**UNIVERSIDADE REGIONAL DE BLUMENAU**

**CENTRO DE CIÊNCIAS EXATAS E NATURAIS**

**CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO**

# FRAMEWORK PARA GERENCIAMENTO E DISPONIBILIZAÇÃO DE INFORMAÇÕES MULTIMÍDIA GEOLOCALIZADAS NA PLATAFORMA ANDROID

**DAVID TIAGO CONCEIÇÃO**

**BLUMENAU**
**2010**

**2010/2-10**

**DAVID TIAGO CONCEIÇÃO**

# FRAMEWORK PARA GERENCIAMENTO E DISPONIBILIZAÇÃO DE INFORMAÇÕES MULTIMÍDIA GEOLOCALIZADAS NA PLATAFORMA ANDROID

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

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

**BLUMENAU**
**2010**

**2010/2-10**

# FRAMEWORK PARA GERENCIAMENTO E DISPONIBILIZAÇÃO DE INFORMAÇÕES MULTIMÍDIA GEOLOCALIZADAS NA PLATAFORMA ANDROID

Por

**DAVID TIAGO CONCEIÇÃO**

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, Dr. – FURB |
| Membro: | Prof. Paulo Fernando da Silva, M.Sc. – FURB |

Blumenau, 13 de dezembro de 2010

Dedico este trabalho a todos os que me apoiaram durante todo o curso, especialmente meus pais, irmãos e amigos.

## AGRADECIMENTOS

A todos aqueles que contribuiram para a realização do presente trabalho e a minha evolução durante o curso.

Aos meus pais e irmãos, que sempre estiveram do meu lado em momentos de dificuldade e sacrifícios.

Aos meus amigos, que sempre auxiliaram meu desenvolvimento e foram amigos de verdade.

Àqueles que foram meus amigos e colegas no desenrolar do curso que, sem dúvida, foram fundamentais para chegar até aqui.

Ao meu orientador, pela confiança e apoio mesmo nos momentos mais delicados.

Até que o sol não brilhe, acendamos uma vela na escuridão.

Kung-Fu-Tse (Confúcio)

# RESUMO

Este trabalho apresenta o desenvolvimento de uma ferramenta para a definição de informações de georreferenciamento vinculadas às mídias de áudio, vídeo, imagens, textos e pontos de interesse em um aplicativo na plataforma Android. A ferramenta permite também o compartilhamento das mídias georreferenciadas entre usuários conectados, além da consulta a localização desses usuários. As informações de georreferenciamento são armazenadas em um banco de dados local e a comunicação com o servidor efetuada através de requisições do *HyperText Transfer Protocol* (HTTP). A visualização e reprodução das mídias é efetuada através de bibliotecas da própria plataforma Android. Já a exibição dos mapas e obtenção das coordenadas selecionadas pelo usuário são efetuadas através de bibliotecas externas disponibilizadas para a plataforma. Durante a evolução desse trabalho, a plataforma Android é apresentada, através de conceitos e utilização de funcionalidades da mesma.

Palavras-chave: Android. Multimídia móvel. *Framework* multimídia. Logística. Mobilidade.

# ABSTRACT

This work presents the development of a tool for the definition of geolocate information connected to audio, video, images, texts and points of interest media in an Android application. The tool also allows the sharing of geolocate media between connected users, beyond the query to the geolocalization of these users. The informations about geolocation are persisted into a local database and the communication with the server is made up over the HyperText Transfer Protocol (HTTP). The visualization and reproduction of media occurs through Android native libraries. The maps exibition and the obtaining of the selected coordinates occur through external libraries available to the platform. During the evolution of this work, the Android platform is presented by concepts and use of funcionalities.

Key-words: Android. Mobile multimedia. Multimedia framework. Logistic. Mobile.

# LISTA DE ILUSTRAÇÕES

# LISTA DE TABELAS

# LISTA DE SIGLAS

AAC - *Advanced Audio Coding*

ADT - *Android Development Tools*

AMR - *Adaptive Multi-Rate*

API - *Application Programmer Interface*

EDGE - *Enhanced Data rates for GSM Evolution*

GPS - *Global Position System*

GSM - *Global System for Mobile Communications*

HTTP - *HyperText Transfer Protocol*

IDE - *Integrated Development Environment*

JDBC - *Java Database Connectivity*

JDK - *Java Development Kit*

JEE - *Java Enterprise Edition*

JSE - *Java Standard Edition*

MER - Modelos de Entidades e Relacionamentos

MIDI - *Musical Instrument Digital Interface*

MP3 - *Moving Picture Experts Group Audio Layer 3*

MPEG-4 - *Moving Picture Experts Group Layer 4*

PDA - *Personal Digital Assistant*

RF - Requisito Funcional

RNF - Requisito Não Funcional

SBL - Serviço Baseado em Localização

SDK - *Software Developer Kit*

SMS - *Short Message System*

SQL - *Structured Query Language*

UML - *Unified Modeling Language*

WAVE - *WAVEform audio format*

Wi-Fi - *Wireless Fidelity*

XML - *eXtensible Markup Language*

# SUMÁRIO

# 1 INTRODUÇÃO

A constante evolução das plataformas móveis tem adicionado às mesmas funcionalidades antes encontradas apenas em computadores pessoais. Entre estas funcionalidades estão a capacidade de gerar e reproduzir as mais variadas mídias. Os celulares, em especial, avançaram significativamente desde a sua popularização incorporando inovações em hardware e software que permitem que os aparelhos mais recentes capturem e reproduzam textos, sons, imagens, vídeos, coordenadas geográficas, entre outras possibilidades. Além disso, as tecnologias de conexões sem fio avançam consideravelmente em velocidade e popularidade, fazendo com que cada vez mais os usuários de celulares busquem meios para obter e distribuir mídias.

Assim, o desenvolvimento de aplicações modernas para celulares não pode mais desconsiderar a capacidade multimídia dos aparelhos. A incorporação de mídias é cada vez mais presente nas aplicações e certamente causa uma experiência mais interativa e motivante aos usuários. É o caso da exploração das coordenadas geográficas, geradas por *Global Position System* (GPS) ou outros meios. Essa tecnologia permite o desenvolvimento de aplicações sensíveis ao contexto do usuário, como a notificação de que o usuário está próximo de um ponto de interesse, por exemplo. Segundo Gonzales (2009), "com a maturidade dos sistemas de localização móvel e a disseminação das redes Wi-Fi, as empresas podem desenvolver mais aplicações contextuais, de detecção de presença e de redes sociais móveis". Assim, as aplicações podem ganhar interatividade na medida em que se adaptam a situação do usuário.

Desenvolver aplicações que exploram ao máximo as capacidades multimídia dos aparelhos exige das equipes constante adição e reformulação de recursos. A grande concorrência, porém, exige agilidade e constante redução de custos, problemas que podem ser atacados com a utilização de *frameworks*[1].

Diante do exposto, este trabalho objetiva analisar a troca de informações multimídia vinculadas a informações de georreferenciamento entre celulares com a plataforma Android, desenvolvendo um aplicativo exemplo capaz de demonstrar a troca de informações entre usuários. Serão utilizadas as ferramentas de desenvolvimento do Android *Software*

[1] Conforme Sauvé (2010), "um framework provê uma solução para uma família de problemas semelhantes" através de "um conjunto de classes e interfaces que mostra como decompor a família de problemas" (SAUVÉ, 2010). Apresenta também as relações entre as classes e pontos específicos para inclusão de novas classes.

*Developer Kit* (Android SDK), buscando explorar os recursos já disponibilizados na plataforma. Os componentes serão desenvolvidos de modo a permitir a maior reusabilidade e adaptação possíveis. Por fim, os componentes criados podem ser empregados no desenvolvimento de novas aplicações com conteúdo multimídia e georreferenciamento.

## 1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é disponibilizar um *framework* para a troca de informações multimídia vinculadas a informações de geolocalização na plataforma Android.

Os objetivos específicos do trabalho são:

a) permitir a troca de textos, sons, imagens, vídeos e coordenadas geográficas entre celulares com a plataforma Android através de um servidor;
b) utilizar interfaces de programação disponibilizadas pela plataforma Android para obtenção das coordenadas de cada aparelho e para exibição das informações multimídia;
c) disponibilizar um aplicativo exemplo desenvolvido utilizando o framework criado.

## 1.2 ESTRUTURA DO TRABALHO

Este trabalho está estruturado em quatro capítulos.

O capítulo dois contém a fundamentação teórica necessária para permitir um melhor entendimento sobre este trabalho.

O capítulo três apresenta o desenvolvimento do *framework*, contemplando os principais requisitos do problema e a especificação contendo os casos de uso, diagramas de componentes, de classes e de sequência. Nesse capítulo são apresentadas também as ferramentas utilizadas na implementação. Por fim são apresentados os resultados e discussão.

O quarto capítulo refere-se às conclusões do presente trabalho e sugestões para trabalhos futuros.

# 2 FUNDAMENTAÇÃO TEÓRICA

Na seção 2.1 é apresentado o conceito de multimídia móvel e algumas de suas implicações. A seção 2.2 contém uma introdução a plataforma Android, incluindo os componentes do sistema operacional e os aspectos básicos do ambiente de execução. Na seção 2.3 são apresentados os principais aspectos do desenvolvimento para a plataforma Android, incluindo os componentes básicos das aplicações. Na seção 2.4 são apresentados os principais componentes multimídia da plataforma Android. Ao final, a seção 2.5 apresenta dois trabalhos correlatos ao trabalho proposto.

## 2.1 MULTIMÍDIA MÓVEL

O aumento da capacidade de processamento e armazenamento dos computadores tem permitido a incorporação de novas mídias, que vão muito além dos textos e pequenos arquivos existentes nos primeiros computadores. Além disso, novos dispositivos de captura permitem a geração e reprodução de sons, imagens, vídeos e outros tipos de mídias. Esses recursos tem sido explorados por novas aplicações e a combinação de mídias tornou-se comum, com aplicativos que incorporam diversas características multimídia. Conforme Ibrahim (2008, p. xli), a combinação de mídias aumenta a qualidade da informação, permitindo melhor representação do mundo real e melhor compreensão das informações, além de potencializar comunicação entre pessoas ou pessoas e sistemas.

Além da geração e utilização das mídias, as aplicações modernas buscam distribuir mídias entre usuários, através da troca e do compartilhamento das informações. As trocas permitem uma maior interação entre as pessoas, com as mais variadas finalidades. Equipes de trabalho podem trocar informações que facilitem a execução de suas tarefas. Amigos podem trocar informações para entretenimento, entre outras possibilidades. Esta constante troca de mídias tem despertado nas pessoas o interesse por acesso a aplicativos multimídia a qualquer momento, em qualquer lugar.

Buscando atender a esta demanda, cada vez mais serviços de multimídia são disponibilizados para plataformas móveis, especialmente celulares e *Personal Digital Assistants* (PDAs). A criação de novas redes sem fio tem aumentado a capacidade de

transferência de informações destes dispositivos, além de novos dispositivos convergirem tecnologias de captura de informações em aparelhos cada vez menores e mais portáteis. Aliado a tudo isso, novos sistemas operacionais simplificam a utilização destes recursos pelas aplicações, disponibilizando interfaces de programação para acesso aos dispositivos e aplicativos já prontos para execução de tarefas ligadas à multimídia.

Surge, assim, o conceito de multimídia móvel que, segundo Ibrahim e Taniar (2008, p. vii), pode ser caracterizada como um conjunto de padrões para troca de informações multimídia através de redes sem fio, permitindo aos usuários processar e transmitir informações multimídia, disponibilizando aos usuários serviços diversos. Ainda segundo Ibrahim e Taniar (2008, p. vii), estes serviços podem ser de diversos tipos, entre eles, área de trabalho, entretenimento, recuperação de informações e serviços baseados em contexto.

Os serviços e aplicações baseados em contexto representam um grande avanço de interatividade, uma vez que permitem aos usuários obter informações de interesse no determinado contexto em que o mesmo se encontra. Assim, aplicações podem buscar e exibir informações que só serão úteis ou só terão sentido no determinado momento.

O principal tipo de serviço baseado em contexto é aquele que utiliza a localização do aparelho como forma de obtenção do contexto. Ou seja, aplicativos que exploram a localização do usuário para obter informações relevantes, como pontos que podem ser de interesse, rotas a serem seguidas ou até mesmo compartilham a localização atual do usuário com outros. Os Serviços Baseados em Localização (SBLs) podem permitir também que o próprio usuário associe informações multimídia a sua localização, vinculando texto, som, imagem ou vídeo a um local, por exemplo.

A constante análise da posição do usuário, porém, enfrenta grandes e complexos problemas. A determinação da posição do usuário não é trivial e, para maiores precisões, exige auxílio de hardware e sistemas de GPS. A atualização constante de informações de interesse pode consumir muitos recursos de rede, ainda escassos em plataformas móveis. Por fim, o filtro incorreto de informações pode ocupar os recursos com informações desnecessárias e até mesmo gerar poluição visual no aplicativo.

Assim, a exploração de recursos multimídia, especialmente os baseados em localização, pode gerar aplicações altamente interativas, porém exige dos desenvolvedores análise criteriosa e grande valorização dos recursos. A utilização de plataformas modernas e preparadas para aplicações móveis pode aumentar significativamente a produtividade, uma vez que libera os desenvolvedores de tarefas simples e triviais.

## 2.2 PLATAFORMA ANDROID

Segundo Open Handset Alliance (2010b), a plataforma Android é um pacote completo para plataformas móveis, incluindo um sistema operacional, um *middleware* e aplicações chave. Tem seu desenvolvimento controlado pela Open Handset Alliance, que é uma reunião de empresas com o objetivo de acelerar a inovação tecnológica e permitir a definição de um padrão aberto de desenvolvimento de software para as plataformas móveis (OPEN HANDSET ALLIANCE, 2010a). Segundo Android Open Source Project (2010), a plataforma Android é a primeira plataforma móvel de código aberto e totalmente customizável disponível.

O sistema operacional da plataforma Android (também conhecido apenas como Android) é organizado em cinco componentes principais: *kernel*[2] Linux, bibliotecas, ambiente de execução, *framework* de aplicação e aplicações.

O *kernel* Linux é responsável pelas funções mais básicas de gerenciamento do sistema, entre elas o gerenciamento de memória, de processos, de *drivers* e de dispositivos. Serve também como uma abstração entre os dispositivos físicos e as camadas de software.

Auxiliando a execução das aplicações há um conjunto de bibliotecas compartilhadas. Estas bibliotecas são escritas nas linguagens C/C++ de forma customizada para dispositivos móveis e são disponibilizadas através de *Application Programmer Interfaces* (APIs) da plataforma (ANDROID DEVELOPERS, 2010k). Dentre as bibliotecas disponíveis, estão uma biblioteca de operações padrão do sistema, biblioteca para gerenciamento de toques (para telas sensíveis ao toque), um motor de visualização de páginas web, bibliotecas para gráficos em duas e três dimensões, biblioteca para geração de bancos de dados relacionais locais e bibliotecas multimídia. Essas últimas são o ponto chave para o desenvolvimento multimídia nesta plataforma, já que permitem a reprodução de diversos formatos de áudio e vídeo através de bibliotecas nativas.

O ambiente de execução é composto por um conjunto de bibliotecas Java e uma máquina virtual Java criada especialmente para dispositivos móveis. Conforme Android Developers (2010k), o conjunto de bibliotecas disponibiliza a maior parte das funcionalidades existente na plataforma *Java Standard Edition* (JSE). Já a máquina virtual Dalvik é bastante

[2] *Kernel*, do inglês núcleo, é a denominação dada ao componente central de um sistema operacional. Normalmente esse componente está no nível mais baixo da camada de software e comunica-se com os dispositivos físicos.

diferente das máquinas virtuais Java existentes em outras plataformas, especialmente por ser baseada em registros e não em pilhas (DALVIKVM, 2008). Utilizando arquitetura baseada em registros, são geradas trinta por cento menos instruções para o processador do que nas máquinas virtuais baseadas em pilhas (HASHIMI e KOMATINENI, 2009, p. 5). A máquina virtual Dalvik utiliza-se de funcionalidades do *kernel* do sistema para tarefas básicas, como o controle de processos e gerenciamento de memória (DALVIKVM, 2008). Permite também a execução de diversas instâncias simultâneas e independentes. Conforme Android Developers (2010k), por padrão cada aplicação é executada em seu próprio processo, com sua instância da máquina virtual. Assim, quando o primeiro componente de uma aplicação é executado, o sistema operacional Android inicia um novo processo Linux contendo uma instância da máquina virtual Dalvik e uma única *thread*[3] de execução (ANDROID DEVELOPERS, 2010c). Esse processo inicial continuará existindo até o sistema operacional decidir que este deve ser encerrado para liberar espaço em memória. Por padrão, todos os demais componentes executarão nesse mesmo processo e *thread* (ANDROID DEVELOPERS, 2010c). Para alterar esse comportamento, é necessária a criação explícita de novas *threads* dentro da mesma instância da máquina virtual ou a definição de componentes a serem executados em novas instâncias da máquina virtual.

Antes da execução no sistema Android, as classes Java de uma aplicação precisam ser combinadas em um ou mais arquivos *Dalvik Executable* (`.dex`), formato desenvolvido especialmente para a máquina virtual Dalvik (HASHIMI e KOMATINENI, 2009, p. 5). Esse formato é otimizado tanto para armazenamento no dispositivo quanto para a execução na máquina virtual, através do agrupamento das partes de classes compiladas em listas e tabelas de informações (DALVIKVM, 2008). Das listas, cita-se as listas de classes, de campos e de métodos. Das tabelas, cita-se as tabelas de *strings* e definições de classes. Essas listas são organizadas de forma sequencial dentro do arquivo `.dex`, precedidas por um cabeçalho (*header*) de controle (DALVIKVM, 2008). Assim, informações duplicadas são eliminadas, reduzindo o tamanho do arquivo final para aproximadamente metade do tamanho de um arquivo *Java archieve* (`.jar`) tradicional (HASHIMI e KOMATINENI, 2009, p. 5). Conforme Dalvikvm (2008), após a compilação, os arquivos `.dex` são compactados para um único arquivo *Android package* (`.apk`). Esse arquivo pode então ser instalado e executado no sistema operacional Android.

[3] Uma *thread* é uma das linhas de execução existentes dentro de um processo. Através da criação de diversas *threads*, um processo pode executar diversas operações de forma concorrente, melhorando o desempenho do aplicativo ou a usabilidade.

O *framework* de aplicação é composto por um conjunto de APIs comum a todas as aplicações Android, inclusive as aplicações básicas disponibilizadas junto ao sistema. É inteiramente escrito na linguagem Java e, conforme Android Developers (2010k), através destas APIs as aplicações podem efetuar diversas operações. Entre elas cita-se: acionar dispositivos de *hardware*, obter informações de localização, executar serviços de forma oculta, configurar alertas, adicionar notificações, entre outras possibilidades.

As aplicações disponíveis incluem um conjunto básico de aplicativos que desempenham as funções primárias de um celular, como efetuar ligações, enviar mensagens de texto através do *Short Message System* (SMS), visualizar contatos e outras aplicações mais avançadas como *browser*, calendário e visualizador de mapas (ANDROID DEVELOPERS, 2010k). Nesse mesmo nível estão todas as demais aplicações desenvolvidas para a plataforma. Ou seja, não há diferenciação entre as aplicações padrão da plataforma e as aplicações desenvolvidas por terceiros (OPEN HANDSET ALLIANCE, 2010b).

## 2.3 DESENVOLVIMENTO NA PLATAFORMA ANDROID

O desenvolvimento de aplicações para a plataforma Android é feito através do Android SDK disponibilizado para a plataforma, sendo o código fonte escrito na linguagem Java, com forte auxílio de arquivos escritos em *eXtensible Markup Language* (XML). O Android SDK possui um conjunto completo de bibliotecas para desenvolvimento, aplicativos para a compilação e geração dos executáveis e um emulador do sistema operacional Android para testes. Conforme Android Developers (2010c), cada aplicação é compilada e empacotada em um arquivo *Android package* (extensão `apk`), que pode então ser disponibilizado e executado nos dispositivos móveis. A integração do Android SDK com o *Eclipse Integrated Development Environment* (IDE) pode ser feita através do *plugin Android Development Tools* (ADT). Esse *plugin* possibilita a execução de um aplicativo no emulador diretamente a partir do Eclipse, além de possibilitar o *debug* da aplicação em execuções passo a passo (LECHETA, 2009, p. 29).

Uma das principais características da plataforma Android é a capacidade de uma aplicação fazer uso de componentes disponibilizados por outras aplicações (ANDROID DEVELOPERS, 2010c). Assim, uma aplicação pode ser formada por componentes próprios e acessar componentes de outras aplicações facilmente. Para tal, sua arquitetura precisa

favorecer esse comportamento através da organização em torno dos componentes básicos da plataforma: *services*, *activities*, *broadcast receivers* e *content providers*. Cada um desses componentes pode ser instanciado pelo sistema Android e executado conforme necessidade. O código fonte das aplicações não é incorporado ou ligado (ANDROID DEVELOPERS, 2010c), ou seja, cada aplicação é executada de forma independente em seu próprio processo. Exemplificando, uma aplicação pode conter uma *activity* na qual o usuário pode selecionar diversas opções, entre elas o envio de uma mensagem de texto através de uma *activity* de outra aplicação. A primeira aplicação estará em execução em seu próprio processo. No momento em que o usuário selecionar a opção de envio da mensagem de texto, o sistema Android interromperá a primeira *activity* e ativará a *activity* da segunda aplicação em outro processo. Ao término dessa última, o sistema Android retomará a execução do primeiro processo. Tudo isso deve ocorrer de forma transparente para o usuário, sem que o mesmo perceba que houve transição entre os processos. Portanto, uma aplicação Android não possui um ponto de entrada, mas vários componentes que podem ser instanciados e executados conforme a necessidade.

Para ativação dos componentes, podem ser usados objetos do tipo `android.content.Intent`, que conforme Lecheta (2009, p. 119) “representa uma ’ação’ que a aplicação deseja executar”. Esses objetos, chamados de intenções (do inglês *intents*), são estruturas de dados que podem ser enviadas ao sistema operacional Android através da solicitação de execução de uma ação. O sistema irá então decidir qual componente é capaz de responder a determinada ação e executar o mesmo. O componente selecionado pode estar na aplicação em execução ou em uma aplicação externa, sendo a execução transparente em ambos os casos. A ativação de componentes externos é transparente para o usuário, que tem a percepção de estar navegando o tempo todo na mesma aplicação. Da mesma forma, a ação do usuário de abrir uma aplicação é uma intenção, que o sistema resolve buscando um componente na aplicação que responda a essa intenção.

Assim, a plataforma Android cria um novo meio de desenvolvimento de aplicações, no qual uma aplicação é composta de componentes próprios e de outras aplicações, de forma transparente para o usuário. Conforme a interação com o usuário, a aplicação dispara intenções, que são resolvidas pelo sistema operacional Android de forma flexível e transparente.

### *2.3.1 Services*

Os *services* são componentes utilizados para processamentos longos e sem interação com o usuário ou para suprir funcionalidades para outras aplicações (ANDROID DEVELOPERS, 2010h). Assim, um *service* é um componente sem interface gráfica e capaz de executar por um período indefinido de tempo. Podem permanecer em execução, inclusive, após o usuário deixar a aplicação ou iniciar outra aplicação (ANDROID DEVELOPERS, 2010c). Outros componentes podem comunicar-se com um *service* em execução e inclusive solicitar a sua interrupção. Por padrão, um *service* será executado na mesma *thread* que disparar sua execução (ANDROID DEVELOPERS, 2010h). Para modificar esse comportamento, é necessário definir a criação de uma nova *thread* ou um novo processo explicitamente. Dessa forma, *services* podem ser utilizados para processamentos assíncronos, fazendo com que a aplicação permaneça respondendo a outros eventos, mesmo quando processamentos longos são disparados. Exemplificando, caso um usuário dispare uma comunicação com um servidor remoto, um *service* pode ser utilizado para realizar esta tarefa em uma nova *thread* enquanto a interface com o usuário é atualizada com a situação da comunicação. Caso a comunicação com o servidor remoto seja feita na *thread* principal do programa, sua interface tende a ficar parada até a conclusão, causando a impressão de travamento da aplicação.

Conforme Lecheta (2009, p. 291), a vantagem de utilizar um *service* em detrimento de uma simples *thread* está relacionada à capacidade de reconhecimento do mesmo pelo sistema operacional Android. Uma *thread* disparada dentro de uma *activity* pode ser encerrada junto com a *activity* pelo sistema operacional. Esse comportamento pode ser indesejado em situações em que é necessário garantir a consistência da execução do processamento. Já um *service* em execução é reconhecido pelo sistema como componente que não deve ser interrompido em casos normais de execução. Além disso, um *service* em execução tem prioridade sobre todos os outros processos em segundo plano (LECHETA, 2009, p. 291).

O desenvolvimento de *services* é feito através da criação de subclasses de `android.app.Service`, sendo obrigatória a implementação do método `onBind(Intent)` e sugerida a implementação dos métodos `onCreate()`, `onStart()` e `onDestroy()` (LECHETA, 2009, p. 293). O método `onBind(Intent)` é invocado para a obtenção de um canal de comunicação com o *service* (ANDROID DEVELOPERS, 2010h). Esse canal é utilizado por componentes que precisam se comunicar com o *service* durante a execução do

mesmo. Os métodos `onCreate()`, `onStart()` e `onDestroy()` estão relacionados ao ciclo de vida de um *service* e são invocados, respectivamente, na criação, início e destruição do *service* (ANDROID DEVELOPERS, 2010h). Através desses métodos é possível disparar novas *threads* de execução, obter valores persistidos e efetuar a persistência de valores para processamento futuro.

### *2.3.2 Activities*

Uma *activity* representa uma interface do usuário com o sistema. Contém uma tela da aplicação com um propósito bem definido e é responsável por tratar os eventos relativos a ela (LECHETA, 2009, p. 77). Uma aplicação pode ser composta de uma ou mais *activities*. Normalmente as *activities* de uma aplicação trabalham em conjunto invocando uma a outra e criando uma interface coesa (ANDROID DEVELOPERS, 2010c). Apesar disso, cada activity é independente das demais e pode ser invocada inclusive por outras aplicações.

As *activities* são desenvolvidas em classes Java, através da criação de subclasses de `android.app.Activity` (LECHETA, 2009, p. 77). Nas subclasses criadas, o método `onCreate(Bundle)` deve ser implementado, tratando o evento de criação da *activity* e efetuando as inicializações necessárias (ANDROID DEVELOPERS, 2010a). Dentre essas inicializações está a criação da interface gráfica e, opcionalmente, a recuperação de uma situação anterior ou inicialização de outros componentes. A interface gráfica é exibida através de hierarquias de *views*, nas quais cada componente gráfico controla uma área retangular específica da tela (ANDROID DEVELOPERS, 2010a). A hierarquia de componentes pode ser definida via código Java ou através de arquivos XML. A definição via código Java acontece através da criação e associação de instâncias dos componentes gráficos dentro do código fonte da *activity*. Como alternativa, a hierarquia pode ser definida via arquivos XML de leiaute. Nesses arquivos, a hierarquia é definida através da organização dos elementos dentro do arquivo XML. Após a criação dos arquivos de leiaute, os mesmos são importados para dentro do código fonte através de classes auxiliares, geradas pelo Android SDK. Antes da execução no dispositivo, os arquivos de leiaute são compilados para arquivos binários, como forma de otimizar a execução (HASHIMI e KOMATINENI, 2009, p. 5).

Os componentes gráficos são desenvolvidos através da criação de subclasses de `android.view.View` (ANDROID DEVELOPERS, 2010j). Cada componente ocupa uma área retangular na tela, sendo responsável por seu desenho e pelo tratamento de eventos relativos a

ele. A partir das *views* são criados os *widgets*, componentes que definem elementos de interação da interface como botões, campos para entrada de texto e campos para exibição de textos (ANDROID DEVELOPERS, 2010j). Outra especialização das *views* é definida pela classe `android.view.ViewGroup`, que é base para a definição de leiautes. Na plataforma Android, um leiaute é uma classe capaz de conter *views* e organizá-las na tela conforme regras específicas. Normalmente uma interface gráfica é composta por um leiaute base, no qual são adicionados outros leiautes ou *widgets*. A plataforma Android possui diversos componentes gráficos disponíveis, que podem ser utilizados através das bibliotecas disponíveis.

### 2.3.3 *Broadcast Receivers* e *Content Providers*

Os *broadcasts receivers* são componentes responsáveis por receber notificações sobre eventos diversos, incluindo os gerados pelo sistema (ANDROID DEVELOPERS, 2010c). Como exemplos de eventos gerados pelo sistema, cita-se: ativação da câmera, mensagem recebida, mudança de fuso horário e bateria fraca. Uma aplicação pode também disparar eventos, como forma de notificar outras aplicações. Um *broadcast receiver* não possui interface gráfica, porém pode disparar a exibição de uma *activity*, capaz de exibir uma interface para tratamento do evento (ANDROID DEVELOPERS, 2010c). Uma aplicação pode conter diversos *broadcasts receivers*, para o tratamento de diversos eventos. É comum que os eventos de tratamento mais demorado sejam processados de forma assíncrona por *services*, para que o *broadcast receiver* possa responder a novos eventos (LECHETA, 2009, p. 290).

B*roadcasts receivers* são criados através de classes que estendem da classe `android.content.BroadcastReceiver` (ANDROID DEVELOPERS, 2010c). Para tal, é necessário implementar o método `onReceive(Context, Intent)`. Esse método é invocado quando o *broadcast receiver* é determinado como receptor de determinada mensagem.

Por padrão, cada aplicação Android possui sua própria área de dados, independente das demais aplicações e com acesso restrito a aplicação que criou as informações (LECHETA, 2009, p. 385). Para o compartilhamento de informações, as aplicações podem criar *content providers*, componentes capazes de compartilhar conjuntos específicos de informações com outras aplicações (ANDROID DEVELOPERS, 2010c). A criação de *content providers* é feita através de subclasses de `android.content.ContentProvider` (LECHETA, 2009, p. 385), com a implementação de métodos que buscam e retornam as informações solicitadas. As

demais aplicações, porém, não invocam esses métodos diretamente. A execução é intermediada por um objeto de controle do tipo `android.content.ContentResolver`. A plataforma Android disponibiliza diversos *content providers* nativos, que podem ser utilizados pelas aplicações. Entre os content providers nativos, cita-se provedores de contatos da agenda, localizador de arquivos, imagens e vídeos (LECHETA, 2009, p. 386).

### 2.3.4 AndroidManifest

Para que cada componente da aplicação possa ser utilizado corretamente, o mesmo precisa estar mapeado no arquivo `AndroidManifest.xml`. Este arquivo é escrito no formato XML e deve existir em todas as aplicações Android, com exatamente este nome (ANDROID DEVELOPERS, 2010i). Contém informações básicas sobre a aplicação que precisam ser interpretadas pelo sistema operacional Android para a correta instalação e execução dos componentes. Antes da instalação de uma aplicação, o sistema operacional Android analisa o arquivo `AndroidManifest` e verifica a compatibilidade e as permissões necessárias. A partir dessa análise, o usuário recebe um resumo das permissões e pode decidir instalar ou não a aplicação.

Por ser um arquivo XML padrão, as informações são dispostas no `AndroidManifest` em *tags* e atributos. A *tag* `<manifest>` indica o início da declaração da aplicação. Nessa *tag*, o atributo `package` indica o pacote Java da aplicação, que é utilizado como identificador único da aplicação (ANDROID DEVELOPERS, 2010i). As tags `<uses-permission>` e `<permission>` indicam, respectivamente, permissões obrigatórias e que podem ser limitadas. Essas permissões são concedidas a todos os componentes da aplicação e, caso não existam no arquivo `AndroidManifest`, serão automaticamente negadas às aplicações que tentarem utilizá-las. Informações sobre o Android SDK utilizado são armazenadas na *tag* `<uses-sdk>`, sendo o atributo `android:minSdkVersion` da mesma o indicador da versão mínima da plataforma na qual o aplicativo pode ser executado. Bibliotecas adicionais que precisam ser suportadas pelo dispositivo são declaradas na *tag* `<uses-library>`.

A declaração dos componentes básicos (*services*, *activities*, *broadcast receivers* e *content providers*) ocorre através de *tags* específicas, cada uma contendo informações sobre as classes que implementam os serviços e as capacidades dos serviços (ANDROID DEVELOPERS, 2010i). Através destas declarações, o sistema operacional Android sabe

quais componentes podem ser executados e em quais condições (ANDROID DEVELOPERS, 2010i). Os *services* são declarados através de *tags* `<service>`. As *activities* são declaradas através de *tags* `<activity>`. *Broadcast receivers* e *content providers* são declarados através de *tags* `<receiver>` e `<provider>`, respectivamente. Dentro das *tags* de cada componente é possível definir a quais intenções o componente responde, através da *tag* `<intent-filter>`. Através da interpretação dessas intenções, o sistema operacional Android é capaz de decidir qual componente será executado em tempo de execução. Aplicações que publiquem as intenções que seus componentes respondem estarão automaticamente disponibilizando esses componentes para outras aplicações.

### 2.3.5 Arquivos e classe auxiliares

Com a finalidade de facilitar o desenvolvimento de aplicações, os projetos de aplicativos Android criados com o *plugin* ADT possuem um conjunto de arquivos e diretórios auxiliares. Recursos utilizados pela aplicação podem ser adicionados a diretórios criados dentro do próprio projeto. Esses diretórios posteriormente serão empacotados no arquivo `.apk` junto com a aplicação e estarão disponíveis apenas para ela. Como recurso pode-se entender uma imagem, um arquivo XML (LECHETA, 2009, p. 57) ou qualquer outro arquivo que seja utilizado pela aplicação. Ao criar um novo projeto através do *plugin* ADT, um conjunto de diretórios é criado automaticamente. Os diretórios `res/drawable-hdpi`, `res/drawable-ldpi` e `res/drawable-mdpi` são indicados para armazenar recursos desenháveis (imagens) em alta, baixa e média resolução, respectivamente. O diretório `res/layout` é indicado para armazenar arquivos de leaiute em XML. O diretório `res/values` é indicado para armazenamento de arquivos de constantes. Esse diretório contém o arquivo `strings.xml`, no qual podem ser declaradas cadeias de caracteres (*strings*) para serem utilizadas nas telas da aplicação. Através desse arquivo é possível reutilizar cadeias de caracteres fazendo com que várias telas estejam relacionadas a mesma cadeia.

A classe auxiliar `R` é gerada automaticamente pelo *plguin* ADT, contendo constantes que referenciam os recursos do projeto (LECHETA, 2009, p. 57). Sempre que o projeto é compilado, o plugin verifica os diretórios padrão e atualiza a classe com constantes inteiras que referenciam os recursos. Diversos métodos das bibliotecas da plataforma Android estão preparados para receber constantes da classe `R`. Dessa forma, a localização dos recursos é

abstraída do desenvolvedor e resolvida pela própria plataforma.

## 2.4 MULTIMÍDIA NA PLATAFORMA ANDROID

A plataforma Android foi desenvolvida para permitir a exploração dos mais modernos recursos multimídia. Prova disso é a grande quantidade de funcionalidades já implementadas na API, capazes de reproduzir diversos formatos de mídia, exibir mapas e imagens de satélite e obter as coordenadas do dispositivo. A reprodução de mídias pode ser efetuada de forma simples, através da utilização de intenções (*intentes*) e atividades (*activities*) disponíveis na API Android. Esta API disponibiliza também classes capazes de gerar mídias, através da captura de dados (capturar sons pelo microfone do aparelho, por exemplo) ou obtenção de informações (coordenadas do aparelho, por exemplo).

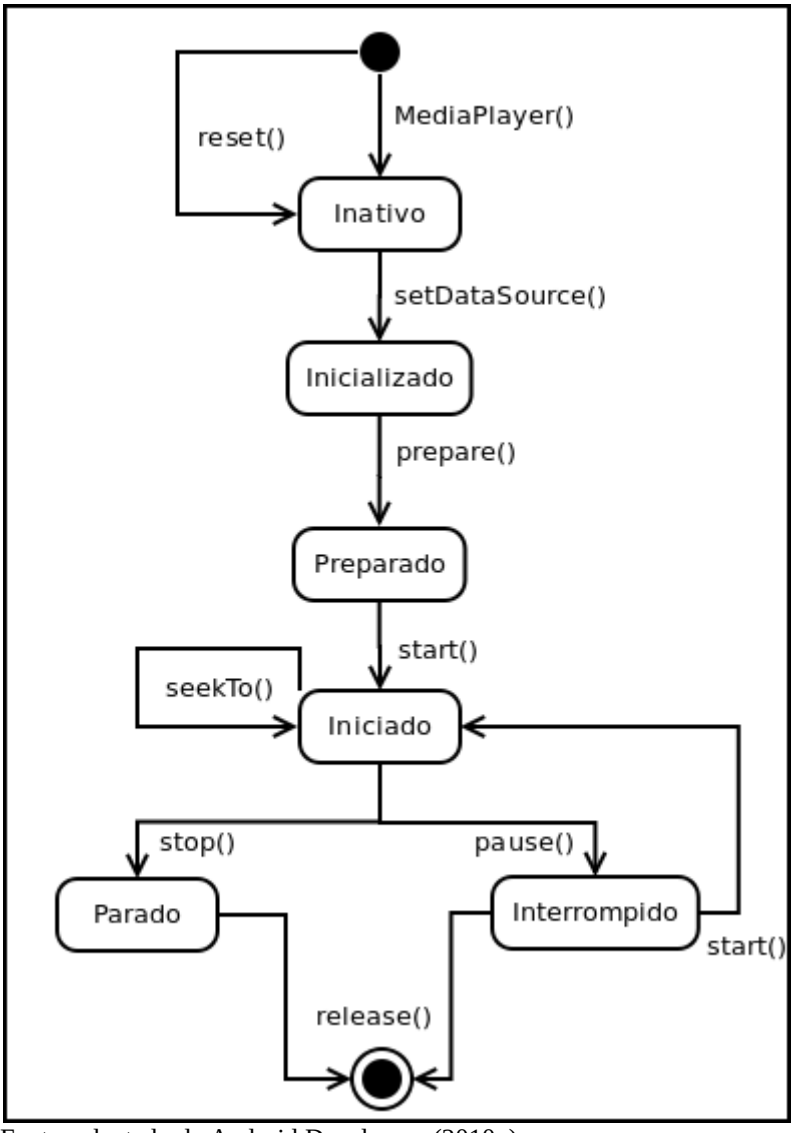
As principais classes para a captura e reprodução de áudio e vídeo da API Android fazem parte do pacote `android.media` (ANDROID DEVELOPERS, 2010d). Entre essas classes estão `MediaPlayer` e `MediaRecorder`. A primeira é capaz de reproduzir áudio e vídeo, enquanto a segunda é capaz de capturar áudio.

A obtenção das coordenadas do aparelho também pode ser feita através da API padrão da plataforma Android. O pacote `android.location` contém a classe `LocationManager`, que permite acesso à posição geográfica (ANDROID DEVELOPERS, 2010f). Não há uma biblioteca nativa na API Android para visualização de mapas, porém há uma biblioteca externa disponibilizada pela Google que permite a incorporação de mapas interativos nas aplicações (ANDROID DEVELOPERS, 2010f).

As aplicações tem disponível também uma série de meios para a troca de informações entre dispositivos ou entre dispositivos e servidores. A API Android contém classes para conexões via *sockets* e protocolo *HyperText Transfer Protocol* (HTTP). Para acesso a *Web Services*, a aplicaçao precisa utilizar bibliotecas externas, uma vez que não há biblioteca nativa para acesso a estes serviços (LECHETA, 2009, p. 508).

### 2.4.1 Reprodução de Áudio e Vídeo

A classe `android.media.MediaPlayer` pode ser utilizada para coordenadar a reprodução de áudio, vídeo ou *streammings* de forma transparente e otimizada. A reprodução é gerenciada por uma máquina de estados implementada pela classe (ANDROID DEVELOPERS, 2010g). Os estados dessa máquina são alterados conforme a invocação dos métodos disponibilizados e são os mesmos para áudio, vídeo e *streamming*. A Figura 1 demonstra os principais métodos e estados da classe `android.media.MediaPlayer`, através de um diagrama de estados válidos para a classe.




Fonte: adaptado de Android Developers (2010g).

Figura 1 – Métodos e estados da classe `MediaPlayer`

Conforme (ANDROID DEVELOPERS, 2010g), em uma nova instância de `android.media.MediaPlayer`, o método `setDataSource()` define a origem das

informações a serem reproduzidas, podendo ser um arquivo de áudio, vídeo ou um *streamming*. O método `prepare()` prepara a execução e só é encerrado quando a instância está pronta para inciar a reprodução. O método `start()` inicia a reprodução, que pode ser interrompida através dos métodos `pause()` ou `stop()`. A reprodução pode ser retomada através da invocação método `start()`, caso tenha sido interrompida pelo método `pause()`. Caso a reprodução seja interrompida pelo método `stop()` é necessário reiniciar a instância de `android.media.MediaPlayer` através do método `reset()`, não sendo possível retomar a reprodução diretamente (LECHETA, 2009, p. 540). Durante a reprodução, é possível também definir o instante a partir do qual a reprodução deve continuar, através do método `seekTo()`. Ao fim da reprodução, é necessário invocar o método `release()` para que os recursos sejam liberados (LECHETA, 2009, p. 541).

Como forma de facilitar a reprodução de vídeos, a biblioteca da plataforma disponibiliza a classe `android.widget.VideoView`. Essa classe encapsula os estados da reprodução de vídeos em um componente que pode ser adicionado a uma tela da aplicação (LECHETA, 2009, p. 559). Esse componente permite a visualização do vídeo e a interação por comandos básicos de interromper, prosseguir, avançar e retroceder.

A plataforma Android suporta nativamente diversos formatos de mídia. Os formatos de áudio suportados nativamente são *Advanced Audio Coding* (AAC), *Adaptive Multi-Rate* (AMR), *Moving Picture Experts Group Audio Layer 3* (MP3), *Musical Instrument Digital Interface* (MIDI), *WAVEform audio format* (WAVE) e Ogg Vorbis (ANDROID DEVELOPERS, 2010g). Da mesma forma os formatos de vídeo suportados são H.263, H.264 e *Moving Picture Experts Group Layer 4* (MPEG-4). Esses formatos nativos de áudio e vídeo são reproduzidos através do Opencore Multimedia Framework, que é um framework multimídia otimizado para dispositivos móveis. Esse é modular e extensível, objetivando permitir a criação de diversos cenários de reprodução de mídias (KOSMACH, 2008, p. 2).

### 2.4.2 Localização e Mapas

Conforme Lecheta (2009, p. 415), uma das funcionalidades mais atraentes da plataforma Android é o desenvolvimento de aplicações de localização sem a escrita de muitas linhas de código. Por padrão, a plataforma Android disponibiliza um serviço de localização através da classe `android.location.LocationManager` (ANDROID DEVELOPERS,

2010f). Uma referência a um objeto desta classe pode ser obtida do sistema através de uma chamada ao método `getSystemService(Context.LOCATION_SERVICE)`. Através dessa referência é possível obter os provedores de localização e a última localização conhecida (ANDROID DEVELOPERS, 2010f). A classe `android.location.LocationManager` permite também registrar um objeto ouvinte para atualizações periódicas de localização. Além disso, é possível registrar um tipo de *intent* a ser disparada quando o aparelho encontrar-se a determinada distância de uma coordenada geográfica. A precisão da localização geográfica, porém, depende do hardware do aparelho em que o sistema está instalado e dos serviços disponíveis na determinada região. Durante a fase de desenvolvimento, é possível testar as aplicações no emulador através da definição manual de coordenadas fictícias.

A visualização de mapas envolve componentes que não são padronizados pela plataforma Android, sendo necessária a utilização de bibliotecas externas. Dentre as bibliotecas disponíveis há uma disponibilizada pela própria Google capaz de obter, renderizar e guardar cache de trechos de mapas (ANDROID DEVELOPERS, 2010f). O principal pacote dessa biblioteca é o `com.google.android.maps`, que contém a classe `MapView`. Essa classe pode ser adicionada ao leiaute de uma *activity* para exibir conteúdo obtido diretamente do serviço Google Maps (ANDROID DEVELOPERS, 2010f). Através dos métodos da classe `com.google.android.maps.MapView` é possível renderizar mapas de forma customizada, definindo as coordenadas nas quais o mapa deve ser centralizado, o nível de *zoom*, controles para aumentar e reduzir o *zoom* e desenhar *overlays* sobre o mapa. Conforme Lecheta (2009, p. 433), *overlays* são especializações de `android.widget.View` utilizadas para adicionar desenhos sobre os mapas. Através desses desenhos é possível destacar pontos de interesse ou adicionar informações sobre determinada localização ao mapa. Uma aplicação que mostra a localização de outras pessoas, por exemplo, pode adicionar um *overlay* no mapa na posição de cada pessoa. Para facilitar a montagem de telas com mapas, *overlays* e controles, o Android SDK disponibiliza a classe `com.google.android.maps.MapActivity` que representa uma atividade na qual será inserido um mapa.

Para que uma aplicação utilize os recursos de localização e mapas, um conjunto de configurações é necessário durante o desenvolvimento da aplicação. Algumas linhas referentes a configurações das bibliotecas e autorizações necessárias precisam ser adicionadas ao arquivo `AndroidManifest.xml`. A obtenção da localização do dispositivo precisa da autorização `ACCESS_FINE_LOCATION`, referente a obtenção da localização do usuário. Essa autorização é solicitada através de uma linha semelhante a esta: `<uses-permission`

`android:name="android.permission.ACCESS_FINE_LOCATION" />`. As classes da biblioteca de mapas necessitam de acesso a internet para obter as informações de mapas. Assim, é necessário adicionar uma solicitação de autorização semelhante a esta: `<uses-permission android:name="android.permission.INTERNET" />`. Por fim, o pacote de classes da biblioteca de renderização de mapas deve ser importado através de uma linha semelhante a esta: `<uses-library android:name="com.google.android.maps" />`.

Por utilizar informações do serviço Google Maps, a biblioteca de mapas do Google para Android está sujeita a alguns termos de utilização e serviço (GOOGLE, 2010c). Dentre estes termos, está o registro da aplicação e obtenção de uma chave de acesso ao Google Maps. Conforme Google (2010c) essa chave de acesso é obtida através de uma conta de usuário Google e do certificado digital gerado pelo Android SDK para a aplicação. Através dessas informações, o desenvolvedor recebe uma chave em forma de cadeia de caracteres. Essa chave deve ser utilizada no construtor da classe `com.google.android.maps.MapView` para permitir a criação de instâncias da mesma. É importante observar que o certificado digital gerado pelo Android SDK para a aplicação acompanha a mesma após o desenvolvimento, na distribuição e instalação da aplicação nos aparelhos. Dessa forma, a integridade da aplicação é garantida, evitando alterações indesejadas.

### 2.4.3 Comunicação HTTP

O protocolo HTTP é um dos protocolos mais importantes da internet hoje (JAKARTA COMMONS, 2008b). Objetivando facilitar a comunicação através desse protocolo, a plataforma Android encapsula a biblioteca `HTTPClient` disponibilizada pela Apache Software Foundation. Essa biblioteca contém um conjunto abrangente de classes (HASHIMI e KOMATINENI, 2009, p. 263), além de ser universalmente utilizada por aplicações *Java Enterprise Edition* (JEE). Nesse conjunto estão as classes necessárias para que aplicações possam fazer requisições HTTP e receber respostas de forma simples e orientada a objetos. A biblioteca `HTTPClient` dá suporte a requisições através de todos os métodos HTTP: `GET`, `POST`, `PUT`, `DELETE`, `HEAD`, `OPTIONS`, e `TRACE` (JAKARTA COMMONS, 2008a). Suporta também outras funcionalidades, entre as quais criptografia, proxies, autenticação, gerenciamento de conexões e troca de cookies (JAKARTA COMMONS, 2008a).

A biblioteca `HTTPClient` disponibiliza classes que representam cada um dos métodos

do protocolo HTTP. As classes `org.apache.http.client.methods.HttpPost` e `org.apache.http.client.methods.HttpGet`, por exemplo, representam respectivamente uma requisição `POST` e uma requisição `GET`. A execução das requisições é feita através do método `execute(HttpUriRequest)` definido pela interface `org.apache.http.client.HttpClient`. Este método tem como retorno uma instância de `org.apache.http.HttpResponse` que contém o resultado do processamento efetuado no recepetor da mensagem HTTP. Dentre as classes que implementam a interface `HttpClient`, cita-se a classe padrão `org.apache.http.impl.client.DefaultHttpClient`.

Cada objeto da classe `HttpPost` está relacionado a uma entidade que representa o corpo da requisição. Essa entidade pode ser montada com os parâmetros e informações a serem processados e depois relacionada ao objeto de `HttpPost` através do método `setEntity(HttpEntity)`. A forma como a montagem desta entidade ocorre depende da aplicação e do tipo de dado a ser transportado. Uma das formas de montagem está descrita no padrão *multipart*, que deve ser utilizado para requisições que contenham arquivos e informações binárias (W3, 2010). Nesse padrão, cada requisição HTTP é formada por um conjunto de partes. Cada parte representa uma submissão válida (W3, 2010), ou seja, na mensagem gerada, cada parte representa um corpo de conteúdo. O resultado é uma mensagem com diversos corpos de conteúdo, separados por marcadores que delimitam cada uma das partes ou corpos de conteúdo. Esses marcadores não devem interferir na informação contida (W3, 2010). Exemplificando, uma entidade pode ser montada por duas partes, cada parte contendo um arquivo distindo. Ao adicionar os arquivos, são adicionados também marcadores para delimitar o início e término de cada parte dentro da entidade. Os marcadores adicionados não podem ser interpretados como conteúdo dos arquivos, sob pena de corromper as informações. No receptor da mensagem HTTP, a leitura dos marcadores permite a separação da mensagem nas partes originais, facilitando a interpretação.

A classe `HttpPost` permite a associação com entidades do tipo *multipart*, porém não há componentes na biblioteca `HTTPClient` que simplifiquem a criação dessas entidades. Para auxiliar neste processo, a biblioteca Java `HTTPMime` pode ser incorporada a aplicação, provendo diversas classes para a criação de partes e entidades. Nessa biblioteca, a classe `org.apache.http.entity.mime.MultipartEntity` representa uma entidade *multipart* e pode receber partes de diversos tipos através do método `addPart(ContentBody)`. Entre os tipos de partes, cita-se arquivos e cadeias de caracteres. Os tipos são representados por classes específicas. Os arquivos são representados por instâncias de

`org.apache.http.entity.mime.content.FileBody`. Já as cadeias de caracteres são representadas por instâncias de `org.apache.http.entity.mime.content.StringBody`.

Assim, a conciliação da biblioteca `HTTPClient` encapsulada na plataforma Android com bibliotecas Java externas permite a criação de aplicativos capazes de efetuar diversos tipos de requisições HTTP de forma simples e com a abstração de vários detalhes.

## 2.5 TRABALHOS CORRELATOS

Diversos trabalhos buscam explorar as características multimídia da plataforma Android. Foram selecionados trabalhos conforme a aderência entre as funcionalidades apresentadas pelos trabalhos disponíveis e as funcionalidades propostas neste trabalho. Desta forma, o MAPBR (WEISS, 2008) busca explorar as funcionalidades de visualização de mapas na plataforma Android. Já o Google Latitude (GOOGLE, 2010a) é um serviço para visualização da localização de contatos via celular ou computador.

### 2.5.1 MAPBR

O Android MAPBR (WEISS, 2008) objetivou explorar as características da plataforma Android, à época recente e com poucos trabalhos disponíveis. Demonstra as características da plataforma Android e a anatomia geral de uma aplicação, detalhando as *activities* e sua utilização. O Android MAPBR apresenta as APIs e bibliotecas disponibilizadas que podem ser utilizadas ou extendidas. O trabalho referencia também outras plataformas móveis, situando a plataforma Android nessa área.

A aplicação desenvolvida permite a visualização de mapas e imagens de satélite através da biblioteca de visualização de mapas disponibilizada pela Google para a plataforma Android. Entre os requisitos da aplicação está a possibilidade de “visualizar, adicionar, selecionar, editar, remover, ocultar, listar e salvar marcadores na aplicação” (WEISS, 2008, p34). Cada marcador representa uma coordenada geográfica no mapa e tem suas informações persistidas no dispositivo. Por fim, o usuário pode visualizar os mapas e imagens de satélite em diversas escalas, aplicando *zoom*, além de permitir o deslocamento por todo o mapa.

### 2.5.2 Google Latitude

O Google Latitude (GOOGLE, 2010a) é um dos serviços gratuitos disponibilizados pela Google. É composto por aplicações móveis e sites para acesso via browser. Atualmente há versões para diversas plataformas móveis, entre elas as plataformas Android, iPhone e BlackBerry (GOOGLE, 2010a). O serviço tem como objetivo principal o compartilhamento da posição atual do usuário e a visualização da posição de outros usuários em listas ou mapas. O Latitude é integrado aos demais serviços da Google, sendo possível adicionar amigos via contatos do Gmail, atualizar o status do aplicativo de mensagens instantâneas Google Talk ou exibir a localização do usuário em *blogs* e redes sociais (GOOGLE, 2010b). Além do compartilhamento da localização através do serviço, o Google Latitude permite o envio da localização do usuário através de SMS ou *e-mail* para contatos que não necessariamente são usuários do Google Latitude.

O serviço disponibiliza também opções para que o histórico de localizações seja armazenado nos servidores da Google. Habilitando estas opções, as atualizações de localização do usuário podem ser visualizadas pelo usuário posteriormente via *browser*. Outra funcionalidade disponível é a notificação de que um amigo esta próximo em um local que não seja rotineiro (GOOGLE, 2010b). Através da análise das atualizações de localização do usuário, o serviço identifica locais de rotina como ambientes de trabalho ou estudo. Com base nessas informações o serviço irá disparar alertas quando o usuário estiver em locais incomuns e se aproximar de amigos. Por fim, a privacidade das informações é controlada por cada usuário, com configurações de atualização manual ou automática e de quais amigos podem ver a sua localização.

# 3 DESENVOLVIMENTO

Neste capítulo são abordadas as etapas de desenvolvimento do *framework*. A primeira seção descreve a instalação dos componentes do ambiente de desenvolvimento. A segunda seção apresenta os principais requisitos do problema trabalhado. A terceira seção descreve a especificação da solução através de diagramas da *Unified Modeling Language* (UML) e de Modelos de Entidades e Relacionamentos (MER). A quarta seção apresenta a implementação da solução, incluindo os principais trechos do código fonte e exemplos de uso do *framework*. Por fim, a quinta seção aborda resultados deste trabalho.

## 3.1 AMBIENTE DE DESENVOLVIMENTO

Considerando as especificidades do aplicativo cliente e do servidor, o presente trabalho foi desenvolvido em dois projetos. Em um projeto o aplicativo cliente foi desenvolvido, de forma independente do servidor. No segundo projeto o servidor foi desenvolvido, de modo a prover suporte para as funcionalidades demonstradas no aplicativo cliente.

O aplicativo cliente foi desenvolvido utilizando o Android SDK, ambiente de desenvolvimento padrão da plataforma. Detalhes sobre a instalação do Android SDK podem ser obtidos através da referência Android Developers (2010e). Para a execução do projeto no simulador é necessário definir um dispositivo virtual. Informações sobre a criação desses dispositivos podem ser encontradas na referência Android developers (2010b).

O servidor foi desenvolvido sobre a plataforma JEE, no ambiente de desenvolvimento Eclipse. Como servidor de aplicação foi utilizado o servidor Apache Tomcat. O acesso ao banco de dados foi efetuado através de um *driver Java Database Connectivity* (JDBC) para o banco de dados MySQL.

## 3.2 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Os Requisitos Funcionais (RFs) do *framework* Mediashare são:

a) RF01: permitir a definição de conexões entre usuários;
b) RF02: atualizar as coordenadas geográficas do usuário no sistema enquanto o aplicativo estiver em execução;
c) RF03: permitir a consulta das coordenadas dos usuários conectados;
d) RF04: permitir a troca de informações multimídia georreferenciadas (textos, imagens, sons e vídeos) entre usuários conectados;
e) RF05: permitir a definição de coordenadas geográficas de interesse;
f) RF06: permitir a troca de coordenadas geográficas de interesse entre usuários conectados.

Os Requisitos Não Funcionais (RNFs) do framework Mediashare são:

a) RNF01: ser implementado para a plataforma Android;
b) RNF02: exibir as informações multimídia existentes no aplicativo utilizando o paradigma de intenções da plataforma Android.

## 3.3 ESPECIFICAÇÃO

A especificação deste trabalho foi desenvolvida utilizando diagramas da UML aliados ao MER. Os casos de uso são apresentados através de um diagrama de casos de uso, seguido da descrição do cenário de cada caso de uso. Em seguida, os componentes da aplicação cliente são apresentados através de um diagrama de componentes. As classes do aplicativo cliente e do servidor são apresentadas através de diagramas de classes. Como forma de facilitar a implementação, dois casos de uso foram representados através de diagramas de sequência. Complementando a especificação, as entidades dos bancos de dados do cliente e do servidor são demonstradas através de diagramas MER, que não pertencem ao conjunto de diagramas da UML.

### 3.3.1 Casos de uso

A partir dos requisitos criados para o Mediashare, foram desenvolvidos onze casos de uso. Esses casos de uso objetivam organizar os requisitos em funcionalidades que possam ser executadas de forma simples pelo usuário. O desenvolvimento dos casos de uso também

levou em consideração os padrões da plataforma Android. Em nenhum dos casos de uso foi disponibilizada opção de voltar ou cancelar na tela. Essa opção já existe nos dispositivos com a plataforma Android na forma de um botão voltar.

Os casos de uso desenvolvidos são desempenhados por dois atores. O ator `Usuário` representa o utilizador do sistema, sendo capaz de interagir através dos casos de uso que apresentam interfaces gráficas. Já o ator `mediashare` representa o aplicativo cliente, que interage através de serviços que rodam em segundo plano e não apresentam interface gráfica. Esses serviços apenas disparam notificações para o sistema em alguns eventos específicos.

O diagrama de casos de uso foi desenvolvido observando os padrões da UML. Cada caso de uso foi detalhado em cenários e vinculado a pelo menos um RF. Essa vinculação objetiva facilitar a identificação do propósito do caso de uso e justificar sua existência. A Figura 2 contém o diagrama de casos de uso do *framework*.

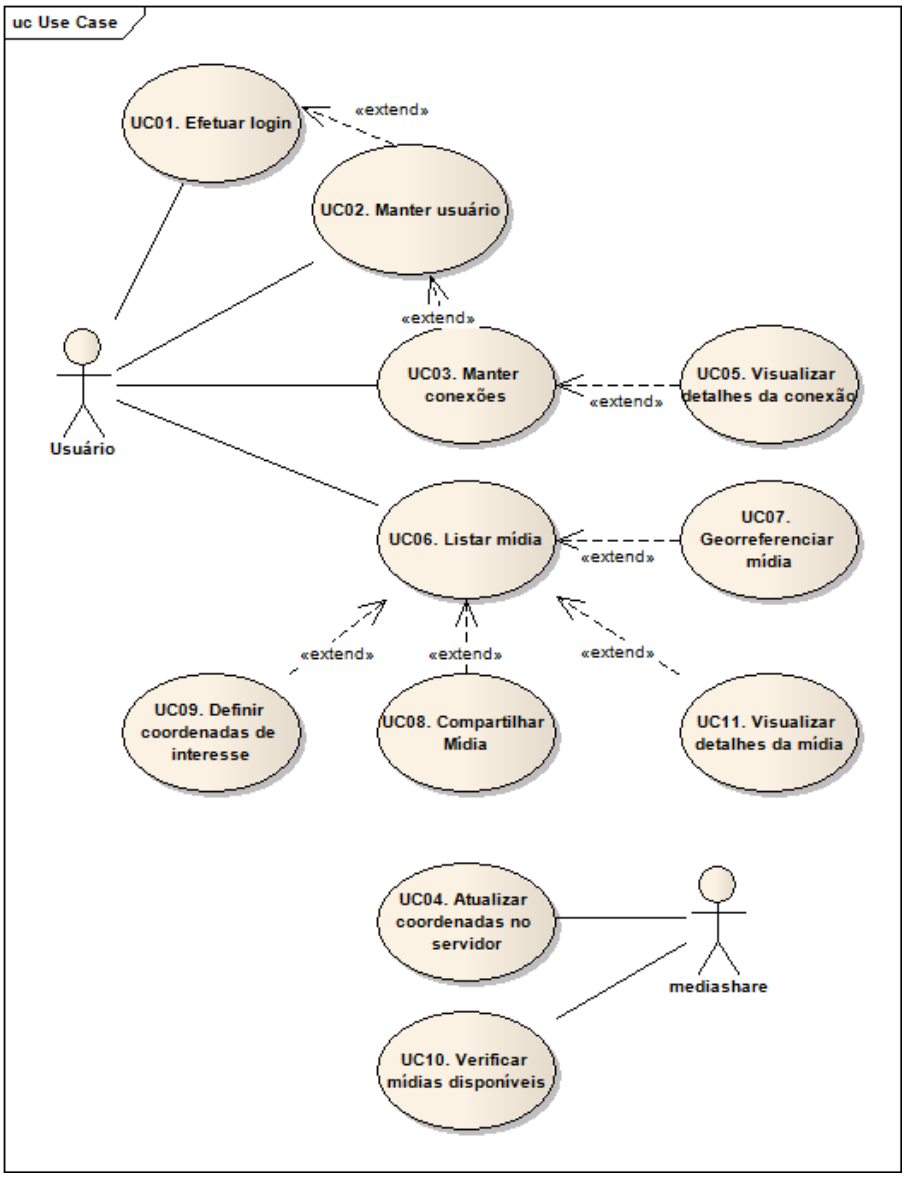



Figura 2 - Diagrama de casos de uso da aplicação desenvolvida

#### 3.3.1.1 Efetuar login

O caso de uso `Efetuar login` é o único caso de uso obrigatório do medishare. Isso ocorre porque todos os outros casos de uso tem como pré-requisito o usuário ter efetuado login no sistema. Esse caso de uso apresenta uma janela para informação do nome de usuário e senha. O usuário deve informar uma combinação válida para poder prosseguir na utilização do sistema. O Quadro 1 contém os cenários do caso de uso `Efetuar login`.

| UC01. Efetuar login: Possibilita ao usuário validar seu nome de usuário e senha da aplicação mediashare e acessar o menu inicial | |
|---|---|
| Requisitos atendidos | RF01 |
| Pré-condições | Nenhuma |
| Cenário principal | 1) o sistema apresenta campos para informar usuário e senha seguida de uma opção para criar usuário,<br>2) o usuário informa seu nome de usuário e senha:<br>3) o sistema verifica se a combinação de nome de usuário e senha existe e informa se a combinação é válida ou inválida, |
| Fluxo alternativo 01 | No passo 1, caso o usuário selecione opção para criar usuário:<br>1) executa o caso de uso UC02 |
| Fluxo alternativo 02 | No passo 3, caso a combinação usuário/senha estiver correta:<br>1) exibe a tela inicial do sistema |
| Pós-condições | Login válido armazenado nas configurações do aplicativo mediashare no cliente |

Quadro 1 - Caso de uso `Efetuar login`

#### 3.3.1.2 Manter usuário

Para facilitar o compartilhamento de mídias e permitir a manutenção de conexões entre usuários, foi definido um cadastro para os mesmos. O caso de uso `Manter usuário` tem como propósito apresentar uma tela em que o próprio usuário pode definir ou alterar suas informações. Ao confirmar essas alterações, as mesmas são refletidas no servidor. O Quadro 2 apresenta os cenários do caso de uso `Manter usuário`.

| UC02. Manter usuário: Permite ao usuário editar suas informações no mediashare ou criar um novo usuário | |
|---|---|
| Requisitos atendidos | RF01 |
| Pré-condições | Para o usuário alterar suas informações, o mesmo deve estar logado no mediashare |
| Cenário principal | 1) o sistema apresenta tela com os campos nome de usuário, nome, senha e confirmação de senha e os botões de Ok,<br>2) o usuário preenche os campos e pressiona Ok para salvar,<br>3) o sistema salva as informações no servidor e atualiza as informações no cliente |
| Exceção 01 | No passo 3, ocorrendo erro ao salvar as informações no servidor:<br>1) o sistema exibe mensagem contento o erro ocorrido ao salvar as informações,<br>2) retorna ao passo 1 do cenário principal |
| Pós-condições | Informações do usuário alteradas e salvas ou nenhuma informação salva |

Quadro 2 - Caso de uso `Manter usuário`

#### 3.3.1.3 Manter Conexões

Cada usuário do Mediashare pode adicionar, excluir ou listar os outros usuários aos quais está conectado. Ou seja, cada usuário pode manter suas conexões. O caso de uso `Manter conexões` objetiva apresentar uma tela em que o usuário possa efetuar essa manutenção. Após alterações, as mesmas são validadas e refletidas no servidor. O Quadro 3 contém os cenários do caso de uso `Manter conexões`.

| UC03. Manter conexões: Permite ao usuário listar, adicionar ou excluir conexões com outros usuários | |
|---|---|
| Requisitos atendidos | RF01 |
| Pré-condições | O usuário está logado no sistema |
| Cenário principal | 1) o sistema busca a lista de usuários conectados no servidor,<br>2) o sistema exibe a lista de usuários já conectados ao usuário logado e as opções adicionar |
| Fluxo alternativo 01 | No passo 1, o usuário pressiona a opção adicionar:<br>1) o sistema exibe campo para informar o nome do usuário a ser adicionado,<br>2) o usuário preenche o nome do usuário a ser adicionado e pressiona Ok,<br>2) o sistema verifica se o nome de usuário informado existe e adiciona a lista de usuários conectados |
| Fluxo alternativo 02 | No passo 2, o usuário pressiona o nome de uma das conexões e em seguida a opção remover:<br>1) o sistema remove o usuário selecionado da lista de usuários conectados |
| Fluxo alternativo 03 | No passo 2, o usuário pressiona o nome de uma das conexões e em seguida a opção visualizar:<br>1) executa o caso de uso UC05 |
| Exceção 01 | No passo 3 do fluxo alternativo 01, ocorre erro ao adicionar o usuário:<br>1) o sistema exibe a mensagem de erro ocorrida ao adicionar o usuário,<br>2) retorna ao passo 1 do fluxo alternativo 01 |
| Exceção 02 | No passo 1 do fluxo alternativo 01, ocorre erro ao remover o usuário:<br>1) o sistema exibe a mensagem de erro ocorrida ao remover o usuário,<br>2) retorna ao passo 1 do cenário principal |
| Pós-condições | Usuário adicionado, excluído ou lista de usuários exibida |

Quadro 3 - Caso de uso `Manter conexões`

#### 3.3.1.4 Atualizar coordenadas no servidor

Durante a execução do aplicativo cliente, o mesmo deve atualizar as coordenadas de cada usuário no sistema. Dessa forma, cada um pode visualizar a localização dos usuários aos quais está conectado. O caso de uso `Atualizar coordenadas no servidor` é executado pelo ator `mediashare` (aplicativo cliente) e, em intervalos de tempo definidos, busca a localização atual do usuário e a envia para o servidor. O Quadro 4 contém os cenários do caso

de uso `Atualizar coordenadas no servidor`.

| UC04. Atualizar coordenadas no servidor: os serviços do sistema são executados em segundo plano (sem intervenção do usuário) e atualizam as coordenadas geográficas do usuário no servidor em intervalos de tempo configurados | |
|---|---|
| Requisitos atendidos | RF02 |
| Pré-condições | O usuário está logado no sistema |
| Cenário principal | 1) o sistema dispara a execução do serviço de atualização das coordenadas,<br>2) o serviço de atualização das coordenadas obtém as coordenadas do dispositivo,<br>3) o serviço envia as coordenadas para o servidor,<br>4) o serviço espera a quantidadade de tempo configurada para nova atualização,<br>5) retorna ao passo 2 |
| Exceção 01 | No passo 2, ocorre erro ao obter as coordenadas do dispositivo:<br>1) executa o passo 4 do cenário principal |
| Exceção 02 | No passo 3, ocorre erro ao enviar as coordenadas do dispositivo para o servidor:<br>1) executa o passo 4 do cenário principal |
| Pós-condições | As coordenadas geográficas do usuário são atualizadas no servidor em intervalos de tempo |

Quadro 4 - Caso de uso `Atualizar coordenadas no servidor`

#### 3.3.1.5 Visualizar detalhes da conexão

O caso de uso `Visualizar detalhes da conexão` apresenta os detalhes de uma conexão do usuário. Esse caso de uso apresenta também um mapa no qual a localização da conexão é indicada, com o objetivo de facilitar a identificação do local. O Quadro 5 contém os cenários do caso de uso `Visualizar detalhes da conexão`.

| UC05. Visualizar detalhes da conexão: Permite ao usuário visualizar os detalhes de sua conexão, contendo nome de usuário, nome completo do usuário, coordenadas geográficas e localização em um mapa | |
|---|---|
| Requisitos atendidos | RF03 |
| Pré-condições | O usuário está logado no sistema, o usuário está conectado a pelo menos um outro usuário |
| Cenário principal | 1) o sistema busca os detalhes do usuário selecionado no servidor,<br>2) o sistema exibe campos com o nome do usuário, nome completo do usuário, coordenadas do usuário e um mapa indicando a localização do usuário,<br>3) usuário visualiza as informações exibidas |
| Exceção 01 | No passo 1 do cenário principal, ocorrendo erro ao buscar os detalhes do usuário:<br>1) o sistema exibe mensagem contendo o erro ocorrido,<br>2) encerra o caso de uso |
| Pós-condições | Detalhes do usuário exibidos |

Quadro 5 - Caso de uso `Visualizar detalhes da conexão`

#### 3.3.1.6 Listar mídias

Para que o usuário possa georreferenciar os objetos de mídia é necessário antes apresentar os objetos disponíveis. O caso de uso `Listar mídias` engloba telas do aplicativo cliente que listam os objetos de áudio, vídeo, imagens, textos ou coordenadas geográficas disponíveis no dispositivo. A partir desse caso de uso o usuário pode disparar outros, de georreferenciamento, visualização, compartilhamento ou criação de coordenadas geográficas. O Quadro 6 contém os cenários do caso de uso `Listar mídias`.

| UC06. Listar mídias: Permite ao usuário listar os arquivos de áudio, vídeo, imagens, registros de texto ou de coordenadas armazenados no dispositivo | |
|---|---|
| Requisitos atendidos | RF04, RF05, RF06 |
| Pré-condições | O usuário está logado no sistema, o usuário selecionou uma das mídias a ter os arquivos ou registros listados |
| Cenário principal | 1) o sistema exibe a tela principal contendo a lista de arquivos de áudio, vídeo, imagens, os textos ou as coordenadas geográficas existentes no dispositivo e opção para adicionar coordenadas geográficas |
| Fluxo alternativo 01 | No passo 1, o usuário pressiona um objeto de mídia e em seguida a opção para definição de coordenadas:<br>1) executa o caso de uso UC07,<br>2) retorna ao passo 1 do cenário principal |
| Fluxo alternativo 02 | No passo 1, o usuário pressiona um objeto de mídia e em seguida a opção para alteração de coordenadas:<br>1) Executa o caso de uso UC07,<br>2) retorna ao passo 1 do cenário principal |
| Fluxo alternativo 03 | No passo 1, o usuário pressiona um objeto de mídia e em seguida a opção para visualizar a mídia:<br>1) executa o caso de uso UC11,<br>2) retorna ao passo 1 do cenário principal |
| Fluxo alternativo 04 | No passo 1, o usuário pressiona um objeto de mídia e em seguida a opção para compartilhamento:<br>1) executa o caso de uso UC08,<br>2) retorna para o passo 1 do cenário principal |
| Fluxo alternativo 05 | No passo 1, o usuário pressiona a opção para adicionar coordenadas geográficas:<br>1) executa o caso de uso UC09,<br>2) retorna ao passo 1 do cenário principal |
| Exceção 01 | No passo 1, ocorre erro ao obter os objetos de mídia:<br>1) o sistema exibe o erro ocorrido,<br>2) encerra o caso de uso |
| Pós-condições | Lista de mídias exibida |

Quadro 6 - Caso de uso `Listar mídias`

### 3.3.1.7 Georreferenciar mídia

O compartilhamento de mídias através do Mediashare exige que as mesmas sejam antes georreferenciadas. Através do caso de uso `Georreferenciar mídia` um objeto de mídia pode ser associado a uma coordenada geográfica (georreferenciado). O Quadro 7

contém os cenários do caso de uso `Georreferenciar mídia`.

| UC07. Georreferenciar mídia: O sistema apresenta uma tela na qual o usuário pode marcar uma coordenada geográfica no mapa e salvá-la no dispositivo | |
|---|---|
| Requisitos atendidos | RF04 |
| Pré-condições | O usuário está logado no sistema |
| Cenário principal | 1) o sistema apresenta uma tela contendo um mapa, uma opção para marcar as coordenadas no mapa e uma opção para concluir,<br>2) o usuário desloca o mapa até a localização desejada, pressiona a opção para marcar coordenadas, pressiona um ponto no mapa e depois a opção para concluir,<br>3) o sistema associa as coordenadas marcadas no mapa ao objeto de mídia selecionado |
| Pós-condições | O objeto de mídia é associado a coordenadas geográficas |

Quadro 7 - Caso de uso `Georreferenciar mídia`

#### 3.3.1.8 Compartilhar mídia

O caso de uso `Compartilhar mídia` permite o compartilhamento de mídias através de um servidor. O sistema apresenta uma tela para seleção do destinatário do compartilhamento. Após confirmar o compartilhamento, o sistema envia uma cópia da mídia para o servidor, onde a mensagem irá aguardar que o destinatário efetue a cópia da mídia para o seu dispositivo. O Quadro 8 contém os cenários do caso de uso `Compartilhar mídia`.

| UC08. Compartilhar mídia: Permite ao usuário compartilhar um objeto de mídia com outro usuário do mediashare. O objeto compartilhado é armazenado no servidor, para posterior recebimento do destinatário | |
|---|---|
| Requisitos atendidos | RF04, RF06 |
| Pré-condições | Há uma mídia associada a coordenadas geográficas no mediashare |
| Cenário principal | 1) o sistema exibe uma tela para selecionar o usuário que irá receber a mídia e um botão para confirmar,<br>2) o usuário seleciona um destinatário para o compartilhamento e em seguida pressiona o botão de confirmação,<br>3) o sistema verifica se o usuário destino existe e armazena o objeto compartilhado no servidor |
| Exceção 01 | No passo 3, caso o usuário destino não exista:<br>1) exibe mensagem informando que o usuário destino não existe,<br>2) retorna ao passo 1 |
| Exceção 02 | No passo 3, caso ocorra erro ao enviar o objeto para o servidor:<br>1) exibe mensagem informando que ocorreu erro ao enviar o objeto,<br>2) retorna ao passo 1 |
| Pós-condições | Objeto de mídia armazenado no servidor |

Quadro 8 - Caso de uso `Compartilhar mídia`

#### 3.3.1.9 Definir coordenadas de interesse

O caso de uso `Definir coordenadas de interesse` apresenta uma tela contendo um mapa no qual o usuário pode definir uma coordenada de interesse. Essa coordenada é então adicionada a lista de coordenadas armazenada no dispositivo, para posterior compartilhamento ou associação com uma mídia. O Quadro 9 contém os cenários do caso de uso `Definir coordenadas de interesse`.

| UC09. Definir coordenadas de interesse: Permite ao usuário definir coordenadas geográficas de seu interesse, selecionando a localização em um mapa | |
|---|---|
| Requisitos atendidos | RF05 |
| Pré-condições | O usuário está logado no sistema |
| Cenário principal | 1) o sistema exibe um mapa interativo, uma opção para marcar coordenada e uma opção para concluir,<br>2) o usuário desloca o mapa até a localização desejada, pressiona a opção para marcar coordenadas, pressiona um ponto no mapa e depois a opção para concluir,<br>3) o sistema persiste as coordenadas selecionadas no dispositivo como coordenada geográfica de interesse |
| Pós-condições | Uma coordenada de interesse foi definida |

Quadro 9 - Caso de uso `Definir coordenadas de interesse`

#### 3.3.1.10 Verificar mídias disponíveis

Para completar o compartilhamento de um objeto de mídia, a mesma deve ser copiada do servidor para o dispositivo do usuário. O caso de uso `Verificar mídias disponíveis` é executado pelo ator `mediashare` com esse objetivo. Após a cópia da mídia para o dispositivo, a mesma é marcada como recebida no servidor e não é mais enviada para o cliente em requisições futuras. O Quadro 10 contém os cenários do caso de uso `Verificar mídias disponíveis`.

| UC10. Verificar mídias disponíveis: O sistema requisita ao servidor as mídias disponíveis para o usuário logado. Caso existam mídias disponíveis, as mesmas são copiadas do servidor para o dispositivo | |
|---|---|
| Requisitos atendidos | RF04 |
| Pré-condições | O usuário está logado no sistema |
| Cenário principal | 1) o sistema dispara a execução do serviço de verificação de mídias disponíveis,<br>2) o serviço de verificação de mídias requisita ao servidor as mídias disponíveis,<br>3) o serviço espera uma quantidade de tempo configurada para nova requisição,<br>4) retorna ao passo 2 |
| Fluxo alternativo 01 | No passo 2, o servidor retorna uma lista de mídias disponíveis:<br>1) o serviço copia cada uma das mídias do servidor para o dispositivo,<br>2) retorna para o passo 3 do cenário principal |
| Exceção 01 | No passo 2, ocorrendo erro ao efetuar a requisição para o servidor:<br>1) retorna para o passo 3 do cenário principal |
| Pós-condições | É efetuada requisição para verificar a disponibilidade de mídias no servidor |

Quadro 10 - Caso de uso `Verificar mídias disponíveis`

#### 3.3.1.11 Visualizar detalhes da mídia

Cada objeto de mídia existente no mediashare apresentará um conjunto de detalhes após ser georreferenciado. Através do caso de uso `Visualizar detalhes da mídia` o usuário pode visualizar os detalhes da mídia, incluindo as coordenadas geográficas associadas a ela. O Quadro 11 contém os cenários do caso de uso `Visualizar detalhes da mídia`.

| UC11. Visualizar detalhes da mídia: Permite a visualização de detalhes dos objetos de mídia, incluindo o nome, o usuário que associou as coordenadas geográficas e as coordenadas geográficas associadas | |
|---|---|
| Requisitos atendidos | RF04 |
| Pré-condições | Há uma mídia associada a coordenadas geográficas no mediashare |
| Cenário principal | 1) o sistema exibe o nome, o usuário que associou as coordenadas geográficas e as coordenadas geográficas associadas a mídia e uma opção para visualizar as coordenadas geográficas em um mapa |
| Fluxo alternativo 01 | No passo 1, caso a mídia em questão seja um vídeo, o sistema exibe uma área para a reprodução do vídeo |
| Fluxo alternativo 02 | No passo 1, caso a mídia em questão seja um arquivo de áudio, o sistema exibe uma área para a reprodução do áudio |
| Fluxo alternativo 03 | No passo 1, caso a mídia em questão seja uma imagem, o sistema exibe uma área com a visualização da imagem |
| Fluxo alternativo 04 | No passo 1, caso a mídia em questão seja um texto, o sistema exibe uma área com o conteúdo do texto |
| Exceção 01 | No passo 1, caso a mídia em questão seja um ponto de interesse, o sistema exibe um mapa indicando a localização com um marcador |
| Pós-condições | Detalhes da mídia exibidos |

Quadro 11 - Caso de uso `Visualizar detalhes da mídia`

### 3.3.2 Componentes do aplicativo cliente

Considerando o modelo de desenvolvimento em torno dos componentes básicos da plataforma Android, o aplicativo cliente foi organizado em *activities* e *services*. Para atender aos requisitos apresentados, os componentes responsáveis pela interação com o usuário foram definidos como especializações de *activities*. Os outros componentes definidos não possuem interação com o usuário, porém efetuam processamentos periódicos de atualização de informações. Esses últimos foram definidos como especializações de *services*. Cada componente teve também uma *intent* associada, com a finalidade de permitir sua ativação através do paradigma de intenções da plataforma Android.

A representação dos componentes definidos foi feita através de um diagrama de componentes da UML. No diagrama, as *activities* são representadas com o esteriótipo `activity`. Já os *services* são apresentados com o esteriótipo `service`. As *intents* disponibilizadas por cada componente foram representadas no diagrama como interfaces expostas. Para facilitar a interpretação, a descrição de cada *intent* foi reduzida a apenas o último elemento que compoe sua descrição. Ou seja, a *intent*

`br.furb.mediashare.intent.action.MediaMapView` foi reduzida para `MediaMapView`. Assim, todas as *intents* tem o prefixo `br.furb.mediashare.intent.action` omitido no diagrama e nos textos deste trabalho, como forma de simplificar a visualização e facilitar a interpretação. A única exceção é `android.intent.action.MAIN`, que é padrão da plataforma e indica a *activity* que inicia a aplicação. A ativação de cada *intent* foi representada através de uma conexão de dependência (linha tracejada com seta). Nesta representação, a origem da seta indica o componente que ativa a *intent* e o destino indica o componente que é ativado pela *intent*. Os componentes que disponibilizam mais de uma *intent* podem ser ativados de mais de uma maneira, apresentando comportamento diferente para cada *intent*. A Figura 3 contém o diagrama de componentes do aplicativo cliente.

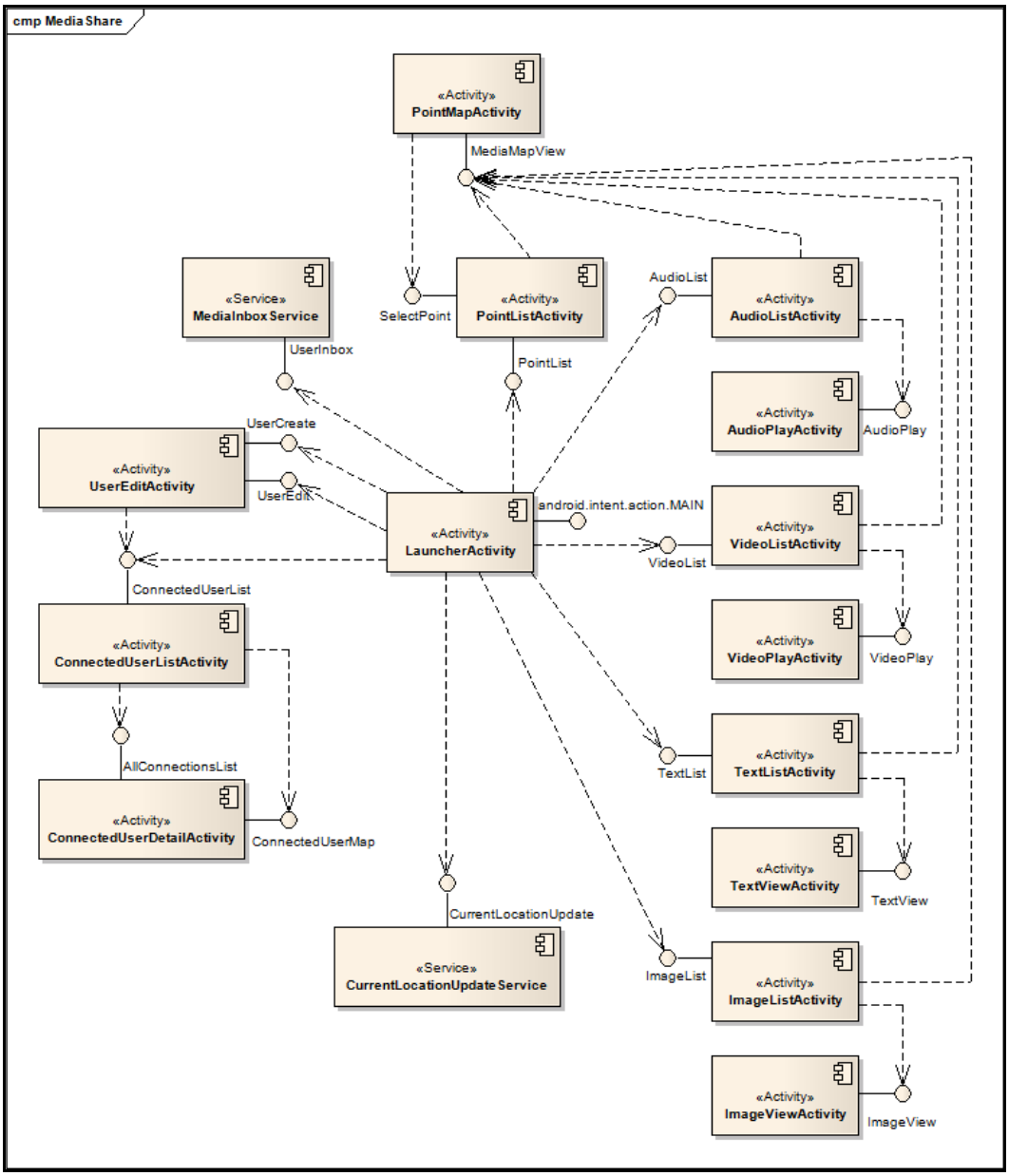



Figura 3 - Componentes do aplicativo cliente

A *activity* `LauncherActivity` é responsável pela inicialização do aplicativo e pela tela inicial. A partir dela o usuário pode disparar as *activities* responsáveis por listar cada uma das mídias disponíveis no aplicativo. A partir da tela inicial é possível também disparar o cadastro de um novo usuário, listar as conexões do usuário ou editar as informações do usuário (*intents* `UserCreate`, `ConnectedUserList` e `UserEdit`, respectivamente). `LaucherActivity` é responsável também por disparar os *services* que rodam de forma oculta e atualizam as informações do usuário no servidor e no cliente. Através da *intent* `CurrentLocationUpdate` o serviço de atualização da localização do cliente no servidor é disparado. Este serviço permanecerá em execução enquanto o aplicativo estiver em execução. As mídias disponíveis no servidor são copiadas para o dispositivo através do serviço `MediaInboxService`. Esse último efetua consultas periódicas ao servidor e é disparado pela *activity* inicial através da *intent* `UserInbox`.

A visualização da lista de mídias existentes no dispositivo é efetuada através de *activities* específicas para cada tipo de mídia. As *activities* `AudioListActivity`, `VideoListActivity` e `ImageListActivity` listam, respectivamente, os arquivos de áudio, vídeo e imagens existentes em determinado diretório do dispositivo. Cada arquivo de mídia pode ter geolocalização associada ou não, porém apenas os arquivos que possuírem a geolocalização definida poderão ser compartilhados. Ao selecionar uma mídia na lista de mídias, o determinado componente irá verificar se a mídia possui coordenadas associadas. Caso não possua, será habilitada a opção para selecionar as coordenadas geográficas em um mapa, através do componente `PointMapActivity`. Esse componente irá permitir a seleção das coordenadas e retornar para o componente que o ativou. Esse último deverá então persistir as coordenadas e associá-las ao arquivo de mídia. Ao selecionar uma mídia na lista, o usuário poderá também ativar uma *activity* de reprodução ou exibição da mídia. As *activities* `AudioPlayActivity` e `VideoPlayActivity` reproduzem, respectivamente, áudio e vídeo. A exibição das imagens é feita através da *activity* `ImageListActivity`.

Os textos e pontos de interesse (coordenadas geográficas) não são armazenados em arquivos. Essas informação são persistidas no espaço de memória da aplicação. Os textos são listados através da activity `TextListActivity`. A partir dessa é possível também editar e criar textos ativando a activity `TextViewActivity`. Cada texto terá sempre coordenadas geográficas associadas. Já os pontos de interesse (coordenadas geográficas) são listados através da activity `PointListActivity`. Da mesma forma que ocorre com as mídias persistidas em arquivos, as coordenadas associadas aos textos e aos pontos de interesse são

criadas ou visualizadas através da activity `PointMapActivity`.

### 3.3.3 Classes do aplicativo cliente

Partindo da definição dos componentes do aplicativo cliente, os mesmos foram organizados em pacotes e classes, de acordo com a proximidade das funções e dos tipos de componentes. Os componentes do tipo *activitiy* são implementados como classes que estendem `android.app.Activity` ou subclasses desta. Com a finalidade de simplificar os diagramas e facilitar a visualização, essas classes são representadas com o esteriótipo `Activity`. Os *services* são implementados como classes que estendem `android.app.Service`. Nos diagramas, os *services* são representados com o esterótipo `Service`. Cada aplicação Android é estruturada sob um pacote raíz, que deve ser único no dispositivo. Assim, a aplicação foi implementada sob o pacote `br.furb.mediashare`. A Figura 4 apresenta o diagrama do pacote raíz da aplicação cliente.

É importante observar que não há conexões entre *activities* ou *activities* e *services* no diagrama de classes. A navegação entre as *activities* é feita através de *intents*, que não tem referência direta às classes. O pacote raíz contém a classe `LaucherActivity`, que implementa o compontente de início do programa. Essa classe está associada a classe `UserLoginDialog`, capaz de exibir uma janela de solicitação das informações do usuário e validá-las no servidor. Caso as informações sejam válidas, as mesmas são persistidas através das classes `UserPersistence` e `MediashareUser`, ambas definidas no pacote `br.furb.mediashare.user`. Esse pacote contém as classes relativas aos componentes de informações de usuário. Já o pacote `br.furb.mediashare.media` contém as classes relativas aos componentes de mídias.

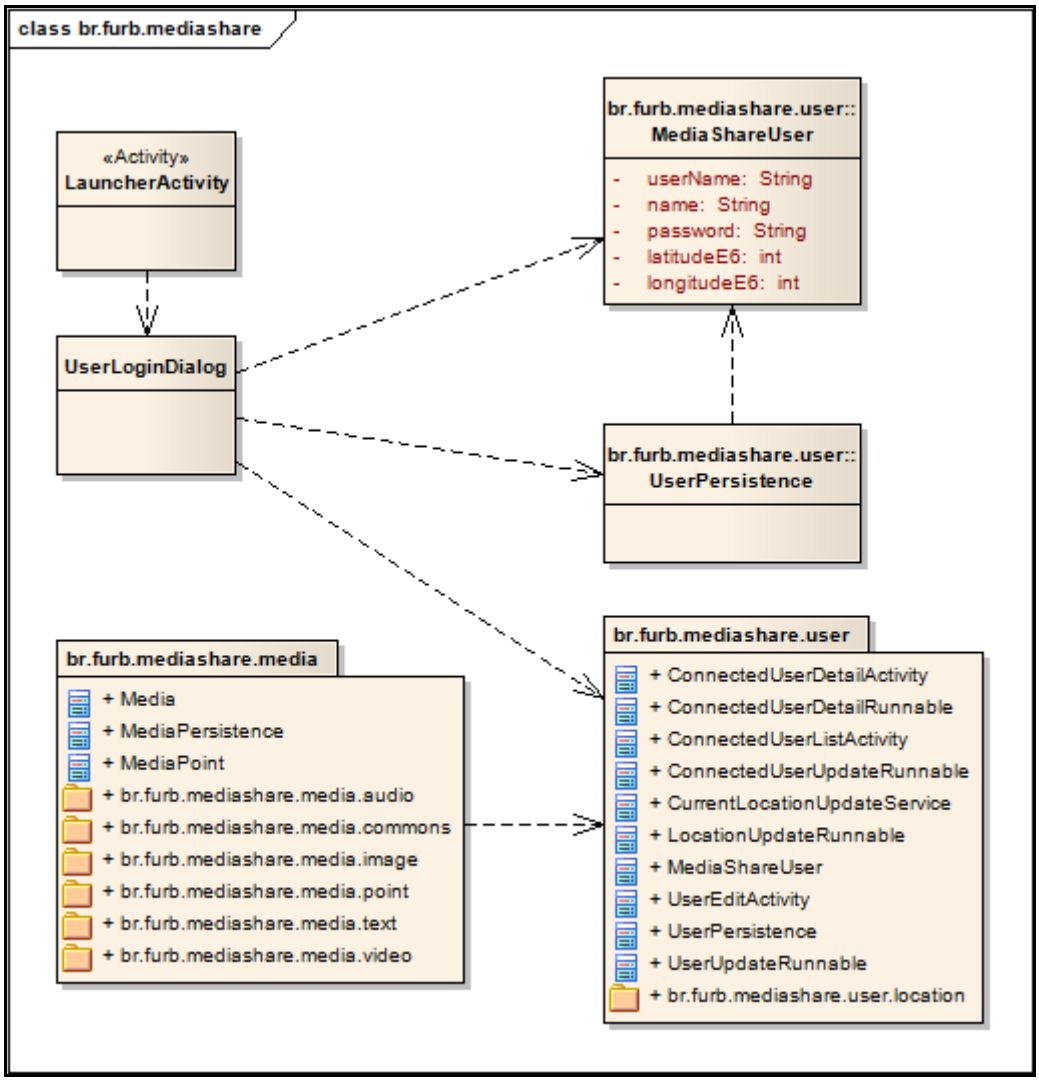



Figura 4 - Pacote `br.furb.mediashare`

#### 3.3.3.1 Pacote `br.furb.mediashare.user`

As classes relativas aos componentes de informações de usuário foram implementadas no pacote `br.furb.mediashare.user`. A Figura 5 contém o diagrama de classes do pacote `br.furb.mediashare.user`.

A classe `MediaShareUser` contém as informações do usuário que está utilizando o aplicativo. Essas informações são persistidas e recuperadas pela classe `UserPersistence` no banco de dados local da aplicação. Essa última classe é responsável também por persistir e recuperar as conexões do usuário do aplicativo. As informações do usuário podem ser editadas através da activity `UserEditActivity`. Essa está associada a classe `UserUpdateRunnable`, que efetua a atualização das informações no servidor em uma nova

*thread*. A visualização e definição dos usuários conectados é efetuada através da activity `ConnectedUserListActivity`. Ela está associada à classe `ConnectedUserUpdateRunnable` que atualiza as informações no servidor. A visualização dos detalhes de cada contato e a posição do mesmo no mapa é efetuada através da classe `ConnectedUserDetailActivity`, que busca as informações no servidor em uma nova *thread* com uma instância de `ConnectedUserDetailRunnable`.

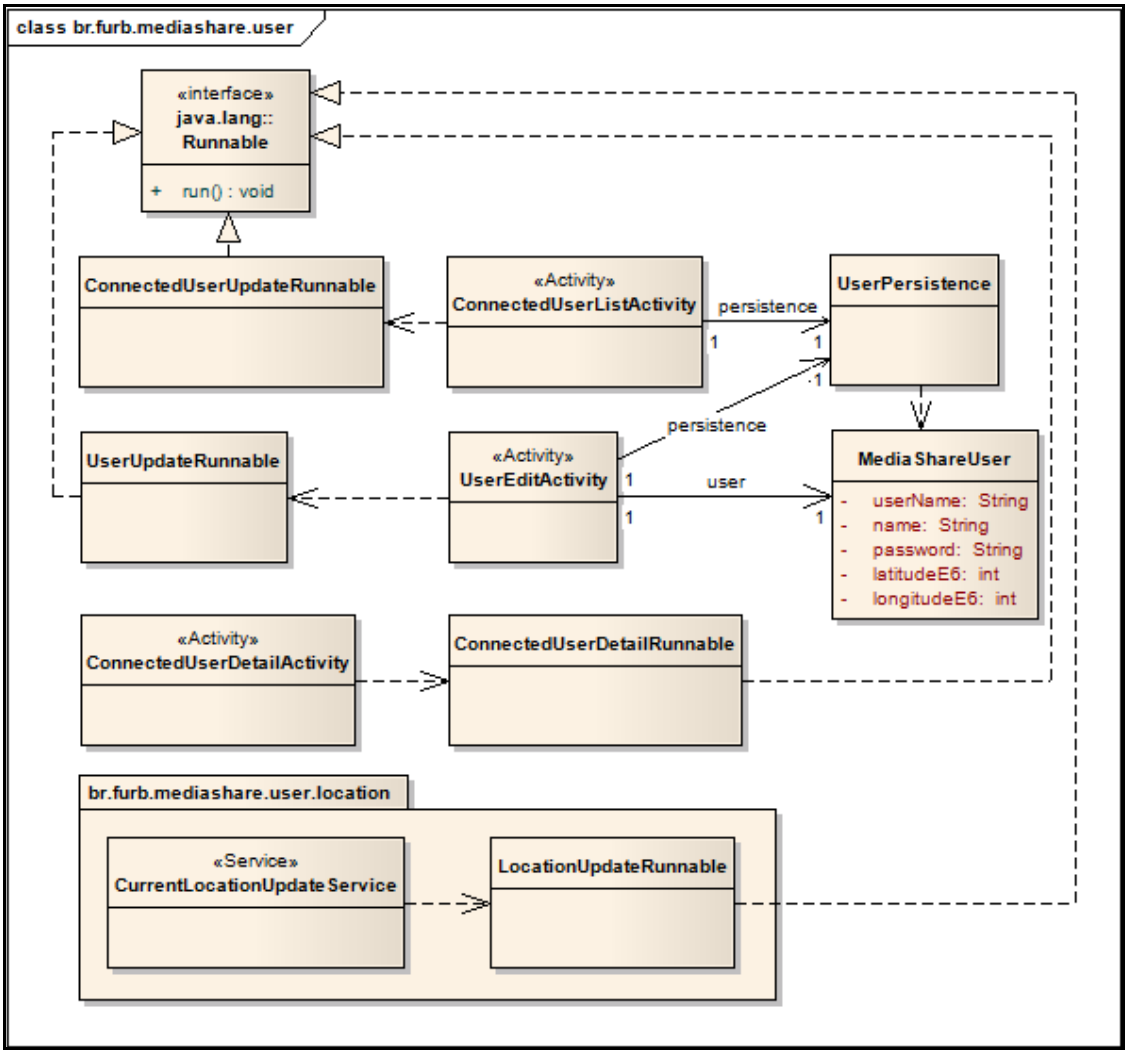



Figura 5 - Pacote `br.furb.mediashare.user`

### 3.3.3.2 Pacote `br.furb.mediashare.media`

Os componentes relativos às mídias manipuladas pelo aplicativo foram implementadas no pacote `br.furb.mediashare.media`. A Figura 6 contém o diagrama de classes do pacote `br.furb.mediashare.media`.

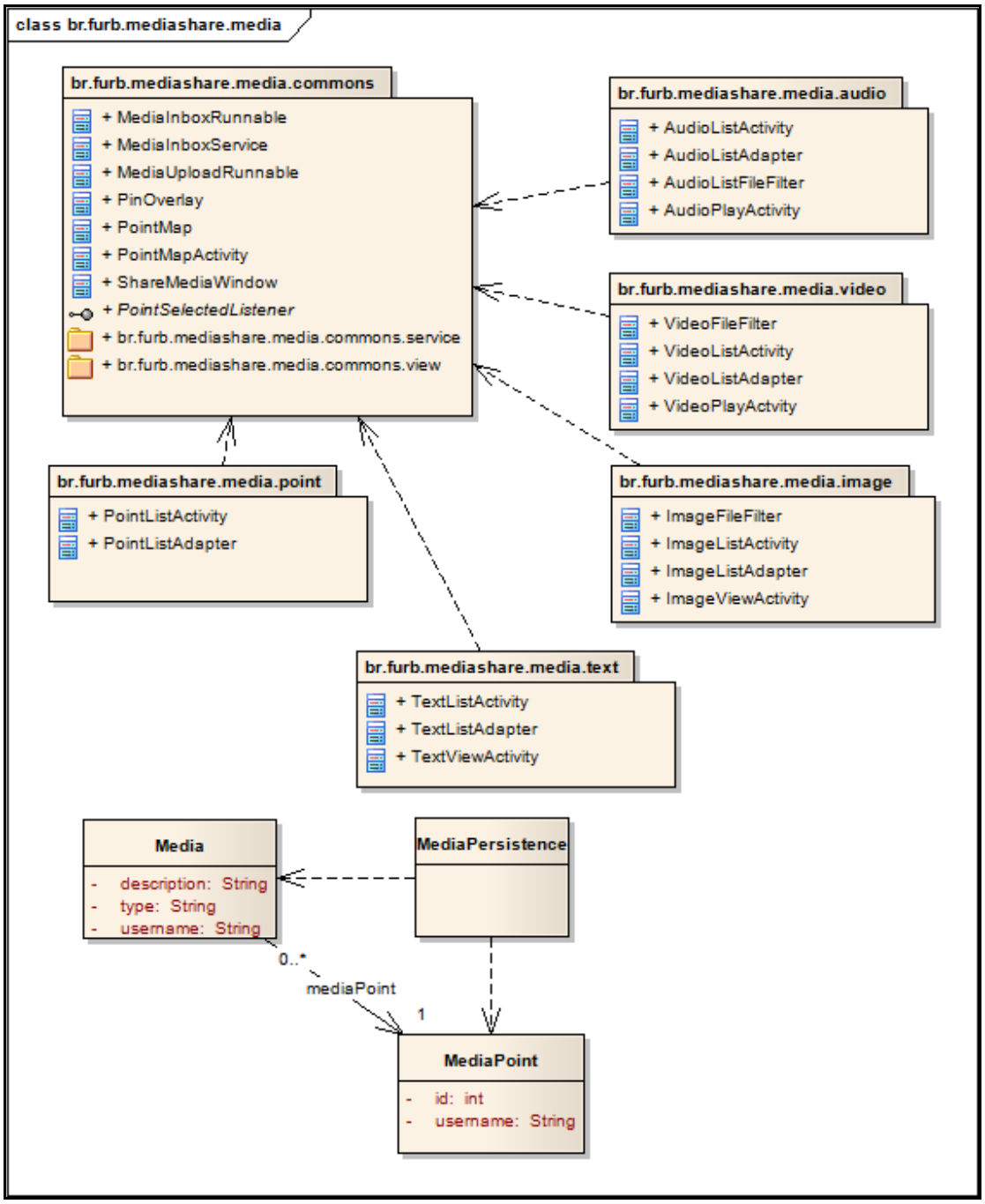



Figura 6 - Pacote `br.furb.mediashare.media`

A classe `Media` representa uma mídia existente na aplicação. Para mídias armazenadas em arquivos, o atributo `description` contém o caminho completo e o nome do arquivo no dispositivo. Para mídias do tipo texto, o atributo `description` contém os caracteres do texto em questão. As instâncias de `Media` estão associadas a uma instância de `MediaPoint`. Essa última representa uma determinada coordenada geográfica e pode ser associada a uma ou nenhuma mídia. Tanto as informações de mídia quanto as informações de coordenadas são persistidas no banco de dados local do aplicativo pela classe `MediaPersistence`. Essa classe é utilizada por todas as activities de mídias.

Com a finalidade de facilitar o desenvolvimento e as modificações do aplicativo cliente, algumas funcionalidades foram organizadas em classes que são utilizadas pelas outras classes de manipulação de mídias. Essas classes foram implementadas no pacote `br.furb.mediashare.media.commons`. Esse pacote contém também classes que atualizam informações relativas a todas as *activities* de mídias. A Figura 7 apresenta o diagrama de classes do pacote `br.furb.mediashare.media.commons`.

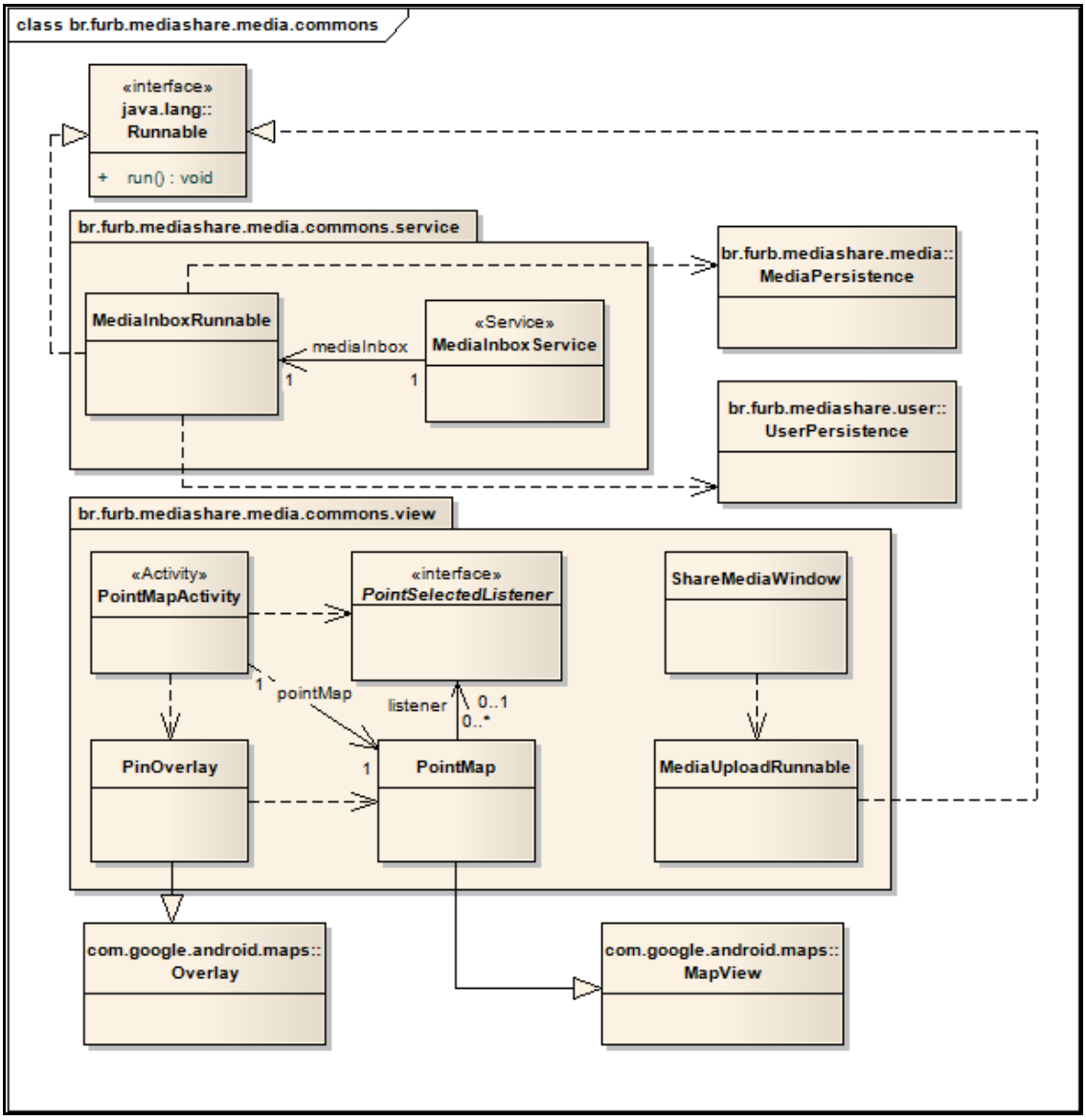



Figura 7 - Pacote `br.furb.mediashare.media.commons`

A classe `MediaInboxService` implementa o componente de *download* das mídias disponíveis no servidor. A mesma contém uma referência para `MediaInboxRunnable`, que efetua o *download* das mídias disponíveis em uma nova *thread* a cada intervalo de tempo definido no arquivo `values/strings.xml`. As mídias recebidas que são persistidas em arquivos (do tipo áudio, vídeo ou imagem) são gravadas em arquivos no dispositivo por uma instância de `MediaFilePersistence`.

As classes de interface gráfica que são reaproveitadas em diversas outras classes foram implementadas no pacote `br.furb.mediashare.media.commons.view`. A classe `ShareMediaDialog` monta uma janela em uma *activity*, através da qual é possível selecionar o destinatário do compartilhamento de uma mídia e compartilhá-la através do servidor. O envio para o servidor é efetuado através de uma instância de `MediaUploadRunnable`, que envia o arquivo ao servidor em uma nova *thread*.

O componente de seleção ou visualização pontos no mapa também foi implementado no pacote `br.furb.mediashare.media.commons.view`, através da *activity* `PointMapActivity`. Esta classe contém uma referência para `PointMap`, que é responsável pela exibição dos mapas e captura do ponto selecionado. Cada ponto no mapa é desenhado através de uma instância de `PinOverlay`, classe que estende `Overlay`. Para a notificação de que um ponto foi selecionado, foi criada a *interface* `PointSelectedListener`. O atributo `listener` da classe `PointMap` é ativado sempre que um novo ponto é selecionado no mapa.

As classes de manipulação de mídias foram separadas em pacotes, de modo que cada pacote contém as classes relativas a manipulação de um tipo de mídia. A Figura 8 contém o diagrama de classes do pacote `br.furb.mediashare.media.audio`, que efetua a manipulação de arquivos de áudio.

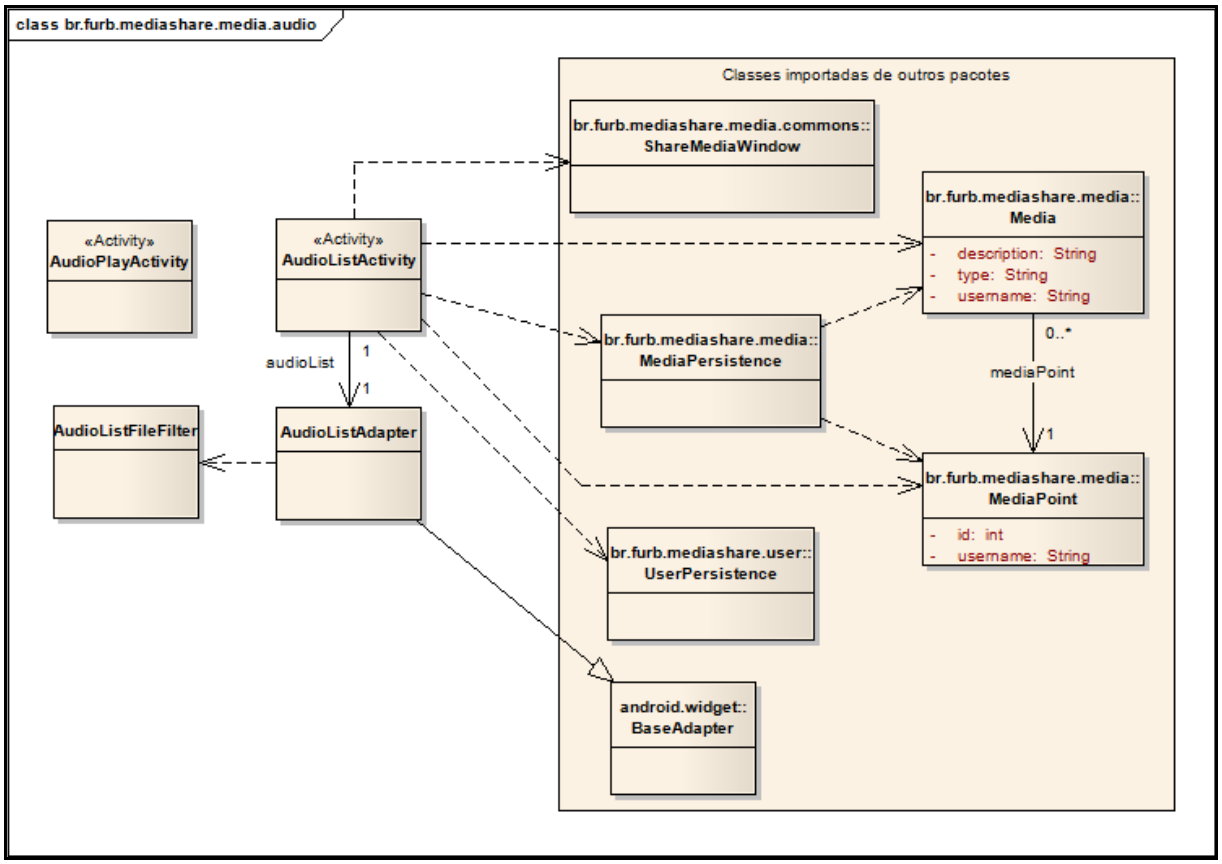



Figura 8 - Pacote `br.furb.mediashare.media.audio`

A classe `AudioListActivity` implementa o componente de listagem e associação de coordenadas a arquivos de áudio. A listagem é feita através de instâncias de `AudioListAdapter`. Essa classe estende `BaseAdapter`, que e é utilizada em activities de listas de informações. O filtro por tipo de arquivo é efetuado por uma instância de `AudioListFileFilter`. As coordenadas de cada arquivo de imagem são recuperadas ou persistidas através de uma instância de `MediaPersistence`. Já as informações do usuário são obtidas através de uma instância de `UserPersistence`. Para o compartilhamento das mídias, são utilizadas instâncias de `ShareMediaDialog`. A reprodução dos arquivos de áudio é efetuada através da ativação da classe `AudioPlayActivity`.

Similar ao pacote de classes de manipulação de áudio, o pacote `br.furb.mediashare.media.video` contém as classes de manipulação de vídeos. A Figura 9 contém o diagrama de classes do pacote `br.furb.mediashare.media.video`.

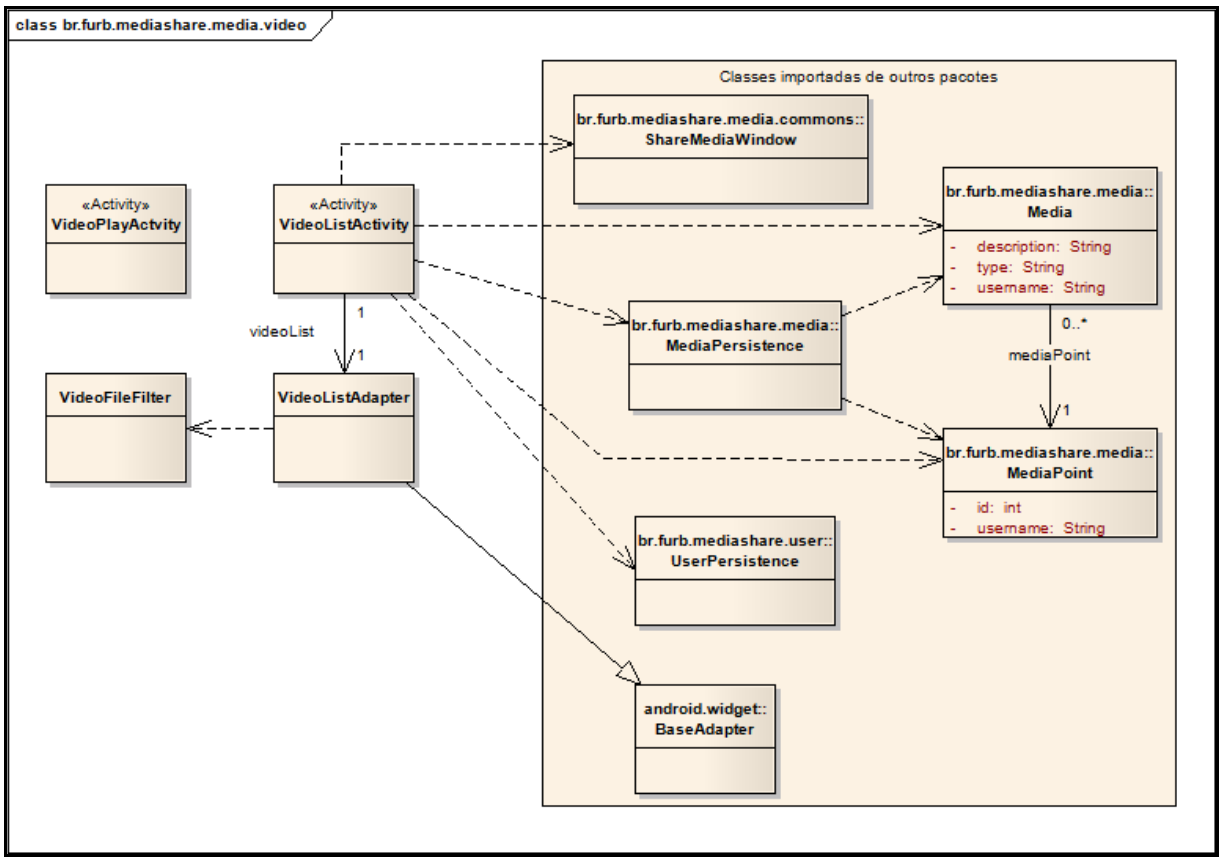



Figura 9 - Pacote `br.furb.mediashare.media.video`

A classe `VideoListActivity` implementa o componente de listagem e associação de coordenadas aos vídeos disponíveis. A listagem dos vídeos é efetuada através de uma instância de `VideoListAdapter`, classe que utiliza `VideoFileFilter` para filtrar os arquivos. A persistência e obtenção de informações é efetuada através de instâncias de `MediaPersistence` e `UserPersistence`. O compartilhamento das mídias é efetuado através

de instâncias de `ShareMediaDialog`. Os vídeos são reproduzidos através da *activity* `VideoPlayActvity`.

Seguindo o padrão de pacotes de manipulação de áudio e vídeo, as classes de manipulação de imagens foram agrupadas no pacote `br.furb.mediashare.media.image`. A Figura 10 contém o diagrama de classes do pacote `br.furb.mediashare.media.image`.

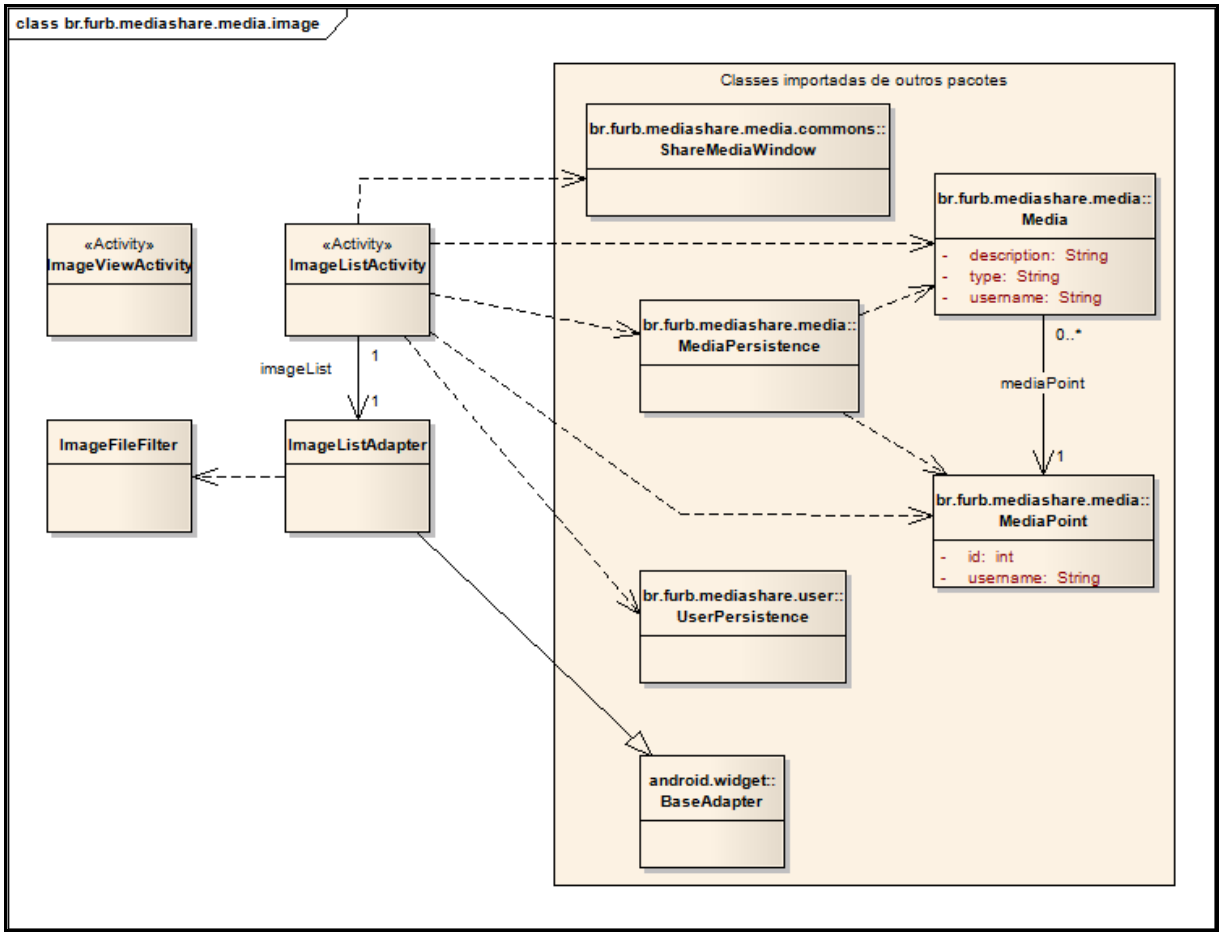



Figura 10 - Pacote `br.furb.mediashare.media.image`

A classe `ImageListActivity` implementa a listagem e associação de arquivos de imagens. Os arquivos são obtidos através de uma instância de `ImageListAdapter` que efetua o filtro de arquivos através da classe `ImageFileFilter`. A persistência e obtenção das informações é efetuada através de instâncias de `MediaPersistence` e `UserPersistence`. O compartilhamento das mídias é efetuado através de instâncias de `ShareMediaDialog`. A exibição de imagens é feita através da ativação de `ImageViewActivity`.

As classes para manipulação de textos foram implementadas no pacote `br.furb.mediashare.media.text`. Este pacote se diferencia do padrão encontrado nos demais pacotes de classes de mídias por não apresentar classe de recuperação de arquivos. O diagrama de classes do pacote `br.furb.mediashare.media.text` é apresentado na Figura 11.

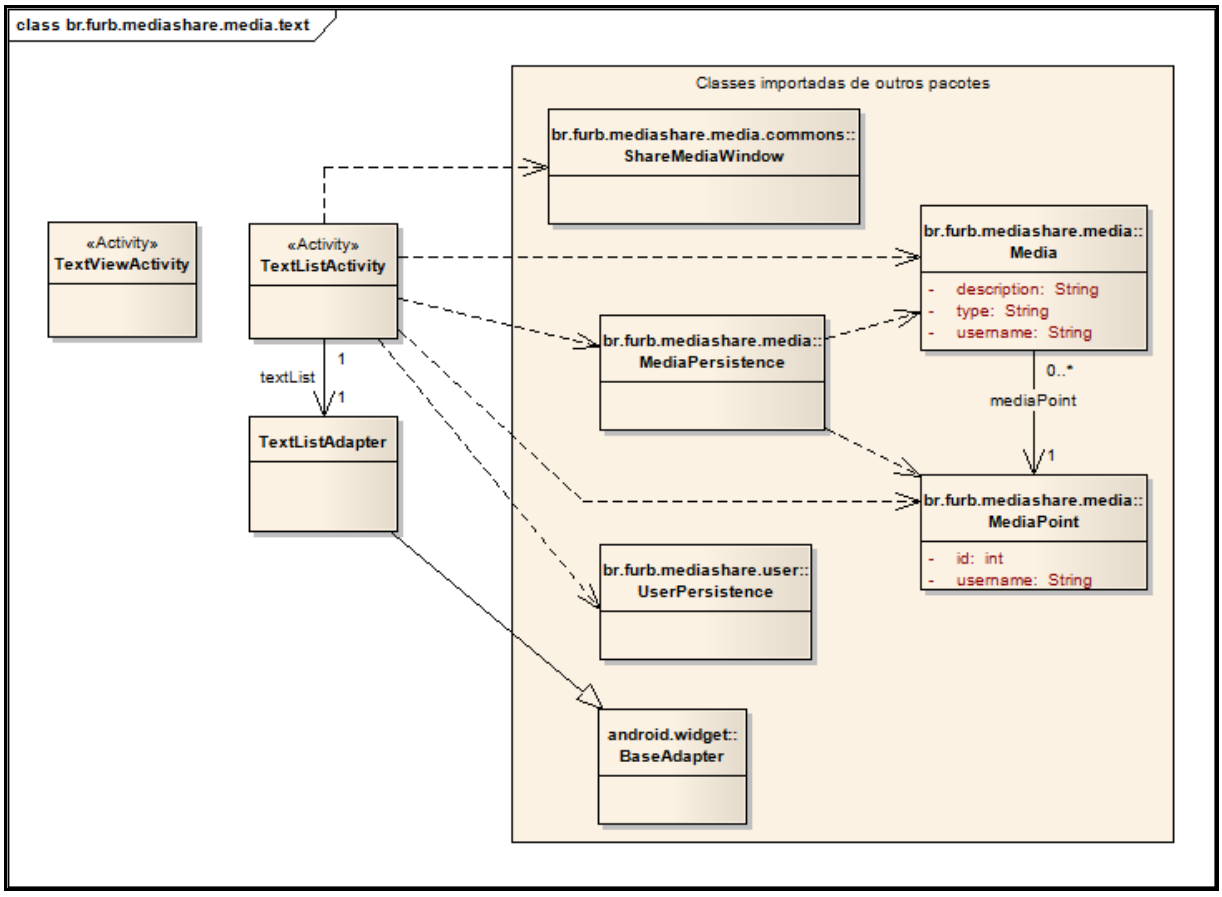



Figura 11 - Pacote `br.furb.mediashare.media.text`

A classe `TextListActivity` efetua a recuperação dos textos através da classe `MediaPersistence`. Esses textos são repassados a instâncias de `TextListAdapter` para exibição na tela. A persistência de novos textos também é efetuada através da classe `MediaPersistence`. O compartilhamento de textos é efetuado através de instâncias de ShareMediaDialog. A visualização e criação de textos é efetuado através da ativação de `TextViewActivity`.

As classes de manipulação de pontos de interesse (coordenadas geográficas) foram organizadas no pacote `br.furb.mediashare.media.point`. A Figura 12 apresenta o diagrama de classes desse pacote.

A listagem de pontos é feita através da *activity* `PointListActivity`, que busca os pontos disponíveis através de uma instância da classe `MediaPersistence` e repassa a uma instância da classe `PointListAdapter`. A classe `PointListActivity` também efetua a recuperação de informações do usuário através de uma instância de `UserPersistence`. Diferente dos outros pacotes de mídia, o pacote de classes de ponto não possui classe para visualização dos pontos. A visualização dos pontos de interesse no mapa é feita através da ativação de `PointMapActivity`, do pacote `br.furb.mediashare.media.commons`.

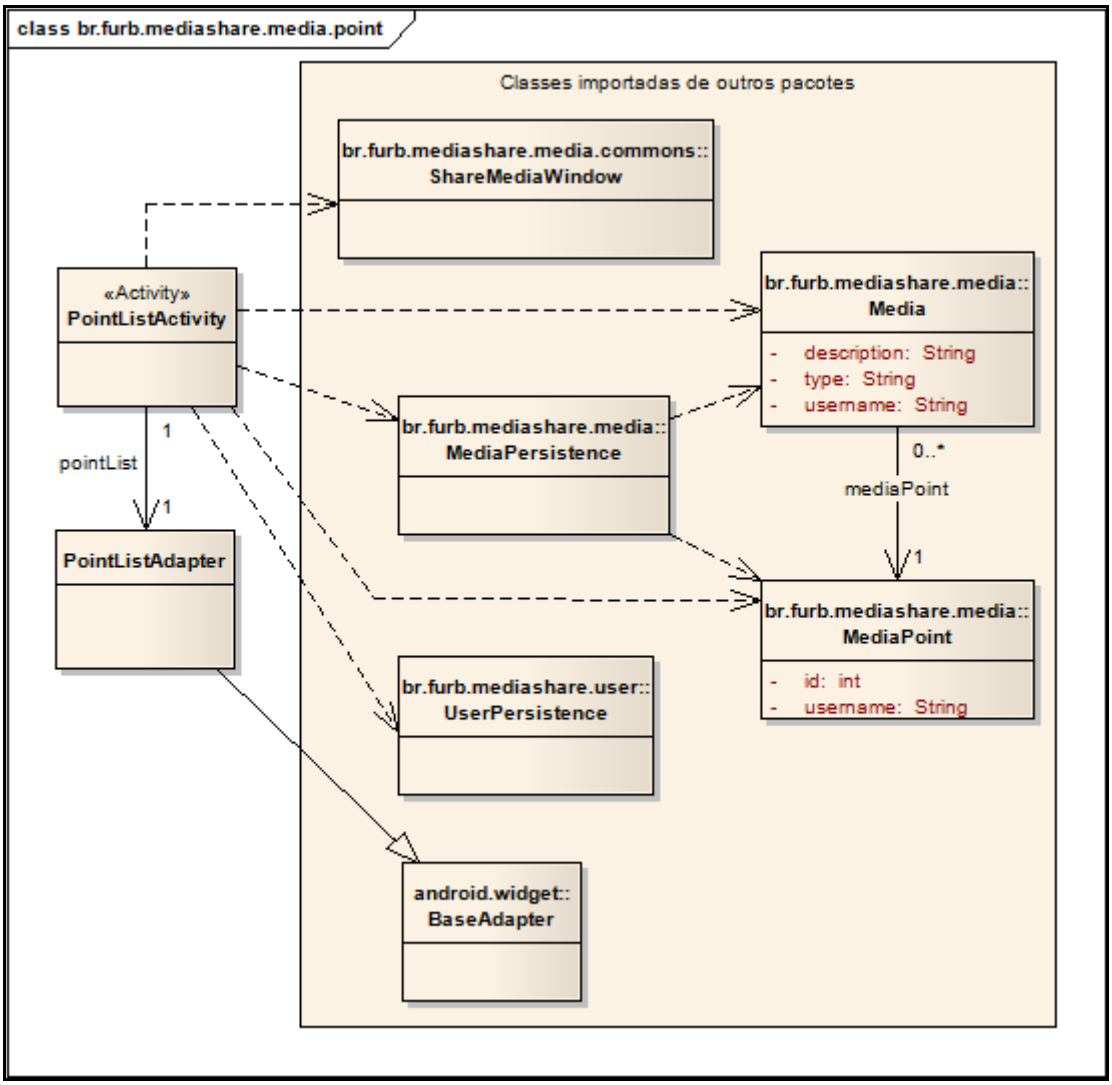



Figura 12 - Pacote `br.furb.mediashare.media.point`

### 3.3.4 Classes do servidor

Partindo da definição dos objetivos deste trabalho, cujo foco é a criação de um aplicativo para a plataforma Android, o servidor foi desenvolvido com o propósito de suprir as necessidades apresentadas no aplicativo cliente. As classes do servidor foram desenvolvidas no pacote `br.furb.mediashare.server`. Cada serviço necessário no servidor é implementado por um *servlet* HTTP que processa as requisições e efetua o retorno em formato XML ou formato binário dos arquivos de mídia. No diagrama, essas classes são apresentadas com o esteriótipo de *servlet*. A Figura 13 apresenta o diagrama de classes do servidor.

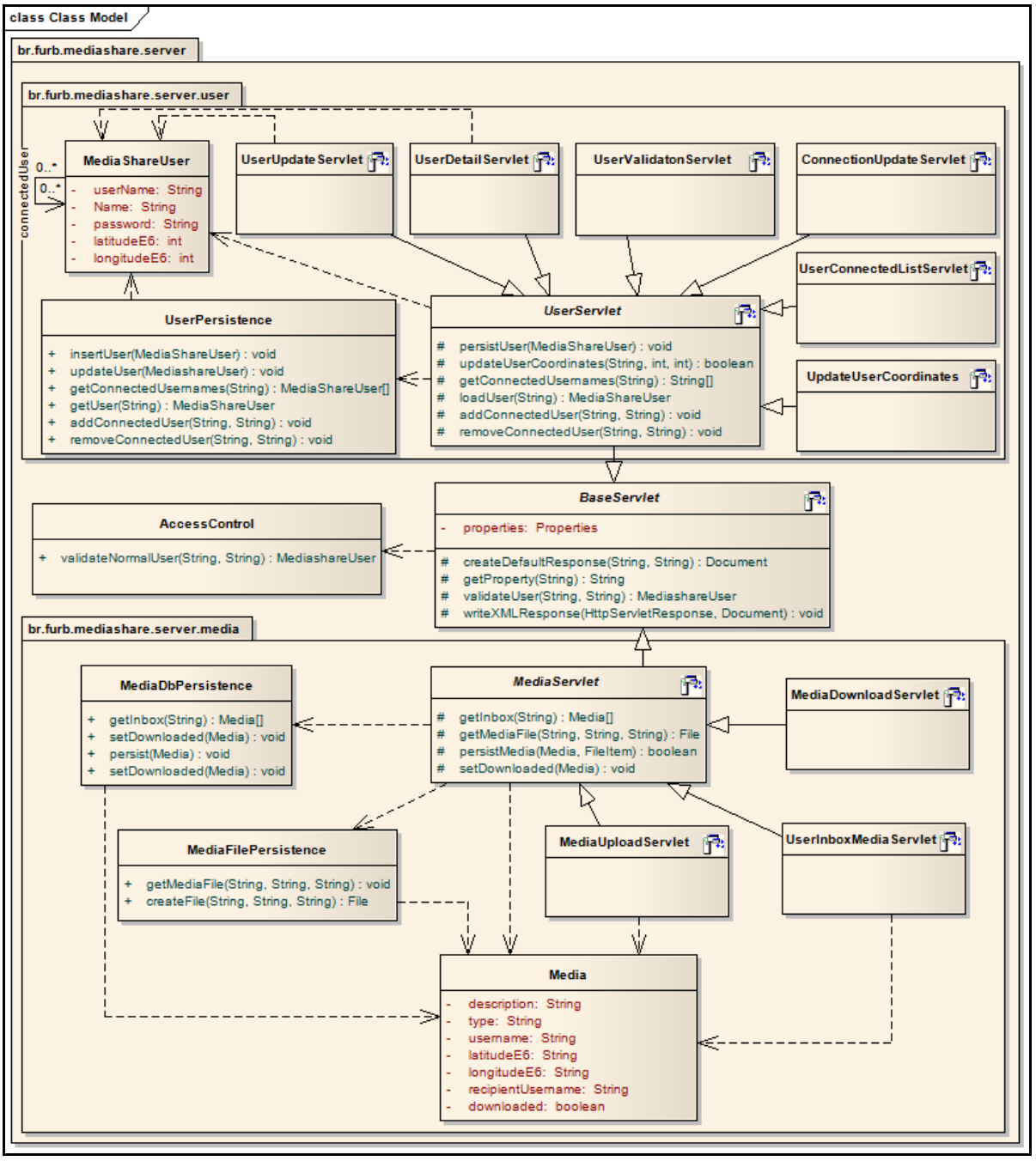



Figura 13 - Classes do servidor

A classe abstrata `BaseServlet` estende `javax.servlet.http.HttpServlet` e contém os métodos básicos para a correta execução dos serviços disponibilizados. É responsável por efetuar as inicializações necessárias e carregar as propriedades da aplicação. Essas propriedades representam configurações do sistema como diretórios de armazenamento de arquivos, por exemplo. A classe `UserServlet` encapsula os métodos básicos de serviços de usuários. Esses métodos invocam instâncias de `UserPersistence` para obter ou persistir as informações de usuário. As informações de cada usuário são armazenadas em instâncias de `MediaShareUser`. Essas instâncias são persistidas em um banco de dados do servidor. A

classe `UserUpdateServlet` efetua a criação ou atualização de informações de usuário no servidor. Os detalhes de cada usuário, incluindo suas coordenadas geográficas, são listados através da classe `UserDetailServlet`. A validação de usuário e senha de aplicação é verificada através da classe `UserValidatonServlet`. Para atualizar as informações de conexões entre usuários, é utilizada a classe `ConnectionUpdateServlet`. Para obter a lista de usuários conectados é necessário efetuar uma requisição a classe `UserConnectedListServlet`.

Os serviços de mídias estão agrupados no pacote `br.furb.mediashare.server.media`. A classe `MediaServlet` encapsula os métodos básicos das classes de mídias. Essa classe efetua a persistência das informações de mídias através de instâncias de `MediaDbPersistence`. Para a persistência dos arquivos de mídias existentes em alguns formatos, são utilizadas instâncias de `MediaFilePersistence`. Cada mídia existente na aplicação é representada por uma instância de `Media`. Essa classe contém as informações da mídia, o usuário que a enviou, o usuário que deverá recebê-la e as corrdenadas geográficas da mídia. Para as mídias armazenadas em arquivos, o campo `description` contém o caminho completo do arquivo. Já para as mídias do tipo texto, o campo `description` contém os caracteres do texto. O envio de mídias para o servidor é efetuado através da classe `MediaUploadServlet`, que efetua a persistência das informações através dos métodos da classe pai. Para verificar se há mídias disponíveis para *download* é necessário enviar uma requisição para `UserInboxMediaServlet`. Esse *servlet* retorna um arquivo XML contendo as informações de mídias disponíveis para o usuário. A partir de cada mídia contida nesse XML é possível enviar requisições para `MediaDownloadServlet` e obter os arquivos de cada mídia, caso necessário.

### 3.3.5 Diagramas de sequências

Os diagramas de sequência tem como finalidade demonstrar a troca de mensagens entre as classes criadas, facilitando a implementação dos casos de uso. Nesta seção são apresentados os diagramas de sequência para os casos de uso `Georreferenciar mídia` (`UC07`) e `Compartilhar Mídia` (`UC08`).

### 3.3.5.1 Diagrama de sequência georreferenciar mídia

O caso de uso `Georreferenciar mídia` (`UC07`) parte da seleção de uma mídia. Assim, a primeira mensagem da sequência ocorre quando o usuário seleciona uma mídia e em seguida a opção para definir coordenada geográfica. Através da *activity* `PointMapActivity`, o usuário pode navegar por um mapa e em seguida marcar a localização desejada. A conversão das coordenadas da tela (ponto selecionado) em coordenadas geográficas é feita pela classe `PointMap`. A Figura 14 contém o diagrama de sequência do caso de uso `Georreferenciar mídia` (`UC07`).

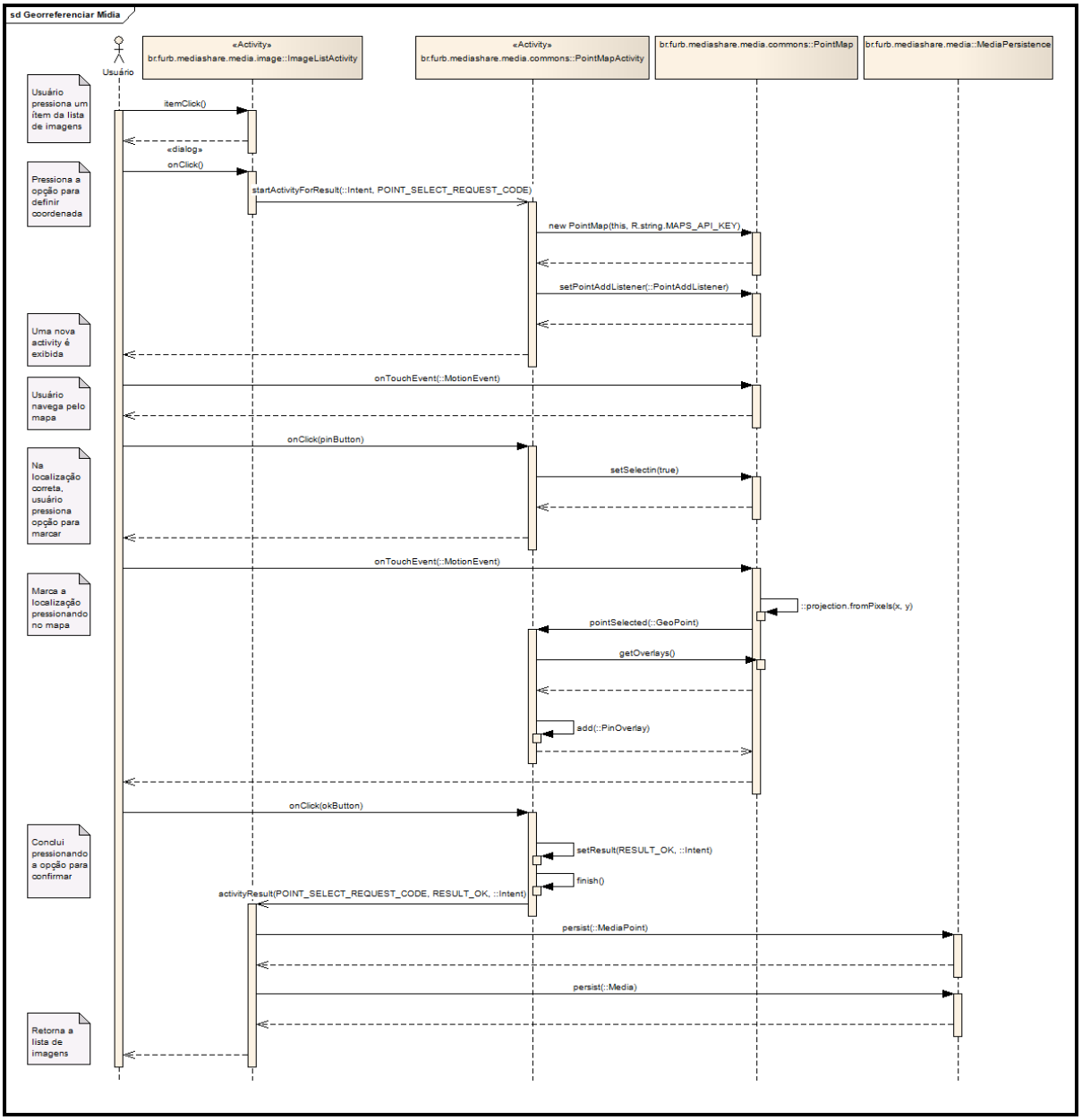



Figura 14 - Diagrama de sequência `Georreferenciar mídia`

#### 3.3.5.2 Diagrama de sequência compartilhar mídia

O caso de uso `Compartilhar Mídia` (`UC08`) define a sequência de passos para o compartilhamento de uma mídia através do servidor. Inicia através de uma instância de `ShareMediaDialog`. Essa classe representa uma janela na qual o usuário informa o destinatário do compartilhamento e confirma a operação. Após a confirmação, o envio das informações ao servidor é efetuado através de uma instância de `MediaUploadRunnable`. Com a finalidade de permitir atualizações da tela enquanto as informações são enviadas, o envio ocorre em uma nova *thread* de execução. A classe `MediaUploadRunnable` efetua uma requisição através de uma requisição HTTP `Post`. Essa requisição recebe os parâmetros, incluindo a mídia a ser compartilhada, através do padrão *multipart* e retorna um XML contendo o resultado do processamento. No servidor, a requisição é atendida pelo *servlet* `MediaUploadServlet`. Mídias do tipo áudio, vídeo ou imagem são persistidas em arquivos físicos e posteriormente referenciadas por um registro no banco de dados. A Figura 15 apresenta o diagrama de sequência do caso de uso `Compartilhar Mídia` (`UC08`).

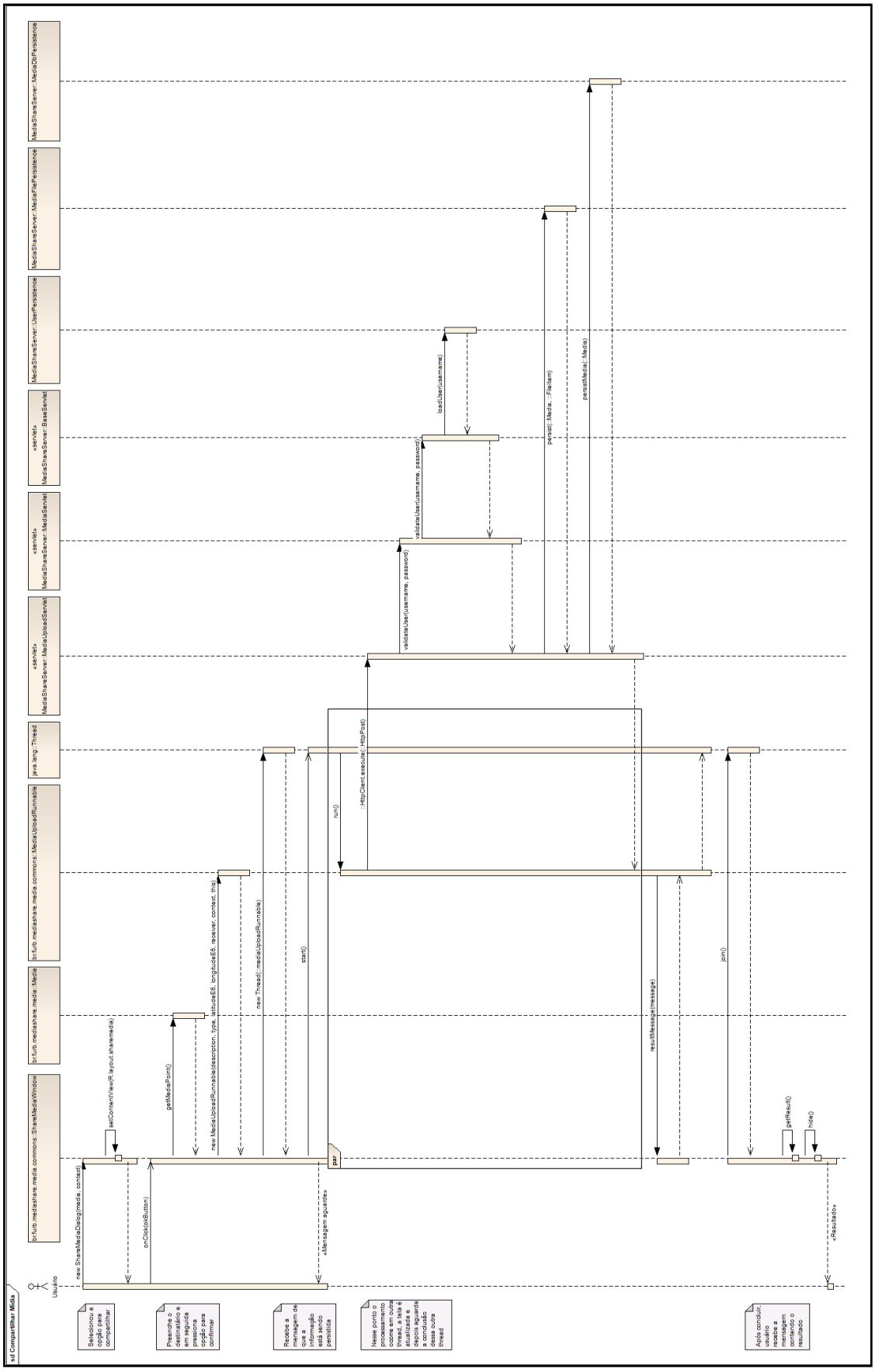



Figura 15 – Diagrama de sequência `Compartilhar mídia`

### 3.3.6 Bancos de dados

As informações básicas do sistema foram persistidas em bancos de dados no cliente e no servidor. O aplicativo cliente utiliza bibliotecas da plataforma Android para criar e manipular um banco de dados SQLite. Esse banco é armazenado no dispositivo móvel na área da memória dedicada ao armazenamento de arquivos da aplicação. Dessa forma, as informações não podem ser acessadas por outro aplicativo sendo executado no dispositivo. A Figura 16 apresenta o MER do banco de dados criado no aplicativo cliente.

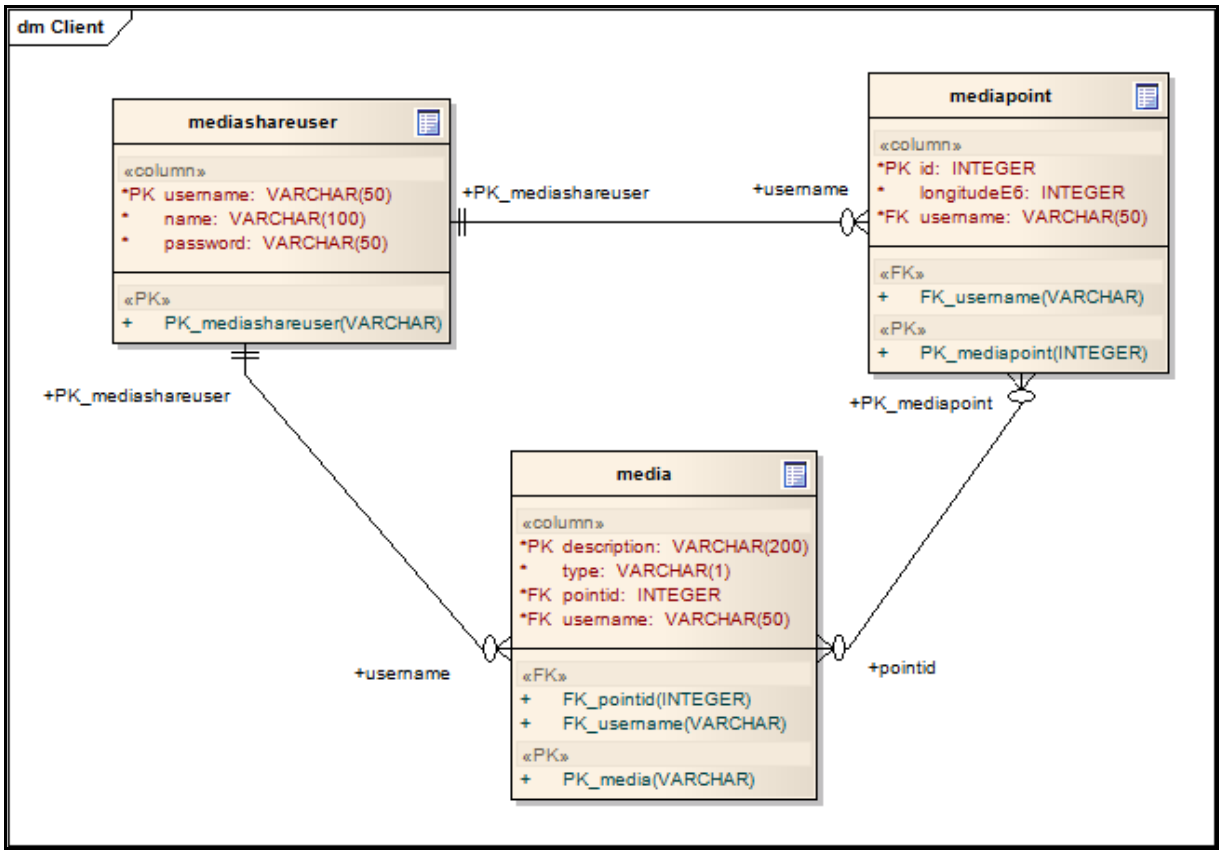



Figura 16 - MER do aplicativo Cliente

A tabela `mediashareuser` contém as informações do usuário que está executando o aplicativo. Essa tabela é atualizada a cada novo login, recebendo as informações do novo usuário. As informações contidas nessa tabela são persistidas e recuperadas pela classe `UserPersistence`. Essa é responsável por converter as informações de uma instância de `MediashareUser` em informações a serem persistidas e efetuar o processo contrário, conforme necessidade. A tabela `mediapoint` contém todos os pontos existentes no dispositivo, tanto criados localmente quanto recebidos de outros usuários. Ao georreferenciar uma mídia, o usuário pode selecionar um ponto já existente ou criar um novo. Como cada mídia existente no sistema deve possuir georreferenciamento, cada mídia estará associada a

exatamente um ponto. Essa relação é mantida também no banco de dados, na forma das chaves que ligam as tabelas `mediapoint` e `media`. Essa última contém as informações de mídias criadas localmente ou recebidas de outros usuários. A classe `MediaPersistence` é responsável por converter instâncias de `MediaPoint` e `Media` em informações a serem persistidas no banco de dados e efetuar o processo contrário, conforme necessidade. Os arquivos de áudio, vídeo e imagens não são persistidos no banco de dados, mas nos diretórios compartilhados do dispositivo. Apenas uma referência para o arquivo é armazenada na tabela `media`, através do campo `description`.

O servidor utiliza um banco de dados MySQL para persistir e recuperar as informações relativas as mídias e aos usuários. Da mesma forma que ocorre no aplicativo cliente, os arquivos de áudio, vídeo e imagem são armazenados em direórios externos, a serem configurados nas propriedades do aplicativo. A Figura 17 contém o MER do banco de dados do aplicativo servidor.

A tabela `mediashareuser` contém as informações de todos os usuários do sistema. A cada operação efetuada a combinação nome de usuário e senha do usuário é verificada nessa tabela. Caso não exista registro correspondente, a operação não é executada. A classe `br.furb.mediashare.server.user.UserPersistence` é responsável por armazenar e recuperar as informações da tabela `mediashareuser`. É responsável também por acessar os registros da tabela `mediashareuserconection`. Essa tabela é resultado da relação de múltiplos registros de para múltiplos `mediashareuser` registros da mesma tabela. Dessa forma, a tabela `mediashareuserconection` armazena as conexões entre os usuários. A tabela `media` contém todas as informações de mídias enviadas para o servidor, inclusive a coordenada geográfica de cada uma das mídias. A classe `br.furb.mediashare.server.media.MediaDBPersistence` é responsável por persistir e recuperar as informações de mídias existentes na tabela `media`.

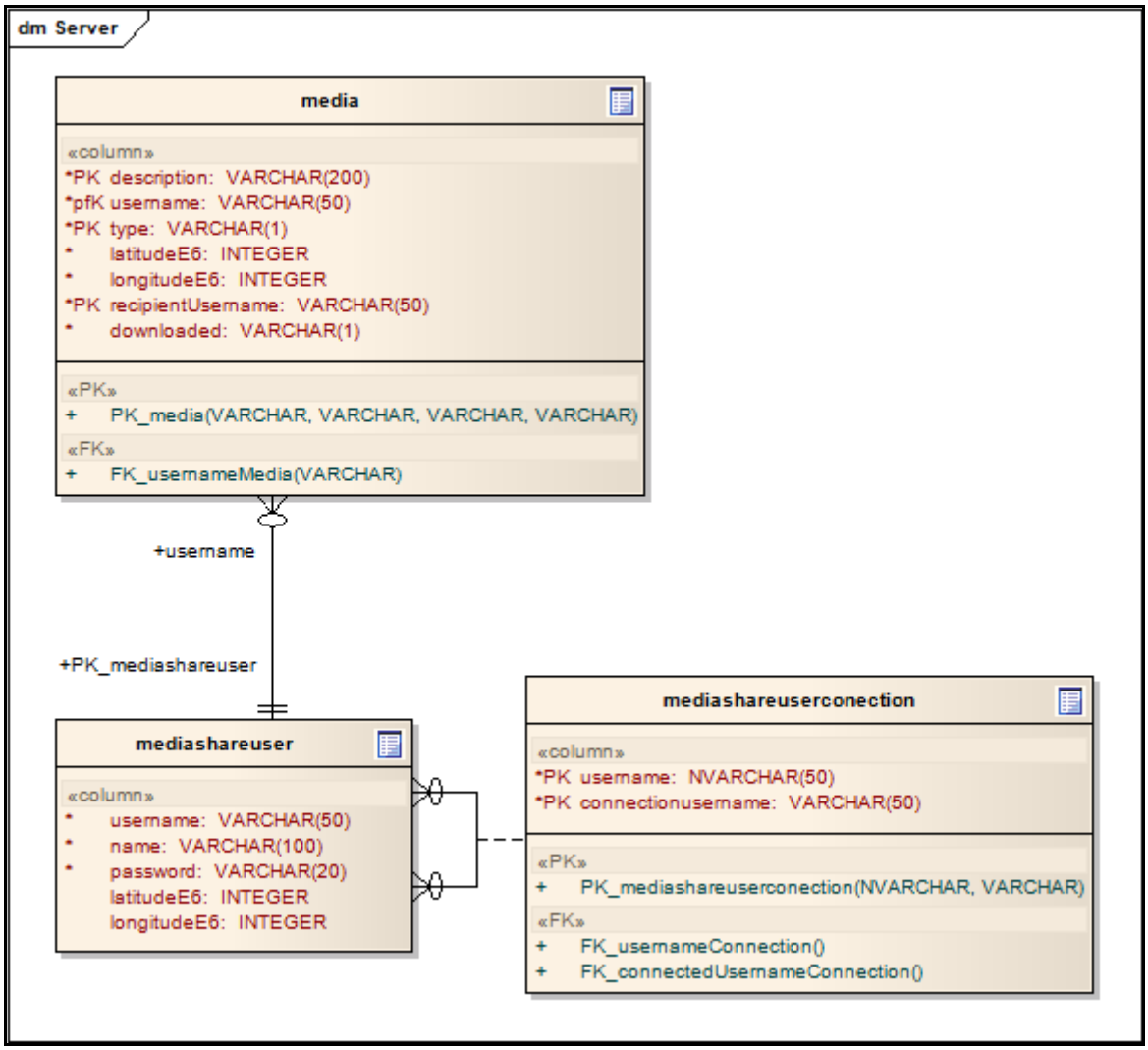



Figura 17 - MER do aplicativo servidor

## 3.4 IMPLEMENTAÇÃO

A seguir são mostradas as técnicas e ferramentas utilizadas e a operacionalidade da implementação.

### 3.4.1 Técnicas e ferramentas utilizadas

O desenvolvimento do sistema proposto foi efetuado em duas partes, de forma paralela. As funcionalidades propostas foram desenvolvidas em um aplicativo cliente, a ser

executado na plataforma Android. Devido a necessidade, um servidor foi desenvolvido para atender as funcionalidades apresentadas.

Para o desenvolvimento da aplicação cliente foi utilizado o ambiente de desenvolvimento Eclipse, versão 3.6.1 (Helios) combinado com o Android SDK, versão 0.9.7. Esse ambiente foi executado sobre o *Java Development Kit* (JDK) versão 1.6.0_20 de 64 bits, para Ubuntu Linux. Foi utilizado o nível 8 da API Android, aliada a biblioteca de mapas da Google, também do nível 8. Assim, o aplicativo desenvolvido é compatível com a versão 2.2 do sistema operacional Android ou versões superiores compatíveis. Para adicionar a capacidade de efetuar requisições *multipart* via HTTP foi adicionada a biblioteca Java `HTTPMime`, versão 4.1 alfa 2. Essa biblioteca é fornecida pela fundação Apache. Os testes do aplicativo cliente foram efetuados no simulador do próprio Android SDK, através de um dispositivo virtual. Para integrar a IDE Eclipse ao Android SDK, foi utilizado o *plugin* ADT.

O servidor foi desenvolvido com base na plataforma JEE, através do ambiente de desenvolvimento Eclipse, versão 3.6.1 (Helios). Como servidor de aplicação foi utilizado o servidor Apache Tomcat, versão 7.0.2. A persistência das informações é efetuada através de um servidor MySQL, versão 5.1.49-1ubuntu8. O acesso ao servidor é efetuado através de um *driver* JDBC para MySQL, versão 5.1.13. Os testes do servidor foram efetuados através do próprio Apache Tomcat.

### 3.4.2 Arquivo AndroidManifest.xml

O arquivo `AndroidManifest.xml` contém informações básicas das aplicações desenvolvidas para a plataforma Android. Algumas informações são obrigatórias e existem em todas as aplicações, como o pacote, o código da versão e o nome da versão. Além dessas, outras são opcionais e existem conforme as necessidades de cada aplicação. Para o correto funcionamento da aplicação Mediashare, foram necessárias as definições de permissões e bibliotecas utilizadas. O Quadro 12 apresenta o primeiro trecho do arquivo `AndroidManifest.xml` do aplicativo Mediashare.

O atributo `package="br.furb.mediashare"` indica o pacote padrão da aplicação. Esse pacote é considerado a raíz da aplicação e usado para definição de *namespaces* das aplicações. Os atributos `android:versionCode="1"` e `android:versionName="1.0"` indicam informações da versão da aplicação. A *tag* `<uses-permission`

`android:name="android.permission.ACCESS_FINE_LOCATION" />` solicita permissão para acesso a localização do usuário que está executando a aplicação. Essa permissão é necessária para que o aplicativo Mediashare possa atualizar as coordenadas do usuário no cliente e no servidor. A *tag* `<uses-permission android:name="android.permission.INTERNET" />` solicita permissão para abertura de conexões com a internet. Essa permissão é necessária para que o aplicativo possa trocar informações com o servidor via HTTP. A *tag* `<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />` solicita permissão para escrita de arquivos no local de armazenagem de arquivos externos. Essa permissão é necessária para que o aplicativo possa escrever os arquivos recebidos do servidor no local de armazenagem padrão de cada tipo de mídia.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="br.furb.mediashare" android:versionCode="1"
    android:versionName="1.0">
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <application android:icon="@drawable/appicon"
        android:label="@string/app_name">
        <uses-library android:name="com.google.android.maps" />
    <!-- ... -->
```

Quadro 12 - Primeiro trecho do arquivo `AndroidManifest.xml`

As informações e componentes da aplicação são informadas dentro da *tag* `application`. Dentro dela é definida a *tag* `<uses-library android:name="com.google.android.maps" />` que indica a utilização da biblioteca de mapas da Google.

Além das informações básicas da aplicação, o arquivo `AndroidManifest.xml` deve conter também a declaração de cada componente da aplicação. O aplicativo Mediashare contém *activities* e *services*, que foram declarados com a inclusão das *intents* as quais respondem. O Quadro 13 apresenta a declaração de uma *activity* e um *service*.

A *activity* foi declarada em uma *tag* com o mesmo nome, na qual o atributo `android:name=".LauncherActivity"` indica a classe que contém a *activity* em questão. Inicia com ponto para abstrair o pacote padrão, simplificando a visualização. A *tag* `intent-filter` contém uma *intent* a qual essa *activity* responde. Dentro dessa, a *tag* `action` indica qual a ação (*string*) que deve ser enviada ao sistema para a ativação desse componente. Nesse caso, o nome `android.intent.action.MAIN` indica que a activity em questão é a inicial de uma aplicação. Da mesma forma, o *service* é declarado em uma *tag* de mesmo nome, também

contendo o indicativo da classe e de qual intent o ativa.

```
<activity android:name=".LauncherActivity" android:label="@string/Launcher_Label">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
    <intent-filter>
        <action android:name="br.furb.mediashare.intent.action.launchApp" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</activity>
<service android:name=".media.commons.service.UserInboxService">
    <intent-filter>
        <action android:name="br.furb.mediashare.intent.action.UserInbox" />
        <category android:name="android.intent.category.DEFAULT" />
    </intent-filter>
</service>
```

Quadro 13 - Segundo trecho do arquivo `AndroidManifest.xml`

### 3.4.3 Arquivos auxiliares

Além do código Java e do arquivo `AndroidManifest.xml`, a aplicação desenvolvida é composta de outros arquivos auxiliares. Esses arquivos definem leiautes de telas da aplicação ou constantes de texto.

A definição e organização dos componentes das telas da aplicação desenvolvida pode ser feita de duas maneiras. Através do código Java, os componentes da interface podem ser instanciados e associados através de métodos da API Android. Por arquivos XML, os componentes podem ser definidos em *tags* e depois acessados. No aplicativo Mediashare, a maior parte das telas é definida em XML e depois importada para as classes Java. Nessas últimas, os objetos são preenchidos com valores e, em algumas situações, associados com outros objetos instanciados no código Java. O Quadro 14 apresenta o arquivo `layout/imagedetail.xml`, que define a interface da tela de visualização dos detalhes de uma imagem.

As *tags* `LinearLayout`, `TableLayout` e `TableRow` declaram componentes de leiaute, capazes de receber e organizar outros componentes. As *tags* `TextView` declaram componentes de texto, usados para apresentar informações textuais ao usuário. Essas informações podem ser definidas de forma fixa, ou atribuídas em tempo de execução. As propriedades de todos os componentes podem ser alteradas via código Java, através da API da plataforma Android. O Quadro 15 apresenta um trecho do código da classe `ImageViewActivity`. Nesse código o

arquivo `layout/imagedetail.xml` é importado e definido como conteúdo da *activity* em questão. Em seguida, alguns componentes de texto são preenchidos com informações e um novo componente para visualização de mapa é adicionado.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent" android:layout_height="fill_parent"
    android:orientation="vertical">
    <LinearLayout android:layout_weight="1"
        android:layout_height="fill_parent" android:layout_width="fill_parent">
        <ImageView android:layout_height="fill_parent"
            android:layout_width="fill_parent" android:id="@+id/imagedetailImageValue"></ImageView>
    </LinearLayout>
    <LinearLayout android:layout_height="wrap_content"
        android:layout_width="wrap_content" android:orientation="vertical">
        <TableLayout android:layout_height="fill_parent"
            android:layout_width="fill_parent">
            <TableRow android:layout_height="wrap_content"
                android:layout_width="wrap_content">
                <TextView android:layout_height="wrap_content"
                    android:layout_width="wrap_content" android:id="@+id/imagedetailDescriptionValue"></TextView>
            </TableRow>
            <TableRow android:layout_height="wrap_content"
                android:layout_width="wrap_content">
                <TextView android:layout_height="wrap_content"
                    android:layout_width="wrap_content" android:text="@string/connectecuserdetailUsername"></TextView>
                <TextView android:layout_height="wrap_content"
                    android:layout_width="wrap_content" android:id="@+id/imagedetailUsernameValue"></TextView>
            </TableRow>
            <TableRow android:layout_height="wrap_content"
                android:layout_width="wrap_content">
                <TextView android:layout_height="wrap_content"
                    android:layout_width="wrap_content" android:text="@string/connectecuserdetaillatitude"></TextView>
                <TextView android:layout_height="wrap_content"
                    android:layout_width="wrap_content" android:text="@string/connectecuserdetaillongitude"></TextView>
            </TableRow>
            <TableRow android:layout_height="wrap_content"
                android:layout_width="wrap_content">
                <TextView android:layout_height="wrap_content"
                    android:layout_width="wrap_content" android:id="@+id/imagedetaillatitudeValue"></TextView>
                <TextView android:layout_height="wrap_content"
                    android:layout_width="wrap_content" android:id="@+id/imagedetaillongitudeValue"></TextView>
            </TableRow>
        </TableLayout>
    </LinearLayout>
    <LinearLayout android:layout_weight="1"
        android:layout_height="fill_parent" android:layout_width="fill_parent"
        android:id="@+id/imagedetailMapRootLayout">
    </LinearLayout>
</LinearLayout>
```

Quadro 14 - Arquivo `layout/imagedetail.xml`

```
public class ImageViewActivity extends MapActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.imagedetail);
        Bundle extras = getIntent().getExtras();
        if (extras != null) {
            String username = extras.getString("unsername");
            String description = extras.getString("description");
            int latitude = extras.getInt("latitudeE6");
            int longitude = extras.getInt("longitudeE6");
            TextView descriptionView = (TextView) findViewById(R.id.imagedetailDescriptionValue);
            descriptionView.setText(description);
            TextView usernameView = (TextView) findViewById(R.id.imagedetailUsernameValue);
            usernameView.setText(username);
            // ...
            LinearLayout rootMapLayout = (LinearLayout) findViewById(R.id.imagedetailMapRootLayout);
            PointMap map = new PointMap(this, getString(R.string.MAPS_API_KEY));
            List<Overlay> overlays = map.getOverlays();
            overlays.add(new PinOverlay(latitude, longitude));
            map.setCenter(latitude, longitude);
            rootMapLayout.addView(map);
```

Quadro 15 - Importação e configuração do leiaute

Outro arquivo XML que contém diversas informações importantes para a aplicação

cliente é o arquivo `values/strings.xml`. Esse arquivo contém cadeias de caracteres importadas para outros componentes na forma de constantes. É utilizado como forma de centralizar e reutilizar as cadeias, facilitando o desenvolvimento. O Quadro 16 apresenta um trecho do arquivo `values/strings.xml` no qual os textos exibidos em uma *activity* são declarados.

```
<!-- Strings da Activity ImageListActivity -->
<string name="ImageList_Label">MediaShare - Imagens</string>
<string name="ImageList_Option_ModifyCoordinates">Modificar coordenadas</string>
<string name="ImageList_Option_ClearCoordinates">Limpar coordenadas</string>
<string name="ImageList_Option_SeeMap">Ver no mapa</string>
<string name="ImageList_Option_Share">Compartilhar</string>
<string name="ImageList_Option_CreateCoordinates">Criar coordenadas</string>
<string name="ImageList_Option_Details">Detalhes</string>
```

Quadro 16 - Trecho do arquivo `values/strings.xml`

### 3.4.4 Persistência no aplicativo cliente

As informações do Mediashare são persistidas pela aplicação cliente no dispositivo na forma de registros de um banco de dados e na forma de arquivos de mídia. O banco de dados é criado através de bibliotecas da plataforma Android. Essas bibliotecas acessam uma base de dados SQLite local. Os arquivos de mídia são persistidos e recuperados dos diretórios de mídia padrão do sistema, através de bibliotecas de entrada e saída de dados da linguagem Java. O Quadro 17 apresenta um trecho do código fonte da classe `MediaPersistence` em que o banco de dados local é inicializado.

```
final String APP_TAG = context.getString(R.string.app_tag);
final String DATABASE = context.getString(R.string.DATABASE);
Log.d(APP_TAG, "Obtendo ou criando database");
SQLiteDatabase db = SQLiteDatabase.openOrCreateDatabase(DATABASE, null);
Log.d(APP_TAG, "Obteve database");
int version = db.getVersion();
Log.d(APP_TAG, "Versao inicial do banco de dados: " + version);
if (version == 0) {
    Log.d(APP_TAG, "Iniciando criacao de tabelas");
    db.execSQL( "CREATE TABLE \"mediashareuser\"([username] VARCHAR(50) PRIMARY KEY, " +
                "[name] VARCHAR(100), [password] VARCHAR2(20))");
    db.execSQL( "CREATE TABLE \"mediapoint\"([id] INTEGER PRIMARY KEY, [latitudeE6] INTEGER, [longitudeE6] INTEGER, " +
                "[username] VARCHAR(50) REFERENCES mediashareuser(username))");
    db.execSQL( "CREATE TABLE \"media\"([description] VARCHAR(200) PRIMARY KEY, [type] VARCHAR(1), " +
                "[pointid] INTEGER REFERENCES mediapoint(id), [username] VARCHAR(50))");
    Log.d(APP_TAG, "Encerrando criacao de tabelas");
    db.setVersion(1);
}
Log.d(APP_TAG, "Versao final do banco de dados: " + db.getVersion());
db.close();
```

Quadro 17 - Inicialização do banco de dados do aplicativo cliente

Através do método `openOrCreateDatabase(String, CursorFactory)`, o banco de dados local é lido ou criado. Uma referência para uma base SQLite é armazenada no objeto `db`. Através desse, é verificada a versão da base local e, caso essa não seja a versão correta, as

tabelas necessárias são criadas através de comandos da *Structured Query Language* (SQL). Ao final, a conexão com a base de dados é encerrada através do método `close()`.

A persistência e recuperação de informações também é efetuada via bibliotecas da plataforma Android. O Quadro 18 demonstra a persistência de informações de mídia através de um trecho do código da classe `MediaPersistence`.

```
public int persist(Media media) {
    final String APP_TAG = this.context.getString(R.string.app_tag);
    if (media == null) {
        Log.w(APP_TAG, "persist(media): media não pode ser null");
        return -1;
    }
    final String DESCRIPTION_FIELD_NAME = this.context.getString(R.string.fieldNameDescription);
    final String TYPE_FIELD_NAME = this.context.getString(R.string.fieldNameType);
    final String POINT_ID_FIELD_NAME = this.context.getString(R.string.fieldNamePointId);
    final String USERNAME = this.context.getString(R.string.fieldNameUsername);
    final String MEDIA_TABLE_NAME = this.context.getString(R.string.tableNameMedia);
    ContentValues values = new ContentValues();
    values.put(DESCRIPTION_FIELD_NAME, media.getDescription());
    values.put(TYPE_FIELD_NAME, media.getType());
    values.put(POINT_ID_FIELD_NAME, media.getMediaPoint().getId());
    values.put(USERNAME, media.getUsername());
    int result = -1;
    Media load = load(media.getDescription());
    SQLiteDatabase database = openDatabase();
    if (load == null) {
        long insert = database.insert(MEDIA_TABLE_NAME, null, values);
        result = (int) insert;
    } else {
        database.update(MEDIA_TABLE_NAME, values, "description=?", new String[] { media.getDescription() });
        result = 1;
    }
    return result;
}
```

Quadro 18 - Trecho da classe `MediaPersistence` em que as informações são persistidas

Inicialmente o método `persist(Media)` obtém constantes do arquivo `values/strings.xml` que contém os nomes dos campos da tabela. Depois uma estrutura de dados do tipo `ContentValues` é criada. Essa estrutura contém os campos da tabela e os respectivos valores a serem armazenados. Em seguida é verificada a existência do registro no banco de dados através de uma chamada ao método `load(String)`. Caso o registro não exista, uma operação de inserção é executada no banco de dados. Caso contário, uma operação de atualização é executada.

O método `load(String)` da classe `MediaPersistence` efetua a recuperação de informações da base local. Esse método abre uma conexão com o banco de dados apenas para a leitura e efetua uma consulta através do método `query(String,String[],String,String[],String,String,String)`. Os parâmetros do último incluem o nome da tabela a ser consultada e qual a cláusula de filtro das informações. Como retorno da execução, o objeto `cursor` recebe uma instância de `Cursor`, que é uma classe para iteração sobre os registros retornados na consulta. A partir desses registros, uma nova instância de `Media` é criada e retornada. O Quadro 19 apresenta o código fonte do método `load(String)` da classe `MediaPersistence`.

```
public Media load(String description) {
    final String APP_TAG = this.context.getString(R.string.app_tag);
    if (description == null || description.length() == 0) {
        Log.w(APP_TAG, "load(String): description não pode ser null ou vazio");
        return null;
    }
    final String MEDIA_TABLE_NAME = this.context.getString(R.string.tableNameMedia);
    Media media = null;
    SQLiteDatabase database = openDatabaseReadOnly();
    Cursor cursor = database.query(MEDIA_TABLE_NAME, null, "description=?", new String[] { description }, null, null, null);
    if (cursor != null) {
        if (cursor.moveToFirst()) {
            String mediaDescription = cursor.getString(0);
            String mediaType = cursor.getString(1);
            int pointId = cursor.getInt(2);
            String username = cursor.getString(3);
            MediaPoint mediaPoint = loadMediaPoint(pointId);
            media = new Media(mediaDescription, mediaType, username, mediaPoint);
        }
        cursor.close();
    }
    closeDatabase(database);
    return media;
}
```

Quadro 19 - Método `load(String)`

### 3.4.5 Comunicação com o servidor

A comunicação com o servidor é efetuada através de requisições `POST`, no padrão *multipart*. As requisições recebem como resposta arquivos XML contendo as informações solicitadas ou arquivos binários com a mídia solicitada. O Quadro 20 apresenta um trecho do código fonte do método `run()` da classe `MediaUploadRunnable`.

O método `run()` inicia obtendo as informações do usuário que está executando a aplicação. Em seguida obtém o endereço do servidor ao qual deve ser feita a requisição. Uma instância de `HttpClient` é criada e armazenada no objeto `client`. Em seguida, uma instância de `HttpPost` é criada. A execução da requisição será feita através do uso combinado dessas instâncias. Antes dessa execução, porém, é necessário montar o conjunto de informações enviada na requisição. Para cada parâmetro da requisição é criado um corpo de requisição dos tipos `StringBody` ou `FileBody`. O primeiro é usado para os tipos texto e o segundo para o arquivo de mídia, quando necessário. Todos os corpos de requisição são agrupados em uma instância de `MultipartEntity`. Essa instância monta uma requisição *multipart*, que é definida como corpo da requisição `POST`. Essa execução é enviada ao servidor pelo método `execute(HttpPost)`. O último retorna um objeto de `HttpResponse`, que contém uma entidade com a resposta. Nesse caso, a resposta é um arquivo XML que indicará se a operação foi processada com sucesso no servidor e, em caso de erros, apresentará uma mensagem de erro. O fluxo de resposta é interpretado como arquivo XML e as *tags* verificadas para, se necessário, exibir uma mensagem de erro para o usuário.

```
UserPersistence userPersistence = new UserPersistence(this.context);
String username = userPersistence.getUsername();
String password = userPersistence.getUserPassword();
final String URL = this.context.getString(R.string.serverUrl) + this.context.getString(R.string.mediaUploadServlet);
HttpClient client = new DefaultHttpClient();
HttpPost post = new HttpPost(URL);
File file = new File(this.description);
FileBody fileBody = new FileBody(file);
StringBody usernameBody = new StringBody(username);
StringBody passwordBody = new StringBody(password);
StringBody typeBody = new StringBody(this.type);
StringBody latitudeBody = new StringBody(Integer.toString(this.latitudeE6));
StringBody longitudeBody = new StringBody(Integer.toString(this.longitudeE6));
StringBody recipientBody = new StringBody(this.recipientUsername);
StringBody descriptionBody = new StringBody(file.getName());
MultipartEntity reqEntity = new MultipartEntity(HttpMultipartMode.BROWSER_COMPATIBLE);
reqEntity.addPart("mediaFile", fileBody);
reqEntity.addPart("username", usernameBody);
reqEntity.addPart("password", passwordBody);
reqEntity.addPart("type", typeBody);
//...
post.setEntity(reqEntity);
InputStream inputStream = null;
HttpResponse response = client.execute(post);
HttpEntity resEntity = response.getEntity();
inputStream = resEntity.getContent();
InputStreamReader reader = new InputStreamReader(inputStream);
char[] readChar = new char[1];
String read = "";
while (reader.read(readChar) != -1) {
    read += new String(readChar);
}
DocumentBuilderFactory documentFactory = DocumentBuilderFactory.newInstance();
DocumentBuilder documentBuilder = documentFactory.newDocumentBuilder();
ByteArrayInputStream byteInput = new ByteArrayInputStream(read.getBytes());
Document document = documentBuilder.parse(byteInput);
inputStream.close();
NodeList childNodes = document.getChildNodes();
Node resultNode = childNodes.item(0);
Node messageNode = childNodes.item(1);
if ("Ok".equals(resultNode.getTextContent())) {
    this.shareMediaDialog.resultMessage(context.getString(R.string.shareMediaSucessMessage));
    return;
} else {
    String message = this.context.getString(R.string.shareMediaErrorMessage);
    message += messageNode.getTextContent();
    this.shareMediaDialog.resultMessage(message);
    return;
}
```

Quadro 20 - Trecho do método `run()` da classe `MediaUploadRunnable`

### 3.4.6 Atualização de coordenada

A atualização das informações de localização do usuário ocorre a partir de bibliotecas da plataforma Android. Essas bibliotecas permitem o registro de uma classe de observação da alteração na localização do usuário. O Quadro 21 apresenta um trecho do código fonte da classe `CurrentLocationUpdateService` em que a mesma se registra como ouvinte de atualização da localização do usuário.

Na criação do serviço `CurrentLocationUpdateService`, o mesmo busca o tempo de atualização e a distância mínima para atualização do arquivo de constantes `values/strings.xml`. Em seguida o método obtém uma referência para o gerenciador de localização do sistema. Através desse gerenciador, o serviço se registra como ouvinte de atualização de localização do usuário invocando o método

`requestLocationUpdates(String, long, float, LocationListener)`.

```
public void onCreate() {
    super.onCreate();
    String updateTimeString = this.getString(R.string.CurrentLocationUpdateServiceUpdateTime);
    String minDistanceString = this.getString(R.string.CurrentLocationUpdateServiceMinDistance);
    long updateTime = Long.parseLong(updateTimeString);
    float inDistance = Float.parseFloat(minDistanceString);
    LocationManager locationManager = (LocationManager) this.getSystemService(Context.LOCATION_SERVICE);
    locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, updateTime, inDistance, this);
}
```

Quadro 21 - Método `onCreate()` da classe `CurrentLocationUpdateService`

Para que uma classe possa ser utilizada para receber atualizações de localização, a mesma deve realizar a interface `LocationListener`. A última define um conjunto de métodos a serem invocados na atualiação da localização ou da situação dos provedores de localização. O Quadro 22 apresenta a implementação de um desses métodos na classe `CurrentLocationUpdateService`.

```
public void onLocationChanged(Location location) {
    LocationUpdateRunnable runnable = new LocationUpdateRunnable(location, this);
    // Processa de forma assÃ-ncrona
    Thread thread = new Thread(runnable);
    thread.start();
}
```

Quadro 22 - Método `onLocationChanged()` da classe `CurrentLocationUpdateService`

O método `onLocationChanged(Location)` ao receber uma atualização de localização delega o processamento a uma instância de `LocationUpdateRunnable`, a ser executada em uma nova *thread*. O Quadro 23 contém o trecho do código fonte em que a localização do usuário é atualizada no cliente.

```
Double latitude = this.location.getLatitude() * 1E6;
Double longitude = this.location.getLongitude() * 1E6;
int latitudeE6 = latitude.intValue();
int longitudeE6 = longitude.intValue();

UserPersistence persistence = new UserPersistence(this.context);
MediashareUser loadUser = persistence.loadUser();
loadUser.setLatitudeE6(latitudeE6);
loadUser.setLongitudeE6(longitudeE6);
persistence.persist(loadUser);
```

Quadro 23 - Trecho do método `run()` da classe `LocationUpdateRunnable`

A classe `LocationUpdateRunnable` efetua o processamento da nova localização convertendo a mesma em valores inteiros. Em seguida, atribui a nova localização ao usuário que está utilizando o aplicativo. O trecho de código em que as coordenadas são enviadas ao servidor foi omitido. Nesse trecho, o aplicativo envia uma requisição `POST` contendo a nova coordenada e recebe um XML com o resultado da operação.

### 3.4.7 Listagem de mídias

As mídias disponíveis no dispositivo são obtidas através da leitura de diretórios padrão existentes para cada tipo de mídia. O Quadro 24 apresenta trecho do código fonte do método `loadImageFilenames()` da classe `ImageLoader` no qual os arquivos de imagens são lidos do sistema.

```
public String[] loadImageFilenames() {
    final String APP_NAME = this.context.getString(R.string.app_name);

    File root = Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_PICTURES);
    root.mkdirs();
    String PATH = root.getPath();
    //...
    ArrayList<String> filenamesArray = new ArrayList<String>();
    String[] list = root.list(new ImageFileFilter());
    if (list != null && list.length != 0) {
        for (String s : list) {
            String fullname = PATH + s;
            Log.i(APP_NAME, "Imagem encontrada: " + fullname);
            filenamesArray.add(fullname);
        }
    } else {
        // Nunca returna nulo
        return new String[0];
    }
    return filenamesArray.toArray(new String[filenamesArray.size()]);
}
```

Quadro 24 - Trecho do método `loadImageFilenames()` da classe `ImageLoader`

Através do método `getExternalStoragePublicDirectory(String)` é obtida uma referência para o diretório público de armazenamento de imagens. Esse diretório pode ainda não ter sido criado, por isso o método `mkdirs()` é invocado. Em seguida, os arquivos da classe são listados, tendo como filtro uma instância de `ImageFileFilter`. O Quadro 25 apresenta o método de filtro da classe `ImageFileFilter`.

```
public boolean accept(File dir, String filename) {
    filename = filename.toLowerCase();
    return  filename.endsWith(".jpeg")
            || filename.endsWith(".jpg") || filename.endsWith(".gif")
            || filename.endsWith(".png") || filename.endsWith(".bmp");
}
```

Quadro 25 - Método `accept()` da classe `ImageFileFilter`

### 3.4.8 Georreferenciamento de mídias

O georreferenciamento de mídias é feito através da persistência de dois registros no banco de dados local. O primeiro registro a ser persistido é o do ponto selecionado pelo

usuário. Após a seleção, esse ponto é vinculado a um registro de mídia, que é o segundo valor persistido. A seleção do ponto é feita na classe `PointMapActivity`. O Quadro 26 apresenta o tratamento do evento de seleção de um ponto no mapa.

```
public boolean onTouchEvent(MotionEvent ev) {
    boolean onTouchEvent = super.onTouchEvent(ev);
    PointSelectedListener listener = PointMap.this.pointAddListener;
    if (this.selecting && listener != null) {
        Projection projection = getProjection();
        // Apenas para debug
        int x = (int) ev.getX();
        int y = (int) ev.getY();
        GeoPoint point = projection.fromPixels(x, y);
        listener.pointSelected(point);
    }
    return onTouchEvent;
}
```

Quadro 26 - Método `onTouchEvent()` da classe `PointMap`

O método `onTouchEvent(MotionEvent)` é invocado a cada toque do usuário no mapa apresentado pela instância de `PointMap`. Caso essa esteja em estado de seleção de ponto e exista objeto observador do evento de seleção, o ponto é convertido em um par latitude e longitude. Para essa conversão, é necessária uma instância de `Projection`. Essa última é capaz de converter uma localização da tela em uma localização geográfica. A conversão é feita através do método `fromPixels(int, int)` que tem como retorno uma instância da classe `GeoPoint`, que representa um ponto no mapa. Essa instância é enviada ao ouvinte de seleção, que efetuará o devido processamento.

Após a definição da coordenada, o aplicativo cliente persiste essas informações no banco de dados local através de uma instância de `MediaPersistence`.

### 3.4.9 Verificação de mídias disponíveis

Como forma de completar o compartilhamento das mídias, o aplicativo cliente efetua requisições ao servidor para verificar a existência de mídias disponíveis para o usuário que está executando o aplicativo. Essas requisições ocorrem em intervalos de tempo definidos e podem ser feitas em duas etapas. Primeiramente o aplicativo verifica se há mídias disponíveis. Caso seja necessário, a segunda etapa é executada e os arquivos de áudio, vídeo ou imagens são buscados em novas requisições, uma para cada arquivo. Para cada arquivo de mídia, o aplicativo cliente efetua uma nova requisição. Todo esse processo é efetuado por uma

instância de `UserInboxRunnable`, que é ativada pelo serviço `UserInboxService`. O Quadro 27 apresenta um trecho do construtor da classe `UserInboxRunnable`.

```
try {
    // Estes objetos não precisam ser recriados.
    // Declara e inicializa antes, para economizar processamento
    final String SERVER_URL = this.context.getString(R.string.serverUrl);
    final String SERVLET = context.getString(R.string.userInboxServlet);
    this.INBOX_URL = SERVER_URL + SERVLET;
    HttpPost post = new HttpPost(this.INBOX_URL);
    StringBody usernameBody = new StringBody(username);
    StringBody passwordBody = new StringBody(password);
    MultipartEntity reqEntity = new MultipartEntity(HttpMultipartMode.BROWSER_COMPATIBLE);
    reqEntity.addPart("username", usernameBody);
    reqEntity.addPart("password", passwordBody);
    post.setEntity(reqEntity);
    Log.d(this.APP_TAG, "Obteve URL e preparoupost");

    HttpClient client = new DefaultHttpClient();
    Log.d(this.APP_TAG, "Obteve httpclient");
    this.inboxHttpPost = post;
    this.client = client;
} catch (Exception e) {
    Log.e(this.APP_TAG, "Ocorreu erro no construtor da thread de userinbox");
    Log.e(this.APP_TAG, "Erro ocorrido: ", e);
    throw new RuntimeException(e);
}
```

Quadro 27 - Trecho do construtor da classe `UserInboxRunnable`

O construtor da classe `UserInboxRunnable` obtém o endereço do servidor para efetuar a requisição do arquivo de constantes `values/strings.xml`. Em seguida instancia um objeto de `HttpPost` e os corpos de requisição com os parâmetros. Esses objetos são então agrupados em uma instância de `MultipartEntity`. A instância é então atribuida ao objeto de `HttpPost`. Em seguida é criada uma instância de `DefaultHttpClient`, a ser utilizada para as requisições. Essas instâncias são armazenadas em atributos, a serem utilizados posteriormente.

Durante a execução do método `run()`, a instância de `UserInboxRunnable` efetua as requisições ao servidor. O Quadro 28 apresenta o trecho do código fonte do método `run()` em que a requisição é efetuada.

Inicialmente o método interrompe a *thread* pelo tempo configurado. Essa interrupção está dentro de um *loop* que é concluído apenas quando ocorrer uma solicitação externa. Após a interrupção, a requisição é efetuada através do método `execute(HttpPost)`. O conteúdo é lido em uma cadeia de caracteres e depois interpretado como um documento XML. Esse documento é então separado em instâncias de `Media` pelo método `parseDocument(Document)`. O Quadro 29 apresenta um trecho desse método em que o documento é interpretado.

```
Thread.sleep(sleepTime);
InputStream inputStream = null;
try {
    HttpResponse response = this.client.execute(this.inboxHttpPost);
    HttpEntity resEntity = response.getEntity();
    Log.d(this.APP_TAG, "Efetuou a requisiÃ§Ã£o e obteve resposta");
    inputStream = resEntity.getContent();
} catch (Exception e) {
    Log.e(this.APP_TAG, "Ocorreu erro na comunicação HTTP. Irá enviar nova requisição");
    Log.e(this.APP_TAG, "Erro ocorrido: ", e);
    continue;
}
InputStreamReader reader = new InputStreamReader(inputStream);
char[] readChar = new char[1];
String read = "";
while (reader.read(readChar) != -1) {
    read += new String(readChar);
}
Document document = null;
try {
    DocumentBuilderFactory documentFactory = DocumentBuilderFactory.newInstance();
    DocumentBuilder documentBuilder = documentFactory.newDocumentBuilder();
    Log.d(this.APP_TAG, "Obteve a factory e o builder de XML");
    ByteArrayInputStream byteInput = new ByteArrayInputStream(read.getBytes());
    document = documentBuilder.parse(byteInput);
    inputStream.close();
    Log.d(this.APP_TAG, "Obteve o documento do XML e fechou o stream de entrada");
} catch (Exception e) {
    Log.e(this.APP_TAG, "Ocorreu erro na obtenÃ§Ã£o dos objetos de manipulaÃ§Ã£o do XML ou na
    Log.e(this.APP_TAG, "Erro ocorrido: ", e);
    continue;
}
```

Quadro 28 - Trecho do método `run()` da classe `UserInboxRunnable`

```
Node inboxRootNode = document.getFirstChild();
NodeList childNodes = inboxRootNode.getChildNodes();
if (childNodes != null && childNodes.getLength() > 0) {
    int mediaLength = childNodes.getLength();
    String description = null;
    String username = null;
    String type = null;
    int latitudeE6 = Integer.MIN_VALUE;
    int longitudeE6 = Integer.MIN_VALUE;
    medias = new Media[mediaLength];
    for (int i = 0; i < mediaLength; i++) {
        Node mediaItem = childNodes.item(i);
        if (!MEDIA_TAG.equals(mediaItem.getNodeName())) {
            continue;
        }
        NodeList detailsNodes = mediaItem.getChildNodes();
        int detailsLength = detailsNodes.getLength();
        for (int j = 0; j < detailsLength; j++) {
            Node detailItem = detailsNodes.item(j);
            String nodeName = detailItem.getNodeName();
            String nodeValue = detailItem.getTextContent();
            if (DESCRIPTION_TAG.equals(nodeName)) {
                description = nodeValue;
            } else if (USERNAME_TAG.equals(nodeName)) {
                username = nodeValue;
            } else if (TYPE_TAG.equals(nodeName)) {
                type = nodeValue;
            } else if (LATITUDEE6_TAG.equals(nodeName)) {
                latitudeE6 = Integer.parseInt(nodeValue);
            } else if (LONGITUDEE6_TAG.equals(nodeName)) {
                longitudeE6 = Integer.parseInt(nodeValue);
            }
        }
        MediaPoint mediaPoint = new MediaPoint(0, latitudeE6, longitudeE6, username);
        Media media = new Media(description, type, username, mediaPoint);
        medias[i] = media;
```

Quadro 29 - Trecho do método `parseDocument()` da classe `UserInboxRunnable`

O método `parseDocument(Document)` obtém o nó raíz do documento e, a partir desse, itera pelos nós filhos. Para cada nó filho, o método verifica se o nó é do tipo mídia. Caso seja, o método obtém os filhos do nó de mídia. Cada filho é então comparado com as *tags* de mídia esperadas e, ao final, uma nova instância de `MediaPoint` é criada. O conjunto

das instâncias obtidas é retornado pelo método. O conjunto das mídias é então percorrido no método `run()` e cada mídia é persistida no banco de dados local através da classe `MediaPersistence`. Para cada mídia do tipo áudio, vídeo e imagem, uma nova requisição é efetuada ao servidor para o recebimento do arquivo. Cada um desses três tipos de mídia é armazenado em um diretório específico existente no sistema de arquivos do dispositivo, através das APIs da plataforma Android. Para exemplificar, o Quadro 30 apresenta um trecho de código do método `run()` da classe no qual um arquivo de imagem é recebido e armazenado no dispositivo.

```
HttpResponse response = this.client.execute(post);
HttpEntity responseEntity = response.getEntity();
InputStream inputStream = responseEntity.getContent();
File externalStoragePublicDirectory =
    Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_PICTURES);
externalStoragePublicDirectory.mkdirs();
File downloadFile = new File(externalStoragePublicDirectory, media.getDescription());
FileOutputStream outputStream = new FileOutputStream(downloadFile);

Bitmap decodedStream = BitmapFactory.decodeStream(inputStream);
decodedStream.compress(Bitmap.CompressFormat.PNG, 100, outputStream);
outputStream.flush();
outputStream.close();
inputStream.close();
```

Quadro 30 - Trecho final do método `run()` da classe `UserInboxRunnable`

O método recebe o retorno do servidor e obtém o fluxo de entrada. Em seguida busca o diretório padrão de armazenamento de imagens do sistema. Em seguida um novo arquivo e um novo fluxo de saída são criados. O fluxo de entrada é então decodificado em uma instância de `Bitmap`. Essa instância então efetua a conversão do fluxo de entrada em um arquivo físico através do método `compress(Bitmap.Format, int, OutputStream)`. Esse método recebe o formato de arquivo destino, no caso `PNG`, a taxa de qualidade a ser mantida, no caso 100%, e o fluxo de saída.

### 3.4.10 Operacionalidade da implementação

As principais funcionalidades do aplicativo Mediashare estão relacionadas ao georreferenciamento de mídias e ao compartilhamento das mesmas. Esta seção apresenta a operacionalidade da aplicação demonstrando o cadastro de um novo usuário, a manutenção de conexões e o georreferenciamento e compartilhamento de uma mídia exemplo.

#### 3.4.10.1 Cadastro de usuário

A tela inicial da aplicação apresenta opções para *login* de um usuário já existente ou cadastro de um novo usuário. Como o *login* é obrigatório, ao utilizar o aplicativo pela primeira vez o usuário deve selecionar a opção para cadastro de um novo usuário. Ao selecionar esta opção, o aplicativo apresentará uma tela contendo os campos relativos ao cadastro do usuário. Após preencher os campos, o usuário poderá salvar as informações através do botão de confirmação. Ao fazê-lo, as informações são enviadas ao servidor e, em caso de sucesso, o aplicativo apresentará a tela inicial do sistema. A Figura 18 apresenta a execução dos passos para o cadastro do usuário dentro do aplicativo.

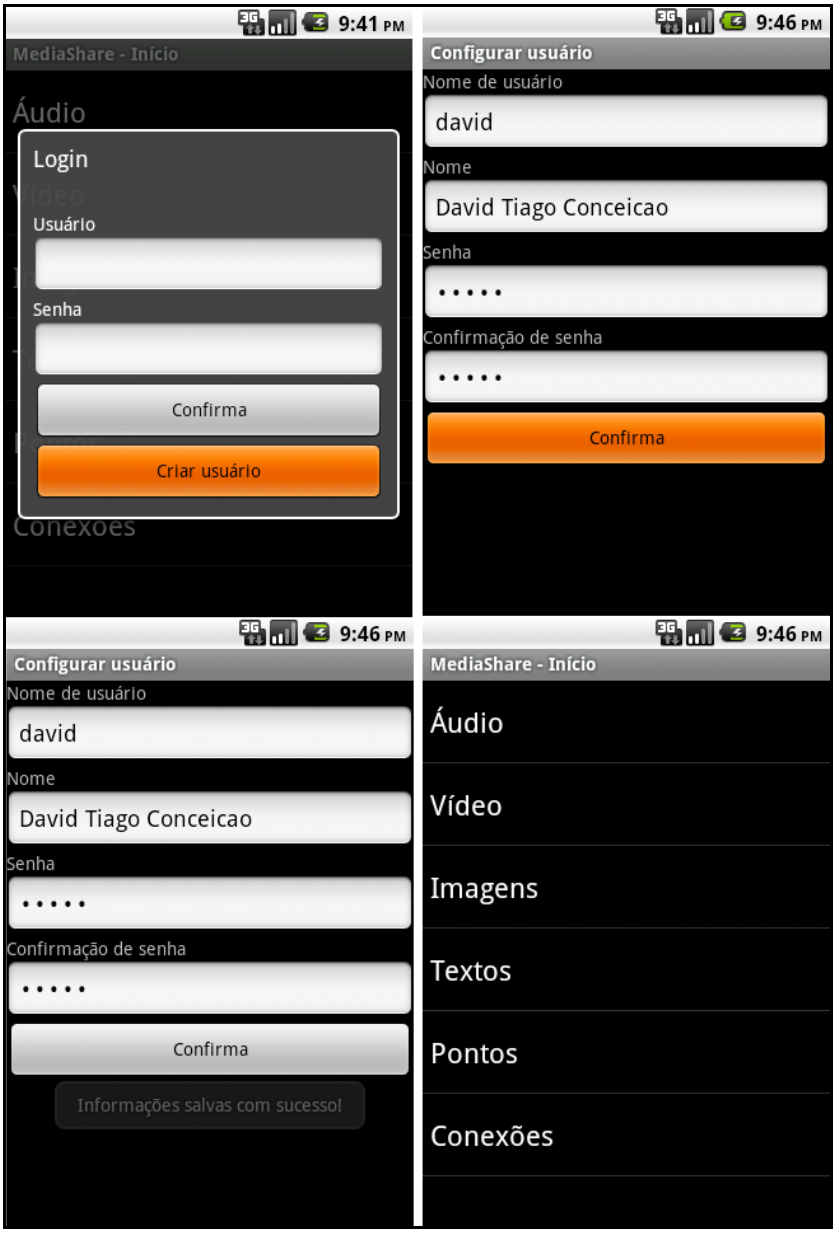



Figura 18 - Passos para a criação de usuário

### 3.4.10.2 Manutenção de conexões

Após seu cadastro no Mediashare, o usuário pode adicionar, remover ou visualizar detalhes de suas conexões. Essas atividades são feitas a partir da opção de conexões da tela inicial. Pressionando esta opção, o aplicativo irá apresentar uma tela com as conexões do usuário. Na primeira utilização esta tela não conterá informações. Pressionando o botão *menu* do dispositivo o usuário terá disponível a opção para adicionar uma conexão. Através dessa opção o usuário pode adicionar uma conexão, que será exibida em sua lista. A Figura 19 apresenta a execução desses passos na aplicação.

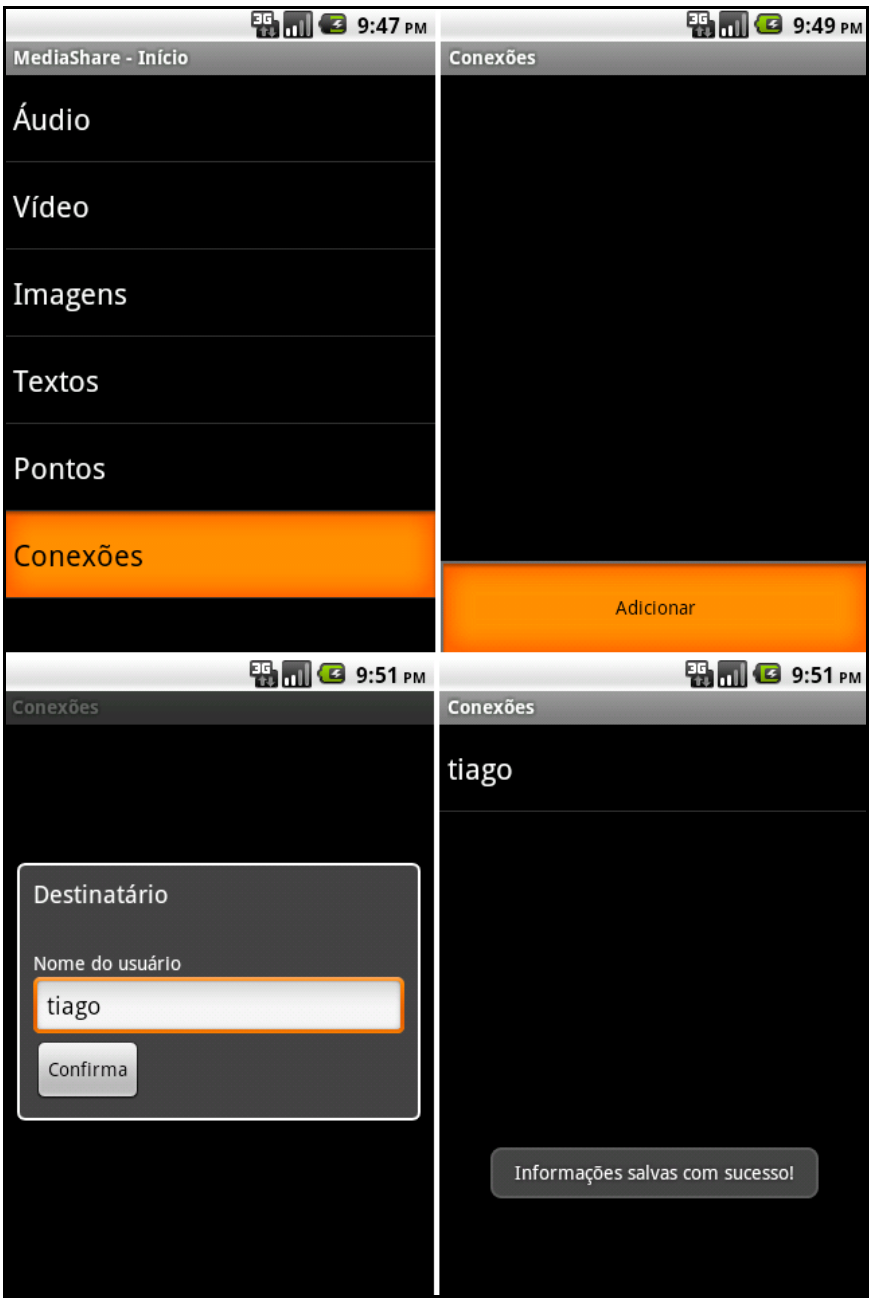



Figura 19 - Passos para adicionar uma conexão

Após adicionar uma conexão, o usuário pode verificar detalhes da mesma pressionando a conexão na lista e em seguida a opção para detalhes. Através desta opção são exibidas informações como nome completo, nome de usuário e localização. A Figura 20 apresenta os passos para a visualização de detalhes de um contato.

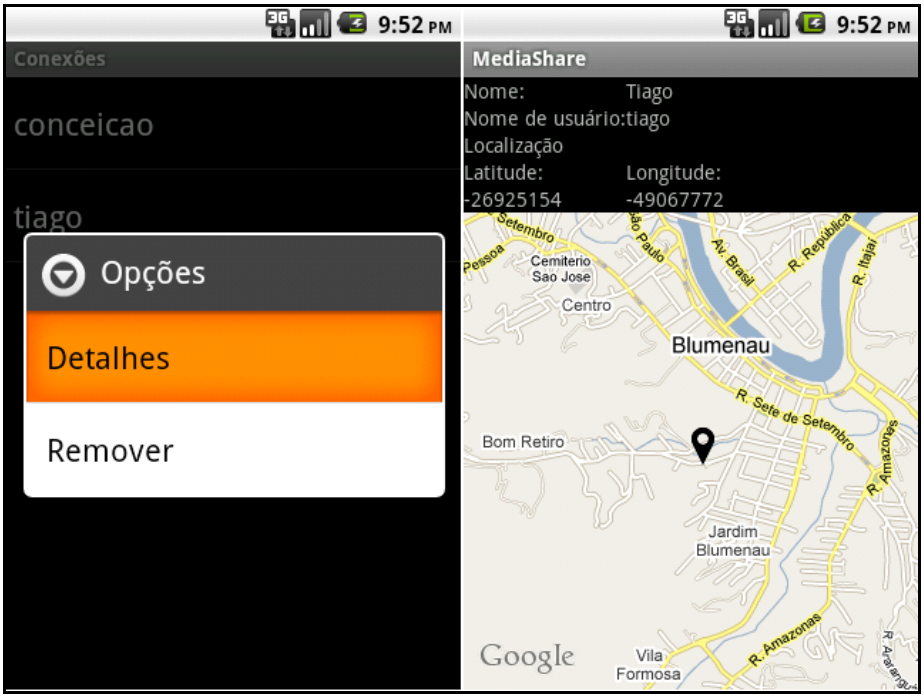



Figura 20 - Passos para visualizar os detalhes de uma conexão

A lista de conexões permite também a remoção de uma conexão, pressionando a mesma e em seguida a opção para remoção. A Figura 21 apresenta a execução dos passos para a remoção de uma conexão da lista.

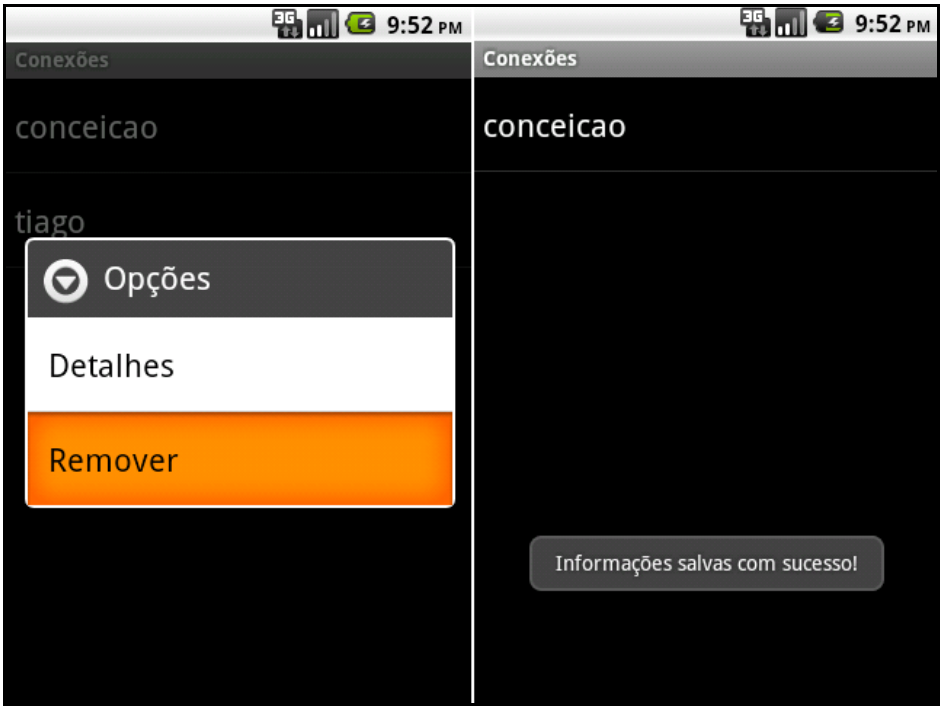



Figura 21 - Passos para remover uma conexão

### 3.4.10.3 Georreferenciamento e compartilhamento

Uma das principais funcionalidades do Mediashare é o compartilhamento de mídias georreferenciadas. Para tal, o usuário deve efetuar dois passos: georreferenciar a mídia e em seguida efetuar o compartilhamento. Assim, a partir da tela inicial, o usuário pode pressionar uma das opções relativas a cada uma das mídias. Uma tela com a listagem de mídias será aberta. A partir dessa tela, o usuário pode pressionar uma das mídias e em seguida a opção para definir a localização. Na tela de mapa, o usuário poderá navegar até o ponto desejado. Para definir o ponto, é necessário pressionar a opção para apontar e em seguida a localização no mapa. Caso a localização informada seja incorreta, a seleção pode ser cancelada através do botão para cancelamento. Para salvar a localização selecionada, é necessário pressionar a opção de confirmação. Com isso, a mídia estará georreferenciada no Mediashare. A Figura 22 demonstra os passos para o georreferenciamento de uma imagem no aplicativo cliente.

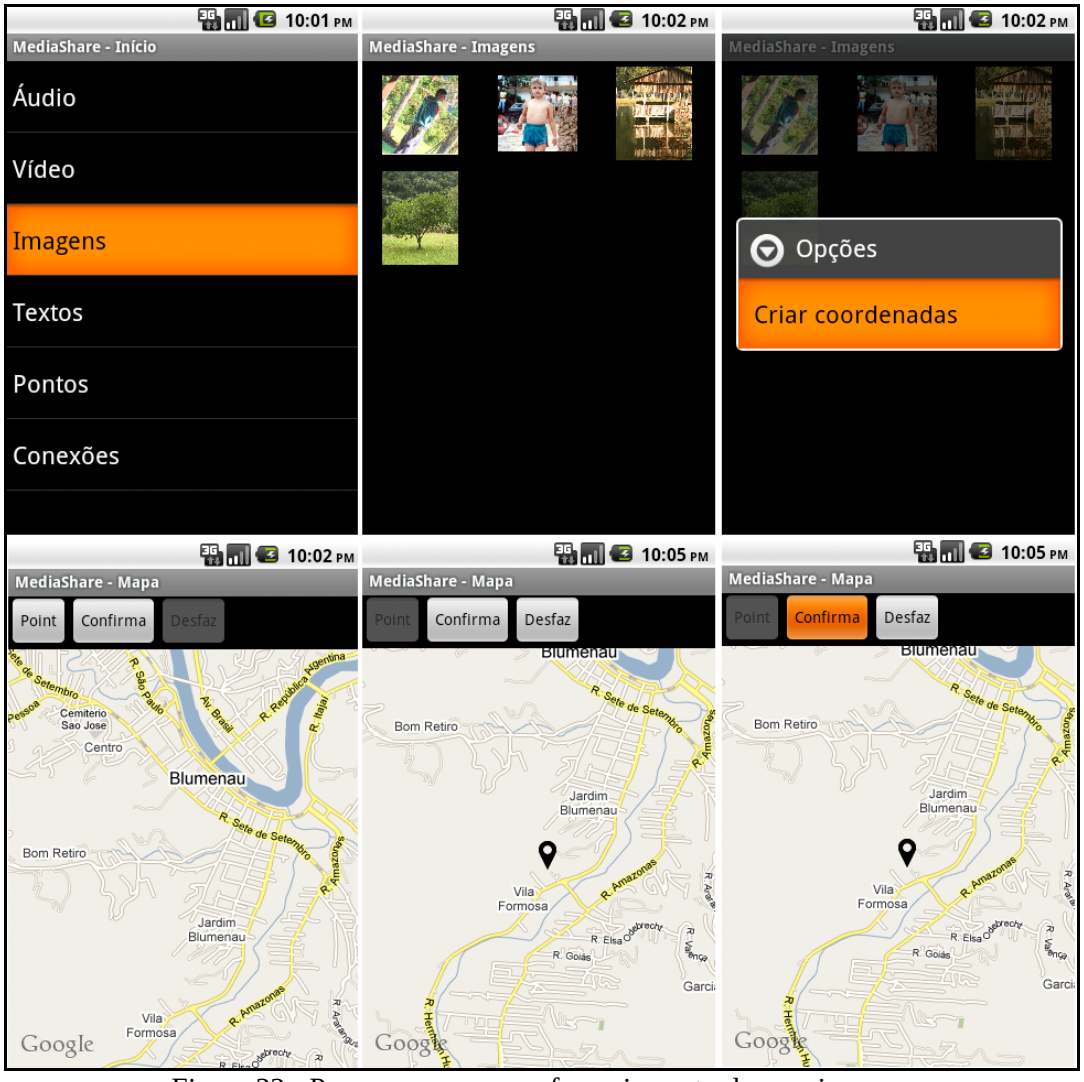



Figura 22 - Passos para o georreferenciamento de uma imagem

Após o georreferenciamento, o aplicativo retornará para a lista de mídias. Pressionando a mídia anteriormente georreferenciada, uma nova lista de opções será aberta. A nova lista inclui a opção de compartilhamento. Essa opção exibe uma janela, na qual deve ser informado o nome do usuário que receberá o compartilhamento. Pressionando a opção de confirmação, a mídia será compartilhada via servidor e o resultado do compartilhamento será exibido. Caso ocorra falha, o resultado será uma mensagem contendo a descrição da falha. Caso contrário, será exibida uma mensagem de conclusão com sucesso. A Figura 23 apresenta a execução do compartilhamento de uma imagem no aplicativo cliente.

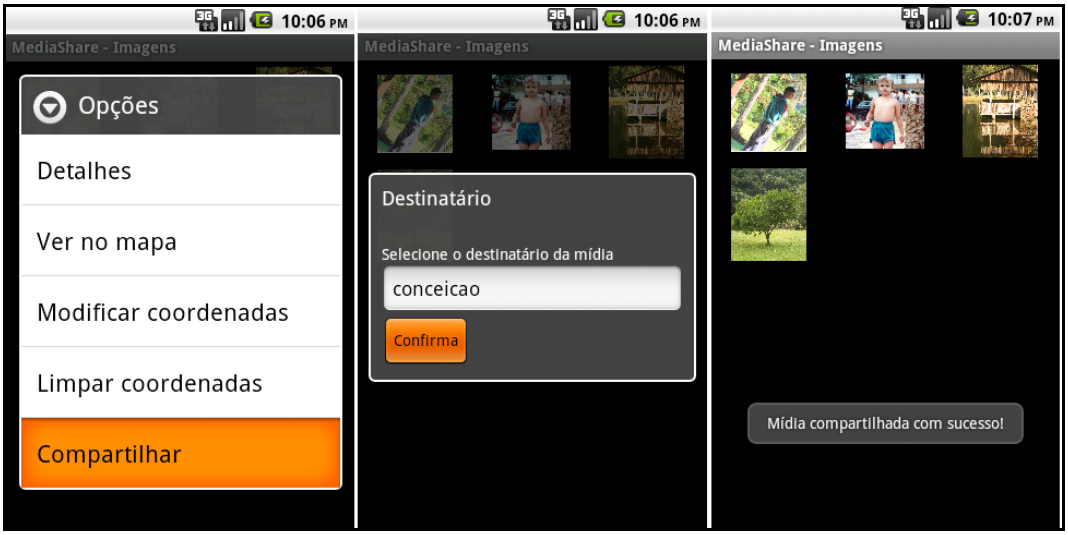



Figura 23 - Passos para o compartilhamento de uma imagem

No dispositivo do destinatário, a mídia será recebida quando o programa estiver em execução ou em segundo plano. Após o recebimento de uma mídia, o Mediashare adicionará uma mensagem na barra de notificações do sistema. Acessando a lista de mídias, pressionando a mídia recebida e em seguida a opção para detalhes, será possível verificar a mídia recebida. Essa opção está disponível para todas as mídias georreferenciadas, recebidas por compartilhamento ou não. A Figura 24 apresenta a notificação de recebimento e a visualização dos detalhes de uma imagem.

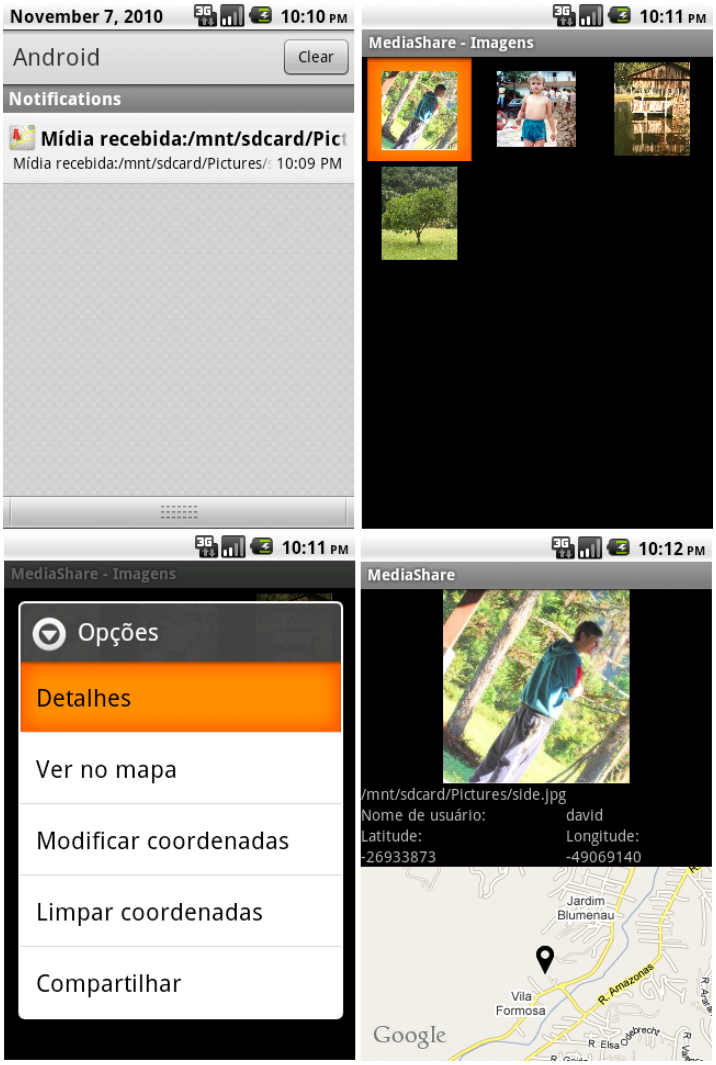



Figura 24 - Notificação de compartilhamento e detalhes de uma imagem

## 3.5 RESULTADOS E DISCUSSÃO

O presente trabalho teve como objetivo inicial o desenvolvimento de uma solução que se enquadrasse nas definições de um *framework*. Sauvé (2010) afirma que um *framework* apresenta uma solução para um conjunto de problemas semelhantes, com classes e interfaces que decompoem o problema. As associações e o fluxo de informações entre essas classes são definidos pelo próprio *framework*, sendo que esse dita a arquitetura do sistema. Afirma também que o *framework* deve ser responsável por invocar classes adicionais da aplicação. Ou seja, uma aplicação desenvolvida sobre um *framework* deve adaptar-se a ele da maneira definida. Para isso, um *framework* deve disponibilizar funcionalidades abstratas, a serem

completadas. Essas funcionalidades definirão as características específicas de cada aplicação.

Diante do exposto, pode-se verificar que o presente trabalho atendeu parte das características de um *framework*. A partir da identificação de um conjunto de problemas (georreferenciamento de mídias e compartilhamento de mídias georreferenciadas) foi desenvolvido um conjunto de classes objetivando solucionar o problema. As classes desenvolvidas apresentam associações e fluxo de informações definidos. Não apresentam, porém, funcionalidades abstratas, de modo que a criação de aplicações que estendam o *framework* foi comprometida. Considerando a grande quantidade de casos de uso, preferiu-se demonstrar a solução de diversos problemas em detrimento da criação de funcionalidades abstratas.

As diferenças entre a plataforma Android e a plataforma JSE causaram diversos problemas de adequação, especialmente aos novos conceitos. A ativação de componentes através de intenções impede a aplicação de modelos tradicionais e obriga a aplicação de modelos mais genéricos. Dado que a ativação deve ser resolvida pelo sistema, as mensagens e as informações devem ser genéricas (independentes de métodos ou classes específicos). Cada componente deve também ser independente de outros. Como forma de contornar esse problema, os componentes da aplicação cliente foram pensados previamente, com a definição da responsabilidade de cada *activity* e cada *service*. Os principais problemas relativos ao aplicativo cliente estiveram relacionados a definição da arquitetura do sistema e criação de um modelo simples e reutilizável dentro do framework. O desenvolvimento das classes e interfaces na já conhecida linguagem Java, aliados a bibliotecas intuitivas serviram de facilitador na implementação da solução.

Durante a implementação optou-se pela simplificação do aplicativo cliente através da importação da menor quantidade de bibliotecas possível. Ao final, apenas as bibliotecas `HTTPMime` e a biblioteca de mapas da Google foram importadas. Considerando a inexistência de bibliotecas nativas na plataforma Android para comunicação com *Web Services* optou-se pela comunicação via conexões HTTP auxiliadas pela biblioteca `HTTPMime`. Essas conexões mostraram-se eficientes e de fácil utilização.

Outra opção relacionada a implementação foi a não utilização de código nativo (escrito na linguagem C++) no aplicativo cliente. Conforme Android Developers (2010l), a utilização de código nativo não beneficia todas as aplicações e não resulta em acréscimo automático de performance, apesar de sempre resultar em acréscimo de complexidade. Ainda segundo Android Developers (2010l), operações candidatas a escrita em código nativo apresentam alto consumo do processador central do dispositivo e pouca alocação de memória. Exemplos

dessas operações são o processamento de sinais e simulações físicas. Assim, nenhum componente do aplicativo cliente justificou o emprego dessas técnicas. Em testes no simulador foi possível verificar que o maior consumo de processamento esteve ligado a renderização dos mapas, operação controlada pela biblioteca de mapas da Google.

Os *services* apresentaram um desafio a parte, já que podem ser desenvolvidos de diversas maneiras. *Services* remotos podem ser desenvolvidos através de interfaces e de comunicação entre processos. Dessa forma podem ser acionados síncronamente a partir de outras aplicações. O desenvolvimento de *services* remotos mostrou-se de complexidade demasiada, uma vez que exige a definição de interfaces remotas, a realização dessas interfaces e a criação de clientes. Além disso, a invocação de serviços remotos é síncrona, sendo necessária a criação de novas *threads* para atender aos requisitos propostos. Os *services* remotos podem ser desenvolvidos para a execução em novos processos, também exigindo a definição de interfaces. Para este trabalho, optou-se pelo desenvolvimento de *services* locais que, ao contrário dos remotos, não necessitam da definição de interfaces. Essa simplificação, porém, permite menor interação com o *service*, que pode apenas ser disparado ou interrompido por classes externas. Mesmo com essa limitação, os *services* locais atendem as necessidades do trabalho através da criação de novas *threads* dentro da instância de cada *service*.

A troca de informações entre os usuários exigiu o desenvolvimento de componentes externos a plataforma Android. Para este trabalho, foi adotada a arquitetura cliente-servidor, com ênfase no desenvolvimento do cliente. O aplicativo servidor foi desenvolvido de forma a suprir as necessidades apresentadas pelo cliente. Assim, o servidor apresenta apenas o estritamente necessário para a correta apresentação das funcionalidades do cliente. O envio de informações do cliente para o servidor também exigiu estudo. Inicialmente foi proposto o envio em duas fases, uma para o georreferenciamento e outra para o envio do arquivo, se necessário. Buscou-se, porém, uma alternativa para a execução em apenas uma requisição, com o objetivo de reduzir a carga da rede. A solução encontrada foi efetuar a requisição através do método `POST` com *multipart*. Dessa forma, as requisições puderam ser efetuadas de maneira intuitiva e sem a necessidade de informações adicionais. As respostas enviadas pelo servidor também foram trabalhadas no sentido de conterem o mínimo de informações possível.

Os requisitos do trabalho foram desenvolvidos a partir do princípio de que apenas mídias georreferenciadas possam ser compartilhadas através do sistema. Essa definição gerou algumas restrições no aplicativo cliente e no servidor. As telas de listagem de mídias

apresentam a opção de compartilhamento apenas para mídias georreferenciadas. A classe de envio para o servidor e o próprio servidor exigem as informações de georreferenciamento para funcionarem. Essas restrições reduziram a complexidade do sistema, porém reduziram também o espectro de aplicação do mesmo, uma vez que, caso o usuário tenha a intenção de compartilhar mídias não georreferenciadas, não poderá fazê-lo através do Mediashare.

Para armazenamento das informações de georreferenciamento, buscou-se um modelo genérico, que pudesse ser aplicado a todas as mídias. Para tal, optou-se por manter essas informações em um banco de dados local, de forma independente dos arquivos de mídia externos. Assim, cada coordenada pode ser associada a mais de uma mídia de forma simplificada. Também não são necessários controles extras específicos para cada tipo de mídia, permitindo grande reusabilidade das classes de persistência. Esse modelo, porém, exige a criação de um novo arquivo de dados, a ser gerenciado pelas bibliotecas de bancos de dados locais. Optou-se por manter esse arquivo no espaço de memória específico da aplicação, impedindo acesso de aplicações externas. Apesar dessa restrição, o arquivo gerado para o banco de dados pode ser visualizado através do Android SDK, sendo possível verificar informações do mesmo ou removê-lo. Com isso, um dos testes de eficiência realizados trata de verificar o tamanho do arquivo em comparação com a quantidade de mídias georreferenciadas. O tamanho do arquivo do banco de dados em comparação com a quantidade de mídias georreferenciadas é apresentado na Tabela 1.

Tabela 1 - Tamanho do banco de dados local

| Banco de dados local | |
|---|---|
| **Mídias** | **Tamanho em bytes** |
| 0 | 9216 |
| 5 | 9216 |
| 10 | 9216 |
| 15 | 11264 |
| 20 | 13312 |
| 25 | 13312 |
| 30 | 13312 |
| 35 | 14336 |
| 40 | 15360 |
| 45 | 15360 |
| 50 | 15360 |

A partir das informações obtidas no teste é possível observar que o arquivo gerado pelo banco de dados local não necessariamente aumenta de tamanho a cada nova inserção de registro. A Figura 25 apresenta um gráfico em que é apresentado o tamanho do banco de dados em função da quantidade de mídias georreferenciadas.

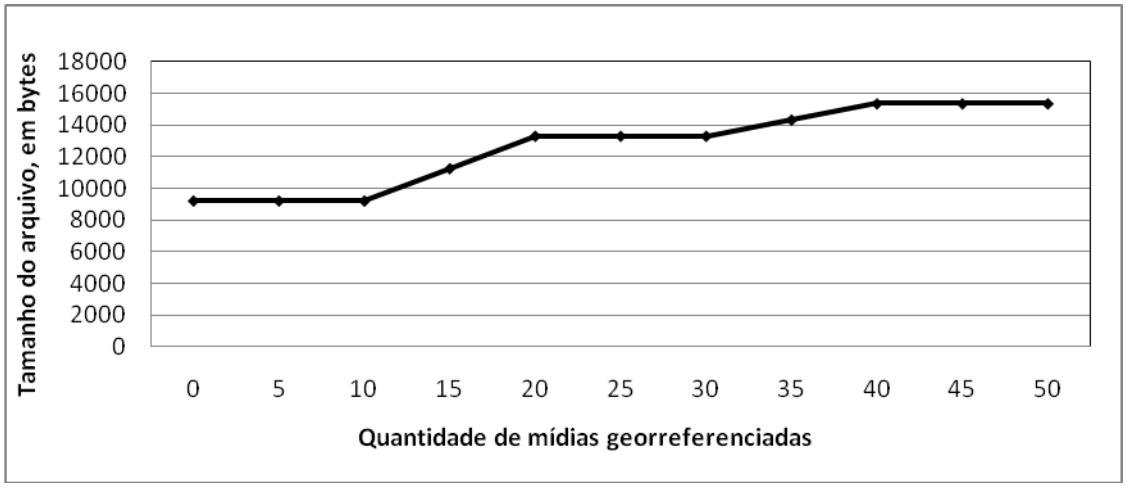



Figura 25 - Tamanho do banco de dados local em função da quantidade de registros

Além da eficiência na quantidade de informações armazenada, os dispositivos móveis exigem eficiência na quantidade de memória principal e de processamento consumidos. Considerando que o aplicativo proposto possui serviços a serem executados em segundo plano, foram efetuados testes para verificar a quantidade de memória utilizada pelo aplicativo e o tempo de processamento. Essas informações podem ser verificadas através do Android SDK, executando a aplicação no emulador. Esse último foi executado em um microcomputador com processador Intel Core 2 Duo T6400 e 4 *GigaBytes* de memória principal. O servidor foi executado no mesmo computador. Foram testadas duas alternativas para a implementação dos services. Na primeira, todos os objetos necessários para o processamento foram instanciados a cada execução do *service*. Ou seja, no tempo em que o *service* ficou ocioso, a maior quantidade de objetos possível foi disponibilizada para remoção. Após cinco requisições de cada serviço, foi possível verificar que o aplicativo ocupava aproximadamente 2,565 *MegaBytes* da memória principal. Cada requisição demorou aproximadamente 225 milissegundos para ser executada. Como segunda alternativa, a maior quantidade de objetos possível foi instanciada nos construtores dos serviços e mantida em memória mesmo quando o serviço estava ocioso. Após cinco requisições de cada serviço, foi possível verificar que o aplicativo ocupava aproximadamente 2,578 MegaBytes da memória principal. Cada requisição foi executada em aproximadamente 220 milissegundos. Em comparação com a primeira alternativa, a última utilizou quantidade de memória semelhante para efetuar requisições em tempos semelhantes. Dessa forma, optou-se pela segunda forma de implementação, mantendo os objetos em memória.

Outros testes para verificar a quantidade de memória consumida foram efetuados no emulador. Em um deles foi efetuado o processo de georreferenciamento de uma mídia seguido do compartilhamento da mesma. A cada nova tela carregada foi consultada a

quantidade de memória ocupada pelo aplicativo. Através desse teste foi possível verificar que, imediatamente após o carregamento, o aplicativo ocupa 2,344 *MegaBytes* da memória principal. Essa foi a quantidade mínima ocupada. Durante o processo de georreferenciamento seguido do compartilhamento o aplicativo alcançou o máximo de 3,283 *MegaBytes* de memória ocupados (cerca de 1,4 vez a quantidade o mínima). Em média, o aplicativo ocupou 2,871 *MegaBytes* da memória principal (cerca de 1,22 vez a quantidade mínima). Para efeito comparativo, foram efetuados testes na aplicação Google Latitude (GOOGLE, 2010a). Ao iniciar uma nova instância da aplicação, a mesma ocupava 3,513 *MegaBytes* da memória principal (cerca de 1,49 vez a quantidade mínima ocupada pelo Mediashare). Após um minuto navegando pelo mapa apresentado na aplicação, a mesma alcançou o máximo de 4,332 *MegaBytes* de memória alocada (cerca de 1,32 vez a quantidade máxima ocupada pelo Mediashare).

Para verificar a usabilidade e aplicabilidade da solução, o tempo de transferência de uma mídia para o servidor foi verificado. O objetivo foi analisar qual o crescimento no tempo de transferência em proporção ao tamanho de cada midia. O emulador foi executado em uma rede Wi-Fi de 54 *Megabits* por segundo, no mesmo computador utilizado para os testes anteriores. O servidor foi executado na mesma máquina, fazendo com que a latência da transmissão fique próxima a zero. Também não foi aplicada carga extra no servidor, para simular concorrência. A banda de rede máxima alocada pelo emulador é equivalente a rede de dados de terceira geração (3G). Em seguida, essa banda foi restrita para a velocidade das redes de celulares do tipo *Global System for Mobile Communications* (GSM) e *Enhanced Data rates for GSM Evolution* (EDGE), através de configurações do emulador. O Quadro 31 apresenta os tempos de execução do compartilhamento de cada mídia em relação ao tamanho da mídia compartilhada.

| Número | Tamanho do arquivo em bytes | Tecnologia de transmissão | Tempo de processamento |
|---|---|---|---|
| 1 | 102.551 bytes | 3G | **344 ms** |
| | | GPRS | **350 ms** |
| | | EDGE | **370 ms** |
| 2 | 305.556 bytes | 3G | **301 ms** |
| | | GPRS | **306 ms** |
| | | EDGE | **340 ms** |
| 3 | 491.912 bytes | 3G | **342 ms** |
| | | GPRS | **361 ms** |
| | | EDGE | **500 ms** |
| 4 | 3.236.574 bytes | 3G | **1.170 ms** |
| | | GPRS | **1.066 ms** |
| | | EDGE | **1.700 ms** |
| 5 | 9.879.583 bytes | 3G | **2.693 ms** |
| | | GPRS | **2.851 ms** |
| | | EDGE | **3.850 ms** |
| 6 | 38.264.944 bytes | 3G | **15.490 ms** |
| | | GPRS | **16.140 ms** |
| | | EDGE | **18.369 ms** |

Quadro 31 - Tempo de processamento do compartilhamento das mídias

Através dos resultados obtidos é possível verificar que o tamanho do arquivo de mídia compartilhado é de grande impacto no tempo de processamento. Embora os tempos obtidos sejam inferiores aos esperados, é possível observar que o crescimento do tempo de processamento não é linear. A mídia número 3, por exemplo, tem aproximadamente 1,61 vez o tamanho da mídia de número 2. O tempo de processamenteo da mídia 3, porém foi aproximadamente 1,14 vez superior ao da mídia 2. A Figura 26 apresenta um gráfico no qual o crescimento do tempo de processamento é apresentado. É possível observar que as tecnologias de transmissão não proporcionaram grandes diferenças nos tempos, especialmente para os arquivos menores.

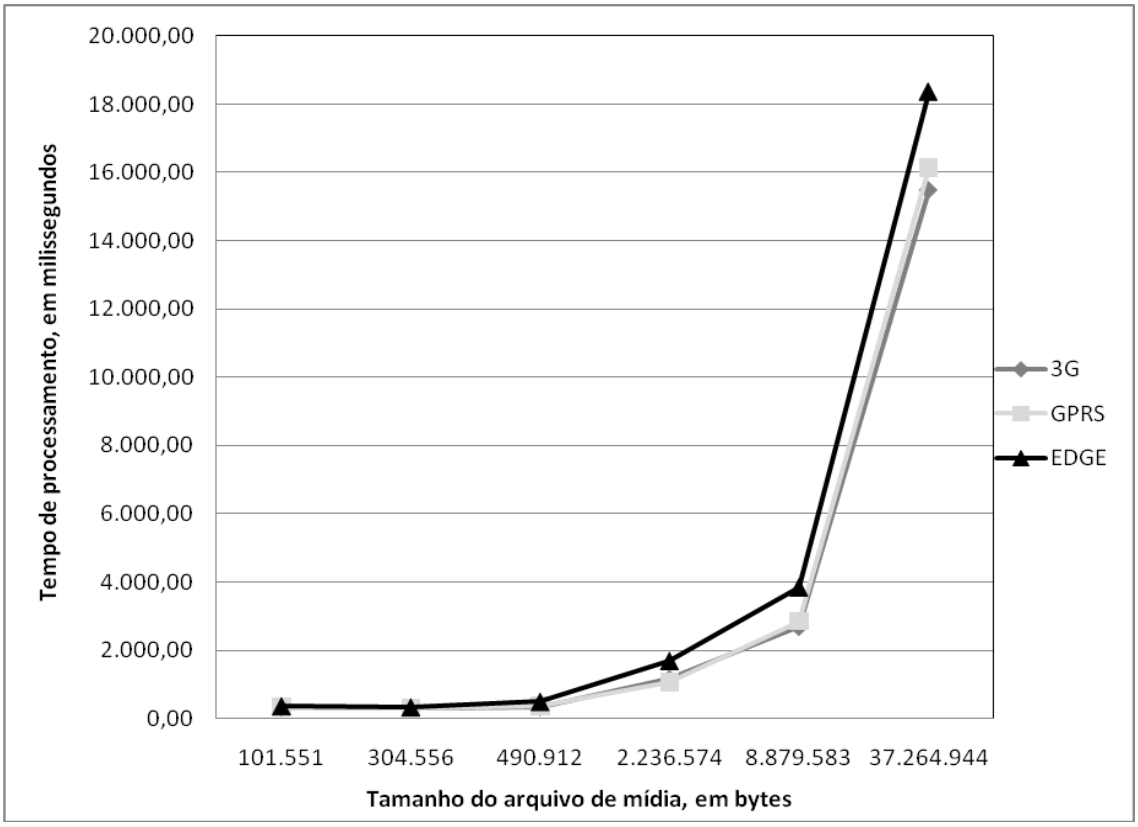



Figura 26 - Tamanho de arquivo e tempo de processamento

# 4 CONCLUSÕES

O presente trabalho apresenta a reunião de formatos de mídia variados em um único aplicativo, permitindo o georreferenciamento e compartilhamento através de funcionalidades padronizadas. Permite também a formação de redes de conexões entre usuários e a visualização das variadas mídias. Dessa forma, contribui para a informática definindo uma forma de georreferenciamento, armazenamento de informações locais e troca de informações com um servidor remoto a partir da plataforma Android. Os diversos aspectos da plataforma são apresentados e, ao final, as opções adotadas são discutidas.

Durante o desenvolvimento do trabalho foi possível observar que o desenvolvimento de um *framework* é um processo complexo que exige constante aprimoramento da solução. Assim, neste trabalho opto-se por atender a parte das características de um *framework*, definindo conjuntos de classes para uma categoria de problemas. Além disso, buscou-se também atender a maior parte dos requisitos propostos, demonstrando diversas características e funcionalidades da plataforma Android. Assim, o presente trabalho lança bases para o estudo de novas funcionalidades ou a adaptação desse para as características completas de um *framework.*

O presente trabalho demonstra também a utilização da plataforma Android como cliente de uma arquitetura cliente-servidor. Para tal, foram analisados os meios de comunicação entre um dispositivo Android e um servidor externo. A alternativa adotada foi a comunicação via método `POST` do protocolo HTTP. Essa decisão foi tomada por diversos motivos, entre eles o suporte nativo na plataforma Android, a possibilidade de inclusão de parâmetros na requisição e a facilidade de desenvolvimento de um servidor que atenda as requisições. Para reduzir o número de requisições e, consequentemente, o tráfego na rede foi utilizado o padrão *multipart*. Através desse, diversas informações, inclusive arquivos, puderam ser agrupadas em uma única requisição.

Considerando as restrições dos dispositivos móveis, diversos cuidados foram necessários. A minimização da quantidade de memória principal ocupada pelo dispositivo foi obtida através do uso combinado de atributos de classes e variáveis locais. Nesse ponto, o resultado alcançado foi positivo, já que, mesmo com serviços executando em segundo plano, a quantidade de memória ocupada foi inferior a ocupada por um trabalho correlato. Outro cuidado esteve relacionado a quantidade de informações armazenadas no banco de dados local. Considerando que esse banco de dados contém todas as informações de

georreferenciamento, foi necessária atenção a quais atributos poderiam ser armazenados no banco de dados e qual a capacidade de cada atributo. O resultado obtido nesse caso foi satisfatório, já que a inclusão de novos registros não acarretou na expansão acelerada do banco de dados.

Como principais limitações do presente trabalho, podem ser apresentadas as limitações inerentes a decisões de especificação. O compartilhamento de mídias não georreferenciadas não é permitido, uma vez que tanto o cliente quanto o servidor foram arquitetados para exigir o georreferenciamento. A interação e comunicação com as conexões também é limitada, não existindo alertas de proximidade, aviso de que um usuário foi adicionado ou afins. Não foi desenvolvida também funcionalidade para troca de informações em tempo real, como mensagens de texto, por exemplo. A vinculação de informações de georreferenciamento através dos formatos específicos de cada mídia também não foi desenvolvida. Alguns formatos de arquivo podem apresentar atributos específicos para a armazenagem do georreferenciamento. Esses atributos não foram explorados no presente trabalho.

Por fim, a plataforma Android mostrou-se ampla e expansível. Ampla por apresentar diversas bibliotecas nativas que facilitaram o desenvolvimento desse trabalho. A exibição de mídias, mapas e a obtenção de coordenadas geográficas, por exemplo, puderam ser desenvolvidas facilmente através das classes já existentes na plataforma ou em bibliotecas auxiliares. Além disso, a plataforma apresenta um ambiente de desenvolvimento completo, com um SDK fortemente integrado com o Eclipse IDE e um emulador com diveras opções de parametrização. A capacidade de expansão da plataforma é dada pela grande facilidade de incorporação de bibliotecas Java externas. É o caso da biblioteca `HTTPMime` disponibilizada pela Apache, que pode ser facilmente incorporada ao projeto.

## 4.1 EXTENSÕES

Como sugestões de extensões para a continuidade do presente trabalho, tem-se:

a) adicionar funcionalidades abstratas ao aplicativo cliente. Ou seja, adicionar pontos em que classes possam ser conectadas, criando novas aplicações. Através dessa alteração, o presente trabalho passaria a atender a todas as características de um *framework*;
b) melhorar o servidor através do desenvolvimento de um aplicativo com arquitetura

robusta e preparada para grandes volumes de dados. Outras funcionalidades sugeridas para o servidor são a aplicação de técnicas de segurança mais avançadas, o gerenciamento automático de arquivos e uma interface gráfica para administração;

c) adicionar opções para acesso a *streamming*, de forma que um vídeo possa ter sua reprodução iniciada antes da conclusão do recebimento. A plataforma Android possui classes preparadas para manipular *streamming*, sendo necessário o desenvolvimento de um servidor, ou adequação a um servidor já existente;
d) permitir o compartilhamento de mídias não georreferenciadas, eliminando as restrições do sistema quanto a isso. Dessa forma, o *framework* aumentaria sua abrangência, podendo ser utilizado na resolução de outros problemas;
e) pesquisar modelos de georreferenciamento específicos para cada tipo de mídia, de forma a permitir que o georreferenciamento seja importado por outras aplicações;
f) adicionar opção para reprodução de vídeos em tela cheia. Como sugestão, pode ser desenvolvida uma *activity* específica para a reprodução de vídeos, com a opção de fazer o vídeo ocupar toda a tela do dispositivo;
g) adicionar opção para visualizar imagem em tela cheia. Como sugestão, pode ser desenvolvida uma nova activity específica para a visualização de imagem, com a opção de *zoom* e deslocamento pela imagem;
h) adicionar funcionalidades de interação entre os usuários conectados. Como sugestões, tem-se a criação de alertas de proximidade com outras conexões, avisos de que o usuário atual foi adicionado como conexão de outro usuário e funcionalidades para comunicação em tempo real, para conversas via texto, por exemplo. Outra sugestão é o emprego de APIs de redes sociais já existentes com o Facebook e o Twitter. Dessa forma o framework seria inserido diretamente no contexto das redes sociais.

# REFERÊNCIAS BIBLIOGRÁFICAS


ANDROID DEVELOPERS. **Activity**. [S.l.], 2010a. Disponível em: <http://developer.android.com/reference/android/app/Activity.html>. Acesso em: 15 ago. 2010.

____. **Android virtual devices**. [S.l.], 2010b. Disponível em: <http://developer.android.com/guide/developing/tools/avd.html>. Acesso em: 11 nov. 2010.

____. **Application fundamentals**. [S.l.], 2010c. Disponível em: <http://developer.android.com/guide/topics/fundamentals.html>. Acesso em: 28 mar. 2010.

____. **Audio and video**. [S.l.], 2010d. Disponível em: <http://developer.android.com/guide/topics/media/index.html>. Acesso em: 28 mar. 2010.

____. **Installing the SDK**. [S.l.], 2010e. Disponível em: <http://developer.android.com/sdk/installing.html>. Acesso em: 02 nov. 2010.

____. **Location and maps**. [S.l.], 2010f. Disponível em: <http://developer.android.com/guide/topics/location/index.html>. Acesso em: 28 mar. 2010.

____. **MediaPlayer**. [S.l.], 2010g. Disponível em: <http://developer.android.com/reference/android/media/MediaPlayer.html>. Acesso em: 17 ago. 2010.

____. **Service**. [S.l.], 2010h. Disponível em: <http://developer.android.com/reference/android/app/Service.html>. Acesso em: 15 ago. 2010.

____. **The AndroidManifest.xml file**. [S.l.], 2010i. Disponível em: <http://developer.android.com/guide/topics/fundamentals.html>. Acesso em: 28 mar. 2010.

____. **View**. [S.l.], 2010j. Disponível em: <http://developer.android.com/reference/android/view/View.html>. Acesso em: 15 ago. 2010.

____. **What is Android?** [S.l.], 2010k. Disponível em: <http://developer.android.com/guide/basics/what-is-android.html>. Acesso em: 28 mar. 2010.

____. **What is the NDK?** [S.l.], 2010l. Disponível em: <http://developer.android.com/sdk/ndk/overview.html>. Acesso em: 13 dez. 2010.

ANDROID OPEN SOURCE PROJECT. **Welcome**. [S.l.], 2010. Disponível em: <http://source.android.com>. Acesso em: 28 mar. 2010.

DALVIKVM. **Dalvik virtual machine**. [S.l.], 2008. Disponível em: <http://www.dalvikvm.com>. Acesso em: 28 mar. 2009.

GONZALES, Max A. **As 8 tecnologias móveis de 2009 e 2010**. São Paulo, 2009. Disponível em: <http://info.abril.com.br/professional/mobilidade/as-8-tecnologias-moveis-de-200.shtml?2>. Acesso em: 28 mar. 2009.

GOOGLE. **Google latitude**. [S.l.], 2010a. Disponível em: <http://www.google.com/intl/en_us/mobile/latitude>. Acesso em: 10 out. 2010.

____. **Google latitude**. [S.l.], 2010b. Disponível em: <http://www.google.com/latitude/apps>. Acesso em: 10 out. 2010.

____. **Google projects for Android**: Google APIs: Obtaining a Maps API Key. [S.l.], 2010c. Disponível em: <http://code.google.com/intl/pt-BR/android/add-ons/google-apis/mapkey.html>. Acesso em: 17 ago. 2010.

HASHIMI, Sayed Y.; KOMATINENI, Satya,. **Pro Android**. Berkeley: Apress, 2009.

IBRAHIM, Ismail K. (Ed.). **Handbook of research on mobile multimedia**. 2. ed. Nova Iorque: Information Science Reference, 2008. Disponível em: <http://books.google.com/books?id=T5EV8lPALSkC>. Acesso em: 28 mar. 2010.

IBRAHIM, Ismail K.; TANIAR, David (Ed.). **Mobile multimedia**: a communication engineering perspective. Nova Iorque: Information Science Reference, 2008. Disponível em: <http://books.google.com/books?id=HTAtI4MopGsC>. Acesso em: 28 mar. 2010.

JAKARTA COMMONS. **HttpClient – HttpClient Features** . [S.l.], 2008a. Disponível em: <http://hc.apache.org/httpclient-3.x/features.html>. Acesso em: 24 set. 2010.

____. **HttpClient**: HttpClient home. [S.l.], 2008b. Disponível em: <http://hc.apache.org/httpclient-3.x>. Acesso em: 24 set. 2010.

KOSMACH, Jim et al. Introduction to the Opencore video components used in the Android platform. In: INTERNATIONAL WORKSHOP ON VIDEO CODING AND VIDEO PROCESSING, 1. , 2008, Shenzhen. **Proceedings…** [S.l.]:[s.n], 2008. p. 1-3. Disponível em: <http://www.opencore.net/files/VCVPpaper2008-v07.pdf>. Acesso em: 17 ago. 2010.

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

OPEN HANDSET ALLIANCE. **Alliance.** [S.l.], [2010a]. Disponível em: <http://www.openhandsetalliance.com/oha_overview.html>. Acesso em: 13 ago. 2010.

____. **Android.** [S.l.], [2010b]. Disponível em: <http://www.openhandsetalliance.com/android_overview.html>. Acesso em: 28 mar. 2010.

SAUVÉ, Jacques P. **Frameworks.** Campina Grande, [2010]. Disponível em: <http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/frame/oque.htm>. Acesso em: 11 nov. 2010.

W3. **Forms in HTML documents**. [S.l.], 2010. Disponível em: <http://www.w3.org/TR/html401/interact/forms.html>. Acesso em: 24 set. 2010.

WEISS, Marco A. de O.. **MAPBR**: estudo sobre a plataforma Android com foco na manipulação de mapas usando interfaces de programação de aplicativos do Google. 2008. 93 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.