa com o nível máximo de zoom.
Fluxo alternativo 02	Editar marcador: 1) Após o passo 4 do cenário principal, caso o usuário desejar editar um marcador, este deve pressionar durante quatro segundos o marcador desejado e optar pela opção `Editar` do menu que é exibido pela aplicação. 2) Ver caso de uso `UC03`. 3) A aplicação atualiza o marcados na lista de marcadores manipulados pela aplicação. 4) Voltar ao passo 4 do cenário principal.

Fluxo alternativo 03	Remover marcador: 1) Após o passo 4 do cenário principal, caso o usuário desejar remover um marcador, este deve pressionar durante quatro segundos o marcador desejado e optar pela opção `Remover` do menu que é exibido pela aplicação. 2) A aplicação solicita a confirmação do usuário, que se for positiva, permite que a aplicação remova o marcador do mapa, da lista de marcadores e delete o marcador do banco de dados. 3) Voltar ao passo 4 do cenário principal.
Fluxo alternativo 04	Voltar ao mapa: 1) Após o passo 4 do cenário principal, caso o usuário desejar voltar a visualizar o mapa, este deve optar pelo botão `Voltar`. 2) Ver caso de uso `UC01`.
Pós-condições	O marcador selecionado será exibido no mapa com nível máximo de zoom, e/ou marcadores serão removidos e/ou ainda editados. Observação: Os marcadores manipulados pela aplicação serão exibidos no mapa de acordo com seu estado de visualização individual e/ou o estado conjunto de visualização de marcadores da aplicação. Os marcadores estarão salvos em banco de dados.

Quadro 7 – Caso de uso `UC04`

3.3.1.5 Pesquisar

O quinto caso de uso (Quadro 8), designado `Pesquisar`, trata da pesquisa de endereços ou locais. O caso de uso possui o cenário principal, um cenário alternativo, e um cenário de exceção.

UC05 – Pesquisar: possibilita ao usuário uma forma de pesquisar endereços e/ou locais.	
Requisitos atendidos	RF07.
Pré-condições	A aplicação deve estar executando o caso de uso `UC01`.
Cenário principal	1) O usuário visualiza o mapa onde são exibidos os marcadores de acordo com seu estado de visualização individual e/ou o estado conjunto de visualização de marcadores da aplicação. 2) O usuário opta por pesquisar um endereço ou local fazendo uso do sub-item `Pesquisar` do item de menu `Pesquisa` do menu principal. 3) A aplicação exibe a tela pesquisa com o campo `Endereço/Local` em branco e com a quantidade máxima de registros posicionada no valor padrão da aplicação, cinco primeiros registros. 4) O usuário informa os dados da pesquisa. 5) O usuário opta por pesquisar pressionando o botão `Pesquisar` da tela de pesquisa de endereços ou locais. 6) Ver caso de uso `UC06`.
Fluxo alternativo 01	Voltar ao mapa: 1) Nos passos 4 ou 5 do cenário principal, caso o usuário desejar voltar a visualizar o mapa, este deve optar pelo botão Cancelar. 2) Ver caso de uso UC01.

Exceção 01	Endereço ou localidade não informada: Após o passo 5 do cenário principal, caso o usuário não tenha informado um endereço ou local, uma exceção é gerada pela aplicação.
Pós-condições	O endereço ou local selecionado será exibido no mapa com nível máximo de zoom, ou o mapa será exibido com seu estado anterior à exibição da tela de pesquisa de endereços ou locais.

Quadro 8 – Caso de uso `UC05`

3.3.1.6 Listar resultados da pesquisa

O sexto caso de uso (Quadro 9), designado `Listar resultados da pesquisa`, trata da visualização dos resultados de endereços e/ou locais da última pesquisa efetuada pelo usuário. O caso de uso possui o cenário principal e dois cenários alternativos. Não existem cenários de exceção para este caso de uso.

UC06 – Listar resultados da pesquisa: possibilita ao usuário uma forma de listar os endereços e/ou locais da última pesquisa efetuada pelo usuário.	
Requisitos atendidos	RF07.
Pré-condições	A aplicação estava executando o caso de uso `Explorar mapa` (`UC01`) quando o usuário desejou listar os endereços e/ou locais retornados pela última pesquisa, optando pelo sub-item `Últimos Resultados` do item de menu `Pesquisa` do menu principal. Também é possível que a aplicação estivesse executando o caso de uso `Pesquisar` (`UC05`) e o usuário tenha concluído o cenário principal.
Cenário principal	1) A aplicação exibe a tela com todos os resultados da última pesquisa. 2) O usuário visualiza a lista de resultados da última pesquisa.
Fluxo alternativo 01	Visualizar resultado: 1) Após o passo 2 do cenário principal, caso o usuário desejar visualizar um resultado, este deve pressionar o resultado desejado. 2) A aplicação exibe o resultado no mapa com o nível máximo de zoom.
Fluxo alternativo 02	Voltar ao mapa: 1) Após o passo 2 do cenário principal, caso o usuário desejar voltar a visualizar o mapa, este deve optar pelo botão `Voltar`. 2) Ver caso de uso `UC01`.
Pós-condições	O resultado selecionado será exibido no mapa com nível máximo de zoom, ou o mapa será exibido com seu estado anterior a exibição da lista de marcadores. Os marcadores manipulados pela aplicação serão exibidos de acordo com seu estado de visualização individual e o estado conjunto de visualização de marcadores da aplicação.

Quadro 9 – Caso de uso `UC06`

3.3.2 Diagrama de classes

O diagrama de classes apresenta uma visão de como as classes estão estruturadas e

relacionadas. Nesta seção são descritas as classes necessárias para o desenvolvimento do Android MapBr, uma aplicação com o intuito disponibilizar algumas das funcionalidades dispostas por ferramentas de visualização de mapas, tal como o Google Maps, no ambiente móvel da plataforma Android. Nesse contexto, para uma visualização que favoreça o melhor entendimento, as classes estão reunidas, conforme as ligações lógicas entre si, em pacotes (Figura 4).

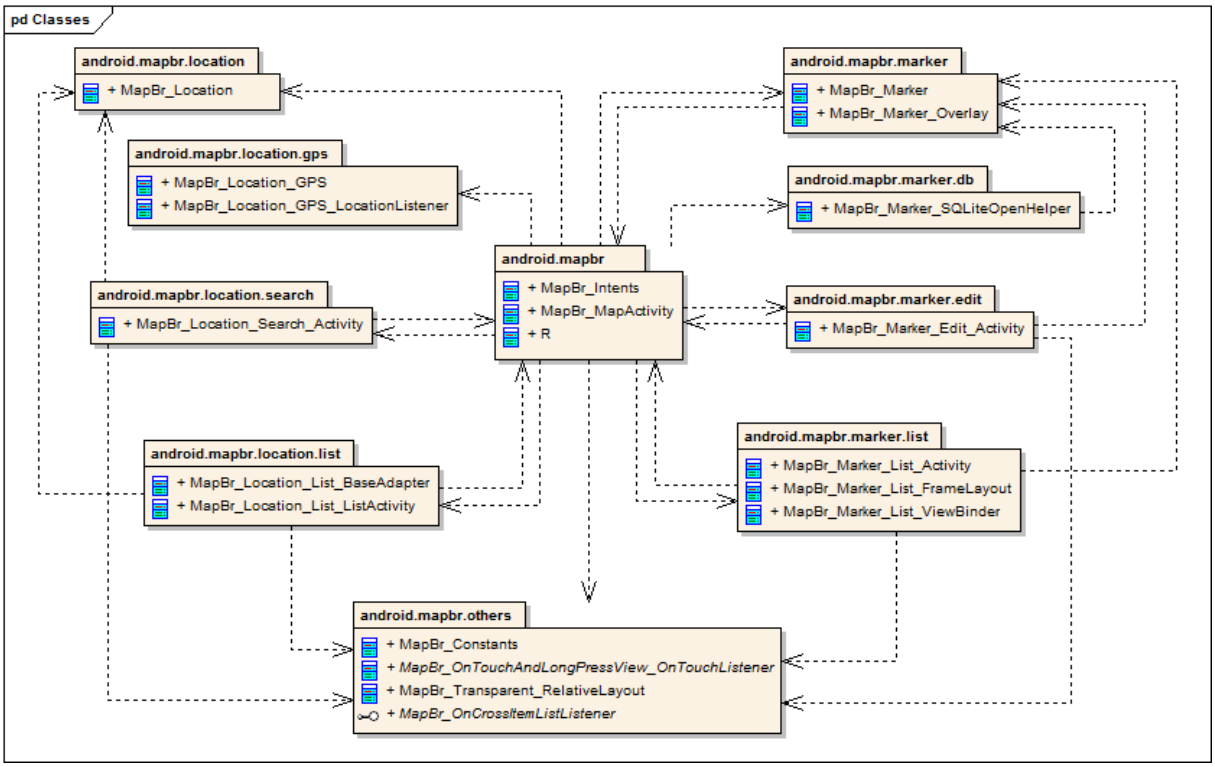


Figura 4 – Pacotes da biblioteca

3.3.2.1 Pacote `android.mapbr`

O primeiro pacote (Figura 5) é denominado `android.mapbr` e contém a classe que centraliza as intenções, a classe principal da aplicação, bem como uma das classes de constantes utilizadas pelas demais classes da aplicação. Fazem parte deste pacote a classe `MapBr_Intents`, a classe `MapBr_MapActivity` e a classe Android `R`.

A classe `MapBr_Intents` representa a classe que centraliza e generaliza a chamada de uma nova intenção. Todas as intenções dessa classe são utilizadas para possibilitar o inicio de uma atividade disponibilizada pela aplicação e dessa forma, a classe `MapBr_Intents` é usada por outras classes da aplicação quando deseja-se o início de uma determinada atividade em função de alguma ação do usuário. Todos os métodos da classe realizam a chamada de uma

intenção em especifico. Mais detalhes sobre essa classe serão descritos na seção 3.4.2.

Já a classe `MapBr_MapActivity` representa a extensão da classe Android `MapActivity`, uma classe base que tem por intuito gerenciar as necessidades de qualquer classe que exiba uma instância da classe Android `MapView`. A classe `MapActivity` tem como algumas de suas responsabilidades controlar o ciclo de vida da atividade e os serviços necessários à classe `MapView`. Esta última por sua vez representa uma `View` que exibe um mapa, cujos dados são obtidos através do serviço Google Maps. A classe `MapView` pode desenhar sobreposições (classe Android `Overlay`) no topo do mapa. A classe `MapBr_MapActivity` é a classe base da aplicação e desse forma é onde encontra-se definida a criação do menu principal, de todos os métodos ligados a cada item do menu, bem como os estados que afetam a disponibilidade de cada item em função de objetos e variáveis locais controladas pela classe. O método `onActivityResult()` é responsável por tratar o retorno das intenções iniciadas a partir da atividade da classe principal, a classe de atividade de mapa `MapBr_MapActivity`. No método `onCreate()` da classe `MapBr_MapActivity` é especificado o *layout* da atividade do mapa. Ainda nessa classe fez-se uma referência à classe `MapBr_OnTouchAndLongPressView_OnTouchListener` para implementar uma forma de simular o evento de pressionamento longo de um ponto de coordenadas `x` e `y` no mapa.

A classe Android `R` contempla classes internas de constantes inteiras, geradas pela compilação de todos os recursos existentes na aplicação, tal como atributos, elementos visuais (ícones, imagens, etc.), elementos de *layout* e cadeias de caracteres de valores diversos. Esta classe é automaticamente gerada pelo ambiente Android quando da construção do projeto. Conteúdo adicional referente aos recursos pode ser obtido na referência Google (2008h).

Figura 5 – Pacote `android.mapbr`

3.3.2.2 Pacote `android.mapbr.marker`

O segundo pacote (Figura 6) é denominado `android.mapbr.marker` e contém todas as classes relacionadas aos tratamentos inerentes a criação e a visualização de um marcador. Fazem parte deste pacote a classe `MapBr_Marker` e a classe `MapBr_Marker_Overlay`.

A classe `MapBr_Marker` define os atributos e métodos de um marcador. É utilizada por outras classes da aplicação quando se faz necessário a criação de um novo marcador ou a obtenção da última instância da classe de marcador a ser criada. Obter esta instância é possível uma vez que a classe implementa o padrão de projeto de software (do inglês *Design Pattern*) *Singleton*[17] com algumas alterações. Optou-se por disponibilizar de forma pública a chamada do método `createInstance()`, responsável pela criação de uma nova instância da classe, de forma a possibilitar a criação um novo marcador sempre que necessário. A última instância da classe marcador pode ser obtida através do método `getInstance()` e caso for necessário, pode ser atribuída através do método `setInstance()`.

A classe `MapBr_Marker_Overlay` representa a extensão da classe Android `Overlay`, uma classe base que atua como uma sobreposição, que pode ser exibida no topo de um mapa. Uma instância dessa classe é adicionada ao mapa da aplicação. Nela existe uma lista de marcadores que é sincronizada com a lista de marcadores da aplicação. É com base na lista interna de marcadores dessa classe que cada marcador é desenhado na sobreposição, ou seja, na camada implementada pela classe `MapBr_Marker_Overlay`. Ainda nesta classe estão disponíveis os métodos `getClickedMapBr_Marker()` e `onTap()` que verificam o evento do clique em marcador, bem como o método `drawInfoWindow()`, que desenha acima do marcador um retângulo contendo o seu respectivo título. Também é nessa classe que pode ser definido o estado conjunto de visualização dos marcadores da aplicação por meio do método `invertShowMarkers()`.

[17] Singleton é um padrão de projeto de software que garante a existência de apenas uma instância de uma classe, mantendo um único ponto global de acesso ao objeto.

Figura 6 – Pacote `android.mapbr.marker`

3.3.2.3 Pacote `android.mapbr.marker.db`

O terceiro pacote (Figura 7) é denominado `android.mapbr.marker.db` e contém a

classes relacionada a persistência de um marcador. Apenas a classe `MapBr_Marker_SQLiteOpenHelper` faz parte deste pacote. Esta classe representa uma extensão da classe Android `SQLiteOpenHelper`, e disponibiliza métodos que atendem a todas as operações de banco de dados inerentes a um marcador, tais como a inserção, alteração, deleção de marcadores. Ela possui um método para o retorno de um cursor e dois outros para a realização da seleção com retorno de um marcador em específico ou de todos os marcadores em forma de lista. Esta classe também dispõe de métodos para a criação e atualização da tabela de marcadores. Encontram-se na classe `MapBr_Marker_SQLiteOpenHelper` constantes relacionadas aos nomes dos campos e índices das colunas da tabela de marcadores.

Figura 7 – Pacote `android.mapbr.marker.db`

3.3.2.3.1 Pacote `android.mapbr.marker.edit`

O quarto pacote é denominado `android.mapbr.marker.edit` e contém a classe relacionada à implementação da atividade que realiza a edição de um marcador (Figura 8). Apenas a classe `MapBr_Marker_Edit_Activity` faz parte deste pacote. Esta classe atua

como uma extensão da classe de atividade Android `Activity`, que representa uma atividade com a qual o usuário pode interagir. A classe de atividade tem como uma de suas responsabilidades a criação de uma nova janela, deixando a cargo do desenvolvedor o direito de especificar a visão (classe Android `View`) que irá servir de interface com o usuário. A classe `MapBr_Marker_Edit_Activity` é usada por outras classes da aplicação quando se faz necessário editar um novo marcador ou alterar um marcador existente. Nela estão descritas as atribuições de valores aos componentes visuais, bem como a definição das ações ligadas aos botões da tela de edição. Também é nesta classe onde são feitas as consistências relacionadas à edição de um marcador. No método `onCreate()` da classe `MapBr_Marker_Edit_Activity` é especificado o *layout* da atividade, ou seja, a interface com o usuário.

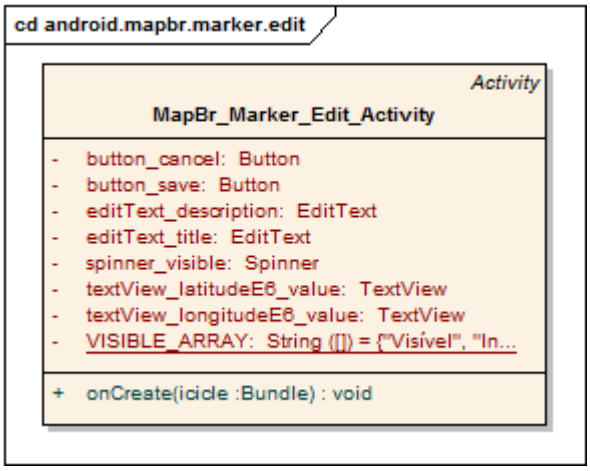


Figura 8 – Pacote `android.mapbr.marker.edit`

3.3.2.3.2 Pacote `android.mapbr.marker.list`

O quinto pacote é denominado `android.mapbr.marker.list` e contém classes relacionadas a exibição e funcionalidades disponibilizadas pela atividade da lista de marcadores (Figura 9). Fazem parte deste pacote a classe `MapBr_Marker_List_FrameLayout`, a classe `MapBr_Marker_List_Activity` e a classe `MapBr_Marker_List_ViewBinder`.

A classe `MapBr_Marker_List_FrameLayout` representa uma extensão da classe Android `FrameLayout` e atua como interface visual da lista de marcadores. A classe Android `FrameLayout` é projetada para bloquear uma área da tela a fim de exibir um único item. Nesta classe é possível adicionar vários elementos a interface visual, os quais são ancorados ao canto superior esquerdo da tela. Os elementos são desenhados em uma pilha, onde o último elemento adicionado está presente no topo. A classe `MapBr_Marker_List_FrameLayout` tem

como intuito interceptar os eventos de toque realizados pelo usuário, verificando se estes representam um movimento horizontal para, em caso positivo, invocar os ouvintes (do inglês *listeners*) associados à classe `MapBr_Marker_List_FrameLayout`, por meio do método `onCross()` de cada ouvinte.

Já a classe `MapBr_Marker_List_Activity` representa uma extensão da classe de atividade Android `Activity`. A classe de atividade tem como uma de suas responsabilidades a criação de uma nova janela, deixando a cargo do desenvolvedor o direito de especificar a visão (classe Android `View`) que irá servir de interface com o usuário. A classe `MapBr_Marker_List_Activity` também implementa a interface `MapBr_OnCrossListener` e é responsável por vincular as respectivas ações a cada um dos botões da atividade de lista de marcadores, bem como definir o menu relacionado a cada marcador da lista. Também é nela que um objeto da classe `MapBr_Marker_List_FrameLayout` é instanciado, para que a implementação do método `onCrossItemList()` seja associado a esta classe. Essa associação se faz necessária para tratar os eventos de toque realizados pelo usuário. Ainda na classe `MapBr_Marker_List_Activity` cria-se uma referência à instância da classe `ListView`, que representa o componente visual em foram de lista que irá exibir todos os marcadores. O método `onActivityResult()` da classe `MapBr_Marker_List_Activity` é responsável por tratar o retorno das intenções iniciadas a partir desta atividade. No método `onStart()` da atividade uma instância da classe `MapBr_Marker_SQLiteOpenHelper` é criada e um cursor com os marcadores gravados em banco de dados é obtido a partir desta instância. Este cursor é usado para que a instância da classe `SimpleCursorAdapter` mapeie as colunas existentes para um marcador em forma de elementos visuais que formarão cada um dos itens da lista de marcadores. Dentro deste mesmo método é atribuído uma instância da classe `MapBr_Marker_List_ViewBinder` ao adaptador da lista de marcadores. O adaptador por sua vez é atribuído a lista de marcadores. No método `onCreate()` da classe `MapBr_Marker_List_Activity` é especificado o *layout* da lista de atividades, ou seja, a interface com o usuário.

Por fim, a classe `MapBr_Marker_List_ViewBinder` implementa a classe Android `ViewBinder`, cuja função é vincular dados de cursores em forma de visões, ou seja, em instâncias da classe Android `View`. A classe `ViewBinder` também pode ser usada para vincular valores que não são diretamente suportados pela classe `SimpleCursorAdapter`, ou para alterar o modo com que as visões suportadas são exibidas. Em outras palavras, ela define como os dados são dispostos e exibidos. A classe `MapBr_Marker_List_ViewBinder` é usada

para que através do método `setViewValue()` seja possível definir o conteúdo das visões existentes para cada item da lista de marcadores.

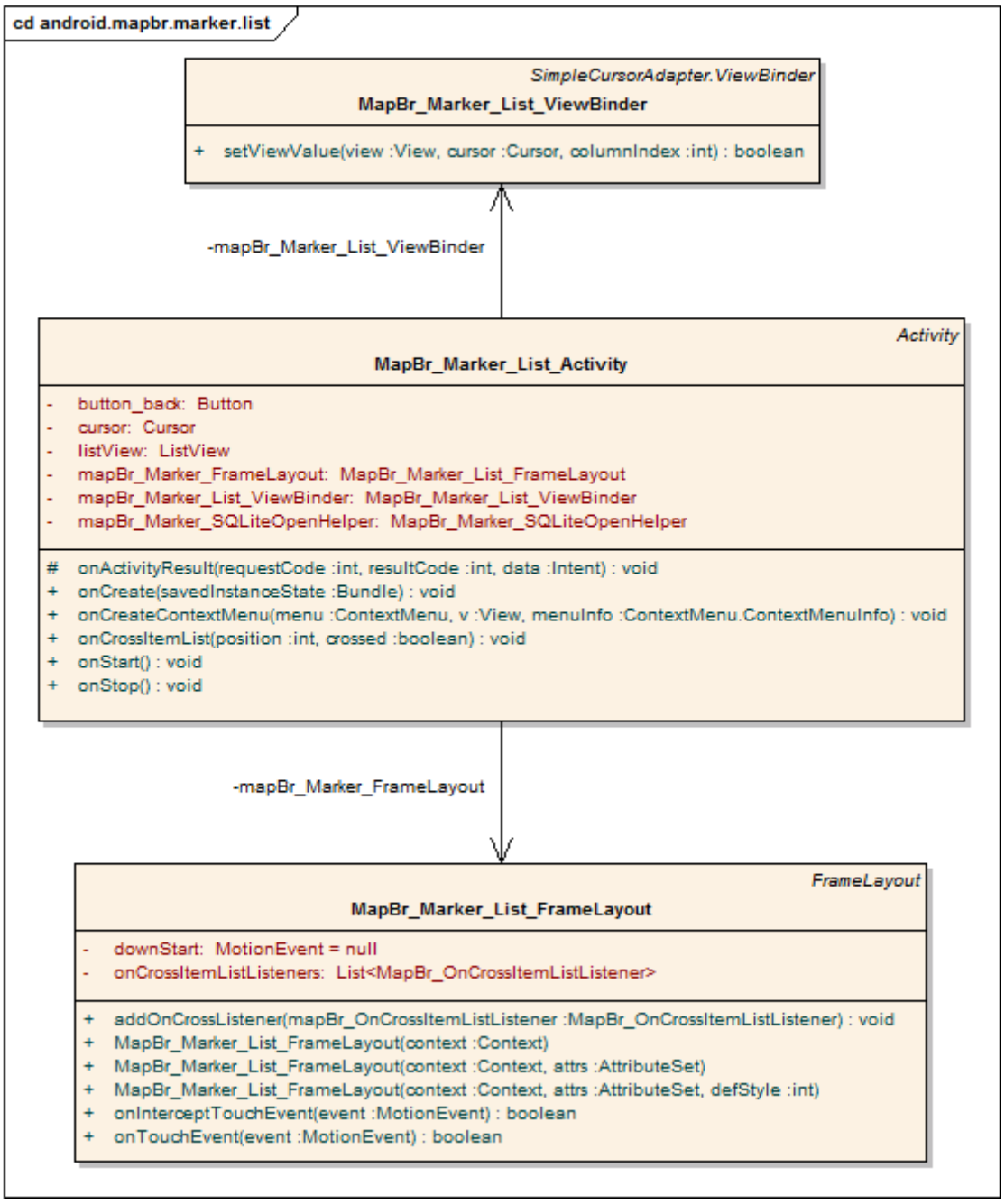


Figura 9 – Pacote `android.mapbr.marker.list`

3.3.2.4 Pacote `android.mapbr.location`

O sexto pacote (Figura 10) é denominado `android.mapbr.location` e contém apenas a classe de localidade `MapBr_Location`. Esta classe contém um atributo estático que representa uma lista de objetos da classe Android `Address`, uma classe que representa um

endereço, ou seja, um conjunto de `Strings` que descreve um local. A classe `MapBr_Location` disponibiliza dois métodos estáticos, o primeiro `getAddresses()`, para retornar a lista de endereços e o segundo `setAddresses()`, para atribuir endereços de uma lista à lista estática da classe `MapBr_Location`.

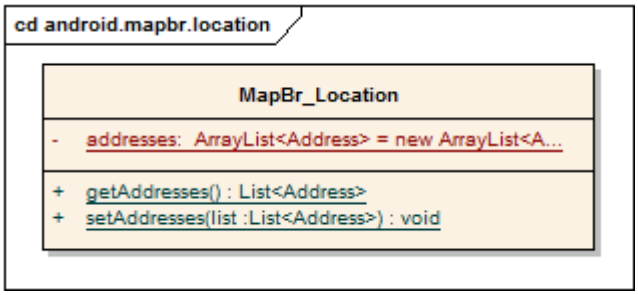


Figura 10 – Pacote `android.mapbr.location`

3.3.2.4.1 Pacote `android.mapbr.location.gps`

O sétimo pacote (Figura 11) é denominado `android.mapbr.location.gps` e contém classes relacionadas a obtenção da posição GPS do dispositivo móvel. Fazem parte deste pacote a classe `MapBr_Location_GPS` e a classe `MapBr_Location_GPS_LocationListener`.

A classe `MapBr_Location_GPS` contempla uma instância da classe Android `LocationManager`, classe que provê acesso à rede local de serviços. Os serviços permitem que as aplicações obtenham atualizações periódicas do dispositivo de localização geográfica. Também faz parte da classe `MapBr_Location_GPS` uma instância da classe `MapBr_Location_GPS_LocationListener`. Ainda na classe `MapBr_Location_GPS` estão disponíveis métodos para o retorno da latitude e longitude referentes à posição GPS do dispositivo móvel.

Já a classe `MapBr_Location_GPS_LocationListener` implementa a interface `LocationListener`. Esta interface é usada para que seja possível receber notificações a partir de uma instância da classe Android `LocationManager` quando a localização for alterada.

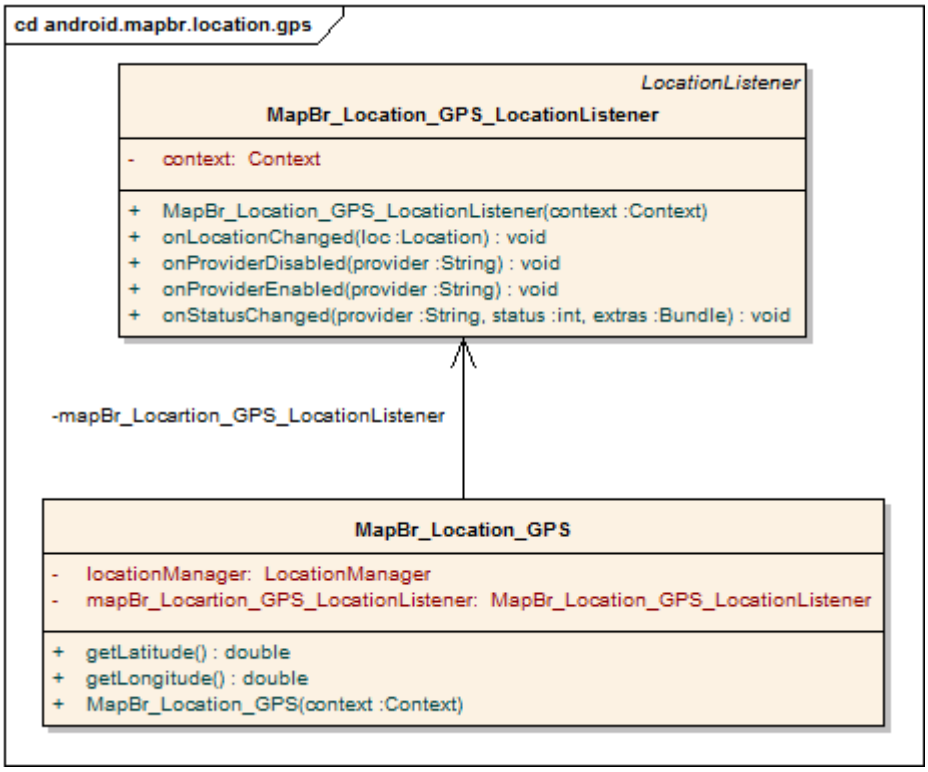


Figura 11 – Pacote `android.mapbr.location.gps`

3.3.2.4.2 Pacote `android.mapbr.location.list`

O oitavo pacote é denominado `android.mapbr.location.list` e contém as classes responsáveis pela implementação da lista de endereços e/ou locais (Figura 12). Fazem parte deste pacote a classe `MapBr_Location_List_BaseAdapter` e a classe `MapBr_Location_List_ListActivity`.

A classe `MapBr_Location_List_BaseAdapter` realiza a extensão da classe Android `BaseAdapter`, uma classe base para a implementação de um adaptador que pode ser usado com os componentes visuais Android `ListView` (implementando a interface Android `ListAdapter`) e `Spinner` (implementando a interface `SpinnerAdapter`). A classe `MapBr_Location_List_BaseAdapter` fornece métodos para o retorno da quantidade de endereços da lista, retorno de um objeto específico ou retorno do identificador de um objeto na lista. O método `getView()` é usado para o retorno das visões que representam cada item da lista de endereços, ou seja, cada endereço da lista. O método `setList()` é usado para atribuir uma lista de endereços à lista interna da classe `MapBr_Location_List_BaseAdapter`.

Já a classe `MapBr_Location_List_ListActivity` realiza a extensão da classe Android `ListActivity`. A classe `ListActivity` representa uma atividade que exibe uma

lista de itens com base em uma fonte de dados, como um *array* ou um cursor. O tratador de evento `onListItemClick()` é disponibilizado para que os tratamentos necessários sejam executados quando o usuário seleciona um item da lista. A classe `ListActivity` hospeda uma `ListView` que é responsável por exibir os itens da lista. No método `onCreate()` da classe `MapBr_Location_List_ListActivity` é especificado o *layout* da lista de atividades, ou seja, a interface com o usuário, são vinculadas as ações ligadas aos botões da tela e atribuída a instância da classe `MapBr_Location_List_BaseAdapter` como adaptador da lista de atividades.

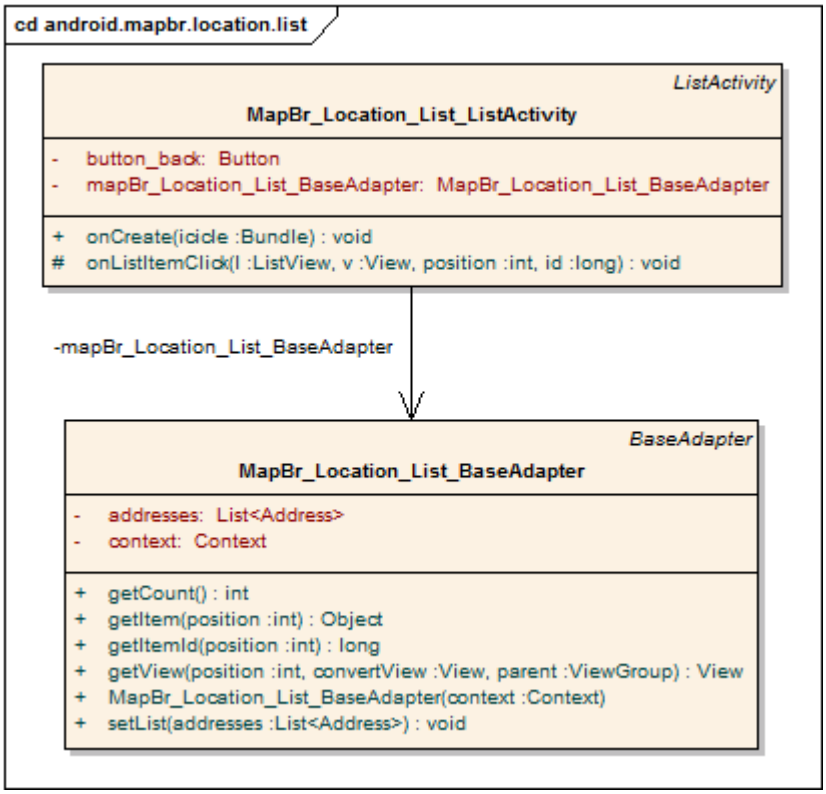


Figura 12 – Pacote `android.mapbr.location.list`

3.3.2.4.3 Pacote `android.mapbr.location.search`

O nono pacote é denominado `android.mapbr.location.search` e contém apenas uma classe, a classe `MapBr_Location_Search_Activity` (Figura 13), cujo intuito é possibilitar a pesquisa de endereços e/ou locais. Esta classe representa uma extensão da classe Android `Activity`. A classe de atividade tem como uma de suas responsabilidades a criação de uma nova janela, deixando a cargo do desenvolvedor o direito de especificar a visão (classe Android `View`) que irá servir de interface com o usuário. No método `onCreate()` da

classe `MapBr_Location_Search_Activity` é especificado o *layout* da atividade, ou seja, a interface com o usuário e são vinculadas as ações ligadas aos botões da tela.

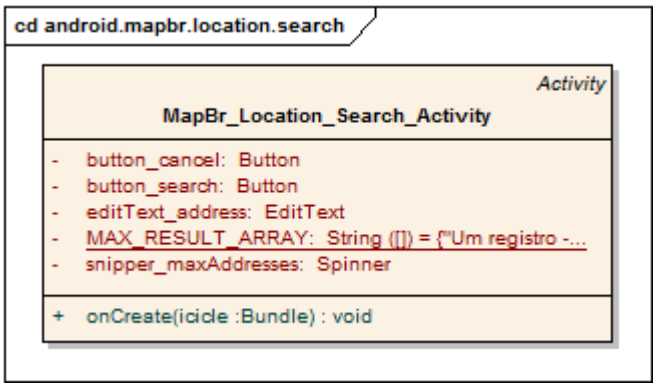


Figura 13 – Pacote `android.mapbr.location.search`

3.3.2.5 Pacote `android.mapbr.others`

O décimo pacote (Figura 14) é denominado `android.mapbr.others` e contém classes de constantes e elementos visuais, bem como interfaces de eventos usados pela aplicação Android MapBr. Fazem parte deste pacote a classe `MapBr_Constants`, a classe `MapBr_OnTouchAndLongPressView_OnTouchListener`, como também a classe visual `MapBr_Transparent_RelativeLayout` e a interface `MapBr_OnCrossItemListListener`.

A classe `MapBr_Constants` representa a classe de constantes globais a todas as demais classes da aplicação. Nesta classe estão definidas as constantes inteiras que representam os códigos de requisição que representam de forma única cada uma das intenções disponíveis à aplicação. Também estão descritos todos os possíveis códigos de resultado a serem tratados no retorno das intenções. Nessa classe estão definidos também as ações que representam as intenções disponibilizadas pela aplicação Android MapBr. Ainda nessa classe existe a definição da constante `E6`, usada pela Google para realizar a conversão entre graus e micro-graus[18].

Já a classe abstrata `MapBr_OnTouchAndLongPressView_OnTouchListener` implementa a interface Android `OnTouchListener`, uma interface para a chamada de evento quando um toque é realizado em uma visão. A chamada será feita antes que o evento de toque seja realizado na visão. A classe `MapBr_OnTouchAndLongPressView_OnTouchListener` é responsável por detectar o evento de toque longo em uma visão.

[18] Micro-graus é o valor de um grau multiplicado por 1000000.

A classe `MapBr_Transparent_RelativeLayout` representa uma extensão da classe Android `RelativeLayout`, uma classe visual onde a posição dos elementos pode ser descrita em relação aos demais elementos ou em relação ao elemento pai. A classe `MapBr_Transparent_RelativeLayout` representa uma classe Android `RelativeLayout` transparente.

Por fim, a interface `MapBr_OnCrossItemListListener` descreve a assinatura do método `onCrossItemList()`, que trata o toque horizontal em um elemento da classe visual Android `ListView`.

Figura 14 – Pacote `android.mapbr.others`

3.3.3 Diagrama de seqüência

O diagrama de seqüência apresenta uma visão interna do processo e da comunicação entre as classes para cada caso de uso. Nesta seção são apresentados apenas os diagramas de seqüência referentes aos casos de uso `UC03` (Figura 15) e `UC05` (Figura 16).

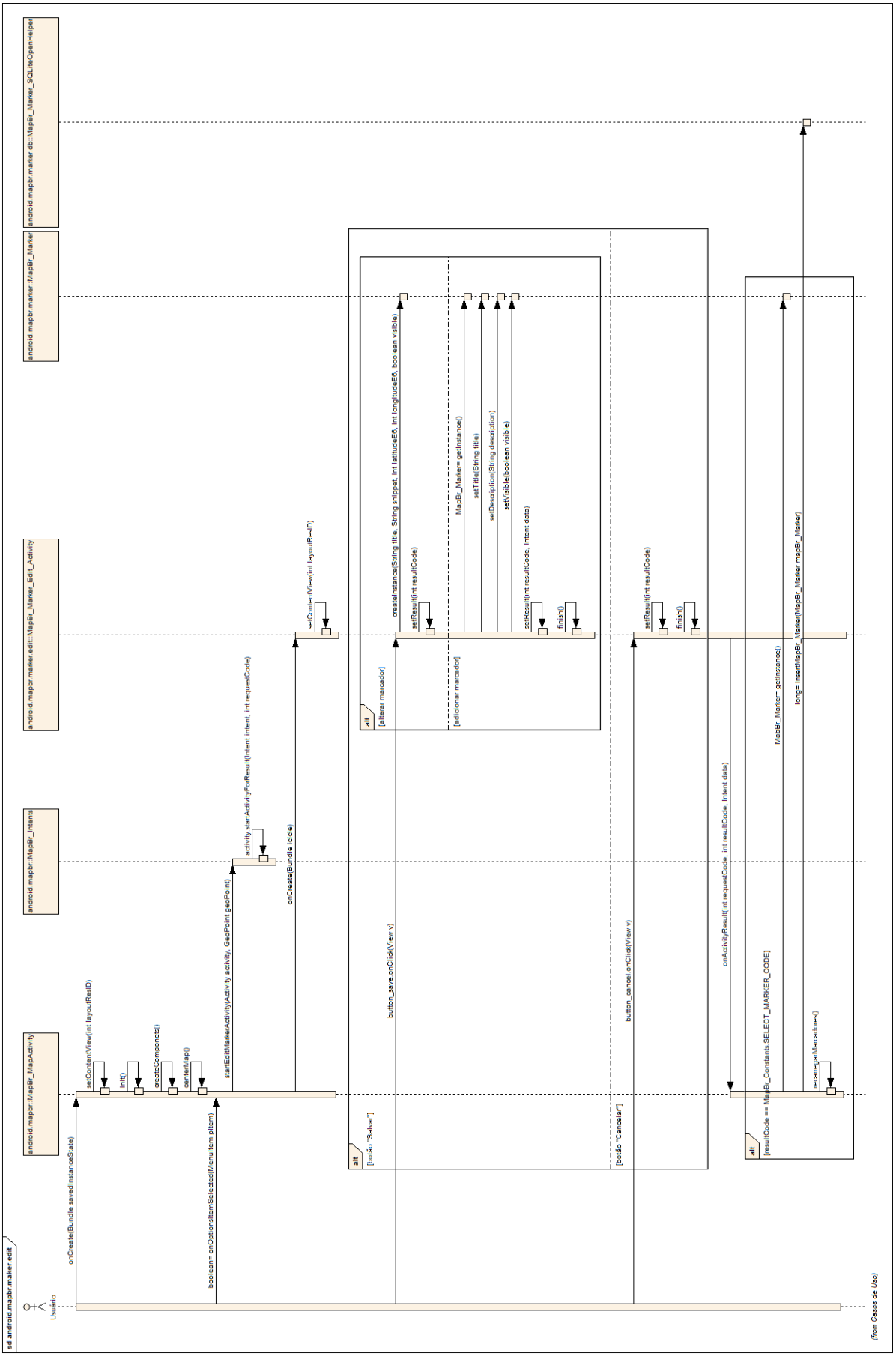


Figura 15 – Diagrama de seqüência do caso de uso UC03

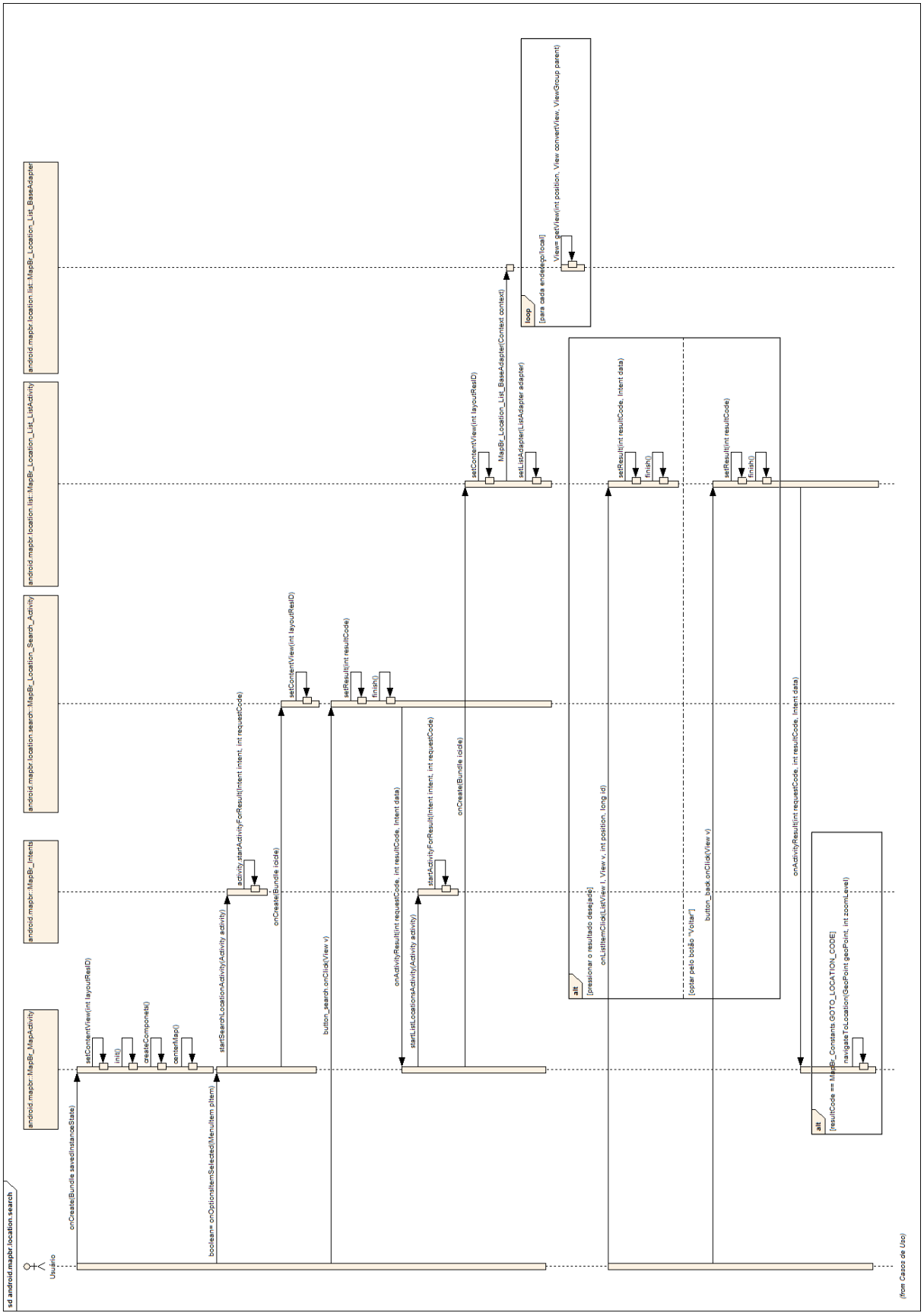


Figura 16 – Diagrama de seqüência do caso de uso UC04

3.4 IMPLEMENTAÇÃO

A seguir são descritas as técnicas e ferramentas utilizadas na implementação, bem como detalhes das principais classes e rotinas implementadas durante o desenvolvimento da aplicação Android MapBr.

3.4.1 Técnicas e ferramentas utilizadas

A aplicação Android MapBr foi implementada sob o paradigma da orientação a objetos na linguagem programação Java, utilizando-se o ambiente de desenvolvimento Eclipse IDE. Em conjunto com os recursos da linguagem de programação da aplicação foi utilizado o pacote de desenvolvimento de software Android, que disponibiliza todas as ferramentas e APIs necessárias para o desenvolvimento de aplicativos para a plataforma Android na linguagem de programação Java. Também foi utilizado o paradigma de intenções proposto em Android.

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

Optou-se por utilizar juntamente com o ambiente de desenvolvimento Eclipse IDE um *plugin* personalizado chamado Android *Development Tools* (ADT), que integra e adiciona suporte para projetos e ferramentas Android. De forma geral, o *plugin* ADT inclui uma variedade de recursos que tornam a criação, execução e a depuração de aplicações Android mais intuitiva e produtiva ao desenvolvedor.

3.4.2 As intenções

Segundo Google (2008d), a classe Android `Intent` representa a descrição resumida de uma operação que deve ser realizada. As intenções pode ser usadas em conjunto com o método `startActivity()` quando se faz necessário iniciar uma atividade, com o método `sendBroadcast()` para encaminhar uma determinada intenção a todos as instancias da classe

Android `BroadcastReceiver` interessadas, ou ainda juntamente com os métodos `startService()` e `bindService()` para efetuar a comunicação com serviços. Diante de todas as formas de desenvolvimento ligadas a classe de intenção, o uso mais significativo para as intenções reside no fato de que esta classe possibilita o início de atividades. Nesse contexto uma intenção atua como o meio de ligação entre as atividades envolvidas.

De uma forma geral, Google (2008c) relata que em Android uma nova atividade, ou seja, uma instância da classe `Activity` pode ser iniciada a partir de dois métodos distintos. O método `startActivity()` dá início a uma nova atividade sem que a aplicação receba qualquer retorno quando esta for encerrada. Da mesma forma, o método `startActivityForResult()` também inicia uma nova atividade a partir da qual pode-se verificar resultados após ter sido encerrada. Quando a atividade iniciada pelo método `startActivityForResult()` termina, o método `onActivityResult()` da atividade origem, ou seja, da atividade que invocou o método `startActivityForResult()` é chamado contendo o código da atividade que teve sua execução encerrada. Ainda nesse método é possível usufruir de códigos de retorno de resultados e de um *container* de valores diversos que foram atribuídos na atividade encerrada.

Diante desses conceitos, a classe `MapBr_Intents` representa a classe que centraliza e generaliza a chamada de uma nova intenção na aplicação Android MapBr. Todas as intenções dessa classe são utilizadas para possibilitar o inicio de uma atividade disponibilizada pela aplicação e dessa forma, a classe `MapBr_Intents` é usada por outras classes da aplicação quando deseja-se o início de uma determinada atividade em função de alguma ação do usuário. Todos os métodos da classe realizam a chamada de uma intenção em especifico. Quatro intenções (Quadro 10) distintas são disponibilizadas pela classe `MapBr_Intents`, sendo estas, intenções relacionadas as tarefas descritas abaixo:

a) editar um marcador;
b) pesquisar localidade;
c) listar os resultados da pesquisa;
d) listar marcadores.

Para a intenção que realiza a edição de um marcador criou-se algumas sobrecargas de método com parâmetros diferentes.

Todos os métodos da classe `MapBr_Intents` possuem como primeiro parâmetro uma instância da classe Android `Activity`, sobre a qual a nova intenção será invocada. Essa referencia é necessária para que a nova intenção seja iniciada a partir da atividade que

necessita da intenção. Ainda sobre esta técnica de desenvolvimento salienta-se que apenas dessa forma foi possível centralizar a chamada de uma intenção específica, mesmo quando a intenção é comum a diversas classes da aplicação, ou seja, quando é usada em atividades diferentes dentro da aplicação. Dessa forma, apensar da existência de intenções que podem ser chamadas de atividades diferentes dentro da aplicação, onde cada atividade trata o retorno da intenção de forma individualizada em seu método `onActivityResult()`, com a referência da atividade ao chamar uma nova intenção garante-se que o retorno da execução seja feito para a origem da chamada, ou seja, para a atividade que requisitou a intenção.

```
public static void startEditMarkerActivity(Activity activity, Intent intent) {
    activity.startActivityForResult(intent, MapBr_Constants.EDIT_MARKER_CODE);
}

public static void startSearchLocationActivity(Activity activity) {
    Intent intent = new Intent(MapBr_Constants.SEARCH_LOCATION_INTENT);
    activity.startActivityForResult(intent,
MapBr_Constants.SEARCH_LOCATION_CODE);
}

public static void startListLocationsActivity(Activity activity) {
    Intent intent = new Intent(MapBr_Constants.LIST_LOCATIONS_INTENT);
    activity.startActivityForResult(intent,
MapBr_Constants.SEARCH_LOCATION_CODE);
}

public static void startListMarkersActivity(Activity activity) {
    Intent intent = new Intent(MapBr_Constants.LIST_MARKERS_INTENT);
    activity.startActivityForResult(intent,
MapBr_Constants.SEARCH_LOCATION_CODE);
}
```

Quadro 10 – Intenções da classe `MapBr_Intents`

É possível ainda iniciar uma atividade relacionando diretamente na intenção, em código Java, qual classe é responsável pela implementação da atividade, conforme exemplificado no Quadro 11.

```
public static void startEditMarkerActivity(Activity activity, Intent intent) {
    activity.startActivityForResult(intent, MapBr_Constants.EDIT_MARKER_CODE);
}

public static void startSearchLocationActivity(Activity activity) {
    Intent intent = new Intent(activity,
MapBr_Location_Search_Activity.class);
    activity.startActivityForResult(intent,
MapBr_Constants.SEARCH_LOCATION_CODE);
}

public static void startListLocationsActivity(Activity activity) {
    Intent intent = new Intent(activity,
MapBr_Location_List_ListActivity.class);
    activity.startActivityForResult(intent,
MapBr_Constants.LIST_LOCATIONS_CODE);
}

public static void startListMarkersActivity(Activity activity) {
    Intent intent = new Intent(activity, MapBr_Marker_List_Activity.class);
    activity.startActivityForResult(intent,
MapBr_Constants.LIST_MARKERS_CODE);
}
```

Quadro 11 – Início de uma atividade a partir de uma intenção relacionada à classe da atividade

Com as intenções definidas, para invocar uma atividade fazendo uso da classe de intenções é necessário apenas realizar a chamada de método da intenção desejada, passando como parâmetro a atividade em execução seguida dos demais parâmetros inerentes a cada um dos métodos da classe de intenções. O Quadro 12 ilustra a chamada da intenção referente à edição de um marcador a partir do item de menu `Editar` da lista de marcadores da classe principal da aplicação.

```
MenuItem menuItem_edit = menu.add("Editar");
menuItem_edit.setOnMenuItemClickListener(new OnMenuItemClickListener() {
    @Override
    public boolean onMenuItemClick(MenuItem item) {
          MapBr_Intents.startEditMarkerActivity(MapBr_Marker_List_Activity.thi
s, cursor, position);
          return true;
    }
});
```

Quadro 12 – Chamada da intenção `android.mapbr.marker.edit` a partir do item de menu `Editar` da lista de marcadores

Torna-se necessário, no entanto, que todas as atividades presente na aplicação, bem como suas respectivas intenções estejam declaradas no arquivo `AndroidManifest.xml` (Quadro 13). Essa necessidade existe para ambas às formas descritas visando o início de uma atividade a partir de uma intenção. Segundo Google (2008o), o aspecto da declaração das atividades e intenções é de extrema importância para uma aplicação em Android. Os *intent filters*, ou seja, os filtros de intenções presentes no arquivo `AndroidManifest.xml` descrevem onde e como uma atividade pode ser iniciada. Quando uma atividade ou mesmo o sistema operacional pretende realizar uma ação, cria-se uma instancia da classe Android `Intent`. Essa instancia pode conter vários descritores que especificam inúmeras informações, dentre as

quais se encontra o que pretende-se realizar. Android compara as informações de uma intenção com cada filtro de intenção exposto por qualquer aplicação para verificar qual a atividade mais adequada a fim de lidar com os dados ou a ação especificada pela intenção. A especificação da estrutura e a descrição detalhada dos marcadores disponíveis no arquivo `AndroidManifest.xml` bem como informações relacionadas podem ser encontradas na referência Google (2008o).

```
<?xml version="1.0" encoding="utf-8"?>

<manifest xmlns:android=http://schemas.android.com/apk/res/android
    package="android.mapbr"
    android:versionCode="1"
    android:versionName="1.0.0">

  <!-- Permissions -->
  <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
  <uses-permission android:name="android.permission.INTERNET" />

  <application android:icon="@drawable/icon" android:label="@string/app_name">

    <!-- Libraries -->
    <uses-library android:name="com.google.android.maps" />

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

    <activity android:name=".marker.edit.MapBr_Marker_Edit_Activity"
              android:label="@string/app_name">
      <intent-filter>
        <action android:name="android.mapbr.marker.edit" />
        <category android:name="android.intent.category.DEFAULT" />
      </intent-filter>
    </activity>

    <activity android:name=".location.search.MapBr_Location_Search_Activity"
           android:label="@string/app_name">
      <intent-filter>
        <action android:name="android.mapbr.location.search" />
        <category android:name="android.intent.category.DEFAULT" />
      </intent-filter>
    </activity>

    <activity android:name=".location.list.MapBr_Location_List_ListActivity"
           android:label="@string/app_name">
      <intent-filter>
        <action android:name="android.mapbr.location.list" />
        <category android:name="android.intent.category.DEFAULT" />
      </intent-filter>
    </activity>

    <activity android:name=".marker.list.MapBr_Marker_List_Activity"
           android:label="@string/app_name">
      <intent-filter>
        <action android:name="android.mapbr.marker.list" />
        <category android:name="android.intent.category.DEFAULT" />
      </intent-filter>
    </activity>

  </application>

</manifest>
```

Quadro 13 – Arquivo `AndroidManifest.xml`

3.4.3 A interface visual em XML

Considera-se a criação da interface visual de uma tela por meio da codificação direta na linguagem programação Java um trabalho oneroso, pois esta tarefa exige visualização imaginária do contexto da interface antes mesmo da sua execução, bem como o conhecimento dos elementos e seus possíveis relacionamentos em termos de aglutinação, alinhamento, visualização, posicionamento, entre outros. Visando tornar esta tarefa mais simples, conforme descrito em Google (2008j), Android disponibiliza o suporte ao desenho de telas com base em uma sintaxe XML.

A sintaxe XML para o desenho das telas utilizada em Android define um grande número de elementos personalizados, cada um representando uma subclasse específica da classe Android `View`. Nesse contexto pode-se projetar uma tela de forma similar a construção de arquivos HTML, especificando uma série de marcadores aninhados. Esses marcadores são guardados em um arquivo XML dentro do diretório `res/layout/` da aplicação.

Cada arquivo XML descreve um único elemento da classe Android `View`, todavia, este elemento pode ser tanto um simples elemento visual, como também um elemento de *layout* que contém uma coleção de objetos filhos representando a própria tela ou uma porção desta. Quando Android compila a aplicação, cada arquivo é compilado em uma única árvore, de uma única raiz. A compilação de cada arquivo dá origem a uma classe Android `View`. A classe então se torna um recurso que pode ser carregado e utilizado sempre que necessário dentro da aplicação. Ainda sobre os arquivos XML, Google (2008j) considera-o como um conjunto de marcadores que correspondem às classes Android de interface gráfica do usuário.

Marcadores, ou seja, elementos de interface possuem atributos que correspondem aproximadamente a métodos de uma determinada classe. Apesar da similaridade, não há uma correspondência exata entre a classe e seus métodos para com um elemento e seus atributos.

Android tende a chamar elementos na ordem em que eles aparecem no arquivo XML e dessa forma, a sobreposição de elementos é feita com o último elemento em um arquivo XML provavelmente desenhado sobre quaisquer elementos indicados anteriormente neste mesmo espaço.

Informações adicionais sobre valores (cores, cadeias de caracteres, estilos de texto e dimensões), elementos de desenho (imagens e cores), animações, *layouts* (elementos e atributos disponibilizados), estilos e temas, bem como conteúdo adicional referente ao formato do arquivo XML podem ser obtidos na referência Google (2008h). A hierarquia dos

elementos de tela está descrita na referencia Google (2008l) enquanto a descrição dos *layouts* de objetos existentes é completamente detalhada em Google (2008i).

Para atribuir um *layout* a uma atividade pode-se fazer uso do método `setContentView()`, que recebe como parâmetro o identificador único de um arquivo do *layout* que será atribuído à atividade, de modo a representar a interface com a qual a atividade se comunica com o usuário. O Quadro 14 ilustra um exemplo do uso do método `setContentView()` da classe de atividade.

```
@Override
public void onCreate(Bundle icicle) {
    super.onCreate(icicle);

    setContentView(R.layout.location_search);
    ...
}
```

Quadro 14 – Atribuição do arquivo XML `location_search` ao conteúdo da classe `MapBr_Location_Search_Activity`

Com base na sintaxe XML definida em Android, o Quadro 15 define o *layout* da atividade de pesquisa de endereços.

```
<?xml version="1.0" encoding="utf-8"?>

// a constante fill_parent define que o elemento deve preencher todo o espaço
disponível dentro do elemento que o abriga, respeitando o espaçamento definido
por este

// a constante wrap_content define que o elemento deve ser suficientemente
grande a ponto de comportar todo o seu conteúdo interno, respeitando o
espaçamento definido para o próprio elemento

// declaração do elemento RelativeLayout (raiz do layout da atividade)
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    // largura do elemento RelativeLayout
    android:layout_width="fill_parent"
    // altura do elemento RelativeLayout    android:layout_height="fill_parent"
    // plano de fundo do elemento RelativeLayout (preto)
    android:background="#ff000000"
    // espaçamento do elemento RelativeLayout (10px)
    android:padding="10px"
    >

    // declaração do elemento TextView
    <TextView
           // identificador único do elemento TextView
           android:id="@+id/location_edit_textView_address"
           // largura do elemento TextView
           android:layout_width="fill_parent"
           // altura do elemento TextView
           android:layout_height="wrap_content"
           // texto do elemento TextView
           android:text="Endereço/Local"
           />

    // declaração do elemento EditText
    <EditText
           // identificador único do elemento EditText
           android:id="@+id/location_edit_editText_address"
           // hint do elemento EditText
           android:hint="@string/input"
           // largura do elemento EditText
           android:layout_width="fill_parent"
           // altura do elemento EditText
           android:layout_height="wrap_content"
           // tamanho do texto do elemento EditText
           android:textSize="18sp"
           // âncora abaixo da qual o elemento EditText é posicionado
           android:layout_below="@id/location_edit_textView_address"
           />

    // declaração do elemento TextView
    <TextView
           // identificador único do elemento TextView
           android:id="@+id/location_edit_textView_maxResults"
           // largura do elemento TextView
           android:layout_width="fill_parent"
           // altura do elemento TextView
           android:layout_height="wrap_content"
           // texto do elemento TextView
           android:text="Número máximo de registros"
           // âncora abaixo da qual o elemento TextView é posicionado
           android:layout_below="@id/location_edit_editText_address"
           />

    // declaração do elemento Spinner
    <Spinner
           // identificador único do elemento Spinner
           android:id="@+id/location_edit_spinner_maxResults"
```

```
            // largura do elemento Spinner
            android:layout_width="wrap_content"
            // altura do elemento Spinner
            android:layout_height="wrap_content"
            // âncora abaixo da qual o elemento Spinner é posicionado
            android:layout_below="@id/location_edit_textView_maxResults"
            />

    // declaração do elemento Button
    <Button
            // identificador único do elemento Button
            android:id="@+id/location_edit_button_cancel"
            // texto do elemento Button (definido em forma de um recurso)
            android:text="@string/cancel"
            // largura do elemento Button
            android:layout_width="wrap_content"
            // altura do elemento Button
            android:layout_height="wrap_content"
            // âncora abaixo da qual o elemento Button é posicionado
            android:layout_below="@id/location_edit_spinner_maxResults"
            // alinhamento à direita
            android:layout_alignParentRight="true"
            />

    // declaração do elemento Button
    <Button
            // identificador único do elemento Button
            android:id="@+id/location_edit_button_search"
            // texto do elemento Button (definido em forma de um recurso)
            android:text="@string/search"
            // largura do elemento Button
            android:layout_width="wrap_content"
            // altura do elemento Button
            android:layout_height="wrap_content"
            // âncora abaixo da qual o elemento Button é posicionado
            android:layout_below="@id/location_edit_spinner_maxResults"
            // âncora a qual o elemento TextView é posicionado a esquerda
            android:layout_toLeftOf="@id/location_edit_button_cancel"
            // âncora a qual o elemento TextView é alinhado ao topo
            android:layout_alignTop="@id/location_edit_button_cancel"
            />

</RelativeLayout>
```

Quadro 15 – *Layout* da atividade de pesquisa de endereços e/ou locais

Observa-se que foram incluídos os códigos de comentários iniciados por barra dupla no Quadro 15 para descrever os elementos e atributos do *layout* da atividade de pesquisa. No entanto, esse tipo de comentário não é aceito pelo editor do *plugin* ADT.

3.4.4 A lista de marcadores

Para implementar a lista de marcadores foi necessário integrar as classes Android `SimpleCursorAdapter`, `ViewBinder` e a classe `ListView`. Além dessas também foi feito uso de outras classes Android, envolvidas para complementar a interface da atividade.

A classe `SimpleCursorAdapter` representa um adaptador usado para mapear colunas

de um cursor relacionando registros de tabelas para objetos das classes Android `TextView` ou `ImageView` definidos em um arquivo XML (GOOGLE, 2008m). No método construtor desta classe especifica-se o contexto de execução, o arquivo de *layout* XML para os itens da lista e um cursor. Também são descritas quais as colunas do cursor serão visualizadas, além de uma visão para cada coluna do cursor (Quadro 16). Nesse contexto, o arquivo XML define a interface do *layout* e das visões, ou seja, uma imagem e dois campos texto, além do layout onde esses estão inseridos (Quadro 17).

```
SimpleCursorAdapter simpleCursorAdapter =
    new SimpleCursorAdapter(this,
           R.layout.marker_list_item,
           cursor,
           new String[] {
                  MapBr_Marker_SQLiteOpenHelper.FIELD_TITLE,
                  MapBr_Marker_SQLiteOpenHelper.FIELD_DESCRIPTION,
                  MapBr_Marker_SQLiteOpenHelper.FIELD_VISIBLE
           },
           new int[] {
                  R.id.marker_list_item_textView_title,
                  R.id.marker_list_item_textView_description,
                  R.id.marker_list_item_relativeLayout
           }
    );
```

Quadro 16 – Criação de um objeto da classe Android `SimpleCursorAdapter`

```
<?xml version="1.0" encoding="utf-8"?>

<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:id="@+id/marker_list_item_relativeLayout"
    android:paddingTop="10dip"
    android:paddingBottom="10dip"
    >

    <ImageView android:id="@+id/marker_list_item_icon"
           android:src="@drawable/checkmark"
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:padding="5dip"
           />

    <TextView android:id="@+id/marker_list_item_textView_title"
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:layout_toRightOf="@id/marker_list_item_icon"
           android:textAppearance="?android:attr/textAppearanceLarge"
           />

    <TextView android:id="@+id/marker_list_item_textView_description"
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:layout_below="@id/marker_list_item_textView_title"
           android:layout_toRightOf="@id/marker_list_item_icon"
           android:textAppearance="?android:attr/textAppearanceSmall"
           />

</RelativeLayout>);
```

Quadro 17 – *Layout* XML dos itens da lista de marcadores

Em execução, a classe Android `SimpleCursorAdapter` realiza o mapeamento dos dados do cursor para elementos visuais de duas formas distintas e exclusivas, para cada elemento do arquivo de *layout*. Na primeira, caso um objeto da classe Android `ViewBinder` estiver atribuído ao adaptador, o método `setViewValue()` deste objeto será chamado. Uma vez que o retorno seja verdadeiro, o mapeamento será feito. Do contrário, ou seja, na segunda forma de mapeamento, se o resultado for falso e a visão a ser vinculada for uma instância da classe Android `TextView`, então o método `setViewText()` desta classe será chamado e o dado do cursor referente a esta visão será atribuído. No entanto, se o resultado for falso e a visão a ser vinculada for um a instância da classe Android `ImageView`, então o método `setViewImage()` desta classe será chamado atribuindo uma imagem ao elemento.

A classe `MapBr_Marker_List_ViewBinder` implementa a classe Android `ViewBinder`, que atua em conjunto com a classe Android `SimpleCursorAdapter`, cuja função é mapear dados de cursores em forma de visões, ou seja, em instâncias da classe Android `View`. A classe `ViewBinder` também pode ser usada para mapear valores que não são diretamente suportados pela classe `SimpleCursorAdapter`, ou para alterar o modo com que as visões suportadas são exibidas. Em outras palavras, ela define como os dados são dispostos e exibidos. A classe `MapBr_Marker_List_ViewBinder` é usada para que através do método `setViewValue()` seja possível definir o conteúdo das visões existentes para cada item da lista de marcadores.

Na aplicação Android MapBr o método `setViewValue()` da classe `MapBr_Marker_List_ViewBinder` verifica a partir do identificador único de cada visão que chama o método, qual elemento está requisitando o mapeamento de dados (Quadro 18). Caso o identificador do elemento for referente ao *layout* dos itens da lista de marcadores, carrega-se o elemento de imagem pertencente ao *layout* em uma variável local. Após isso, o estado de visibilidade do marcador representado pelo elemento em questão é verificado através do cursor, disponibilizado como parâmetro através no método `setViewValue()`. Dessa forma, pode-se atribuir a informação da visibilidade do marcador ao elemento de imagem do item da lista de marcadores presente no arquivo de *layout* a ser exibido ao usuário com o intuito de representar as informações do marcador. No entanto, caso o elemento passado ao método `setViewValue()` for responsável por representar o título ou a descrição do marcador, não se faz necessário tratamento algum, uma vez que a classe Android `SimpleCursorAdapter` encarrega-se de mapear os dados de forma automática para elementos da classe Android `TextView`.

```
public boolean setViewValue(View view, Cursor cursor, int columnIndex) {
    switch    (view.getId()) {
        case R.id.marker_list_item_relativeLayout :
            ImageView icon;
            Icon = (ImageView) view.findViewById(R.id.marker_list_item_icon);

            boolean visible = cursor.getString(columnIndex).equals("true");
            if    (visible) {
                icon.setImageState(new int[] { android.R.attr.state_checked },
true);
            } else {
                icon.setImageState(new int[] { }, true);
            }

            return true;
        }

    return false;
});
```

Quadro 18 – Método `setViewValue()` da classe `MabBr_Marker_List_ViewBinder`

3.4.5 O método `onCrossItemList`

Com o intuito de detectar o movimento horizontal em um item da lista de marcadores definiu-se a interface `MapBr_OnCrossItemListener`, onde se faz presente a declaração do método `onCrossItemList()` a ser codificado pelas classes que implementarem a interface. Para este método devem ser passados dois parâmetros. O primeiro representa a posição do marcador na lista, necessário para que seja possível atualizar o mesmo quando de sua alteração, enquanto o segundo parâmetro representa o indicativo de movimentação, entendido como verdadeiro quando a movimentação for realizada da esquerda para a direita ou falso quando esta ocorrer da direita para a esquerda.

A classe `MapBr_Marker_List_Activity` implementa a interface `MapBr_OnCrossItemListListener`, onde a partir do parâmetro de posição presente no método `onCrossItemList()` carrega-se o marcador sobre o qual foi feita a movimentação. A carga é realizada em função de um cursor, que executa uma consulta direta à tabela de marcadores mantida pela aplicação Android MapBr. O atributo relacionado à visibilidade do marcador é armazenado em uma variável a fim de ser comparado com o parâmetro do indicativo de movimentação do método `onCrossItemList()`. Caso não existir diferença de valores a função retorna, caso contrário atualiza-se o marcador em banco de dados, bem como as visões relacionadas à lista de marcadores, conforme ilustrado no Quadro 19.

```
public void onCrossItemList(int position, boolean crossed) {
    int id;
    boolean visible;

    synchronized (cursor) {
        cursor.moveToPosition((int)position);
        id = cursor.getInt(MapBr_Marker_SQLiteOpenHelper.FIELD_INDEX_ID);
        visible =
cursor.getString(MapBr_Marker_SQLiteOpenHelper.FIELD_INDEX_VISIBLE).equals("tr
ue");
    }

    if (visible == crossed) return;

    mapBr_Marker_SQLiteOpenHelper.updateMapBr_Marker(id, crossed);
    cursor.requery();

    int viewIndex = position - listView.getFirstVisiblePosition();
    View child = listView.getChildAt(viewIndex);
    If (child != null) child.invalidate();
}
```

Quadro 19 – Método `onCrossItemList()`

O movimento horizontal é detectado através da implementação realizada no método `onTouchEvento()`, referente ao evento de toque do *layout* que contempla a lista de marcadores (Quadro 20).

```
public boolean onTouchEvent(MotionEvent event) {
    float targetWidth = this.getWidth() / 4;
    float deltaX = event.getX() - downStart.getX(),
    deltaY = event.getY() - downStart.getY();

    boolean movedAcross = (Math.abs(deltaX) > targetWidth);
    boolean steadyHand = (Math.abs(deltaX / deltaY) > 2);

    if    (movedAcross && steadyHand) {
        boolean crossed = (deltaX > 0);

        ListView list;
        list = (ListView)this.findViewById(R.id.marker_list_listView);
        int position = list.pointToPosition((int)downStart.getX(),
(int)downStart.getY());

        if   (position >= 0) {
            for  (MapBr_OnCrossItemListListener listener :
onCrossItemListListeners) {
                listener.onCrossItemList(position, crossed);
            }

            return true;
        }
    }
    return false;
}
```

Quadro 20 – Método `onTouchEvent()` da classe `MapBr_Marker_List_FrameLayout`

3.4.6 Operacionalidade da aplicação

A operacionalidade da aplicação é apresentada em função dos casos de uso da

aplicação, fazendo o uso de imagens para facilitar o entendimento de cada uma das funcionalidades disponibilizadas.

De uma forma geral a aplicação Android MapBr tem por objetivo disponibilizar algumas das funcionalidades dispostas por ferramentas de visualização de mapas, tal como o Google Maps, no ambiente móvel da plataforma Android. Mais especificamente, de uma forma detalhada, o funcionamento da aplicação ocorre conforme descrito abaixo:

a) a aplicação exibe o mapa ao usuário (Figura 17);
b) o usuário faz uso das funcionalidades disponibilizadas pela aplicação por intermédio de eventos de toque no mapa ou em função dos itens de menu da aplicação (Figura 18);
c) determinadas ações executadas pelo usuário invocam intenções que associadas a atividades implementadas pela aplicação criam novas janelas exibidas em primeiro plano, interagindo diretamente com o usuário;
d) algumas atividades são apenas executadas com o intuito de exibir informações, enquanto outras retornam dados à atividade de origem ou atualizam informações no mapa.

Figura 17 – Android MapBr ao iniciar

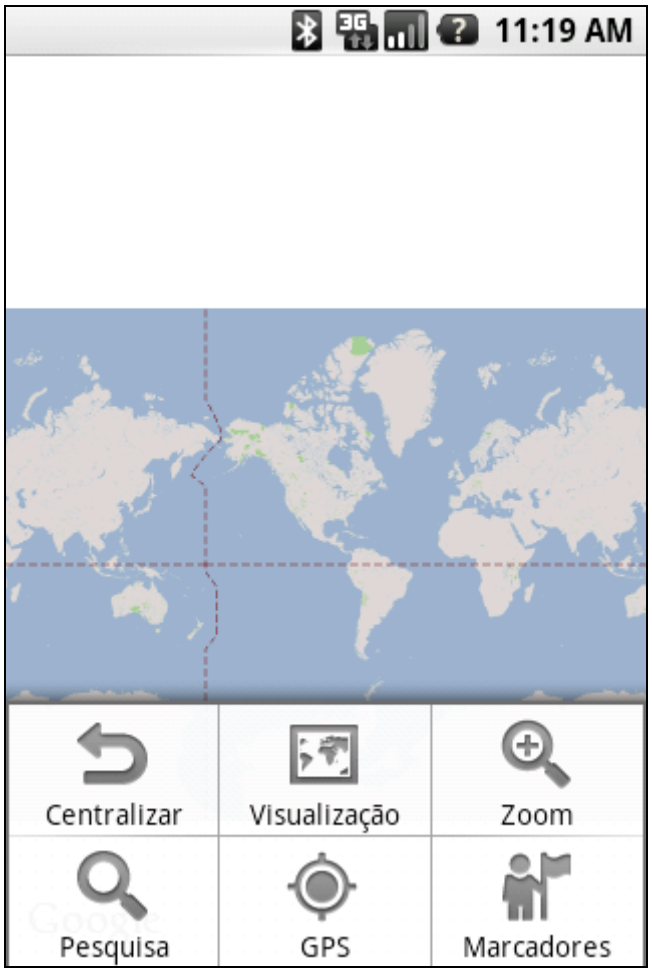


Figura 18 – Menu principal da aplicação Android MapBr

Com base no exposto, nas subseções seguintes são apresentadas as funcionalidades da aplicação Android MapBr.

3.4.6.1 Explorar mapa

Esta operacionalidade leva como base o caso de uso `Explorar mapa`, descrito na seção 3.3.1.1 que possibilita ao usuário uma forma de centralizar o mapa em posicionamento global, definir o modo de visualização ou o nível de zoom, bem como reposicionar o mapa na localização GPS do dispositivo móvel.

Ao iniciar a aplicação, o usuário visualiza o mapa em posicionamento global com zoom em nível mínimo, sob o ponto de vista do modo de desenho, padrão da aplicação. São exibidos os marcadores de acordo com seu estado de visualização individual e/ou o estado conjunto de visualização de marcadores da aplicação. A partir de então o usuário pode realizar a movimentação pelo mapa, bem como optar por executar as funcionalidades descritas abaixo:

a) alterar modo de visualização: optar entre os dois modos visualização existentes (Figura 19) para o item de menu `Visualização` do menu principal, ou seja, entre o modo `Desenho` e o modo `Satélite` (Figura 20);
b) alterar o nível de zoom: optar por realizar mais ou menos zoom, utilizando os controles na base do mapa (Figura 21);
c) posicionar mapa na localização GPS do dispositivo móvel: deve optar pelo item de menu `GPS` do menu principal;
d) centralizar mapa: optar pelo item de menu `Centralizar` do menu principal.

Figura 19 – Modos de visualização do mapa

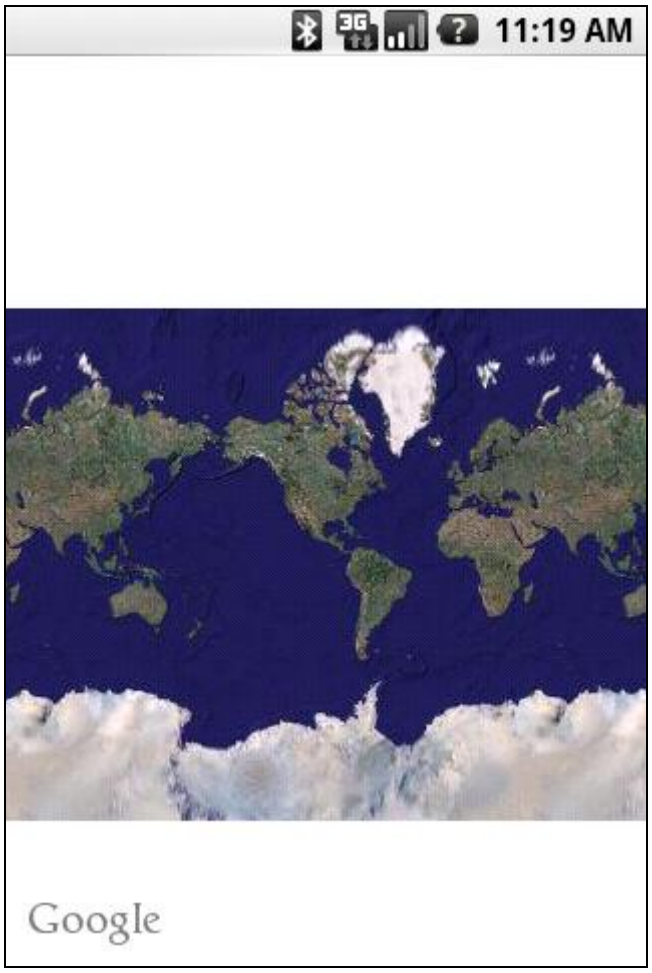


Figura 20 – Modo de visualização `Satélite`

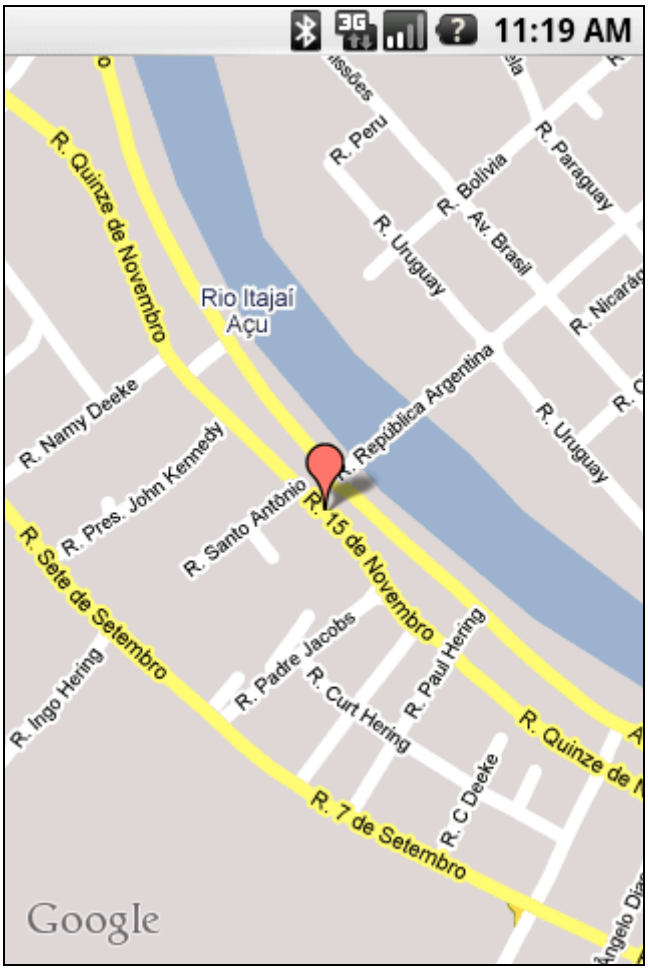


Figura 21 – Zoom máximo no centro da cidade de Blumenau, Santa Catarina

3.4.6.2 Manipular marcadores

Esta operacionalidade leva como base o caso de uso `Manipular marcadores`, descrito na seção 3.3.1.2 que possibilita ao usuário uma forma de adicionar, selecionar, editar e remover marcadores no mapa, bem como definir a visualização dos marcadores de forma individual ou conjunta.

De uma forma geral o usuário visualiza o mapa onde são exibidos os marcadores de acordo com seu estado de visualização individual e/ou o estado conjunto de visualização de marcadores da aplicação (Figura 22), de onde pode optar por uma das funcionalidades descritas abaixo:

a) adicionar marcador: pressionar no mapa durante quatro segundos o ponto onde deseja adicionar o marcador. Caso existir um resultado da pesquisa posicionado pode-se optar pelo sub-item `Adicionar` do item de menu `Marcadores` (Figura 23)

do menu principal para adicionar um marcador ao resultado;

b) selecionar marcador: pressionar o marcador desejado no mapa;

c) editar marcador: pressionar durante quatro segundos o marcador desejado e optar pela opção `editar` do menu *popup*. Caso existir um marcador selecionado pode-se optar pelo sub-item `Editar` do item de menu `Marcadores` do menu principal para editar o marcador selecionado;

d) remover marcador: pressionar durante quatro segundos o marcador desejado e optar pela opção `remover` do menu *popup*. Caso existir um marcador selecionado pode-se optar pelo sub-item `Remover` do item de menu `Marcadores` do menu principal para remover o marcador selecionado;

e) remover todos os marcadores: optar pelo sub-item `Remover Todos` do item de menu `Marcadores` do menu principal;

f) alterar estado de visualização conjunta: optar por alterar o estado do sub-item `Exibir` do item de menu `Marcadores` do menu principal.

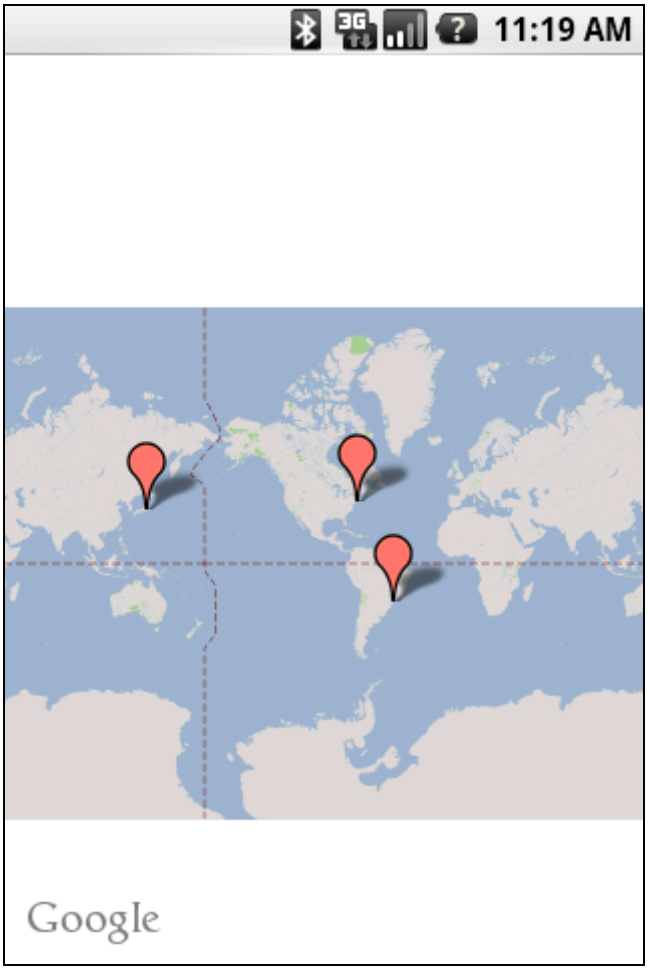


Figura 22 – Mapa com marcadores

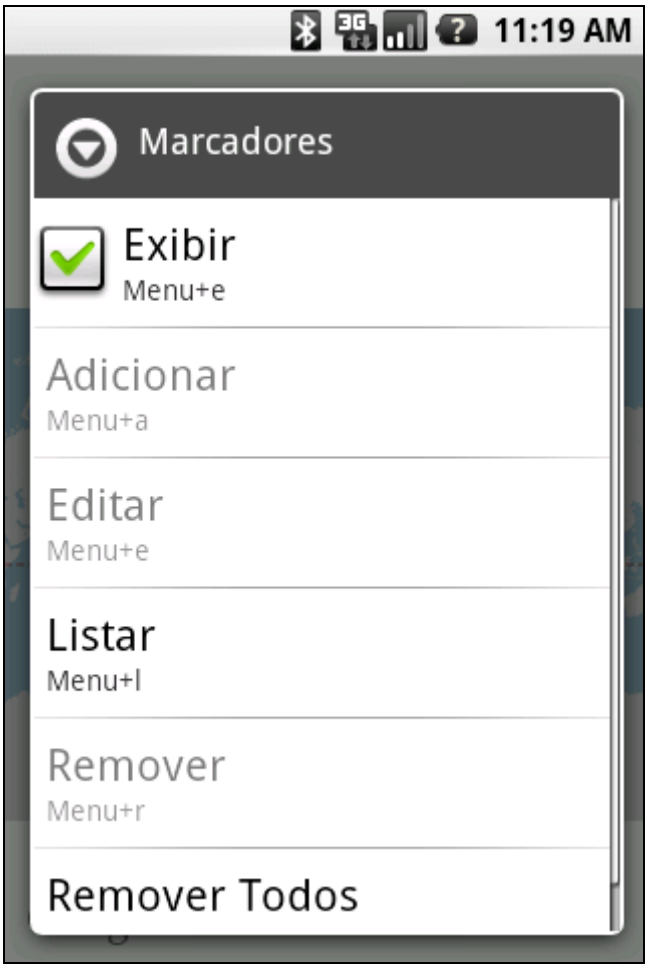


Figura 23 – Menu Marcadores

3.4.6.3 Editar marcador

Esta operacionalidade leva como base o caso de uso `Editar marcador`, descrito na seção 3.3.1.3 que possibilita ao usuário uma forma de inserir ou alterar um marcador no mapa.

Para editar um marcador é necessário que o usuário opte por adicionar ou editar um marcador durante a execução da operacionalidade que possibilita manipular marcadores. Também é possível editar um marcador a partir da execução da operacionalidade responsável por listar marcadores.

Respeitando as precondições citadas acima a aplicação exibe a tela de edição de marcador (Figura 24). Os dados do marcador estarão preenchidos caso o usuário estiver alterando um marcador já existente. Sempre serão exibidos valores para a latitude e a longitude, seja em uma inclusão ou em uma alteração. A partir desse momento o usuário pode optar por uma das funcionalidades descritas abaixo:

a) editar marcador: informar os dados de marcador e optar por salvar o marcador fazendo uso do botão `Salvar` da tela de edição de marcador;
b) cancelar a adição ou a alteração do marcador: optar pela opção `Cancelar` da tela de edição de marcador.

Figura 24 – Tela de edição de marcadores

3.4.6.4 Listar marcadores

Esta operacionalidade leva como base o caso de uso `Listar marcadores`, descrito na seção 3.3.1.4 que possibilita ao usuário uma forma de listar os marcadores do mapa, bem como editar, remover ou definir o estado de visualização individual de cada marcador manipulado pela aplicação.

De uma forma geral o usuário visualiza o mapa onde são exibidos os marcadores de acordo com seu estado de visualização individual e/ou o estado conjunto de visualização de marcadores da aplicação, de onde opta por listar os marcadores fazendo uso do sub-menu

`Listar` do item de menu `Marcadores` do menu principal. A aplicação exibe a tela listando todos os marcadores manipulados pela aplicação (Figura 25). Os marcadores visíveis são exibidos com um indicativo de seleção checado, enquanto os marcadores ocultos são exibidos com o mesmo indicativo não checado. A partir desse momento o usuário pode optar por uma das funcionalidades descritas abaixo:

a) visualizar marcador: pressionar o marcador desejado;
b) editar marcador: pressionar durante quatro segundos o marcador desejado e optar pela opção `Editar` do menu que é exibido pela aplicação;
c) remover marcador: optar pela opção `Remover` do menu que é exibido pela aplicação (Figura 26);
d) voltar ao mapa: optar pelo botão `Voltar`.

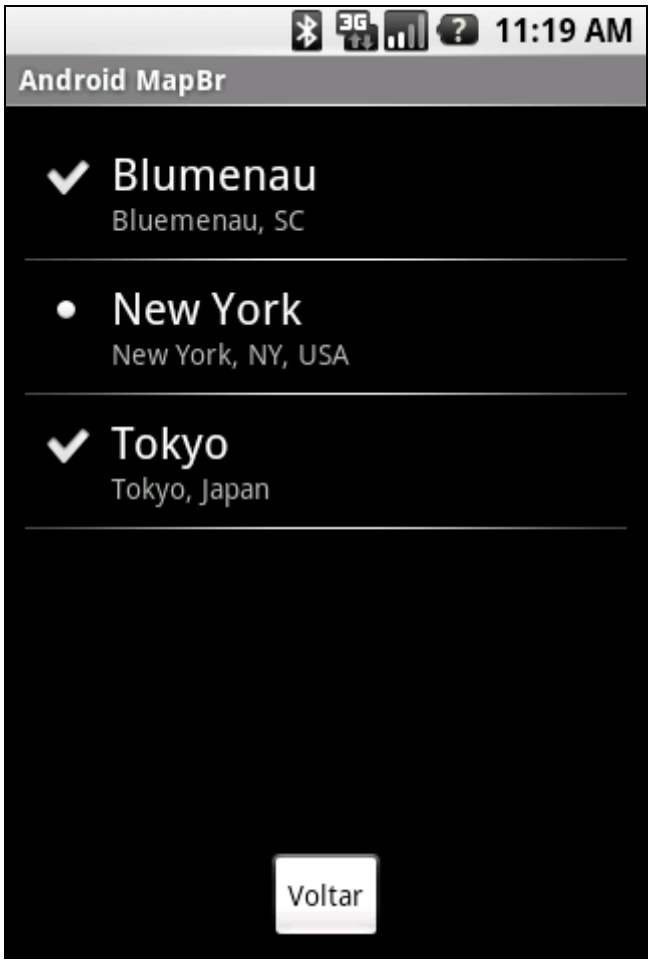


Figura 25 – Lista de marcadores

Figura 26 – Menu de um item da lista de marcadores

3.4.6.5 Pesquisar

Esta operacionalidade leva como base o caso de uso `Pesquisar`, descrito na seção 3.3.1.5 que possibilita ao usuário uma forma de pesquisar endereços e/ou locais.

De uma forma geral o usuário visualiza o mapa onde são exibidos os marcadores de acordo com seu estado de visualização individual e/ou o estado conjunto de visualização de marcadores da aplicação, de onde pode optar por pesquisar um endereço ou local fazendo uso do sub-item `Pesquisar` do item de menu `Pesquisa` do menu principal. A aplicação exibe a tela pesquisa (Figura 27) com o campo `Endereço/Local` em branco e com a quantidade máxima de registros posicionada no valor padrão da aplicação, cinco primeiros registros. A partir desse momento o usuário pode optar por uma das funcionalidades descritas abaixo:

a) pesquisar: informar os dados da pesquisa e optar por pesquisar pressionando o botão `Pesquisar` da tela de pesquisa de endereços ou locais;

b) voltar ao mapa: optar pelo botão `Cancelar`.

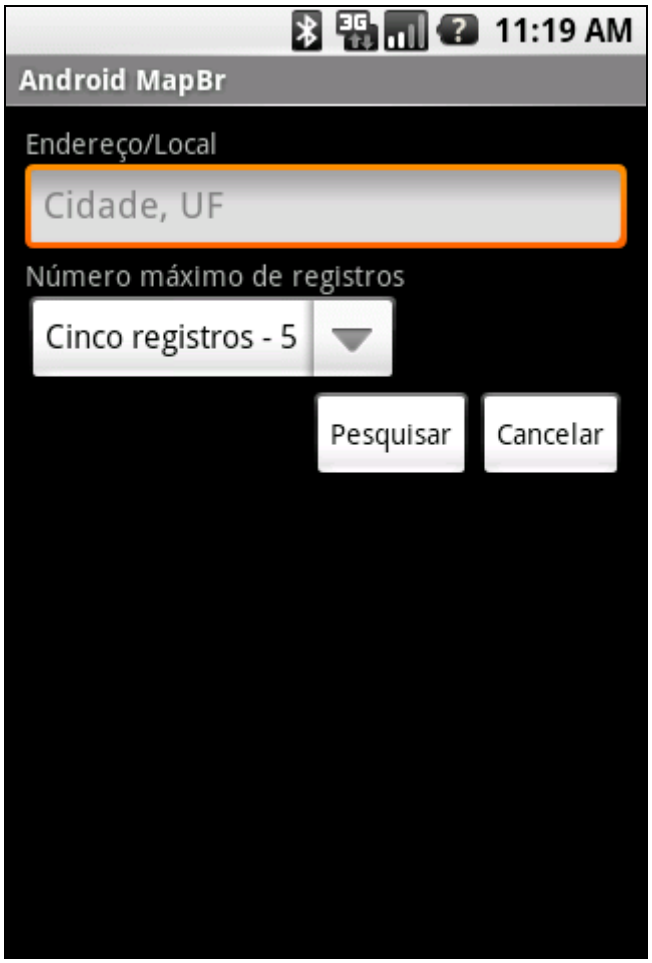


Figura 27 – Tela de pesquisa de endereços e/ou locais

3.4.6.6 Listar resultados da pesquisa

Esta operacionalidade leva como base o caso de uso `Listar resultados da pesquisa`, descrito na seção 3.3.1.6 que possibilita ao usuário uma forma de listar os endereços e/ou locais da última pesquisa efetuada pelo usuário.

Para listar os resultados da pesquisa é necessário que o usuário opte por listar os endereços e/ou locais retornados pela última pesquisa, optando pelo sub-item `Últimos Resultados` do item de menu `Pesquisa` do menu principal. A exibição da lista dos resultados da pesquisa também é feita automaticamente pela aplicação ao concluir a execução desta operacionalidade.

Respeitando as precondições citadas acima a aplicação exibe a tela com todos os resultados da última pesquisa (Figura 28). O usuário visualiza a lista de resultados da última

pesquisa. A partir desse momento o usuário pode optar por uma das funcionalidades descritas abaixo:

a) visualizar resultado: pressionar o resultado desejado;

b) voltar ao mapa: optar pelo botão `Voltar`.

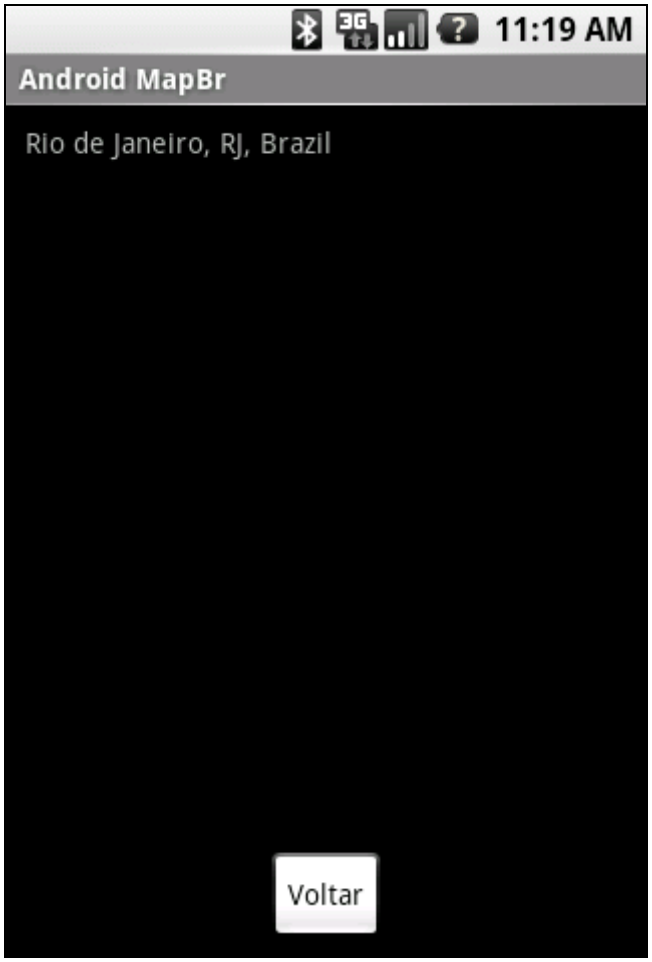


Figura 28 – Lista de resultados da pesquisa

3.5 RESULTADOS E DISCUSSÃO

A realização dos testes relacionados à aplicação Android MapBr deu-se em paralelo ao desenvolvimento da aplicação. Assim, a cada nova operacionalidade adicionada ao projeto procurou-se testar todas as demais funcionalidades já existentes, buscando garantir a correta execução das rotinas de visualização do mapa, de manipulação de marcadores e de pesquisa de endereços, seja em função de novas implementações no SDK, como também devido a correções ou melhorias nos códigos da aplicação desenvolvida.

Todos os testes foram efetuados usando o emulador disponibilizado pela ambiente de desenvolvimento Android. Optou-se por utilizar essa metodologia de testes, uma vez que durante todo o desenvolvimento da aplicação não se encontrava disponível no mercado nenhum equipamento móvel que utilizasse o sistema operacional da Google, necessário para rodar a aplicação. Durante o decorrer das validações percebeu-se a necessidade de testes unitários, a fim de validar não apenas visualmente a correta execução de cada unidade da aplicação, que muitas vezes passam despercebidos. Dessa forma, no âmbito das extensões foi adicionada como sugestão a implementação de testes unitários para a aplicação.

Ainda sobre o desenvolvimento, durante todas as etapas da elaboração da aplicação algumas dificuldades foram encontradas, dentre as quais se cita como causa o conhecimento regular da linguagem de programação Java, a imaturidade e a instabilidade do ambiente de desenvolvimento de sistemas Android bem como a existência de fontes de pesquisa disponibilizadas somente na língua inglesa.

A principal dificuldade encontrada no desenvolvimento utilizando a linguagem de programação Java residiu-se na necessidade de conhecer previamente tópicos fundamentais de qualquer linguagem, como as classes, conceitos e métodos, bem como padrões de projeto e os paradigmas de programação existentes. O domínio do conteúdo relacionado aos tópicos mencionados é de grande importância, uma vez que facilita o desenvolvimento, tornando o código mais robusto e organizado, ao passo que em muitos casos automatiza e poupa esforços desnecessários. No entanto, em inúmeras outras situações durante o desenvolvimento o entendimento e a utilização de alguns conceitos vieram a acrescentar um grau de complexidade maior à aplicação, o que exigia uma constante busca por documentações relacionadas ao tema pesquisado.

No mesmo contexto das dificuldades decorrentes do uso de uma nova linguagem de programação pode-se incluir toda a biblioteca Android. Apenas através da pesquisa e da codificação de rotinas não inerentes ao contexto da aplicação é que foi possível encontrar as classes necessárias para a implementação das funcionalidades propostas. Por diversas vezes encontrou-se formas distintas, com resultados próximos para a resolução de um único problema. Situações desta natureza dificultavam a decisão de qual seria o melhor caminho a ser seguido. Procurou-se sempre optar por abordagens distintas, alternando entre as resoluções possíveis para um problema previamente conhecido, o que possibilitou explorar um número maior de recursos dentro da linguagem de programação Java e principalmente do ambiente de desenvolvimento Android.

A imaturidade e a instabilidade do ambiente Android também foram empecilhos ao

desenvolvimento de mais funcionalidade ao projeto, uma vez que cada nova versão liberada da interface de programação aplicativos do ambiente de desenvolvimento Android era acompanhada de inúmeras mudanças relacionadas à adição, modificação e remoção de pacotes, classes, métodos e/ou atributos. As modificações no SKD por muitas vezes geraram a necessidade de adaptar ou reescrever rotinas da aplicação. Pode-se destacar, no entanto que as alterações existentes são documentadas e disponibilizadas em relatórios de diferenças, onde são abordados os novos recursos disponíveis, bem como alterações que podem ou devem ser integradas ao código das aplicações. De forma geral, os relatórios ilustram todas as diferenças existentes entre a última versão da interface de programação em relação à nova versão disponibilizada, onde cada referência a uma mudança inclui uma breve descrição e uma explicação. Em alguns casos também é possível encontrar uma sugestão de contorno.

Da mesma forma como os relatórios de diferenças, toda a documentação relacionada ao SDK é bastante completa, rica em detalhes visuais explicativos, englobando desde definições até exemplos de uso e melhores práticas de programação, seja ela voltada a dispositivos móveis ou a plataforma Android. Todas as classes, bem como cada um de seus métodos e atributos possuem documentação, que muitas vezes aborta um nível de detalhamento suficiente para que se possa ter ciência sobre o funcionamento interno da implementação, o que acaba dando base para um melhor entendimento sobre outras classes e recursos disponíveis ao programador.

De forma geral, conforme descrito no Quadro 21, a aplicação atende aos requisitos propostos, no entanto, devido a alterações significativas na plataforma Android, a implementação da visualização de rotas não foi incorporada a aplicação, uma vez que a classe Android `com.google.googlenav.DrivingDirection` não é mais disponibilizada publicamente a partir da versão 0.9 do ambiente de desenvolvimento, conforme descrito na referência SHARKEY (2008). Esta classe fornecia todo o suporte necessário para encontrar uma rota entre duas coordenadas no mapa, gerando a lista de pontos da coordenada de origem até a coordenada de destino. A classe `com.google.googlenav.DrivingDirection` ainda pode ser encontrada na versão onde se iniciou o desenvolvimento da aplicação Android MabBr, ou seja, na versão `m5-rc15` do SDK Android. A partir desta versão, outra versão *beta* surgiu, até o lançamento do SDK 1.0 que contempla o HTC G1[19]. Contudo, em função de alterações que exigiram mudanças significativas nos códigos e *layouts* optou-se por compatibilizar toda a aplicação em detrimento da visualização de rotas. Acredita-se que

modelos de programação e recursos de um SDK acrescentam uma relevância maior à aplicação, uma vez que desde a proposta inicial, outros requisitos foram adicionados ao trabalho.

REQUISITOS FUNCIONAIS	CONCLUÍDO
RF01: Ser desenvolvido usando a plataforma Android.	Sim
RF02: Permitir ao usuário visualizar o mapa global disponibilizado pelo serviço Google *Maps* sob o ponto de vista do modo desenho e do modo satélite.	Sim
RF03: Permitir ao usuário realizar movimento por todo o mapa.	Sim
RF04: Permitir ao usuário aplicar níveis de zoom ao mapa.	Sim
RF05: Permitir ao usuário posicionar o mapa na localização GPS do dispositivo móvel.	Sim
RF06: Permitir ao usuário visualizar, adicionar, selecionar, editar, remover, ocultar, listar e salvar marcadores na aplicação.	Sim
RF07: Permitir ao usuário pesquisar endereços ou locais, bem como listar os resultados da pesquisa.	Sim

Quadro 21 – Requisitos funcionais concluídos

Por fim, dentre as limitações existentes no cenário dos dispositivos móveis, cita-se o baixo poder de processamento, a reduzida quantidade de memória disponível, a visualização comprometida por uma pequena resolução, bem como restrições na interatividade devido a um teclado limitado. Como não foi realizado um enfoque nesse campo, torna-se possível adicionar como extensões a análise e compatibilização do código às técnicas de desenvolvimento de aplicações móveis visando o melhor desempenho e o mínimo consumo de memória.

[19] HTC G1 é o primeiro *smartphone* do mundo a ser vendido com o sistema operacional Android desenvolvido pela Google, o Android.

4 CONCLUSÕES

A concepção de uma aplicação para a visualização de mapas em dispositivos móveis consegue consolidar a união de duas necessidades do mundo real, vinculando informação de grande relevância, como localização geográfica, rotas, entre outros, ao âmbito da mobilidade. Nesse contexto, a relevância computacional do projeto se traduz pela complexidade e limitações inerentes ao desenvolvimento voltado a dispositivos móveis. É importante salientar também a convergência do mercado em relação às plataformas abertas, o que faz do Android uma escolha de peso, pois além de inúmeras inovações, contempla um sistema baseado em Linux.

Após a pesquisa, especificação, modelagem e desenvolvimento de uma aplicação em Android, concluiu-se que esta plataforma pode ser caracterizada por um pacote de software para dispositivos móveis que inclui um sistema operacional, um *middleware* e aplicativos de características fundamentais. Dentre esses aplicativos encontram-se o emulador e um *debbuger*, juntamente com toda a biblioteca de desenvolvimento. O emulador é necessário para simular o sistema, inexistente em qualquer dispositivo móvel durante toda a implementação da aplicação. A plataforma contempla também, incorporada as suas bibliotecas, interfaces de programação para que o desenvolvedor faça uso dos mesmos recursos que a Google disponibiliza na Web, tais como calendário, mapas, mensagens e previsão do tempo. Além disso, Android traz inovações quando demonstra novos paradigmas de programação sob o aspecto de intenções.

Em relação aos trabalhos correlatos apresentados, verifica-se que o trabalho proposto atende grande parte das funcionalidades existentes nas ferramentas disponíveis no mercado, uma vez que leva como base uma das interfaces de programação disponibilizadas pela Google, o que torna o comportamento da ferramenta semelhante ao existente no Google Maps. A principal diferença em relação aos trabalhos correlatos apresentados reside no ambiente para o qual a aplicação está disponível, no caso, dispositivos móveis, uma tendência para os dias atuais. É fundamental mencionar também a característica única do uso da plataforma Android desde o desenvolvimento, testes e execução da aplicação.

Por fim, obteve-se uma aplicação funcional utilizando uma grande variedade de conceitos, paradigmas e recursos do ambiente de desenvolvimento da plataforma Android. Ainda nesse contexto, a documentação gerada durante a implementação contempla referências e explicações que até então eram encontradas quase que exclusivamente na língua inglesa,

facilitando a iniciação ao desenvolvimento em Android.

4.1 EXTENSÕES

Durante o desenvolvimento do trabalho foram verificadas diversas possíveis melhorias na aplicação Android MapBr, muitas das quais não puderam ser contempladas neste trabalho. Portanto, as sugestões para extensão do trabalho são mostradas através de uma lista de tarefas. A lista de tarefas é apresentada no Quadro 22, onde há a descrição de cada tarefa e sua complexidade – apresentada em escala crescente de complexidade, de um a dez.

TAREFA	COMPLEXIDADE
Desenvolver recurso para a criação e a visualização de rotas entre endereços e/ou marcadores.	6
Desenvolver recurso para a visualização e monitoração dos contatos no mapa por intermédio de posicionamento GPS.	6
Desenvolver testes unitários para a aplicação Android MapBr.	5
Desenvolver recurso para a troca de mensagens entre os contatos.	5
Analisar e compatibilizar as classes para fazer uso das técnicas de desenvolvimento de aplicações móveis visando o melhor desempenho e o mínimo consumo de memória conforme sugerido pela Google na referência Google (2008q).	5
Desenvolver rotinas que façam uso de serviços (`services`).	4

Quadro 22 – Lista de tarefas da aplicação Android MapBr

REFERÊNCIAS BIBLIOGRÁFICAS

APONTADOR. **Apontador mobile**. [S.l.], [2008]. Disponível em: <http://www.apontador.com.br/mobile/index.html>. Acesso em: 22 mar. 2008.

AZAMBUJA, Wagner. **Symbian**. [S.l.], 2007. Disponível em: <http://www.vivasemfio.com/blog/symbian/>. Acesso em: 16 mar. 2008.

GOOGLE. **Anatomy of an Android application**. [S.l.], 2008a. Disponível em: <http://code.google.com/android/intro/anatomy.html>. Acesso em: 21 mar. 2008.

_____. **Android glossary**. [S.l.], 2008b. Disponível em: <http://code.google.com/android/reference/glossary.html>. Acesso em: 17 mar. 2008.

_____. **android.app.Activity**. [S.l.], 2008c. Disponível em: <http://code.google.com/android/reference/android/app/Activity.html>. Acesso em: 21 ago. 2008.

_____. **android.content.Intent**. [S.l.], 2008d. Disponível em: <http://code.google.com/android/reference/android/content/Intent.html>. Acesso em: 5 nov. 2008.

_____. **android.widget**. [S.l.], 2008e. Disponível em: <http://code.google.com/android/reference/android/widget/package-descr.html>. Acesso em: 24 ago. 2008.

_____. **android.widget.SimpleCursosAdapter**. [S.l.], 2008f. Disponível em: <http://code.google.com/android/reference/android/widget/SimpleCursorAdapter.html>. Acesso em: 6 nov. 2008.

_____. **Android:** an Open Handset Alliance Project. [S.l.], 2008g. Disponível em: <http://code.google.com/android/>. Acesso em: 17 mar. 2008.

_____. **Available resource types**. [S.l.], 2008h. Disponível em: <http://code.google.com/android/reference/available-resources.html>. Acesso em: 6 nov. 2008.

_____. **Common layout objects**. [S.l.], 2008i. Disponível em: <http://code.google.com/android/devel/ui/layout.html>. Acesso em: 6 nov. 2008.

_____. **Conheça o Google Maps**. [S.l.], 2007. Disponível em: <http://www.google.com.br/help/maps/tour/#driving_directions>. Acesso em: 22 mar. 2008.

_____. **Designing your Screen in XML**. [S.l.], 2008j. Disponível em: <http://code.google.com/android/devel/ui/xml.html>. Acesso em: 6 nov. 2008.

_____. **Develop and debug**. [S.l.], 2008k. Disponível em: <http://code.google.com/android/intro/develop-and-debug.html>. Acesso em: 9 nov. 2008.

_____. **Hierarchy of screen elements**. [S.l.], 2008l. Disponível em: <http://code.google.com/android/devel/ui/hierarchy.html>. Acesso em: 6 nov. 2008.

_____. **Installing the SDK**. [S.l.], 2008m. Disponível em: <http://code.google.com/android/intro/installing.html>. Acesso em: 9 nov. 2008.

_____. **Life cycle of an Android application**. [S.l.], 2008n. Disponível em: <http://code.google.com/android/intro/lifecycle.html>. Acesso em: 21 mar. 2008.

_____. **The AndroidManifest.xml File**. [S.l.], 2008o. Disponível em: <http://code.google.com/android/devel/bblocks-manifest.html>. Acesso em: 24 ago. 2008.

_____. **What is Android?** [S.l.], 2008p. Disponível em: <http://code.google.com/android/what-is-android.html>. Acesso em: 17 mar. 2008.

_____. **Writing efficient android code**. [S.l.], 2008q. Disponível em: <http://code.google.com/android/toolbox/performance.html>. Acesso em: 6 nov. 2008.

MENDONÇA, Aderval. **Para onde vão os sistemas operacionais móveis?** [S.l.], 2007. Disponível em: <http://www.devmedia.com.br/articles/viewcomp.asp?comp=4568>. Acesso em: 16 mar. 2008.

NEIVA, Daniele. **Que tendência tecnológica vai se consolidar em 2008?** [S.l.], 2007. Disponível em: <http://oglobo.globo.com/tecnologia/mat/2007/12/23/327741180.asp>. Acesso em: 21 mar. 2008.

NOKIABR. **Você sabe o que é Maemo?** [S.l.], 2007. Disponível em: <http://nokiabr.blogspot.com/2007/10/voc-sabe-o-que-maemo.html>. Acesso em: 23 fev. 2009.

OPEN HANDSET ALLIANCE. **FAQ**. [S.l.], 2007. Disponível em: <http://www.openhandsetalliance.com/oha_faq.html>. Acesso em: 17 mar. 2008.

OPEN MOBILE ALLIANCE. **Frequently asked questions**. [S.l.], [2008?]. Disponível em: <http://www.openmobilealliance.org/AboutOMA/FAQ.aspx>. Acesso em: 16 mar. 2008.

PALMBRASIL. **Notícias Palm OS PDAs smartphones Treo LifeDrive TX Tungsten Zire**. [S.l.], 2008. Disponível em: <http://www.palmbrasil.com.br/noticias/noticias-fev-08.html>. Acesso em: 30 mar. 2008.

_____. **Palm OS Cobalt**. [S.l.], [2004?]. Disponível em: <http://www.palmbrasil.com.br/vocab/palmos-cobalt.html>. Acesso em: 16 mar. 2008.

_____. **Visão geral do Windows Mobile #2**. [S.l.], [2007?]. Disponível em: <http://www.palmbrasil.com.br/windows-mobile/visao-geral-2.html>. Acesso em: 16 mar. 2008.

SHARKEY, Jeffrey. **Driving directions in Android 0.9 SKD**. [S.l.], [2008]. Disponível em: < http://www.jsharkey.org/blog/2008/08/22/driving-directions-in-android-09-sdk/>. Acesso em: 11 nov. 2008.

SPARX SYSTEMS. **Enterprise Architect**. [S.l.], [2007?]. Disponível em: <http://www.sparxsystems.com.au/products/ea.html>. Acesso em: 24 out. 2007.

TECSINAPSE. **Smartphones levam produtividade e convergência ao mundo corporativo**. [S.l.], 2006. Disponível em < http://www.tecsinapse.com.br/?index=viewnews&view=53>. Acesso em: 21 mar. 2008.