UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA

ÁLVARO LUIZ LAGO DE MENEZES

SISTEMA DE VISUALIZAÇÃO E CONEXÃO DE SERVIÇOS VIA WI-FI DIRECT

VITÓRIA/ES
2014

ÁLVARO LUIZ LAGO DE MENEZES

SISTEMA DE VISUALIZAÇÃO E CONEXÃO DE SERVIÇOS VIA WI-FI DIRECT

Monografia apresentada ao Curso de Engenharia de Computação do Centro Tecnológico, Departamento de Informática, da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do título de Bacharel em Engenharia de Computação.

Orientador: Profª. Roberta Lima Gomes

VITÓRIA/ES

2014

AGRADECIMENTOS

Aos meus pais, à minha irmã, à Alyne e à minha orientadora Roberta.

RESUMO

Este trabalho consiste em definir um sistema que permita a visualização de serviços publicados numa rede *ad-hoc*, utilizando a tecnologia *WI-FI Direct* e que através desses serviços, dois ou mais dispositivos possam estabelecer uma conexão. O sistema é desenvolvido para dispositivos móveis Android, utilizando a biblioteca oferecida pela API deste sistema operacional. É estudado o desenvolvimento em *Android,* as redes WI-FI, o modo *ad-hoc* e a tecnologia *WI-FI Direct*. Também é apresentado e discutido o funcionamento da descoberta de serviços, suas arquiteturas, modos e protocolos. Por fim, como exemplo, é desenvolvido um aplicativo de *chat*, que publica um serviço na rede e após a conexão ser estabelecida, permite a troca de mensagens instantâneas entre os dispositivos.

Palavras-chave: *WI-FI Direct*, Android, Descoberta de serviços, WI-FI ad-hoc, desenvolvimento para dispositivos móveis.

ABSTRACT

This paper consists in defining a system that allows the visualization of services published in an ad-hoc network, using the WI-FI Direct technology and that through these services, two or more devices could establish a connection. That system is developed for mobile devices that run Android, using the API offered by this operational system. It's also studied Android development, WI-FI networks, ad-hoc mode and WI-FI Direct technology. Likewise, the operation of service discovery, its architectures, modes and protocols is presented and discussed in this work. Finally, as an example, a chat application is developed, that publishes a service in the network and after the connection is established, allows the exchange of instant messages between the devices.

Keywords: *WI-FI Direct*, Android, Service Discovery, WI-FI ad-hoc, mobile devices development.

SUMÁRIO

LISTA DE FIGURAS 7
1 INTRODUÇÃO 8
1.1 OBJETIVOS 10
1.2 METODOLOGIA 10
1.3 ESTRUTURA DA MONOGRAFIA 11
2 FUNDAMENTAÇÃO TEÓRICA E TRABALHOS RELACIONADOS 12
2.1 DESENVOLVIMENTO *ANDROID* 12
2.1.1 Arquitetura *Android* 13
2.1.2 Estrutura Básica de um Aplicativo 15
2.1.3 Componentes Básicos 16
2.1.4 Ciclo de vida de uma *Activity* 17
2.2 WI-FI, MODO *AD-HOC* E WI-FI *DIRECT* 19
2.2.1 Wi-Fi 19
2.2.2 Modo *ad-hoc* 19
2.2.3 Wi-Fi *Direct* 20
2.3 DESCOBERTA DE SERVIÇOS 22
2.3.1 Arquitetura 22
2.3.2 Modos de Operação 23
2.3.3 Protocolos de Descoberta de Serviços 24
2.4 TRABALHOS RELACIONADOS 25
2.5 CONSIDERAÇÕES SOBRE O CAPÍTULO 27
3 FUNCIONAMENTO E IMPLEMENTAÇÃO DO APLICATIVO 28
3.1 TECNOLOGIAS UTILIZADAS 28
3.2 FUNCIONAMENTO 29
3.3 IMPLEMENTAÇÃO 33
3.3.1 Módulo DnsSdHelper 33
3.3.2 Módulo MainApplication 37
3.3.2.1 DevicesActivity 37
3.3.2.2 DevicesListAdapter 39
3.3.2.3 ServicesActivity 40
3.3.2.4 ServicesListAdapter 41

3.3.3 Módulo ChatExample 41
3.3.3.1 ChatActivity 41
3.3.3.2 ClientSocketHandler 44
3.3.3.3 GroupOwnerSocketHandler 44
3.3.3.4 ChatManager 45
3.3.3.5 ChatInfo 46
3.4 CONSIDERAÇÕES FINAIS SOBRE A IMPLEMENTAÇÃO 47
4 CONCLUSÕES E TRABALHOS FUTUROS 48
4.1 TRABALHOS FUTUROS 48
REFERÊNCIAS 50

LISTA DE FIGURAS

Figura 1 - Arquitetura do sistema *Android* 13
Figura 2 - Pirâmide do ciclo de vida da *Activity*, ilustrando os métodos invocados nas mudanças de estado desta 18
Figura 3 - *Activity* inicial do aplicativo mostrando um dispositivo com dois serviços 29
Figura 4 - *Activity* listando os dois serviços disponíveis no dispositivo selecionado 30
Figura 5 - Pedido de conexão Wi-Fi Direct 31
Figura 6 - Tela inicial do aplicativo de *chat* 32
Figura 7 - Chat entre dois dispositivos 32
Figura 8 - Módulos da aplicação 33
Figura 9 - Diagrama de Classes do Módulo DnsSdHelper 34
Figura 10 - Diagrama de Classes do Módulo MainApplication 37
Figura 11 - Diagrama de Classes do Módulo ChatExample 41

1 INTRODUÇÃO

A evolução da tecnologia móvel permitiu o acesso a inúmeros dados e informações em qualquer lugar e a qualquer momento, revolucionando o cotidiano das pessoas, modificando suas rotinas e alterando as formas de tomarem suas decisões. Em razão dessa abrangência, o crescimento deste setor gerou diversos assuntos a serem estudados e desenvolvidos.

Devido ao avanço da tecnologia, telefonia móvel, redes sem fio e banda larga certas restrições quanto a custo, disponibilidade, padrões e segurança foram superados, tornando os dispositivos móveis cada vez mais acessíveis, superando o número de pessoas em 2013 com projeção de aproximadamente 1,4 dispositivos por habitante em 2017 (Cisco, 2013).

O poder de processamento desses dispositivos evoluiu em um nível antes considerado ficção científica. Os aparelhos atuais têm poder de processamento superior a toda a NASA em 1969 quando esta enviou dois astronautas à Lua (KAKU, 2011). Paralelamente ao poder de processamento, a necessidade de conectividade entre os aparelhos implicou na evolução desta área. Tecnologias como *Wi-Fi, Bluetooth, NFC, DLNA* possibilitam a conexão para transferência de dados e redes de comunicação entre esses dispositivos.

O padrão 802.11, criado em 1997 no Instituto de Engenheiros Eletricistas e Eletrônicos (IEEE) com especificações de implementação de redes locais sem fio e atualizado ao longo do tempo, foi utilizado como base para os produtos *Wi-Fi* atuais. A marca *Wi-Fi* foi criada pela *Wi-Fi Alliance*, organização global não lucrativa, formada em 1999 por líderes da indústria de tecnologia em redes, com o intuito de dirigir a adoção de redes *wireless* de alta velocidade e se tornou o termo mais utilizado quando nos referimos a este tipo de rede. Em três anos o número de membros da organização alcançou a marca de 100 empresas, incluindo as principais fabricantes de eletrônicos mundiais. Atualmente uma em cada dez pessoas utilizam *Wi-Fi* de diversas maneiras em

sua rotina e a organização conta com mais de 500 empresas membro (Wi-Fi Alliance, 2013).

A tecnologia *Wi-Fi* permite o estabelecimento de redes sem fio sem a necessidade de uma infraestrutura de rede. Este modo de conexão é conhecido como *ad hoc*, sendo muito útil em cenários onde a infraestrutura é inexistente, inacessível ou possui falhas como: falta de energia, queda ou sobrecarga de rede e utilização em localidades isoladas. Também se busca uma simplicidade na transmissão de dados, tornando este procedimento o mais simples possível para o usuário.

O modo *ad hoc* foi definido no padrão 802.11, utilizado para constituir uma rede ponto a ponto, onde cada nó desempenha os papéis de cliente e ponto de acesso (CHLAMTAC, 2003). Entretanto esse padrão de conexão possui limitações que desestimulam sua utilização, dificuldades da configuração da rede e banda muito limitada (11 Mbps) são alguns exemplos.

Ciente desse cenário a *Wi-Fi Alliance* criou um novo padrão de comunicação *ad hoc*, o *Wi-Fi Direct,* que utiliza a tecnologia *Wi-Fi* e é totalmente compatível com as redes de comunicação anteriores, com ou sem infraestrutura, utilizando o *hardware* atual. Além disso, permite uma conexão segura mais fácil e é capaz de alcançar velocidades de até 250 Mbps, com um alcance de até 200 metros (Wi-Fi-Direct FAQ, 2010).

Atualmente os dois sistemas operacionais mais utilizados nos dispositivos móveis são o *iOS e Android (NetMarketShare, 2013)*. Para este trabalho, utilizaremos apenas o *Android*, desenvolvido pela *Google*, atualmente na versão 4.4, por este ser um sistema de código aberto, com suporte ao *Wi-Fi Direct* desde sua versão 4.0 e pela maior disponibilidade de dispositivos no ambiente onde o trabalho foi desenvolvido.

As redes *ad-hoc*, principalmente as redes móveis, também chamadas de *Mobile Ad-hoc Networks* (MANETs), representam o cenário máximo de dinamismo, onde a rede é operada sem suporte de infraestrutura. Neste

ambiente, diferentes nós oferecendo diferentes serviços, entram e saem da rede a qualquer momento. Para a utilização otimizada dos recursos compartilhados na rede uma tecnologia que possibilite uma descoberta de serviços eficiente e oportuna é crucial. (LENDERS, et.al., 2005)

Dado o dinamismo de uma MANET, as informações dos serviços devem estar descentralizadas, sendo possível aos dispositivos localizarem automaticamente e anunciarem seus serviços para o resto da rede. (CHO, et.al., 2005)

1.1 OBJETIVOS

Este trabalho tem como objetivo geral o estudo de tecnologias de redes *ad-hoc* e mecanismos de descoberta de serviços utilizando essas tecnologias. Este objetivo surgiu primeiramente do interesse em se estudar a viabilidade do modo *ad-hoc* numa configuração *multi-hop*, em que os nós trabalham como roteadores entre redes *ad-hoc*, e desdobrou-se nos seguintes objetivos específicos:

- Pesquisar as tecnologias atuais para conexão *ad-hoc*;
- Estudar o padrão *Wi-Fi Direct* para conexão de dispositivos móveis;
- Estudar protocolos de descoberta de serviços no contexto do trabalho;
- Estudar uma plataforma móvel que possibilitasse colocar em prática os estudos realizados no trabalho;
- Implementar uma aplicação móvel que utilizasse os conceitos estudados.

1.2 METODOLOGIA

A fase inicial do projeto consistiu em leitura e análise de obras relacionadas a plataformas móveis, especialmente *Android*, redes sem fio, o modo *ad-hoc* e descoberta de serviços e seus padrões.

Após este estudo, foi realizada uma pesquisa mais específica em cima do *Wi-Fi Direct*, seguida de testes com essa tecnologia em cima da plataforma *Android*. Nesta etapa, foi verificado o suporte entre as versões do sistema e as implementações do modo *Wi-Fi Direct* nos dispositivos disponíveis.

A última etapa do trabalho consistiu no desenvolvimento do aplicativo para visualização e conexão de serviços, visando demonstrar as funcionalidades e a aplicabilidade das tecnologias estudadas.

1.3 ESTRUTURA DA MONOGRAFIA

Esta monografia está estruturada em cinco capítulos, incluindo esta introdução, e está organizada da seguinte forma:

- No Capítulo 2 encontra-se a fundamentação teórica dos assuntos abordados neste trabalho.
- O Capítulo 3 apresenta os estudos iniciais de viabilidade de projetos utilizando o *Wifi-Direct.*
- O Capítulo 4 contém a definição e a implementação do aplicativo de visualização e conexão de serviços.
- Encerrando com o Capítulo 5 em que se encontra a conclusão e trabalhos futuros sugeridos.

2 FUNDAMENTAÇÃO TEÓRICA E TRABALHOS RELACIONADOS

Neste capítulo são apresentados os conceitos necessários para o entendimento e o desenvolvimento do trabalho. A seção 2.1 trata do desenvolvimento no *Android*, sua arquitetura, sua estrutura de desenvolvimento, componentes básicos e o ciclo de vida de uma aplicação. A seção 2.2 trata da tecnologia Wi-Fi Direct. Na seção 2.3 é explicado um pouco do funcionamento da descoberta de serviços e por fim, na seção 2.4 são mostrados alguns trabalhos relacionados.

2.1 DESENVOLVIMENTO *ANDROID*

Esta seção trata dos conceitos básicos necessários para o desenvolvimento na plataforma *Android*.

A implementação foi feita na plataforma móvel Android, devido à forte presença deste sistema operacional nos dispositivos móveis atuais e por apresentar vantagens como código aberto. Também é a plataforma utilizada no laboratório onde foi desenvolvido o projeto (LPRM).

A plataforma de desenvolvimento foi feita para abstrair diferenças de *hardware*, isolando a parte de *software*, o que facilita o desenvolvimento de aplicativos para diversos dispositivos. Os recursos dos dispositivos estão em estruturas de alto nível, facilitando seu acesso. (STEELE, 2010)

Além disso, a linguagem base de desenvolvimento é Java, amplamente utilizada para o desenvolvimento, possuindo desta forma uma grande comunidade de desenvolvedores, facilitando a pesquisa de materiais de apoio.

2.1.1 Arquitetura *Android*

O *Android* é uma plataforma de *software*, além de apenas um sistema operacional, essencialmente formada por uma pilha de componentes de *software* como é mostrado na Figura 1 a seguir.

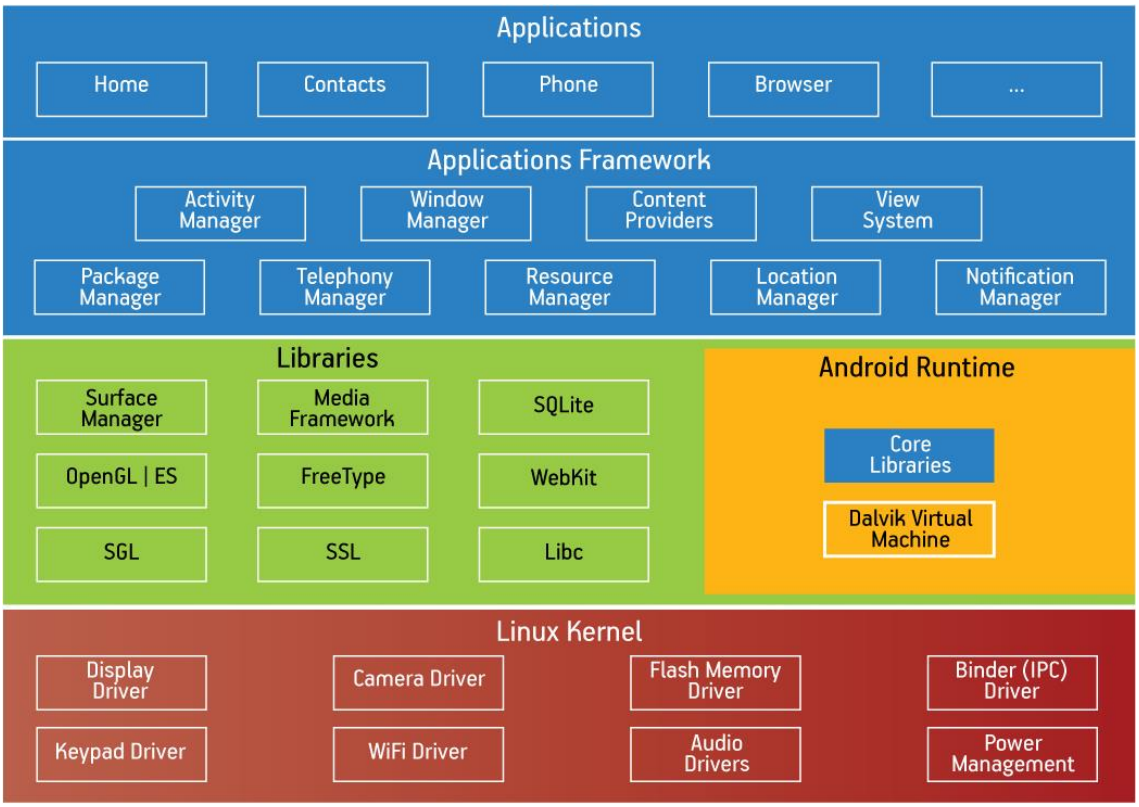

Figura 1 - Arquitetura do sistema *Android*

Fonte: http://www.techdesignforums.com/practice/technique/android-beyond-the-handset/

Na base da plataforma temos o *kernel* Linux, responsável pelas funcionalidades essenciais do sistema, como gerência de processos e memória e segurança. O *kernel* cuida de tarefas relativas a redes, processamento de dados e possui uma vasta coleção de *drivers*.

Um conjunto de bibliotecas está acima do *kernel*, essencialmente desenvolvidas em C ou C++, dando suporte a áudio, vídeo, gráficos e a banco de dados. Nessa altura da pilha está o componente chave do *Android*, a *Dalvik VM*. Os programas feitos em Java e compilados em *bytecode* são convertidos

do formato *.class*, compatíveis com as máquinas virtuais *Java*, para o formato *.dex* (*Dalvik Executable*), compatível com a máquina virtual *Dalvik*. Diferentemente das máquinas virtuais *Java*, que se baseiam em pilha, a *Dalvik* utiliza uma arquitetura baseada em registradores.

Na conversão dos arquivos *.class* para *.dex*, feita pela ferramenta *dx*, vários cuidados são tomados para conservar espaço de memória, como retirar constantes iguais utilizadas em diferentes classes. Este passo é importante, pois o *Android* aborda o multiprocessamento (capacidade do sistema executar simultaneamente dois ou mais processos) utilizando múltiplas instâncias dessa máquina virtual, tornando a eficiência de espaço desta um aspecto crítico.

A camada de *Framework,* por sua vez, fornece serviços básicos às aplicações, por meio de classes *Java*. Dentro outros aspectos, ela fornece acesso em alto nível aos recursos do dispositivo, como telefonia, serviços de localização, rede, atividades e notificações.

No topo da pilha encontram-se as aplicações nativas ou de terceiros. As mesmas são escritas em Java e executadas na máquina virtual *Dalvik*.

Também é disponibilizado o *Android* NDK, para alguns tipos de aplicação onde é interessante o reuso de códigos C e C++. O NDK é um conjunto de ferramentas que permitem a implementação de partes do aplicativo nessas linguagens, no código nativo do *Android* para o processador ARM. Possibilita também uma melhor performance no caso de aplicativos que necessitam de alta utilização de processamento.

O NDK provê bibliotecas padrões estáveis como libc (biblioteca C), libm (biblioteca de funções matemáticas), OpenGL ES (biblioteca para gráficos 3D), bibliotecas de compressão, log, entre outras. (Android NDK)

2.1.2 Estrutura Básica de um Aplicativo

Um aplicativo para *Android* tem basicamente duas camadas, a de apresentação, escrita em *XML* e a camada responsável pelas funcionalidades do sistema, em *Java*. (Android Masterclass)

Todo projeto no *Android* possui as seguintes pastas (Basic structure of an Android project):

- *src:* Contém os arquivos com os códigos *Java.*
- *gen:* Bibliotecas geradas pelo Java, apenas para uso interno do *Android.*
- *Res*: Pasta com os recursos (*Resources)* como imagens, arquivos *XML* dos layouts, vídeos. Dentro desta pasta temos as seguintes divisões:
    - *Drawable:* Onde ficam armazenados os diversos arquivos gráficos. Como existem dispositivos *Android* com variadas resoluções, por padrão existem diversas versões desta pasta como d*rawable-mdpi, drawable-hdpi,* entre outras.
    - *Layout:* Local onde são armazenados os arquivos de *layout*, em *XML*. Estes definem como os objetos das telas são organizados nestas.
    - *Values:* Arquivos *XML* que armazenam várias *strings* utilizadas na aplicação.

Além disso, todo projeto *Android* possui três arquivos indispensáveis:

- *AndroidManifest.xml:* Arquivo com todas as definições requeridas pelo *Android.* Contém informações sobre a aplicação como versão mínima do *Android* requerida pela aplicação, permissões de acesso às funcionalidades do dispositivo, ícone, nome e versão da aplicação.
- *MainLayout.*xml: Não necessariamente com este nome, este arquivo descreve o *layout* inicial da aplicação. Existe uma referência a este arquivo no *AndroidManifest.*

- Classe da *Activity:* Toda aplicação que ocupe toda a tela do dispositivo deve ter ao menos uma classe que herde da classe *Acitivity*. Esta possui um método principal chamado *OnCreate*, que inicia a aplicação e carrega o *layout* da página. Esta classe deve ser referenciada no arquivo *AndroidManifest*.

2.1.3 Componentes Básicos

Qualquer ferramenta possui um grupo básico de componentes, para que se entenda minimamente os componentes, esses componentes estão resumidos abaixo. Nem todos são acessados pelo usuário e alguns são interdependentes, porém cada um existe como uma entidade única e possui um objetivo específico: (Application Fundamentals)

- *Activities:* Uma *activity* representa uma tela do sistema. Temos como exemplo o aplicativo de telefone do próprio SO, este possui uma *activity* para discagem e outra para os contatos. Apesar de interagirem entre si, cada uma é independente.
- *Services:* Um *service* é um componente que executa em segundo plano, e não possui uma *interface*. É usado, por exemplo, para buscar dados na rede sem bloquear o acesso do usuário à *activity.*
- *Content providers:* Um *content provider* organiza conjuntos de dados utilizados pelas aplicações. Ele permite o armazenamento de dados em arquivos, banco de dados, na *web*, ou qualquer outro local de persistência de dados. Por meio do *content provider* é possível compartilhar dados e acessar dados de outras aplicações.
- *Broadcast receivers:* Por último um *broadcast receiver* é o componente responsável pela resposta a *broadcasts* feitos ao longo do sistema como um todo por meio do método *sendBroadcast()*, por exemplo, um *broadcast* avisando que a tela foi desligada ou que a bateria está acabando. Uma aplicação além de receber um *broadcast* pode também gerar um, com intuito de anunciar por todo o sistema alguma ação.

Assim como os outros componentes, exceto as *activities*, um *broadcast receiver* não possui *interface* e é normalmente utilizado como um intermediário entre outros componentes.

- *Intent* é uma descrição abstrata de uma operação a ser realizada como lançar uma *Acitivity* ou *Service*, para fazer um *broadcast* de uma mensagem. Podem ser iniciados pelo usuário ou pelo sistema.
- *IntentFilter* é uma descrição estruturada de valores de *Intents* que podem ser ações, categorias e dados. Esses *Intents* são enviados pelo método do sistema *sendBroadcast()*. (IntentFilter, 2014)
- *Listeners* são interfaces que possuem um ou mais métodos utilizados em *callbacks*.

2.1.4 Ciclo de vida de uma *Activity*

Os aplicativos *Android* são iniciados sempre a partir de uma instância de uma *Activity*, invocando métodos de *callback* que acionam a *Activity*.

Os estados da *Activity* são geralmente descritos em forma de uma pirâmide (Figura 2): o topo desta significa o ponto em que a *Activity* está em primeiro plano; a cada passo do usuário em direção a deixar a *Activity*, o sistema chama métodos em direção à base da pirâmide. Em alguns casos, a *Activity* move-se parcialmente em direção à base da pirâmide e fica em estado de espera, podendo retornar ao topo, retomando o estado onde o usuário estava (Starting an Activity).

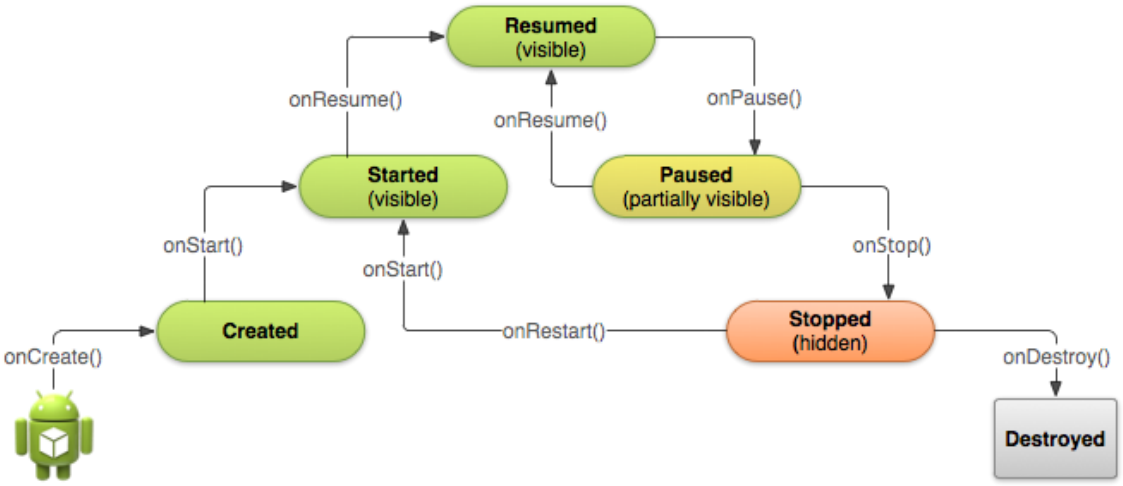

Figura 2 - Pirâmide do ciclo de vida da *Activity*, ilustrando os métodos invocados nas mudanças de estado desta

Fonte: http://developer.android.com/training/basics/activity-lifecycle/starting.html

Apesar da Activity transitar entre todos estes estados, ela só permanece por um período maior de tempo em três deles:

- *Resumed*: A *Activity* está em primeiro plano e há interação com o usuário.
- *Paused*: A *Activity* está parcialmente obscurecida por outra A*ctivity* que está em primeiro plano, esta é semi-transparente ou não cobre toda a tela. Neste estado nenhum código pode ser executado, nem há interação com o usuário.
- *Stopped*: A *Activity* está completamente escondida e não é visível para o usuário. Está completamente em segundo plano. Neste estado nenhum código pode ser executado, nem há interação com o usuário.

2.2 WI-FI, MODO *AD-HOC* E WI-FI *DIRECT*

2.2.1 Wi-Fi

O padrão IEEE 802.11, conhecido como *Wi-Fi* é o responsável por descrever as operações de transmissão de dados sem fio nas frequências de 2.4 e 5 GHz. Utilizando-se esse padrão, é possível formar uma rede local entre dispositivos que possuam o *hardware* compatível e redes físicas estruturadas.

Nas redes infraestruturadas, um dispositivo (ponto de acesso) age centralizando o controle de tráfego e conexão à rede sem fio, também sendo responsável por rotear esta a qualquer outra rede existente, inclusive as cabeadas. O número de pontos de acesso e dispositivos que utilizam as interfaces compatíveis com o padrão é crescente e dada a constante atualização do padrão, este deve permanecer como referência durante algum tempo. Efetivamente, este padrão tornou-se o principal em se tratando de conectividade sem fio para redes locais.

2.2.2 Modo *ad-hoc*

Tendo em vista a larga adoção dos rádios *Wi-Fi* em dispositivos variados, um avanço natural seria a conexão entre dispositivos, em qualquer lugar, sem necessidade de uma infraestrutura. Para tal, foi pensado já no padrão IEEE 802.11, uma solução para criação de uma rede sem um administrador central. P.orém este modo de conexão sofre limitações como oferecer mínima segurança na rede comparado ao modo de conexão com infraestrutura, o nível de sinal não é indicado, tornando mais difícil a melhor localização para a conexão e suporte a uma banda de apenas 11 Mbps.

Apesar da conexão sem dependência de uma infraestrutura ser uma ideia promissora, devido às limitações do modo, esta se tornou apenas uma

contingência para situações em que uma rede centralizada não pode ser instaurada.

2.2.3 Wi-Fi *Direct*

Mais de uma década após seu *design* inicial, o padrão IEEE 802.11 ou *Wi-Fi* tornou-se um dos meios mais comuns de acesso a redes. Entretanto, com o intuito de manter o sucesso, a tecnologia precisa evoluir. A conexão *peer-to-peer* já era possível desde a origem do padrão, utilizando-se do modo de operação *ad-hoc,* todavia este modo nunca foi amplamente utilizado.

A tecnologia *Wi-Fi Direct* apresenta uma abordagem diferente para a conexão *peer-to-peer*. Ao contrário do modo *ad-hoc* em que o usuário criava a rede em um dispositivo e este se tornava o ponto de acesso, no *Wi-Fi Direct* os dispositivos negociam entre si qual deles terá as funcionalidades de ponto de acesso, normalmente de acordo com a disponibilidade de recursos. A tecnologia também manteve as melhorias desenvolvidas para as redes que utilizam infraestrutura, como economia de energia e mecanismos de segurança.

Na conexão dos dispositivos utilizando o *Wi-Fi Direct,* estes formam grupos (*P2P Groups*), como seriam formados nas redes com infraestrutura, um dos dispositivos assume a posição de ponto de acesso e é referido como *P2P Group Owner (P2P GO),* os outros dispositivos são chamados de *P2P Clients.* Como dito, os papéis na rede são normalmente negociados na formação do *P2P Group.* Após um grupo ser formado, outros clientes podem se unir a rede.

A formação dos grupos pode ser feita de diversas maneiras pelos dispositivos, dependendo da negociação ou não do *P2P Group Owner* ou se já existem informações de segurança previamente compartilhadas sobre o grupo.

Um diferencial do *Wi-Fi Direct* é o suporte à descoberta de serviços na camada mais baixa do *TCP/IP*, possibilitando acesso às informações de conjuntos de

serviços disponíveis antes da formação do *P2P Group.* A partir disto o dispositivo pode decidir se formará o grupo ou não. A implementação desta funcionalidade utiliza o *Generic Advertisement Protocol* (GAS), especificado no 802.11u, para transportar as consultas geradas nas camadas mais altas, por meio de protocolos como SSDP ou DNS-SD, para as camadas mais baixas. O GAS é um protocolo de consulta e resposta implementado utilizando-se *action frames*, permitindo a comunicação entre dois dispositivos 802.11 não associados.

Na utilização do *Wi-Fi Direct* os dispositivos agem normalmente como *Group Owner* fazendo o papel de um roteador, portanto a utilização eficiente de energia, principalmente nos dispositivos móveis é de grande importância. Apesar dos mecanismos de economia de energia estarem apenas definidos para os clientes, nas redes *Wi-Fi* tradicionais (também utilizados no *Wi-Fi Direct*), foram definidos dois novos mecanismos de economia de energia: o *Opportunistic Power Save protocol* e o *Notice of Absence (NoA) protocol* para o *P2P GO*, baseados nos períodos de ausência dos clientes na rede.

As razões da escolha desta nova tecnologia para realização de conexões entre os dispositivos foram:

- A presença desta tecnologia numa biblioteca que disponibiliza funções de alto nível para conexão e comunicação via essa tecnologia.
- Suporte à conexões de rede sem infraestrutura com taxas de transferências muito superiores às fornecidas pelas conexões *ad-hoc* Wi-Fi anteriores; (CONTI, 2013)
- Melhor gestão de energia; (CAMPS-MUR, 2011)
- Suporte a tecnologias mais atuais de criptografias, contribuindo para o estabelecimento de conexões mais seguras. (CONTI, 2013)
- Biblioteca nativa para descoberta de serviços utilizando esta tecnologia.

2.3 DESCOBERTA DE SERVIÇOS

No contexto de redes, qualquer funcionalidade fornecida por um dispositivo que pode ser útil a outro pode ser chamado de serviço. Este pode ser um serviço de *software,* como um algoritmo de decodificação que ao ser requisitado por outro dispositivo pode ser utilizado ou um serviço de *hardware*, como uma impressora permitindo a utilização via rede a um dispositivo.

Nas redes sem fio os dispositivos se movimentam livremente. Especialmente nas redes *ad-hoc*, as limitações para que um dispositivo seja móvel são que ele seja leve e pequeno ao máximo, por isso eles possuem menos recursos que os dispositivos fixos. Torna-se importante, portanto, a utilização de recursos disponibilizados por outros dispositivos. Porém, para utilizá-los e formar uma rede, requer-se conhecimento dos serviços disponíveis, em quais dispositivos eles se encontram e como interagir com eles. Esses aspectos são o foco dos protocolos de descoberta de serviços, não apenas possibilitando localizar serviços particulares, mas também publicar um serviço, invocar um serviço, selecionar um serviço dentre mais de um disponível do mesmo tipo.

2.3.1 Arquitetura

A arquitetura dos protocolos de descoberta de serviço depende, sobretudo, de haver ou não um diretório. Um diretório é uma estrutura responsável por armazenar informações sobre os serviços disponíveis numa rede, auxiliando o processo de descoberta de serviços. A respeito dessa característica um protocolo pode ser baseado em diretórios, sem diretórios ou híbrido. (VERVERIDIS, POLYZOS, 2008)

Nas arquiteturas baseadas em diretórios um ou alguns nós da rede são escolhidos para serem coordenadores de serviço. Estes anunciam sua presença na rede periodicamente por meio de um *flood* de anúncios. Esse

flooding é limitado a certo número de saltos, determinado por um parâmetro no anúncio.

Nas arquiteturas sem diretórios não existe essa atribuição de coordenadores de serviço, para descobrir um serviço o cliente faz um *broadcast* do pedido de descoberta de serviço aos próximos nós da rede. Da mesma forma, os saltos são limitados por um parâmetro do anúncio.

Nas arquiteturas híbridas, os serviços são registrados em diretórios, caso estes sejam encontrados. Caso contrário, o serviço é apenas anunciado pela rede por meio de um *broadcast*. As consultas a serviços são feitas aos diretórios que o cliente conheça, caso ele não esteja ciente de nenhum diretório a consulta é transmitida na rede por meio de um *broadcast*.

No caso de uma rede com alto grau de mobilidade e baixa frequência de requisição de serviços uma arquitetura sem diretórios foi provada mais eficiente. Entretanto, se a mesma rede se diferenciar por uma alta frequência de requisição de serviços, uma arquitetura baseada em diretórios pode ser mais eficiente. Portanto, a escolha de uma arquitetura de descoberta de serviços deve ser feita baseando-se nas características de uma rede (VERVERIDIS, POLYZOS, 2008)

2.3.2 Modos de Operação

Além da arquitetura presente, o protocolo pode operar de três modos diferentes: reativamente, pró ativamente ou de modo híbrido. No modo reativo, o cliente faz uma consulta sob demanda aos provedores de serviços ou à estrutura de diretórios. No modo pró-ativo os provedores anunciam seus serviços aos diretórios ou aos potenciais clientes em tempos de intervalo discreto. No modo híbrido a comunicação pode ser feita tanto pró-ativamente quanto reativamente, o provedor pode anunciar seus serviços e o cliente pode fazer requisições.

2.3.3 Protocolos de Descoberta de Serviços

Os protocolos de descoberta de serviços são padrões desenvolvidos com o intuito de simplificar a detecção de dispositivos e seus serviços em uma rede. São exemplos de protocolos de descoberta de serviços:

- **Bluetooth Service Discovery Protocol (SDP):** Utilizado para permitir aos dispositivos descobrirem quais serviços cada um suporta e quais os parâmetros necessários para conectarem-se. Determina os perfis suportados, por exemplo, numa conexão a um *headset bluetooth*. Cada serviço é identificado por um UUID (*Universally Unique Identifier*), o que permite uma consulta a um serviço específico ou a vários, utilizando um UUID referente à raiz dos serviços desejados. (Service Discovery, Bluetooth)
- **DNS Service Discovery (DNS-SD):** Componente do conjunto de tecnologias *Zero Configuration Networking*. Permite que os clientes descubram uma lista de nomes dos serviços por tipo, utilizando requisições padrões do DNS (*Domain Name System*) sem nenhuma configuração especial. Os registros de tipos de serviço são mantidos e publicados em dns-sd.org. É um protocolo simples e leve, adequado para pequenas redes e dispositivos móveis. (VERVERIDIS, POLYZOS, 2008)
- **Simple Service Discovery Protocol (SSDP):** O SSDP é um protocolo utilizado como base para o protocolo de descoberta do UPnP (*Universal Plug and Play*). O UPnP é um conjunto de protocolos que busca expandir o conceito *Plug and Play* para o ambiente de redes. O SSDP visa atender redes residenciais ou de pequenos escritórios. As entidades básicas consideradas no protocolo são os *control points* (entidade opcional que age como um diretório do serviço) e os dispositivos conectados. A descoberta se dá utilizando HTTP em *unicasts* ou *multicasts* sobre UDP. (VERVERIDIS, POLYZOS, 2008)

- **Jini:** Jini é o protocolo de descoberta de serviço desenvolvido pela *Sun* para ser utilizado entre dispositivos compatíveis com Java. Dentre os componentes centrais do Jini estão os *lookup servers.* Estes anunciam sua presença em resposta a um *multicast* e agem como diretórios do serviço, armazenando serviços publicados e também respondendo às requisições dos clientes. (VERVERIDIS, POLYZOS, 2008)

2.4 TRABALHOS RELACIONADOS

Durante o levantamento bibliográfico foram encontrados alguns trabalhos que tratam de descoberta de serviços em redes móveis, outros relativos a experimentações utilizando o *Wi-Fi Direct*, porém pouco foi encontrado sobre os dois assuntos interligados. Os trabalhos mais relevantes serão descritos a seguir.

Em Oliveira (2011) um estudo muito similar ao feito neste trabalho foi realizado, dando enfoque à descoberta de serviços em redes *ad-hoc* e dispositivos móveis. Na época de publicação do trabalho, a plataforma Android era muito recente e ainda não dava suporte à conexão via *wi-fi direct,* nem a protocolos de descoberta de serviços. A solução foi a utilização de uma API Java, a JmDNS que mostrou-se pouco estável pela utilização da classe *InetAddress*, causando uma falha na resolução dos serviços. Além disso, não foi possível a criação da rede *ad-hoc* pelos dispositivos móveis, tornando necessário o uso de um terceiro aparelho intermediando as conexões e criando a rede.

Em Pitkänen, Kärkkäinen, Ott (2012), é feito um estudo sobre os diversos fatores que afetam a descoberta de serviços disseminados em redes móveis, utilizando os dispositivos existentes na época e levando-se em conta suas limitações práticas e o impacto destas à descoberta de serviços. Para isso são usadas simulações de situações do mundo real.

Ververidis, Polyzos (2008) discorre sobre problemas e tecnologias na descoberta de serviços em redes móveis *ad-hoc,* serve de grande fonte de conhecimento para este trabalho principalmente na parte referente à descoberta de serviços e seu funcionamento. O artigo traz resumidamente as características de alguns protocolos e suas arquiteturas e as compara. Apesar de ser muito útil como informação, esse trabalho não traz nenhuma visão prática do ambiente e traz à discussão algumas questões a serem revistas quanto à descoberta de serviços nas redes móveis.

A comunicação entre dispositivos utilizando *Wi-Fi Direct* é testada em Camps-Mur, Garcia-Saavedra, Serrano (2013). A arquitetura da tecnologia é explicada, informação que não foi encontrada em nenhum outro artigo pesquisado, assim como as formas de formações dos grupos de dispositivos na rede. Também há nesse artigo informações quanto à segurança da rede *Wi-Fi Direct* e os métodos utilizados para diminuir o consumo de energia dos dispositivos. Nas experimentações são analisados os *delays* de formação de grupos e a eficiência da tecnologia na utilização de energia. Mesmo sendo praticamente o único artigo relevante tratando-se do *Wi-Fi Direct*, ele não discorre sobre nenhum projeto prático utilizando a tecnologia.

O único trabalho encontrado que relacionou os dois assuntos foi um exemplo da equipe de suporte ao Android, disponibilizado como um *sample* da utilização da descoberta de serviços, desenvolvido durante este trabalho. Este utiliza o *Wi-Fi Direct* e a descoberta de serviços entre dispositivos, permitindo a conexão dos dispositivos, caso estes possuam um serviço específico. Neste exemplo apenas o serviço específico registrado por esse mesmo aplicativo é buscado, com o intuito de demonstrar a funcionalidade da biblioteca.

2.5 CONSIDERAÇÕES SOBRE O CAPÍTULO

O capítulo trouxe os conceitos fundamentais sobre a plataforma Android, as tecnologias de rede relacionadas ao trabalho e sobre descoberta de serviços. A descoberta de serviços permite um melhor uso e aproveitamento dos recursos da rede, tornando mais simples a descoberta deles pelos nós da rede, que nas redes *ad-hoc* móveis são substituídos a todo instante.

Por ser um ambiente livre e *open source*, utilizando linguagem de programação Java, possuindo suporte ao *Wi-Fi Direct* e à descoberta de serviços, o Android foi escolhido para desenvolvimento do projeto. Além desses motivos, o laboratório onde o trabalho foi desenvolvido possui alguns aparelhos disponíveis para testes. O desenvolvimento foi feito para dispositivos que utilizem a versão 4.1 ou mais recente, por ser esta a primeira a disponibilizar suporte a descoberta de serviços em redes *Wi-Fi Direct* nativamente.

A biblioteca de descoberta de serviços nativa do Android possui o DNS-SD e o UPnP como opções de protocolos. Para este trabalho foi escolhido o DNS-SD por ser um protocolo *muticast*, utilizado amplamente em diversos dispositivos, inclusive móveis e apresentar um ambiente descentralizado e que utiliza a colaboração dos componentes da rede na descoberta de serviços.

3 FUNCIONAMENTO E IMPLEMENTAÇÃO DO APLICATIVO

Para concretizar e colocar em prática os estudos realizados, uma etapa de implementação foi realizada. O foco foi a visualização dos serviços de uma forma prática por parte dos nós da rede, utilizando a plataforma *Wi-Fi Direct*. Além disso, a conexão de um exemplo através de um dos serviços foi implementada.

Neste capítulo as tecnologias utilizadas e a arquitetura e implementação da aplicação são descritas e explicadas.

3.1 TECNOLOGIAS UTILIZADAS

Utilizou-se a plataforma Android, a partir da versão 4.1, que possui suporte a descoberta de serviços em redes *Wi-Fi Direct*, com o protocolo DNS-SD.

Dentro do pacote do Wi-Fi Direct no Android, há uma biblioteca com funções pré-estabelecidas para descoberta de serviços utilizando DNS-SD e UPNP, com esta podemos registrar um serviço local na rede e disparar um pedido de descoberta de serviços na rede, utilizando um desses dois protocolos.

O Android SDK é inteiramente compatível com o Eclipse através do plugin ADT (ADT Plugin, 2014), ambiente utilizado para desenvolvimento e testes da aplicação.

A IDE também dá suporte a depuração e testes utilizando diretamente aparelhos ou emuladores Android, além de ser conhecida anteriormente como uma das principais IDEs para desenvolvimento Java.

3.2 FUNCIONAMENTO

A aplicação principal é formada por duas *Activities*, a primeira, em que o aplicativo inicia, responsável pela exibição dos dispositivos que possuam serviços publicados na rede *Wi-Fi Direct* (Figura 3).

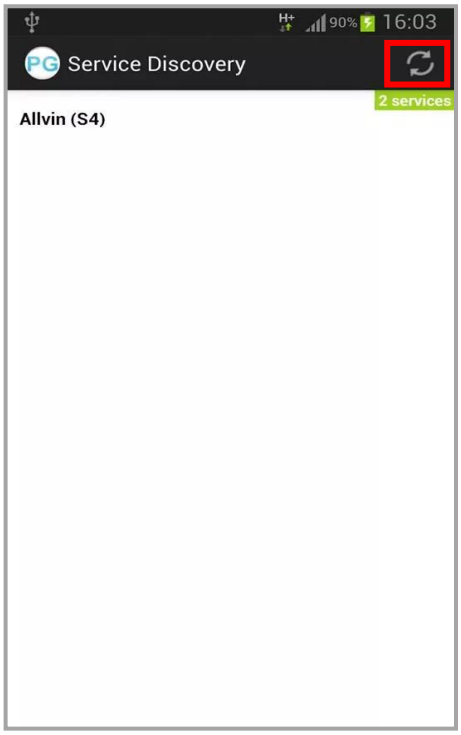

Figura 3 - *Activity* inicial do aplicativo mostrando um dispositivo que possui dois serviços publicados

Ao abrirmos o aplicativo esta *Activity* é aberta, dois serviços utilizados como exemplo são registrados (*laserwriter 8500* e *chat example*) e a descoberta de serviços é iniciada automaticamente.

Nesta tela é exibida uma lista de dispositivos que possuam serviços publicados na rede *Wi-Fi Direct*, listados por seus nomes e com a informação no canto superior direito de cada item informando quantos serviços o dispositivo publicou na rede.

Destacado em vermelho na Figura 3 está o botão responsável por atualizar a lista, executando novamente a descoberta de serviços.

Caso haja algum problema na requisição de descoberta de serviços, o aplicativo exibe uma mensagem de erro através de um *Toast*[1] para o usuário.

Ao clicar em um dispositivo listado passamos à segunda *Activity*. Esta relaciona os serviços registrados pelo dispositivo, selecionado na tela principal (Figura 4).

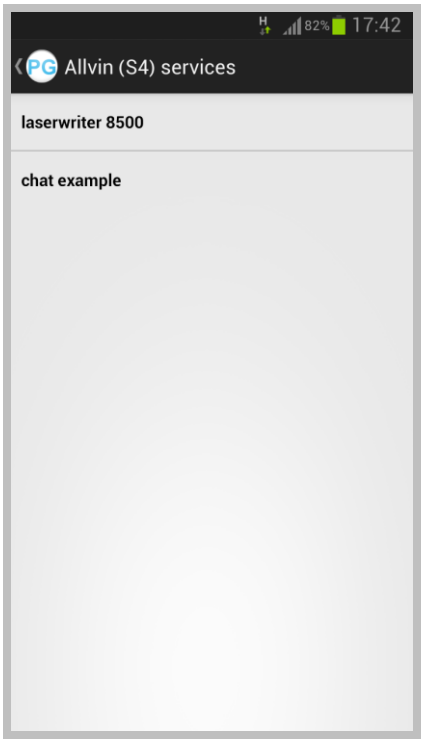

Figura 4 - *Activity* listando os dois serviços disponíveis no dispositivo selecionado

Nessa tela são exibidos os serviços publicados pelo dispositivo selecionado, utilizando os nomes com os quais estes foram registrados. Ao clicar em algum dos serviços, caso haja alguma ação implementada no aplicativo para o serviço selecionado, será iniciada uma conexão *ad-hoc* utilizando a rede *Wi-Fi Direct* entre os dispositivos. Apenas foi desenvolvido uma ação para o serviço *chat example* neste projeto.

[1] *Toasts* são utilizados para fornecerem uma informação simples sobre uma operação com um formato de um pequeno *popup* na parte inferior da tela do dispositivo e desaparecem automaticamente após um determinado período. (Toasts, 2014)

Na Figura 4 estão sendo exibidos os serviços publicados como exemplo, o primeiro representaria uma impressora conectada ao dispositivo (*laserwriter 8500*) e o segundo um aplicativo de chat (*chat example*), que é implementado no módulo *ChatExample*.

O botão de voltar dos dispositivos *Android* tal qual o localizado na parte superior esquerda da tela voltam para a *Activity* inicial, exibindo novamente a lista de dispositivos.

Ao clicarmos no serviço *chat example*, uma conexão *ad-hoc* utilizando o *Wi-Fi Direct* é solicitada ao dispositivo que publicou o serviço (Figura 5). Caso a conexão seja aceita, esta é concluída e o aplicativo de *chat* é iniciado (Figura 6).

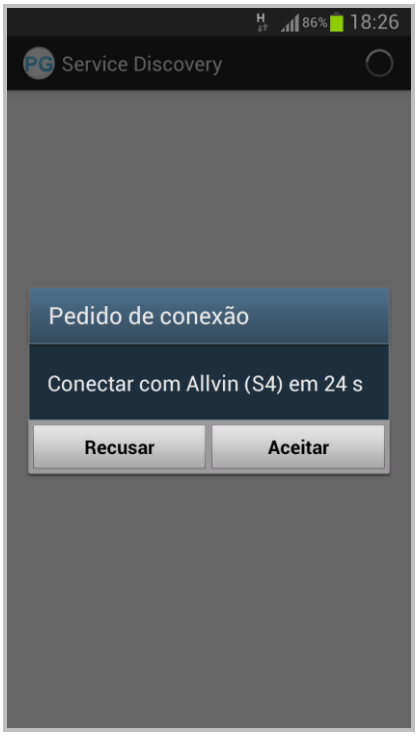

Figura 5 - Pedido de conexão Wi-Fi Direct

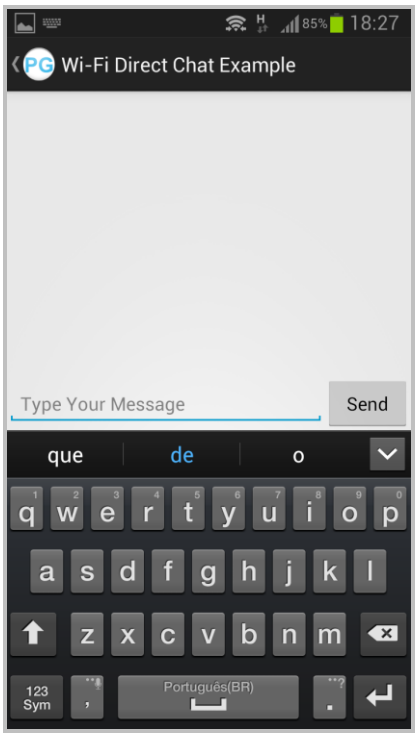

Figura 6 - Tela inicial do aplicativo de *chat*

A seguir, o funcionamento do aplicativo de *chat* é padrão, disponibilizando uma troca de mensagens entre os dispositivos (**Erro! Autoreferência de indicador não válida.**).

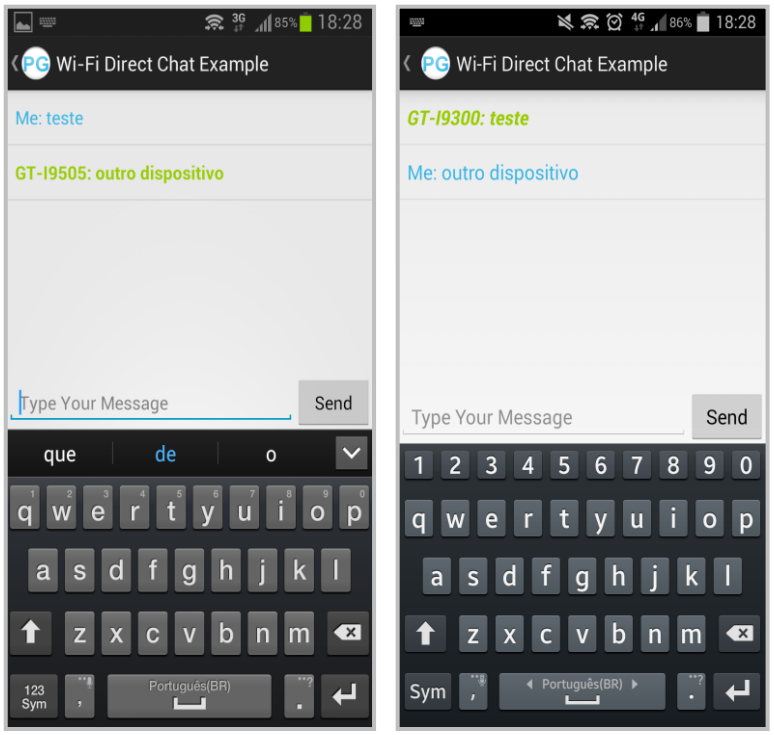

Figura 7 - Chat entre dois dispositivos

3.3 IMPLEMENTAÇÃO

Na proposta do trabalho foi definido que o sistema exibiria e se conectaria aos serviços publicados utilizando a rede *Wi-Fi Direct*. Além disso, existiria um exemplo que se utilizasse de um serviço destes para a conexão.

Os módulos da aplicação e seus relacionamentos são apresentados na Figura 8. Essa organização visou simplificar o entendimento do funcionamento do sistema independente da linguagem utilizada.

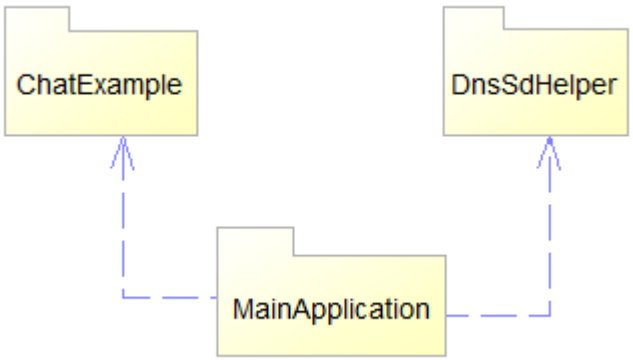

Figura 8 - Módulos da aplicação

3.3.1 Módulo DnsSdHelper

Para simplificar e modularizar o funcionamento da aplicação, toda a parte responsável pela utilização das bibliotecas relativas ao *Wi-Fi Direct*, foi separada neste módulo. Apenas serviços que utilizam o protocolo DNS-SD estão suportados por este *Helper*.

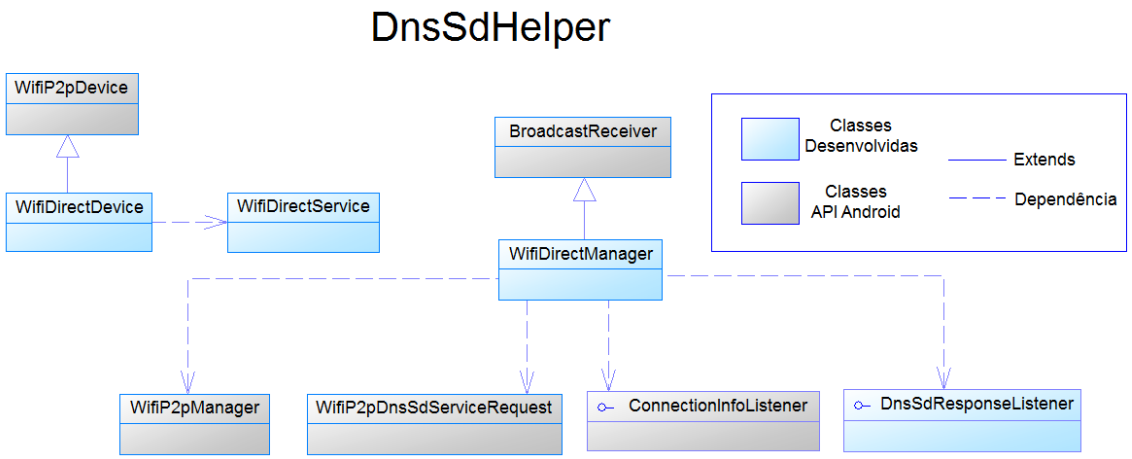

Figura 9 - Diagrama de Classes do Módulo DnsSdHelper

Na Figura 9 é possível visualizar as principais classes desenvolvidas (ou utilizadas) neste módulo. A classe *WifiDirectManager* é a responsável pelo funcionamento do módulo. Na construção da classe alguns objetos devem ser fornecidos (Código-Fonte 1): o serviço WifiP2pManager do sistema, o canal obtido na inicialização deste serviço, e dois *Listeners*[2] responsáveis pela resposta da obtenção de serviços e de informações de uma conexão *Wi-Fi Direct.* O exemplo de utilização é descrito na seção que discute o módulo *MainApplication*. A propriedade *getIntentFilter* (Código-Fonte 2) permite que a *MainApplication* tenha acesso ao *IntentFilter*[3] criado pelo *WifiDirectManager* para que seja registrado um *BroadcastReceiver*[4] com este filtro. Os métodos disponibilizados pela classe são os descritos no Código-Fonte 3.

```
public WifiDirectManager(WifiP2pManager, Channel channel,
                  final DnsSdResponseListener responseListener,
                  ConnectionInfoListener connListener){…}
```

Código-Fonte 1 - Construtor da classe WifiDirectManager

```
public IntentFilter getIntentFilter(){…}
```

Código-Fonte 2 - Propriedade getIntentFilter da classe WifiDirectManager

---

[2] *Listeners* são interfaces que possuem um ou mais métodos utilizados em *callbacks.*
[3] *IntentFilter* é uma descrição estruturada de valores de *Intents* que podem ser ações, categorias e dados. Esses *Intents* são enviados pelo método do sistema *sendBroadcast()*. (IntentFilter, 2014)
[4] *BroadCastReceiver* é uma classe base para códigos que irão receber *Intents* enviados pelo método do sistema *sendBroadcast()*. (BroadcastReceiver, 2014)

```
public void addLocalService(String serviceInstance,
                  String serviceRegType, Map<String, String> record){…}

public void discoverServices(ActionListener listener){…}

public void onPause(){…}

public void onResume(){…}

public void connect(String deviceAddress){…}

public void disconnect(){…}

public void onReceive(Context context, Intent intent){…}
```

Código-Fonte 3 - Métodos disponibilizados pela classe WifiDirectManager

A criação desses métodos tem como intenção tornar o mais transparente possível a descoberta de serviços utilizando DNS-SD e a conexão *Wi-Fi Direct* para a aplicação que a utilize:

- *addLocalService* registra um serviço local com as informações padrões do DNS-SD (nome do serviço, tipo do serviço e uma tabela *hash*[5] com informações do serviço.
- *discoverServices* faz uma chamada assíncrona ao sistema para que sejam descobertos serviços DNS-SD na rede. O retorno é dado para a interface *DnsSdResponseListener* atribuída na construção do *Manager*.
- *onPause* deve ser chamado caso o aplicativo seja pausado, remove as requisições de descoberta de serviços.
- *onResume* deve ser chamado no retorno do aplicativo, configura as requisições de descoberta de serviços.
- *connect* inicia a conexão com algum dispositivo em uma rede *Wi-Fi Direct*.
- *disconnect* desconecta o dispositivo da rede em que ele está atualmente conectado.

[5] Tabela *hash* é uma estrutura de dados especial que associa chaves de pesquisa a valores. Neste caso em especial, tanto as chaves como os valores são *strings.*

- *onReceive* é um evento utilizado pelo sistema, implementação da interface *BroadcastReceiver* e não é utilizado diretamente.

Além disso, neste módulo foram criadas duas classes representando os objetos Dispositivo (*WifiDirectDevice*) e Serviço (*WifiDirectService*) para melhor visualizarmos estes. Na implementação do *WifiDirectDevice*, há um objeto contendo uma lista de serviços para que os serviços possam ser agrupados por dispositivo (Código-Fonte 4). Já a classe *WifiDirectService* reúne as informações de um serviço DNS-SD que são disponibilizadas na descoberta de serviços (Código-Fonte 5).

```
public class WifiDirectDevice extends WifiP2pDevice {

	public ArrayList<WifiDirectService> servicesList =
			new ArrayList<WifiDirectService>();

	public WifiDirectDevice(){…}

	public WifiDirectDevice(WifiP2pDevice device){…}
}
```

Código-Fonte 4 - Membros públicos da classe WifiDirectDevice

```
public class WifiDirectService {

	public HashMap<String, String> record;

	public WifiDirectService(String fullDomainName){…}

	public String getInstanceName(){…}

	public String getRegistrationType(){…}
}
```

Código-Fonte 5 - Membros públicos da classe WifiDirectService

3.3.2 Módulo MainApplication

O módulo principal é chamado *MainApplication*. Nele estão contidas as *Activities* responsáveis pela lógica de visualização e conexão dos serviços, assim como classes de suporte à visualização destes em listas, chamadas *ListAdapters*.

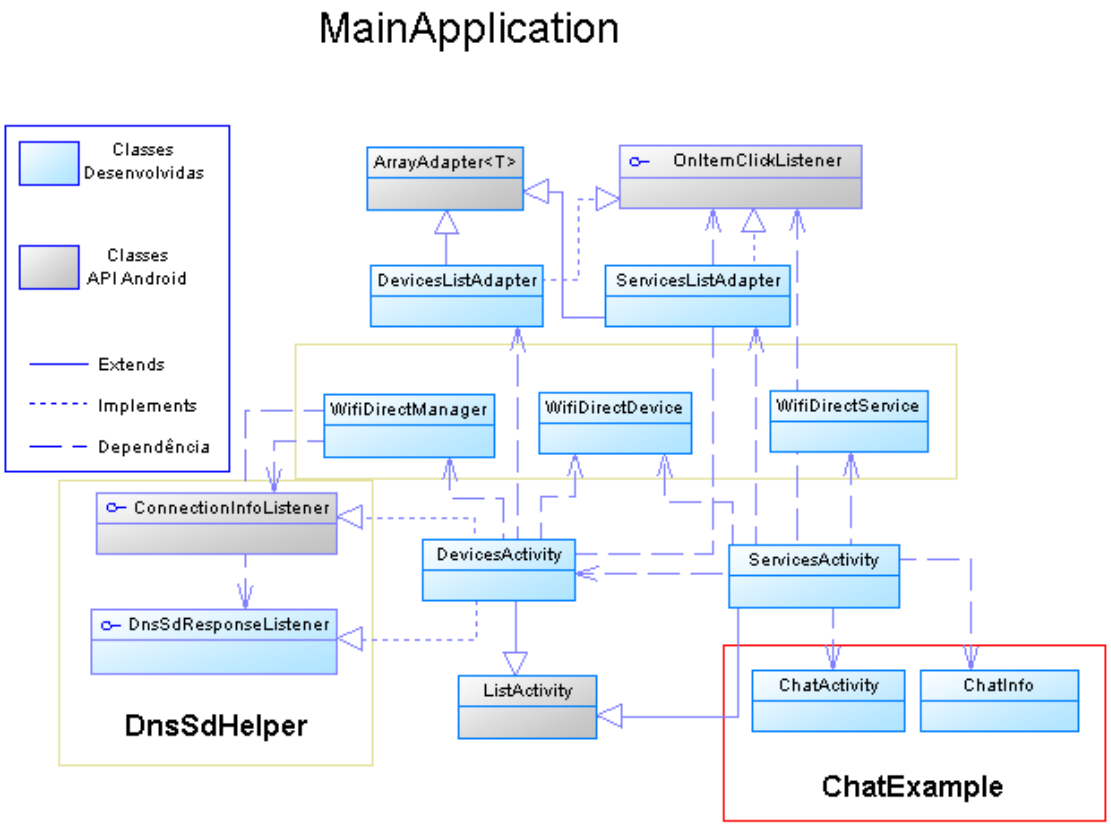

Figura 10 - Diagrama de Classes do Módulo MainApplication

3.3.2.1 DevicesActivity

A *Activity* inicial foi nomeada *DevicesActivity*. Ela é responsável principalmente pela descoberta de serviços na rede *Wi-Fi Direct* e listagem dos dispositivos com serviços publicados. Além disso, também registra dois serviços na rede, utilizados como exemplo. Na criação (*onCreate*) desta *Activity* chamamos dois métodos *setupInterface* e *setupWifiDirect*, responsáveis pela configuração inicial da *interface* e do *Wi-Fi Direct*, o segundo utilizando a classe de apoio

WifiDirectManager do módulo DnsSdHelper descrito na seção 3.2.1 (Código-Fonte 6).

```
private void setupInterface() {

	listAdapter = new DevicesListAdapter(this,
				R.layout.list_item_devices);
	setListAdapter(listAdapter);

	toastDiscovery = Toast.makeText(getBaseContext(),
				"Failed Requesting Service Discovery.",
				Toast.LENGTH_LONG);

	ListView lv = getListView();
	lv.setOnItemClickListener(listAdapter);
}

private void setupWifiDirect() {

	WifiP2pManager manager = (WifiP2pManager)
				getSystemService(Context.WIFI_P2P_SERVICE);
	Channel channel = manager.initialize(this, getMainLooper(), null);

	wifi = new WifiDirectManager(manager, channel, this, this);

	wifi.addLocalService(ChatActivity.getInstanceName(),
				ChatActivity.getRegistrationType(),
				ChatActivity.getRecords());
	wifi.addLocalService("LaserWriter 8500", "_printer._tcp", null);
}
```

Código-Fonte 6 - Implementação dos métodos setupInterface e setupWifiDirect

Em *setupInterface* é definido o adaptador que será responsável pelo controle da lista de dispositivos encontrados, criamos um *Toast* de erro de requisição de serviços e adicionamos o tratamento de clique na lista de dispositivos. Já em *setupWifiDirect* é buscado o *WifiP2pManager* do sistema (responsável pelo controle do *Wi-Fi Direct* no *Android*) e instanciado um objeto da classe de apoio *WifiDirectManager*. Esta recebe dois *callbacks* (dois últimos parâmetros) que serão tratados na classe *DevicesActivity*. Além disso, são registrados os dois serviços utilizados como exemplo, por meio do objeto da classe de apoio.

O evento *onResume* é implementado para executar o *onResume* da classe de apoio e registrar a classe de apoio como B*roadcastReceiver* com os filtros desta mesma classe. No *onPause* é feito o oposto (Código-Fonte 7). Ao

encontrar serviços registrados na rede, *onTxtRecordAvailable*, um *callback* invocado pela classe de apoio é invocado e tratado para que o item seja adicionado na lista (Código-Fonte 8).

```
protected void onResume() {

	super.onResume();

	wifi.onResume();

	registerReceiver(wifi, wifi.getIntentFilter());
}

@Override
protected void onPause() {

	super.onPause();

	wifi.onPause();

	unregisterReceiver(wifi);
}
```

Código-Fonte 7 - Implementação dos métodos onResume e onPause

```
public void onTxtRecordAvailable(String fullDomainName,
                    Map<String, String> record, WifiP2pDevice device) {

	WifiDirectService service = new WifiDirectService(fullDomainName);

	service.record = (HashMap<String, String>) record;

	addItem(new WifiDirectDevice(device), service);
}
```

Código-Fonte 8 - Implementação do *callback onTxtRecordAvailable*

3.3.2.2 DevicesListAdapter

Nesta classe está implementado o tratamento de exibição da lista de dispositivos, atribuída na criação da *Activity* e o tratamento de clique em um item da lista. Ao clicar na lista a *ServicesActivity* é iniciada e as informações do dispositivo selecionado são enviados a esta.

3.3.2.3 ServicesActivity

Nesta *Activity* são listados os serviços publicados pelo dispositivo selecionado na tela anterior. No evento *onCreate* desta, é chamado apenas o método *setupInterface* (Código-Fonte 10). No método *setupInterface,* o titulo da Activity é alterado para corresponder ao nome do dispositivo selecionado e o adaptador da lista de exibição dos serviços é atribuído com tratamento do clique, da mesma forma como é feito na *Activity* anterior. Esta *Activity* também é responsável pelo tratamento diverso devido ao serviço selecionado na lista. No caso deste projeto temos apenas o tratamento caso o serviço selecionado seja o *chat example* (Código-Fonte 11). Em *connectChat* uma conexão via *Wi-Fi Direct* é requisitada e os dados do serviço são enviados à *Activity* do aplicativo de chat. (Código-Fonte 11)

```
private void setupInterface() {

	setTitle(device.deviceName + " services");
	getActionBar().setDisplayHomeAsUpEnabled(true);

	listAdapter = new ServicesListAdapter(this,
					R.layout.list_item_services);
	setListAdapter(listAdapter);
	listAdapter.addAll(device.servicesList);
	listAdapter.notifyDataSetChanged();

	ListView lv = getListView();
	lv.setOnItemClickListener(listAdapter);
}
```

Código-Fonte 9 - Implementação do método setupInterface

```
public void connectChat(WifiDirectService service) {

	wifi.connect(device.deviceAddress);

	ChatInfo.setRecords(service.record);
}
```

Código-Fonte 11 - Implementação do método que trata o clique no serviço *chat example*

3.3.2.4 ServicesListAdapter

Similar ao *DevicesListAdapter*, esta classe é responsável pelo tratamento de exibição da lista de serviços, atribuída na criação da *ServicesActivity*. Também trata o clique em algum item da lista de serviços.

3.3.3 Módulo ChatExample

Este módulo foi criado para exemplificar a conexão via *Wi-Fi Direct* através de um serviço publicado na rede. Possui apenas uma *Activity* e assume que a conexão entre os dispositivos já tenha ocorrido e as informações do serviço tenham sido recebidas e armazenadas na classe *ChatInfo*.

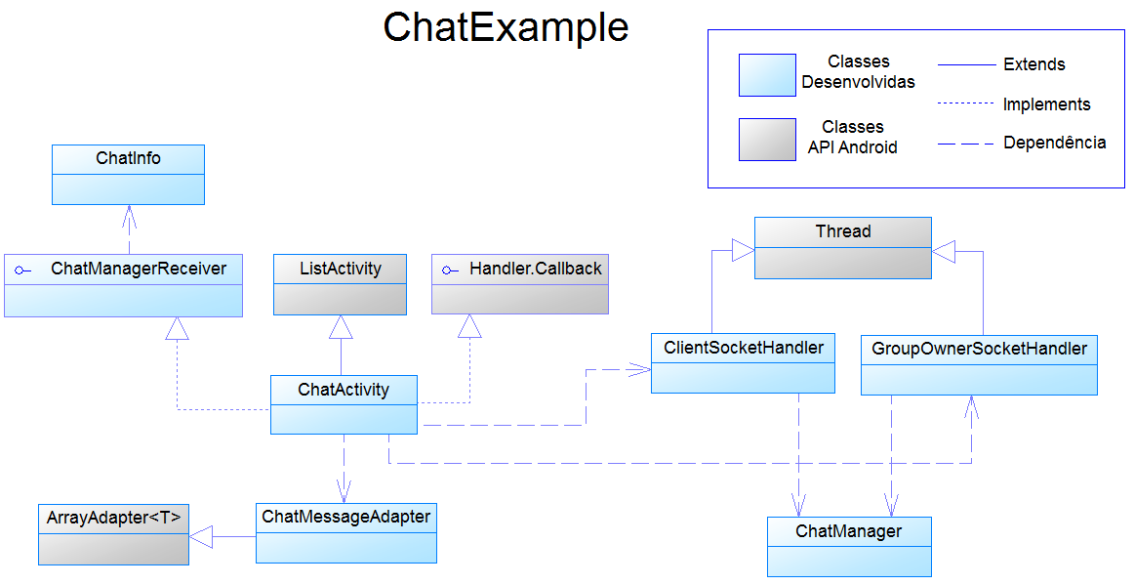

Figura 11 - Diagrama de Classes do Módulo ChatExample

3.3.3.1 ChatActivity

Ao ser aberta, esta *Activity* presume que a conexão entre os dispositivos já tenha sido concluída e que as informações da conexão estejam armazenadas na classe *ChatInfo*. Esta *Acitivity* é responsável pela exibição da troca de mensagens entre os dispositivos conectados anteriormente. No seu evento

onCreate são chamados os métodos *setupInterface* e *setupChatEngine*, este último responsável pela *engine* do *chat* (Código-Fonte 12**Erro! Fonte de referência não encontrada.**). No mesmo arquivo é definida a classe *ChatMessageAdapter* responsável pelo tratamento da exibição das mensagens na *Activity*, atribuída na criação desta.

```
private void setupInterface() {

	listAdapter = new ChatMessageAdapter(this, android.R.id.text1);
	setListAdapter(listAdapter);

	findViewById(R.id.btnSend).setOnClickListener(
		new OnClickListener() {
			public void onClick(View v) {
				btnSend_Click();
			}
		}
	);
}

private void setupChatEngine() {

	Thread thread = new Thread();

	if (ChatInfo.isGroupOwner()) {
		thread = new GroupOwnerSocketHandler(this,
					Integer.parseInt(myPort), handler);
		thread.start();
	} else {
		int port = Integer.parseInt(
				ChatInfo.getRecords().get(RECORD_PORT));

		thread = new ClientSocketHandler(this,
				ChatInfo.getGroupOwnerAddress(), port, handler);
		thread.start();
	}
}
```

Código-Fonte 12 - Implementação dos métodos setupInterface e setupChatEngine

Em *setupInterface* é instanciado o adaptador responsável pelo controle da lista de mensagens e o evento do botão *Send* é atribuído. O método *setupChatEngine* diferencia a situação do dispositivo na rede, se for um *GroupOwner* ele inicia a thread *GroupOwnerSocketHandler* que iniciará um pool de sockets para conexão, caso contrário ele inicia a thread *ClientSocketHandler* que tenta se conectar através do endereço do *groupOwner* fornecido na *activity* anterior. Ambas as *threads* executarão o

ChatManager, para controlar a troca de mensagens. A classe da *Activity* também possui dois métodos que são utilizados como *call-backs* na execução das *Threads* (Código-Fonte 13).

```
public void updateChatManager(ChatManager manager) {

	chatManager = manager;
}

public boolean handleMessage(Message msg) {

	byte[] readBuf = (byte[]) msg.obj;
	String readMessage = new String(readBuf, 0, msg.arg1);

	addMessage(readMessage);

	return true;
}
```

Código-Fonte 13 - *Callbacks* utilizados pelas *Threads*

O método *updateChatManager* recebe como parâmetro um objeto da classe *ChatManager* que será utilizado no envio de mensagens pelo *socket* (Código-Fonte 14). O método *handleMessage* é utilizado pelo *ChatManager* para exibir as mensagens na *activity*.

```
private void btnSend_Click() {

	TextView chatLine = (TextView) findViewById(R.id.txtChatLine);
	String msg = chatLine.getText().toString();

	if (chatManager != null){

		chatManager.write((myNick + ": " + msg).getBytes());
		chatLine.setText("");
		chatLine.clearFocus();
	}

	addMessage("Me: " + msg);
}
```

Código-Fonte 14 - Evento de clique do botão Enviar

No evento de clique do botão, caso haja um *ChatManager* definido, a mensagem digitada na *acitivity* é enviada e o campo de texto da mensagem é apagado. A lista de mensagens é atualizada com o texto da mensagem.

3.3.3.2 ClientSocketHandler

Esta classe é uma thread disparada quando o dispositivo não for um groupOwner na rede Wi-FI Direct. Tem por finalidade tentar uma conexão via socket com o groupOwner, instanciar um ChatManager e atualizar este na activity para que as mensagens possam ser enviadas.

```
public void run() {

       Socket socket = new Socket();
        try {
             socket.bind(null);
             socket.connect(new
InetSocketAddress(serverAddress.getHostAddress(), serverPort), TIMEOUT);

             ChatManager chat = new ChatManager(socket, handler);
             new Thread(chat).start();

             chatManagerReceiver.updateChatManager(chat);

         } catch (IOException e) {
             e.printStackTrace();
             try {
                 socket.close();
             } catch (IOException e1) {
                 e1.printStackTrace();
             }
             return;
         }
}
```

Código-Fonte 15 - Método executado pelo ClientSocketHandler

3.3.3.3 GroupOwnerSocketHandler

Esta classe, assim como a anterior, é uma *thread, desta vez* disparada quando o dispositivo for o *groupOwner*. Na sua execução, aguarda uma conexão de outro dispositivo e, instancia um *ChatManager* e atualiza este na *activity*, com o mesmo intuito.

```
public void run() {

       while (true) {
        try {
             // A blocking operation. Initiate a ChatManager instance when
             // there is a new connection
            ChatManager chat = new ChatManager(socket.accept(), handler);
             pool.execute(chat);

            chatManagerReceiver.updateChatManager(chat);

        } catch (IOException e) {
             try {
                 if (socket != null && !socket.isClosed())
                     socket.close();
             } catch (IOException ioe) {

             }
             e.printStackTrace();
             pool.shutdownNow();
             break;
         }
    }
}
```

Código-Fonte 16 - Método executado pelo GroupOwnerSocketHandler

3.3.3.4 ChatManager

Esta classe é executada pelo *ClientSocketHandler* ou pelo *GroupOwnerSocketHandler* e é responsável pelo controle da troca de mensagens. Durante a sua execução fica aguardando uma mensagem e caso algo seja recebido via socket, os bytes são enviados para um handler (no caso a *activity*) para serem tratados. Possui também um método *write* para envio de *bytes* pelo *socket*. Este método é utilizado pela *activity* no envio das mensagens.

```
public void run() {

    try {

        iStream = socket.getInputStream();
        oStream = socket.getOutputStream();
        byte[] buffer = new byte[1024];
        int bytes;

        while (true) {
            try {
                // Read from the InputStream
                bytes = iStream.read(buffer);
                if (bytes == -1) {
                    break;
                }

                // Send the obtained bytes to the UI Activity
                handler.obtainMessage(-1, bytes, -1,
                    buffer).sendToTarget();

            } catch (IOException e) {
                    e.printStackTrace();
            }
        }
    } catch (IOException e) {
        e.printStackTrace();
    } finally {
        try {
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

public void write(byte[] buffer) {
       try {
             oStream.write(buffer);
       } catch (IOException e) {
             e.printStackTrace();
       }
}
```

Código-Fonte 17 - Código da Classe ChatManager

3.3.3.5 ChatInfo

Este módulo possui ainda a classe *ChatInfo*, esta classe tem a função de armazenar as informações obtidas na conexão e disponibilizá-las para utilização no exemplo de *chat*, através das propriedades de *groupOwner*, *groupOwnerAddress* e *records* (dicionário de informações do serviço). As informações contidas nesta classe são necessárias para o funcionamento do

aplicativo de *chat*. São informações básicas: se o dispositivo é o *GroupOwner*, o endereço IP do *GroupOwner* e as informações contidas no *hashMap*, disponível na classe *ChatActivity* através do método *getRecords*.

3.4 CONSIDERAÇÕES FINAIS SOBRE A IMPLEMENTAÇÃO

Neste capítulo foi discutido a implementação do aplicativo. A implementação do registro e descoberta de serviços na rede *WI-FI Direct* foi implementada, juntamente com a conexão dos dispositivos nessa rede através dos serviços. Essas funcionalidades foram separadas numa classe *Helper* para facilitar a utilização. A visualização é feita primeiramente por uma listagem de dispositivos e posteriormente os serviços agrupados por dispositivo, ao clicar no serviço, a conexão é iniciada. Um aplicativo de chat, utilizado como exemplo, foi implementado utilizando a conexão feita anteriormente por meio do serviço. Este exemplo pode ser utilizado utilizando-se outras redes e outras formas de publicação de serviço, contanto que as informações da classe *ChatInfo* sejam preenchidas corretamente.

Portanto a proposta de implementação definida inicialmente foi cumprida, levando-se em conta a visualização dos serviços na rede e um exemplo de conexão utilizando um dos serviços.

4 CONCLUSÕES E TRABALHOS FUTUROS

Neste trabalho foi desenvolvido um estudo em plataformas móveis, redes, redes *ad-hoc*, publicação e descoberta de serviços. Complementando os estudos da graduação. Após o entendimento da plataforma *Android,* das redes *ad-hoc*, mais especificamente da rede *WI-FI Direct* e do mecanismo de descoberta de serviços foi possível iniciar o processo de desenvolvimento de uma aplicação que pudesse corresponder ao que foi proposto.

Durante o desenvolvimento do programa, foi possível construir um módulo (DnsSdHelper) com o intuito de facilitar a utilização da publicação e descoberta de serviços, utilizando DNS-SD na rede *WI-FI Direct* e a conexão entre os dispositivos, centralizando as responsabilidades relativas à rede.

Sendo uma tecnologia ainda não tão difundida, a *WI-FI Direct* mostra-se promissora, dado sua facilidade de utilização pelo usuário e a velocidade proporcionada, diferenciada na área de redes *ad-hoc* sem fio. A partir da padronização proposta para a tecnologia pela *WI-FI Alliance*, espera-se um aumento da sua popularização.

4.1 TRABALHOS FUTUROS

Testes mais completos devem ser executados no sistema, utilizando mais dispositivos na rede simultaneamente. Estudos de interferências na rede causadas por movimentação e objetos ou pessoas que atravessem o raio de comunicação devem ser feitos para melhor avaliação da rede. O fluxo do aplicativo encerra-se no momento da conexão e inicio do exemplo, não finalizando as conexões quando devido.

Além disso, são algumas sugestões de trabalhos futuros:

- Comunicação *multi-hop.* Este trabalho limita-se apenas a uma comunicação direta entre o *group owner* e o cliente. Seria interessante permitir a um cliente que não esteja no alcance do *group owner* transmitir suas informações de serviços através de outro dispositivo que por sua vez possa se comunicar com o *group owner*.
- Identificação da aplicação que publicou o serviço. Este trabalho utiliza uma aplicação de exemplo que faz parte do projeto e é aberta ao concretizar-se a conexão feita por meio do serviço. Seria interessante uma desvinculação dos aplicativos que registram o serviço, do aplicativo de visualização e conexão, mantendo a possibilidade da chamada do aplicativo correto após a conexão.
- Visualização e conexão de serviços independente do protocolo. Este trabalho utiliza o protocolo DNS-SD para publicação e descoberta de serviços, porém a própria API *Android*, já proporciona a utilização nativa do protocolo UPNP. Além desses, outros protocolos também podem ser utilizados.

REFERÊNCIAS

ADT Plugin, *Android Developers.* Disponível em: <http://developer.android.com/tools/sdk/eclipse-adt.html>. Acesso em 10 jul. 2014.

Android NDK, *Android Developers*. Disponível em: <https://developer.android.com/tools/sdk/ndk/index.html>. Acesso em 15 set. 2014.

Application Fundamentals, *Android Developers.* Disponível em: <http://developer.android.com/guide/components/fundamentals.html>. Acesso em: 30 set. 2013.

Basic structure of an Android project, *Code Project.* Disponível em: <http://www.codeproject.com/Articles/395614/Basic-structure-of-an-Android-project>. Acesso em: 30 set. 2013.

BroadcastReceiver, *Android Developers*. Disponível em: <http://developer.android.com/reference/android/content/BroadcastReceiver.html>. Acesso em 10 jul. 2014.

CAMPS-MUR, D., GARCIA-SAAVEDRA, A., SERRANO, P. **Device to device communications with WiFi Direct:** overview and experimentation. 2013.

CAMPS-MUR, D., PÉREZ-COSTA, X., SALLENT-RIBES, S. **Designing energy eficiente access points with wi-fi direct:** Computer Networks, Elsevier, v. 55. N 13. p. 2838-2855. 2011.

CHENG, X., HUANG, X., DU, D-Z. **Ad Hoc Wireless Networking.** Massachusetts: Kluwer Academic Publishers, 2004.

CHLAMTAC, I., CONTI, M., Liu, J.N. **Mobile ad hoc networking:** imperatives and challenges. 2003.

CHO, C., LEE, D. **Survey of Service Discovery Architectures for Mobile Ad hoc Networks**. 2005.

Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2012-2017. San Jose: Cisco Systems, fev. 2013.

CONTI, M. et. al. **Experimenting opportunistic networks with wifi direct:** IEEE Wireless Days (WD), 2013 IFIP. 2013. p 1-6.

IntentFilter, *Android Developers*. Disponível em: <http://developer.android.com/reference/android/content/IntentFilter.html>. Acesso em 10. jul. 2014.

KAKU, Michio. **Physics of the Future:** How Science Will Shape Human Destiny and Our Daily Lives by the Year 2100. 1. ed. New York: Doubleday. 2011.

LENDERS, V. et. al. **Service Discovery in Mobile Ad Hoc Networks: A Field Theoretic Approach**. 2005.

MEDNIEKS**,** Z., DORNIN, L., MEIKE, G.B., NAKAMURA, M. **Programming Android:** Java Programming for the New Generation of Mobile Devices. 2. ed. California: O'Reilly. 2012.

MIAN, A. N., BERALDI, R. BALDONI, R. **Survey of Service Discovery Protocols in Mobile Ad Hoc Networks:** Technical Report 4/06. 2009.

Mobile/Tablet Operating System Market Share, *NetMarketShare*. Disponível em: <http://www.netmarketshare.com>. Acesso em: 30 set. 2013.

OLIVEIRA, F. A., **Estudo sobre Redes *Ad-Hoc* Móveis com Suporte à Descoberta de Serviços**. 2011.

Organization, *Wi-Fi Alliance Website*. Disponível em: <http://www.wi-fi.org/about/organization>. Acesso em: 17 set. 2013.

PITKÄNEN, M., KÄRKKÄINEN, T., OTT, J. **Mobility and Service Discovery in Opportunistic Networks.** 2012**.**

Service Discovery, *Bluetooth Special Interest Group*. Disponível em: < https://www.bluetooth.org/en-us/specification/assigned-numbers/service-discovery>. Acesso em: 10 out. 2013.

Starting an Activity, *Android Developers*. Disponível em: <http://developer.android.com/training/basics/activity-lifecycle/starting.html>. Acesso em 27. jul. 2014.

STEELE, J., TO, N. **The Android developer's cookbook:** Building applications with the Android SDK. Pearson Edication, 2010.

Toasts, *Android Developers*. Disponível em: <http://developer.android.com/guide/topics/ui/notifiers/toasts.html>. Acesso em 10. jul. 2014.

VASA, R. **Android Masterclass:** Building a simple Android app, *APC Magazine*. Disponível em: <http://apcmag.com/building-a-simple-android-app.htm>, Acesso em: 30 set. 2013.

VERVERIDIS, C. N., POLYZOS, G. C. **Service Discovery for Mobile Ad Hoc Networks:** A Survey of Issues and Techniques. 2008.

Wi-Fi-Direct FAQ., *Wi-Fi Alliance Website*. Disponível em: <http://www.wi-fi.org/files/faq_20100916_Wi-Fi_Direct_FAQ.pdf>. Acesso em: 30 set. 2013.