UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
CURSO SUPERIOR DE TECNOLOGIA EM
SISTEMAS DE TELECOMUNICAÇÕES

EVERTON JULIANO DOS ANJOS
MAURICIO FISZT

SISTEMA DE TRANSMISSÃO DE ÁUDIO PARA DEFICIENTES AUDITIVOS EM AMBIENTES RUIDOSOS

TRABALHO DE CONCLUSÃO DE CURSO

CURITIBA

2013

EVERTON JULIANO DOS ANJOS
MAURICIO FISZT

SISTEMA DE TRANSMISSÃO DE ÁUDIO PARA DEFICIENTES AUDITIVOS EM AMBIENTES RUIDOSOS

Trabalho de Conclusão de Curso de graduação, apresentado a disciplina de Trabalho de Diplomação, do Curso Superior de Tecnologia em Sistemas de Telecomunicações do Departamento Acadêmico de Eletrônica – DAELN - da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para obtenção do título de Tecnólogo.

Orientador: Prof. Francisco Muller Machado.

CURITIBA

2013

EVERTON JULIANO DOS ANJOS
MAURICIO FISZT

SISTEMA DE TRANSMISSÃO DE ÁUDIO PARA DEFICIENTES AUDITIVOS EM AMBIENTES RUIDOSOS.

Este trabalho de conclusão de curso foi apresentado no dia 17 de dezembro de 2012, como requisito parcial para obtenção do título de Tecnólogo em Sistemas de Telecomunicações, outorgado pela Universidade Tecnológica Federal do Paraná. Os alunos foram arguidos pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____
Prof. César Janeczko
Coordenador de Curso
Departamento Acadêmico de Eletrônica

_____
Prof. Dr. Décio Estevão do Nascimento
Responsável pela Atividade de Trabalho de Conclusão de Curso

BANCA EXAMINADORA

Curitiba, 17 de dezembro de 2012

_____
Prof. Dr. Jamea Cristina Batista
UTFPR

_____
Prof. Alexandra Miziara
UTFPR

_____
Prof. Francisco Muller Machado
Orientador – UTFPR

A Folha de Aprovação assinada encontra-se na Coordenação do Curso

AGRADECIMENTOS

Ao professor Francisco Machado Muller pelo seu empenho e auxílio a nós para que esse trabalho de diplomação se concretizasse.

Aos professores Alexandre Miziara e Jamea Cristina Batista pelo incentivo a nossa ideia de unir o nosso conhecimento acadêmico em benefício de pessoas que venham a utilizar o sistema desenvolvido.

Aos amigos e familiares que sempre nos apoiaram no decorrer do curso para que nos tornássemos profissionais nessa instituição em que temos orgulho de nos graduar.

RESUMO

DOS ANJOS, Everton Juliano; FISZT, Mauricio. **Sistema de transmissão de áudio para deficientes auditivos em ambientes ruidosos**. 2012. 96 f. Trabalho de Diplomação – Curso Superior de Tecnologia em Sistemas de Telecomunicações – Universidade Tecnológica Federal do Paraná. Curitiba, 2013.

Salas de aula ou auditórios nem sempre possuem uma acústica adequada para que sejam ministradas palestras de maneira a se compreender o que está sendo falado à frente, por vezes, fenômenos acústicos como a reverberação e o eco, podem interferir na inteligibilidade. Esse problema afeta com uma maior intensidade pessoas que já possuem algum grau de deficiência auditiva. Sabendo disso, o seguinte projeto é idealizado a fim de desenvolver um aplicativo em plataforma Android com transmissão via tecnologia Bluetooth em conjunto com um circuito eletrônico que efetue a filtragem de ruídos e amplificação do sinal de áudio visando promover a inclusão social de deficientes auditivos. A metodologia utilizada para cumprimento da proposta elaborada se dá através da análise de estudos correlacionados relativos à correção auditiva utilizando equalizadores juntamente ao uso de linguagem de programação avançada para construção de um *software* de auxilio auditivo destinado a transmissão de áudio em formato digital. Como resultado final tem-se a obtenção de um circuito alternativo capaz de transmitir áudio de boa qualidade em ambientes ruidosos com ótimo custo beneficio cumprindo o objetivo proposto de maneira eficiente.

Palavras-chave: Deficiência auditiva. Plataforma Android. Tecnologia Bluetooth. Inclusão social de deficientes auditivos. *Software* de auxilio auditivo.

ABSTRACT

DOS ANJOS, Everton Juliano; FISZT, Mauricio. **Audio transmission system for hearing impaired in noisy environments.** 2012. 96 f. Trabalho de Diplomação – Curso Superior de Tecnologia em Sistemas de Telecomunicações – Universidade Tecnológica Federal do Paraná. Curitiba, 2013.

Classrooms or auditoriums quite often have some problems concerning their acoustic construction and do not offer a comfortable environment for lectures in order to understand what is being spoken, acoustic phenomena such as reverb and echo, can interfere with intelligibility. This problem affects with a greater intensity people who already have some degree of hearing loss. Knowing this issue, the project is conceived to develop an application on Android platform with Bluetooth technology transmission together with an electronic circuit that makes noise filtering and signal amplification for hearing impaired audio to promote social inclusion of hearing impaired. The methodology used to achieve the proposal prepared is obtained through analysis of related studies on the hearing correction using equalizers along with the use of advanced programming language to create hearing aid software focused on transmission of audio in a digital format. As final result has the attainment of an alternative circuit able to broadcast audio signals with good quality in noisy environments with an excellent cost benefit which meets the proposed objective efficiently.

Keywords: Hearing loss. Android Platform. Bluetooth Technology. Social inclusion of hearing impaired. Hearing aid software.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE SIGLAS

ACL	Asynchronous Connection Oriented
ANATEL	Agência Nacional de Telecomunicações
API	Application Programming Interface
APK	Android Package File
CI	Circuitos Integrados
DEX	Dalvik executable
DVM	Dalvik Virtual Machine
FH-CDMA	Frequency Hopping - Code Division Multiple Access
FHS	Frequency Hopping Synchronization
FH/TDD	Frequency Hopping/Time Division Duplex
FIFO	First-In First-Out
FIR	Finite Impulse Response
GPS	Global Positioning System
GSM	Global System for Mobile communication
HCI	Host Controller Interface
IBGE	Instituto Brasileiro de Geografia e Estatística
IEEE	Institute of Electrical and Electronics Engineers
ISM	Industrial, Scientific, Medical
LIBRAS	Língua Brasileira de Sinais
LMP	Link Management Protocol
L2CAP	Logical Link Control and Adaptation Protocol
OBEX	OBject Exchange
OHA	Open Handset Alliance
ONU	Organização das Nações Unidas
OSI	Open Systems Interconnection
PAIR	Perda Acústica Induzida pelo Ruído
PPP	Point-to-Point Protocol
RF	Radio Frequency
RFCOMM	Radio Frequency Communication
QoS	Qualidade de Serviço
RAM	Random Access Memory
SCO	Synchronous Connection Oriented
SDK	Software Development Kit
SDP	Service Discovery Protocol
SIG	Special Interest Group
TCP/IP	Transmission Control Protocol / Internet Protocol
TCS	Telephony Control protocol Specification
TDMA	Time Division Multiple Access
UUID	Universally Unique Identifier

SUMÁRIO

1 INTRODUÇÃO

O tema proposto como trabalho de conclusão de curso tem como foco o auxílio a deficientes auditivos em situações específicas. Dados de um relatório divulgado na Organização das Nações Unidas (ONU) em 2011 mostram que mais de um bilhão de pessoas no mundo possuem algum tipo de deficiência (WHO, 2011). No Brasil os dados estão um tanto defasados quanto à estimativa de deficientes, mas baseado no censo demográfico realizado no ano de 2000 pelo Instituto Brasileiro de Geografia e Estatística (IBGE) existem mais de 24 milhões de habitantes com alguma deficiência e destes cerca de 5,5 milhões possuem algum tipo de déficit auditivo (INSTITUTO BRASILEIRO DE GEOGRAFIA E ESTATÍSTICA, 2000).

Normalmente essas pessoas utilizam os chamados aparelhos auditivos que visam a amplificação de ondas sonoras como forma de suprimir a perda da capacidade de deficientes em ouvir sons em níveis mais baixos, os quais uma pessoal sem deficiência consegue perceber.

Esses aparelhos auditivos possuem transdutores, que basicamente captam ondas sonoras e são convertidas em energia elétrica com o intuito de entregar aos deficientes sons amplificados em uma faixa de frequência fazendo, com que ele consiga entender claramente o que está sendo comunicado.

Para um resultado satisfatório, cada aparelho auditivo possui parâmetros que devem ser calibrados por um profissional qualificado a fim de torná-lo o mais eficiente possível para compensação da perda auditiva do deficiente.

De uma maneira geral, hoje o único controle que o paciente possui para regulagem é o volume do som que está sendo entregue pelo aparelho. Porém, isso não é o bastante para corrigir outros fatores que podem interferir no correto entendimento do que está sendo falado no caso de um seminário.

É de conhecimento que muitas soluções já foram desenvolvidas a fim de se corrigir esse tipo de problema. Podemos citar entre soluções, inclusive, os chamados equalizadores perceptuais baseados em filtros com tecnologia digital incorporado aos aparelhos de surdez. No entanto, não há relato de que tanto o problema eventual de má acústica em ambientes fechados quanto o problema de ruído gerado em um ambiente com muitas pessoas conversando com timbres diversos tenha sido

eliminado de modo a tornar a compreensão do som por um deficiente auditivo mais inteligível.

1.1 PROBLEMA

Muitas vezes em nossas atividades cotidianas nosso ouvido é exposto aos mais diferentes tipos de ruído. O ruído está presente em muitas situações do nosso dia-a-dia de diversas formas, porém nem sempre damos importância a seus efeitos maléficos que geralmente causam problemas em longo prazo. O principal fator a ser abordado aqui não diz respeito à intensidade do ruído, e sim à interferência que ele causa durante uma transmissão de áudio, podendo deixar uma comunicação incompreensível.

É de conhecimento que salas ou auditórios nem sempre apresentam uma acústica ideal para se ministrar uma aula ou palestra. Em muitos desses locais pode ocorrer geração de ruídos que representa dificuldade para a correta compreensão por parte das pessoas com algum déficit auditivo (DREOSSI, 2005; INÁCIO, 2009).

Esses ruídos podem ser decorrentes de vários motivos, entre eles, a distorção de componentes de áudio presentes na voz do palestrante, seja por difração, reflexão, ou reverberação, que consiste em um fenômeno acústico de prolongamento dos sons, ocasionando ao ouvinte uma montagem de componentes de som diferente da originalmente transmitida, ou seja, uma distorção. Isto está relacionado ao fato de que o som tem diferentes índices de reflexão quando em contato com diferentes materiais empregados no ambiente em uma transmissão em local fechado, e todos esses fenômenos interferem diretamente na inteligibilidade da voz.

Ambientes tais como salas de aula e anfiteatros são bastante vulneráveis a ruídos gerados tanto pelas conversas entre pessoas que se sentam como ouvintes, assim como também pela má acústica eventual (FERNANDES, 2006). Deste modo, mesmo um deficiente que possua um aparelho bastante sofisticado não conseguirá compreender bem o que é explanado à frente do ambiente devido a pouca inteligibilidade do som que chega a seu aparelho.

1.2 JUSTIFICATIVA

Para pessoas que possuem algum tipo de deficiência auditiva é comum a utilização de aparelhos que venham a compensar essa perda para que se tenha uma correta compreensão do que está sendo comunicado. Esses aparelhos atuam no espectro de frequências de áudio e possibilitam um ganho diferenciado em diferentes faixas de frequência.

Convém mencionar que muitos desses aparelhos não possuem uma resposta em frequência adequada para todos os usuários em diversas situações do cotidiano devido à perda auditiva ocorrer em diferentes faixas de espectro de áudio, variando para cada pessoa e ambiente. Algumas soluções já foram desenvolvidas com o intuito de customizar a resposta em frequência de acordo com cada usuário, porém em nenhuma delas se leva em consideração a introdução do ruído no processo de transmissão até a chegada ao aparelho auditivo.

A proposta do projeto é baseada na elaboração de mais uma solução destinada a deficientes auditivos como auxílio para uma melhor compreensão do que está sendo comunicado por um palestrante em ambientes fechados e ruidosos, como auditórios ou salas de aula, onde podem ocorrer ruídos em função da acústica mal planejada do local.

Outro fator que apoia a proposta está no fato que aparelhos auditivos dotados de tecnologia analógica não conseguem eliminar ruídos em ambientes fechados e mesmo aparelhos de tecnologia digital conseguem somente reduzir esse incômodo ao deficiente.

O desenvolvimento desse projeto se torna interessante devido à sua capacidade em se aplicar e agregar os conhecimentos estudados durante o curso, tais como as disciplinas de fundamentos de comunicação, comunicação de dados, processamento digital de sinais entre outras correlacionadas, além de possibilitar a pesquisa sobre novas maneiras mais eficazes e baratas para transmissão de dados, assim como as novas funcionalidades nelas incorporadas.

O conhecimento de diferentes tecnologias empregadas para comunicações nos dá uma motivação extra na aplicação em algo que cause um impacto positivo na sociedade, por isso nosso foco é o auxílio aos deficientes auditivos no que diz respeito a seu desenvolvimento intelectual.

1.3 OBJETIVOS

Os objetivos compreendem a realização de um trabalho que desenvolva e demonstre uma opção alternativa de se amenizar problemas acústicos acoplado à uma maneira moderna para a transmissão de sinais de áudio para deficientes auditivos especificamente em ambientes fechados e ruidosos.

Esse projeto busca uma alternativa tecnológica para entregar ao deficiente auditivo uma transmissão com amplificação diferenciada de cada uma das componentes de frequência presentes no sinal de forma simples e direta, oferecendo uma saída de áudio com sinal tratado de forma a auxiliá-los em ambientes ruidosos de uma maneira mais inovadora.

1.3.1 Objetivo Geral

Construir um circuito eletrônico que se comporte como uma interface entre o palestrante e o deficiente auditivo realizando a filtragem de possíveis ruídos que o ambiente possa apresentar para transmissão destes sinais de áudio através de uma tecnologia digital sem fio até um receptor, onde o som deve ser entregue de maneira inteligível e equalizado.

1.3.2 Objetivos Específicos

Para alcançarmos os resultados desejados foram definidos como objetivos específicos as seguintes etapas:

- Verificar as faixas de frequência no espectro eletromagnético que podem ser utilizadas para este tipo de aplicação;
- Levantar soluções comerciais para a transmissão e recepção em radiofrequência que utilize modulação digital para as faixas de frequência a serem utilizadas;

- Identificar circuitos capazes de realizar a digitalização e regeneração de áudio;
- Analisar as soluções em radiofrequência buscando as que possuam a capacidade de transmitir um fluxo de bits de um sinal de áudio digitalizado e que sejam compatíveis com o circuito de digitalização e regeneração de áudio;
- Elaborar um filtro para a fonte de sons amenizando efeitos acústicos prejudiciais para entendimento na comunicação;
- Elaborar um filtro que realize a equalização do som recebido pelo receptor de acordo com as preferências do usuário.

1.4 PROCEDIMENTOS METODOLÓGICOS

Para que o projeto possa ser utilizado dentro do território brasileiro, necessita-se inicialmente de uma pesquisa para se levantar as faixas de frequência no espectro eletromagnético que são permitidas pelo órgão regulador, a Agência Nacional de Telecomunicações (ANATEL) para o tipo de aplicação desejada. Dentre as frequências regulamentadas pela agência na resolução nº 506 (AGÊNCIA NACIONAL DE TELECOMUNICAÇÕES, 2008), deve-se consultar os seguintes aspectos:

- Alcance;
- Potência máxima de transmissão;
- Tipo de modulação digital compatível para esta faixa do espectro;
- Tamanho final da antena;
- Gasto de energia para transmissão e recepção;
- Existência de outros serviços que já utilizem a mesma faixa de espectro para se averiguar possíveis interferências.

Tendo este rol de frequências disponíveis no espectro eletromagnético e suas propriedades também conhecidas, pode-se realizar a pesquisa por soluções comercias para transmissão e recepção em radiofrequência com modulação digital para estas faixas.

A partir deste conjunto de soluções encontradas deve-se fazer uma seleção envolvendo não só o transmissor e receptor, mas também a faixa de frequência no espectro eletromagnético, pois através da correta seleção se pode assegurar o correto funcionamento do circuito.

Outra tarefa envolvida consiste na identificação de circuitos eletrônicos, integrados ou não, que possam realizar o trabalho de amostragem do sinal de áudio gerado pela fonte de áudio, normalmente um microfone. Este circuito deverá amostrar o sinal analógico e transformá-lo em um sinal digital a uma taxa de bits suficiente para abranger a frequência da fala humana, sendo este utilizado no transmissor. Para o receptor tem-se o circuito análogo ao transmissor. Este circuito deverá regenerar o sinal analógico de áudio a partir da sequência de bits.

De posse das pesquisas sobre o espectro eletromagnético permitido pela Agência Nacional de telecomunicações, dos transmissores e receptores possíveis e dos circuitos de digitalização e regeneração do sinal de áudio analógico, serão analisadas todas as possibilidades em busca da melhor solução técnica para a solução das principais etapas do projeto que são a transmissão e recepção de uma fonte sonora.

Será elaborado um circuito de áudio que amenize efeitos acústicos como eco e reverberação provenientes da acústica do ambiente, assim como realize a equalização do som recebido e regenerado de acordo com as preferências do usuário, podendo este mudar a amplitude de uma faixa de frequência do sinal.

2 FUNDAMENTAÇÃO TEÓRICA

O sentido da audição nos seres humanos desempenha papel fundamental para sua apropriada comunicação. Afinal, entender o que se comunica é essencial, porém é de conhecimento que nossa percepção aos sons não se comporta da mesma maneira durante toda a vida. Entre os fatores que podemos citar como determinantes para sua alteração estão causas naturais como a idade ou ainda doenças e exposição a ruídos de alta intensidade.

Como podemos ver, a perda auditiva é um problema de afeta às pessoas pelas mais diferentes causas. Por conta disso, os deficientes que utilizam algum tipo de aparelho de auxílio necessitam de uma análise de caso específica para correta determinação de quais faixas de frequência foram mais afetadas e assim determinar o melhor tratamento.

Normalmente, para se determinar a percepção dos sons pelo ouvido humano é realizado o exame de audiometria que visa demonstrar graficamente como está o comportamento limiar de sensibilidade do ouvido em um intervalo de frequências situados na faixa entre 250 Hz e 8 kHz.

Os deficientes auditivos, segundo Rapoport e Petraglia (2002), podem ser divididos em dois grupos conforme a forma em que a sua deficiência foi adquirida. Em um primeiro grupo estão pessoas que possuem um grau de perda auditiva de ordem genética ou fisiológica, onde conforme cita o mesmo autor, o grau da deficiência é elevado, o que significa dizer que as pessoas escutam muito pouco ou por vezes sequer escutam. No segundo grupo estão contidas as pessoas que adquiriram a deficiência devido a traumas acústicos. Hoje esses traumas são referenciados como “Perda Acústica Induzida pelo Ruído” (PAIR). Esse tipo de perda ocorre por uma variedade imensa de condições que nos afetam diariamente. Nesse mesmo grupo podem ser divididos indivíduos que apresentam perda devido à exposição de curto período a uma intensidade de som muita alta ou perda devido à exposição prolongada a um nível de sons acima de 85 dB (RAPOPORT; PETRAGLIA, 2002).

É de conhecimento que o ouvido humano possui a capacidade de captar sons em uma faixa de frequências que compreende 20Hz à 20kHz. De acordo com

Rapoport e Petraglia (2002), existem frequências que determinam alguns parâmetros de como os sons são captados. Em frequências menores que 2kHz os sons seriam responsáveis pela sensação de volume enquanto que acima dessa frequência estaria relacionada à sua inteligibilidade.

Para determinar um sistema de auxílio para deficientes auditivos, Rapoport e Petraglia (2002) expõem a ideia da construção de vários filtros do tipo passa-faixas organizados paralelamente de forma a serem sintonizáveis de acordo com as necessidades de cada pessoa. Dessa maneira o circuito é customizado para cada um. Isto é feito tendo-se em mãos um audiograma realizado pelo deficiente contendo graficamente pontos de frequência múltiplos de 125Hz chegando à 8kHz. Partindo-se do gráfico do exame pode-se configurar um equalizador específico para posterior configuração. Como as perdas auditivas ocorrem mais comumente na faixa dos 3kHz, os amplificadores são dimensionados para um ganho maior nessa frequência.

Existem outras soluções de auxílio para deficientes auditivos, Neto e Lima Filho (2003) construíram um circuito que considerava diferentes ganhos de potência para 3 faixas de frequência definidas de acordo com a perda auditiva do tipo “colher”. Esse tipo de perda concentra um grande número de deficientes e por isso foi escolhida. As frequências centrais das 3 faixas são 600 Hz, 2 kHz e 3 kHz e foram determinadas baseado no estudo de Petraglia e Barúqui (2002). Essas frequências podem ser alteradas de acordo com a resposta em frequência de cada deficiente. O objetivo consiste na realização de um equalizador gráfico que pode ser definido como um conjunto de filtros sintonizados em diferentes frequências centrais e operem com tensão e potência de baixa intensidade, buscando auxiliar o deficiente auditivo de acordo com o seu audiograma.

Mais recentemente, foram desenvolvidos os chamados equalizadores perceptuais onde o objetivo é produzir um equalizador adequado para o sinal de voz utilizando filtros do tipo *Finite Impulse Response (FIR)*. Nesse tipo de circuito são consideradas seis faixas de frequências que compõem o equalizador com base nas faixas contidas nos exames de audiometria. Para cada faixa é definido um ganho e assim personalizado para cada deficiente auditivo *(*GOMES E SILVA; DA SILVA; OLIVEIRA, 2008).

3 EMBASAMENTO TEÓRICO

3.1 PLATAFORMA ANDROID

O Android é uma plataforma do tipo *Open Source* criada pela Google com foco voltado para utilização em dispositivos móveis. Ela foi disponibilizada em 2007 e tem como base a plataforma Java e núcleo Linux. O mantimento do Android no mercado dos dispositivos móveis é realizado por um consórcio denominado *Open Handset Alliance* (OHA) que compreende mais de 40 empresas relacionadas entre os setores de tecnologia da informação e telecomunicações, que unidas são lideradas pela Google Inc. (LECHETA, 2010).

Entre os objetivos principais do Android está o desenvolvimento de aplicativos para dispositivos móveis e, para tanto, conta com uma grande variedade de bibliotecas, interface gráfica, além de ferramentas próprias para criação (LECHETA, 2010).

Essas ferramentas disponibilizadas oferecem uma plataforma flexível e rápida conferindo a possibilidade de personalização de aplicativos e permitindo aos desenvolvedores a criação de aplicativos dos mais variados tipos que podem ser utilizados em nosso cotidiano.

3.1.1 Recursos da Plataforma Android

Para desenvolvimento de aplicativos, o Android conta com o *Software Development Kit* (SDK) que utiliza a linguagem de programação Java como base. Basicamente esse kit provê aos programadores uma melhor integração para com o *software* proposto. No caso do Android o SDK utilizado é o Eclipse, que se trata de um ambiente integrado para desenvolvimento, que possui código aberto para uso (ANDROID DEVELOPERS, 2012a).

É no SDK que se encontram disponibilizados um conjunto de rotinas muito importante, a chamada *Application Programming Interface* (API) que compreende uma série de funções para utilização em aplicativos.

Cada API desenvolvida é sempre referenciada a um número inteiro que a identifica com relação a seu nível. O nível de uma API tem como intuito informar a revisão de um conjunto de classes usados em uma versão da plataforma Android. Isso significa que a cada nova versão disponibilizada, podemos ter novas atualizações e incrementos, além de manter em seu escopo a compatibilidade com versões anteriores. O quadro 1 demonstra os níveis e versões lançadas até o momento (ANDROID DEVELOPERS, 2012d).

Versão da Plataforma	Nível da API	Nome da Versão
Android 4.1, 4.1.1	16	*Jelly Bean*
Android 4.0.3, 4.0.4	15	*Ice Cream Sandwich MR1*
Android 4.0, 4.0.1, 4.0.2	14	*Ice Cream Sandwich*
Android 3.2	13	*Honeycomb MR2*
Android 3.1.x	12	*Honeycomb MR1*
Android 3.0.x	11	*Honeycomb*
Android 2.3.4 Android 2.3.3	10	*Gingerbread MR1*
Android 2.3.2 Android 2.3.1 Android 2.3	9	*Gingerbread*
Android 2.2.x	8	*Froyo*
Android 2.1.x	7	*Eclair MR1*
Android 2.0.1	6	*Eclair 0 1*
Android 2.0	5	*Eclair*
Android 1.6	4	*Donut*
Android 1.5	3	*Cupcake*
Android 1.1	2	*Base 1 1*
Android 1.0	1	*Base*

Quadro 1 - Versões de plataforma e suas respectivas API´s
Fonte: Android Developers (2012e).

Entre os demais recursos disponibilizados aos programadores, se pode citar suporte multimídia, telefonia GSM, câmera, GPS, acelerômetro, Bluetooth, navegador e o ambiente de desenvolvimento do Eclipse, que oferece um emulador de dispositivo para depuração. Mais adiante se descreve maiores detalhes sobre alguns desses recursos.

3.1.2 Arquitetura

Para desenvolvimento de uma aplicação eficiente é de suma importância que conheçamos as capacidades e limitações referentes à arquitetura da plataforma. Basicamente o Android inclui em sua plataforma o sistema operacional, o *middleware,* que faz a intermediação entre *software* e outras aplicações e os aplicativos em si.

A arquitetura do sistema operacional Android fica dividida em 4 camadas que agrupam diversos programas visando suportar funções do próprio sistema operacional (ANDROID DEVELOPERS, 2012d). Na sequência se mostra um pouco mais sobre cada uma delas.

3.1.2.1 *Kernel*

Dispositivos eletrônicos como computadores e celulares possuem arquiteturas próprias, o que torna necessário, a construção de um sistema operacional compatível que se adeque a esses diferentes tipos de *hardware*.

Para um correto funcionamento, o sistema operacional de um equipamento deve ser capaz de gerenciar serviços de *hardware* como memória, processamento, dispositivos de entrada e saída, drivers, etc (TANENBAUM, 2003). No Android tem-se como base para gerenciamento desses serviços o *kernel*.

A Google se baseou na versão 2.6 do *kernel* do Linux para a construção do *kernel* do Android. Mesmo assim, há pouco em comum entre ambas as versões e se pode dizer que o Android utiliza o *kernel* do Linux somado a pacotes que conferem outras funcionalidades.

3.1.2.2 *Bibliotecas*

Ao desenvolver um novo aplicativo percebeu-se que muitas das rotinas utilizadas são comuns em praticamente todos os softwares já desenvolvidos, como por exemplo a função de abrir ou fechar arquivos, entre outras. Essas rotinas não

precisam ser escritas por cada programador em cada aplicativo que se crie, pois já estão prontas para serem incluídas no aplicativo quando necessário.

Esse conjunto de instruções já definidas no sistema Android são as chamadas bibliotecas. As bibliotecas auxiliam na execução de serviços, além de permitir ao programador a possibilidade de dividir o aplicativo em módulos, o que facilita muito o seu desenvolvimento e organização.

Entre as bibliotecas disponíveis no Android estão as seguintes (TURNER, WILHELM, LEMBERG, 2006; PROJECT, 2012; ANDROID DEVELOPERS, 2012d):

- *System C Library*: consiste de uma implementação derivada da biblioteca C. Basicamente define as chamadas ao sistema;
- *Media Libraries*: essa biblioteca é responsável pelo suporte à execução dos formatos mais comuns de vídeo, imagem e áudio;
- *Surface Manager*: gerencia o acesso ao subsistema e possui capacidade de composição de gráficos em 2D ou 3D através de aplicações de múltiplas camadas;
- *Lib Webcore*: é o programa responsável pela renderização de páginas web no navegador do Android, possui código aberto;
- *FreeType*: consiste em um gerenciador de fontes otimizado para utilização em diversas plataformas;
- *Bibliotecas 3D*: baseia-se em OpenGL ES 1.0 utilizando aceleração por *hardware* ou renderização 3D por *software* ;
- *SQLite*: um sistema leve que gerencia um banco de dados relacional, possui código aberto e é indicado para utilização em dispositivos que possuem *hardware* limitado, por exemplo, celulares.

Essas bibliotecas são utilizadas por vários componentes do sistema, sendo a capacidade destas expostas para os desenvolvedores através da camada *Framework* que será descrita mais adiante.

3.1.2.3 *Android RunTime*

Os aplicativos Android rodam em seu próprio processo e são interpretados de maneira similar à plataforma Java, porém ao invés de se gerarem códigos *.class*,

são gerados códigos no formato Dalvik *executable* (*.dex),* onde são interpretados pela chamada *Dalvik Virtual Machine* (DVM).

A DVM consiste na execução de máquinas virtuais onde, após a compilação do código, converte-se os arquivos *.dex* em arquivos no formato *Android Package File (.apk),* assim tem-se o aplicativo pronto para instalação e execução em dispositivos rodando o Android (LECHETA, 2010).

3.1.2.4 *Framework do Aplicativo*

Esta camada é a responsável pela interface com as aplicações Android. O *framework* auxilia no desenvolvimento dos programas permitindo, por exemplo, a criação de botões e caixas de texto no aplicativo, pois provê bibliotecas para acesso e alteração de diferentes recursos do sistema operacional. Esse, aliás, é um dos grandes méritos do Android, onde a modificação de componentes possibilita ao programador adequar o sistema a seu aplicativo. Entre os componentes disponibilizados estão (ANDROID DEVELOPERS, 2012d):

- Provedor de Conteúdo: responsável pelo gerenciamento de acesso aos dados executados pelos aplicativos;
- Gerenciador de Notificações: controla o recebimento de notificações e exibição de informações na barra de status do dispositivo;
- Gerenciador de Atividades: permite a execução de aplicativos em segundo plano assim como gerencia o ciclo de execução destes;

A figura 1 ilustra as camadas envolvidas na arquitetura da plataforma Android.

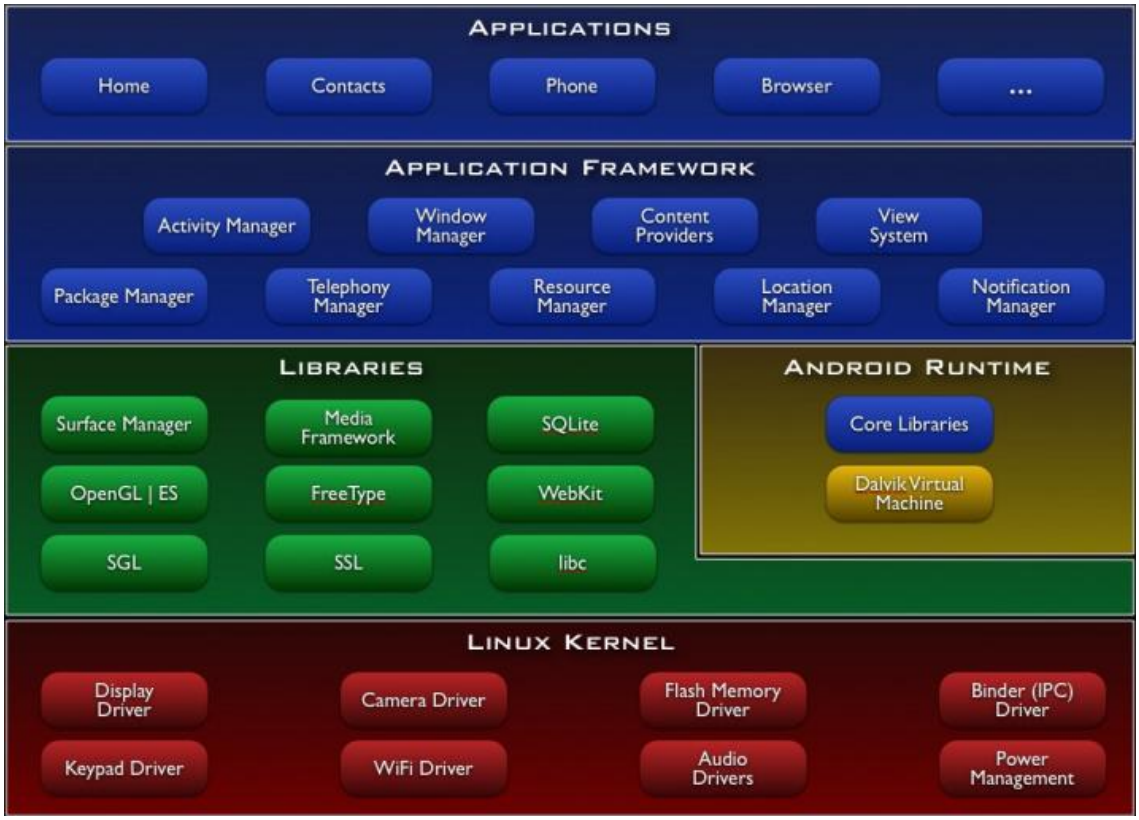

Figura 1 – Arquitetura da plataforma Android
Fonte: Android Developers (2012d).

3.1.3 Bloco de Aplicações

As aplicações desenvolvidas em plataforma Android necessitam de blocos de construção para sua criação. Dentre estes tem-se quatro principais definidos como *activity*, *intent receiver*, *service* e *content provider*. A seguir é detalhado um pouco mais sobre cada um.

Uma *activity* (atividade) pode ser definida como uma tela criada em uma aplicação Android implementada através de uma única classe. Essa classe mostra uma interface gráfica ao usuário e esta responde a determinados eventos.

Um aplicativo pode conter várias telas e cada uma delas deve ser implementada com uma *activity*, nesse caso, quando o usuário alterna entre uma tela e outra, a anterior permanece ativa, porém totalmente parada e armazenada em uma pilha. Caso o usuário deseje retornar a ela, esta estará disponível para consulta. Esse processo de alternância entre telas é realizado com o uso de uma classe especial chamada *Intent*, a qual descreve o que uma aplicação quer efetuar.

O *intent receiver* (receptor de intenção) é usado no código fonte de uma aplicação quando esta necessita executar alguma ação devido à ocorrência de algum evento externo, por exemplo, quando o telefone toca durante a execução de um aplicativo. Este processo não é transparente ao usuário, porém é possível a ocorrência de uma notificação a ele quando o *intent receiver* recorre ao gerenciador de notificações para alertar sobre algum evento de interesse.

O uso de um bloco definido como *service* (serviço) se resume a um código que permanece em execução em segundo plano enquanto outras telas podem ser exibidas ao usuário, ou seja, ele não possui uma interface gráfica. Um bom exemplo é o tocador de música, nele existem *activities* exibindo telas com diferentes opções ao usuário, porém a música em execução por trás dessas telas se trata de um *service*.

Por último tem-se o content provider (provedor de conteúdo) que tem como objetivo prover o compartilhamento de dados armazenados de uma aplicação com outros aplicativos. Isso é possível devido ao *content provider* ser uma classe que realiza um conjunto de métodos que permite a aplicativos terceiros o armazenamento e recuperação de dados tratados por ele (ANDROID DEVELOPERS, 2012d).

Convém lembrar que o uso de todos os quatro blocos mencionados é comum em muitos dos aplicativos já feitos, porém não é regra a utilização de todos, uma vez que o código fonte pode conter somente um ou mais deles.

3.1.4 Áudio no Android

Entre as API´s disponibilizadas na plataforma Android estão as que oferecem suporte a áudio e vídeo. Para uso do recurso de áudio em uma aplicação, duas classes são de suma importância a serem utilizadas no código fonte, a *AudioRecord* e a *AudioTrack* (ANDROID DEVELOPERS, 2012c).

A classe *AudioRecord* possui a capacidade de realizar a gravação de áudio a partir de qualquer fonte de áudio disponível no dispositivo no formato PCM (*Pulse-Code Modulation)*, permitindo a configuração da taxa de amostragem do sinal de áudio, seleção da fonte de áudio (estéreo ou monaural) e seleção do grau de precisão das amostras (8 ou 16 bits). A utilização desta classe em um aplicativo

Android é permitida através da permissão *android.permission.RECORD_AUDIO* (ANDROID DEVELOPERS, 2012c).

Todas estas opções são dependentes do dispositivo que está executando a aplicação. Contudo, há configurações que devem ser suportadas em todos os dispositivos para que haja alguma compatibilidade padrão entre os fabricantes. Por questões de economia de memória RAM (*Random Access Memory*) no dispositivo, esta classe requer que seja especificado um tamanho mínimo para o buffer de gravação de áudio. Isto é feito através da consulta realizada pela função *AudioRecord.getMinBufferSize(),* sendo que o valor retornado também é dependente do dispositivo. Para a gravação efetiva das amostras de áudio é necessário iniciar a gravação através do método *AudioRecord.startRecording()* e solicitar novas amostras de áudio através do método *AudioRecord.read(),* que grava e copia estas amostras em um buffer indicado na chamada a ele (ANDROID DEVELOPERS, 2012c).

A classe *AudioTrack* possui as mesmas características técnicas da classe *AudioRecord* quanto a modulação e amostragem, diferenciando-se apenas em sua funcionalidade, que é a de reprodução de áudio em qualquer uma das saídas disponíveis no dispositivo. Esta classe é capaz de reproduzir uma sequência de amostras com tamanho conhecido, ou seja, com uma duração pré-determinada ou ainda realizar a reprodução contínua de um fluxo de amostras de áudio. Estes dois modos são conhecidos como *Static* e *Streaming*. Em ambos os modos é necessária a especificação de um tamanho mínimo de buffer para o correto funcionamento da classe, sendo que este é dependente do dispositivo utilizado. O uso de cada modo é dependente da funcionalidade desejada, maiores detalhes são descritos abaixo (ANDROID DEVELOPERS, 2012c):

- *Static* (Estático): esse modo é desejável quando se precisa de execução de sons com a menor latência possível, mas que somente pode ser usado caso os sons tratamos sejam curtos o suficiente para serem armazenados na memória;
- *Streaming* (Contínuo): no modo *streaming*, a aplicação grava um fluxo contínuo de dados para o *AudioTrack* utilizando métodos próprios de escrita, onde basicamente se efetua o bloqueio e retorno quando os dados forem transferidos da camada nativa para a fila de reprodução. Esse modo é indicado para cenários onde se tenham blocos de áudio que sejam grandes

demais para serem armazenados em memória, seja pela alta taxa de amostragem ou quantidade de bits por amostra;

Uma vez que se tenham as amostras de áudio necessárias para reprodução, estas são escritas no buffer interno da classe através da *AudioTrack.write()*. A partir desse ponto é possível iniciar a reprodução das amostras (mesmos elas estando parcialmente disponibilizadas) através da *AudioTrack.play()*. Caso a classe *AudioTrack* esteja reproduzindo um fluxo de áudio, sucessivas chamadas ao método *AudioTrack.write()* irão ser continuamente reproduzidas, caso contrário a reprodução é paralisada. Por fim, faz-se menção a outra classe que complementa as demais acima citadas. Trata-se da classe *AudioManager,* que tem como objetivo o gerenciamento das fontes de áudio de entrada e saída nos dispositivos (ANDROID DEVELOPERS, 2012c).

3.2 TECNOLOGIA BLUETOOTH

O Bluetooth consiste em uma tecnologia de comunicação de dados sem fio de padrão global que se mantem em utilização no mercado de dispositivos móveis graças à sua forma de transmissão simples, segura e barata. Através dela pode-se efetuar o envio e recebimento de dados entre dois ou mais aparelhos em pequenas distâncias, independente de sua posição desde que esteja dentro da área de alcance.

Essa tecnologia é empregada nas mais diversas aplicações e, visando atender da melhor maneira possível, com o mínimo de gasto de energia, foi dividida em três classes de potência. A tabela 1 demonstra a relação entre potência e alcance de cada uma delas.

Tabela 1 – Classes de potência e alcance do bluetooth

Classe	Potência Máxima	Alcance
1	100 mW	100 metros
2	2,5 mW	10 metros
3	1 mW	1 metro

Fonte: Bluetooth SIG (2009).

3.2.1 Frequência e Comunicação

Como essa tecnologia está difundida no mundo todo, foi necessária a utilização de uma frequência de rádio que se comportasse como um padrão mundial para uso independente do lugar onde se esteja. Para tanto, a faixa ISM (*Industrial, Scientific, Medical*) foi adotada para implementação. Ela é aberta e usada em diversos países, operando na faixa de frequência de 2,45 GHz, podendo existir variações entre 2,4 GHz e 2,5 GHz (BLUETOOTH SIG, 2012).

Uma das preocupações ao se utilizar a faixa ISM consiste no fato de se garantir que o Bluetooth não venha a sofrer ou gerar interferências. A banda de operação utilizada é a mesma de telefones sem fios, equipamentos *Wi-fi* e fornos micro-ondas, ou seja, esses aparelhos podem, sim, provocar interferências na comunicação durante o funcionamento, porém o Bluetooth foi concebido prevendo a atuação em ambientes tidos como de alta interferência. A tecnologia é muito mais robusta que as demais sem fios (PHILIPS, 2012).

Para assegurar essa imunidade é usada a técnica de modulação FH-CDMA (*Frequency Hopping - Code Division Multiple Access*), que efetua a divisão da frequência em diversos canais. Assim, uma vez estabelecida a conexão do dispositivo, ocorrem rápidas alternâncias entre estes canais fazendo com que largura de banda se torne pequena, o que contribui para diminuição de possíveis interferências.

Com o Bluetooth, pode-se utilizar um máximo de 79 frequências com um espaçamento de 1 Mhz dentro da faixa de ISM. Em alguns países o número máximo de frequências fica limitado a 23.

Dispositivos dotados com a tecnologia Bluetooth tem ao seu dispor o modo de comunicação *full-duplex*, ou seja, podem tanto enviar dados para outro aparelho como também receber. Isso é possível através da alternância entre slots dedicados a função de envio ou recebimento através do esquema FH/TDD (*Frequency Hopping/Time Division Duplex*) (ALECRIM, 2008).

Nesse esquema, os slots ficam divididos em canais com períodos iguais a 625µs, logo, a cada segundo, tem-se 1600 saltos, já que cada salto deve ser preenchido por um slot. A figura 2 demonstra melhor essa descrição.

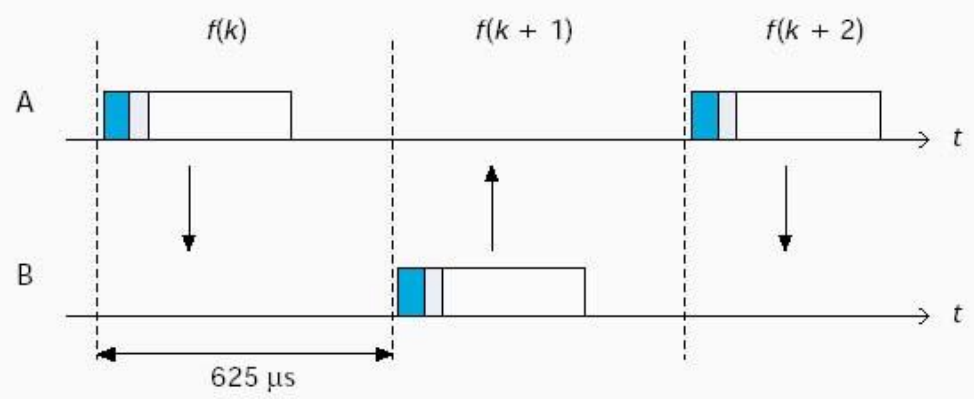

Figura 2 – Canal FH/TDD aplicado ao Bluetooth
Fonte: Ericsson History (1998).

3.2.2 Redes Bluetooth

Por definição, uma rede de comunicação consiste na ligação entre dois ou mais dispositivos. No caso do Bluetooth, as redes possuem uma denominação própria chamada de *piconet* (pico=pequena, net=rede). Em uma rede *piconet*, o dispositivo que inicia a comunicação é tido como mestre. É a ele destinada a função de adequar o sincronismo e transmissão de dados para outros dispositivos que nesse caso são denominados dispositivos escravos (MORIMOTO, 2008).

Uma *piconet* possui uma limitação de 8 conexões possíveis, sendo uma reservada para o dispositivo mestre e as outras sete para os demais dispositivos escravos, porém no Bluetooth tem-se também a possibilidade de comunicação entre redes *piconet* dentro de alcance. Esse esquema é chamado de *scatternet*. Em uma rede *scatternet* pode-se fazer com que dispositivos escravos de *piconets* diferentes venham a se comunicar, porém o mesmo não é válido para dispositivos mestres que

só podem fazer parte de uma rede. A figura 3 a seguir demonstra melhor a conexão entre as redes (ALECRIM, 2008).

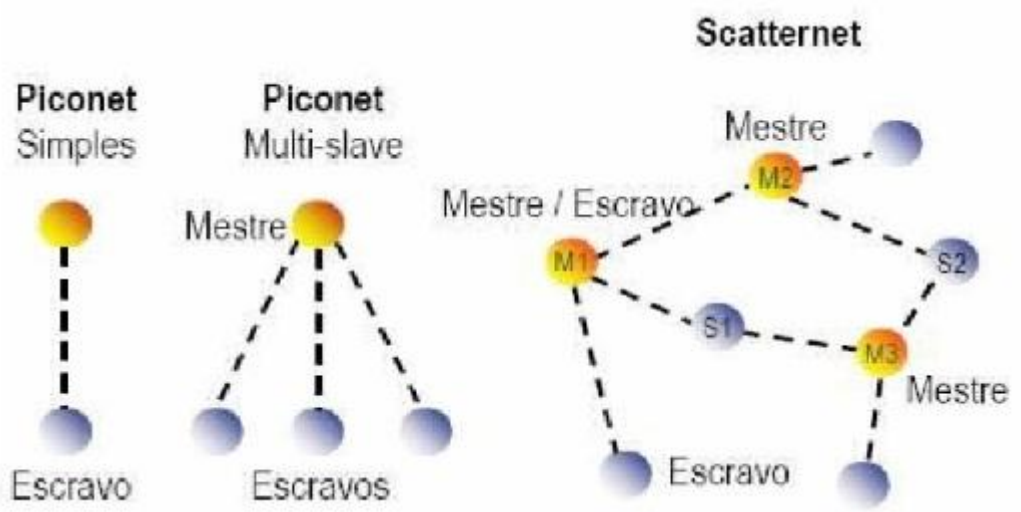

Figura 3 – Topologia Bluetooth
Fonte: Ericsson History (1998).

Uma rede *piconet* possui um esquema de identificação a fim de estabelecer a comunicação entre os dispositivos que fazem parte dela. Isso é possível através de um sinal chamado *Inquiry.* O *Inquiry* consiste de uma mensagem enviada por um dispositivo que deseja identificar quais outros dispositivos estão dentro da área de alcance.

No momento em que um dispositivo tenta efetuar conexão em um *piconet* já estabelecida, o sinal *Inquiry* é enviado e os dispositivos que venham a receber este respondem com um pacote denominado *Frequency Hopping Synchronization* (FHS). O pacote FHS contém dados de identificação do dispositivo e de sincronismo da rede *piconet*, então uma vez que o dispositivo solicitante receba essas informações, este pode enviar um segundo sinal denominado *page.*

Em suma, o "*page"* é enviado pelo dispositivo que deseja efetuar uma conexão em duas portadoras diferentes em um espaço de tempo de 1,25ms, quando então efetua duas escutas para verificar se existe algum retorno para estabelecimento de conexão.

Além dos sinais de identificação acima citados, o Bluetooth também possui um sinal chamado *scan* que visa poupar energia de dispositivos que estejam inativos fazendo que estes entrem em estado ocioso. Após certo intervalo de tempo, esses

dispositivos voltam ao estado ativo a fim de checar se existe algum dispositivo tentando estabelecer conexão. Esta janela de varredura dura em torno de 10ms (ALECRIM, 2008).

3.2.3 Versões de Bluetooth

Desde o seu surgimento em meados de 1994 até o momento, o Bluetooth evoluiu e teve novas versões lançadas que melhoraram principalmente sua performance quanto à velocidade de transmissão, alcance e economia de energia. Até a presente data são de conhecimento as seguintes versões (ALECRIM, 2008):

- Bluetooth 1.0 e 1.0B: nestas versões foram encontradas muitas restrições por parte dos fabricantes. A principal delas foi relativa à falta de interoperabilidade entre os dispositivos;
- Bluetooth 1.1: esta foi considerada a primeira versão de sucesso do Bluetooth, sendo definida com um padrão IEEE 802.15. Como principal benefício, foram corrigidos problemas de interoperabilidade da versão anterior;
- Bluetooth 1.2: esta versão tem como benefícios a compatibilidade com a versão 1.1, melhor velocidade de transmissão, redução de interferências, aperfeiçoamento de qualidade de áudio e redes *scatternets*;
- Bluetooth 2.0: entre as melhorias dessa versão estão um novo aumento de velocidade de transmissão para 3Mb/s, menor consumo de energia e melhor comunicação entre os dispositivos;
- Bluetooth 2.1: como principais benefícios dessa versão tem-se o aperfeiçoamento em recursos de criptografia, menor consumo de energia e adição de informações ao *Inquiry*, o que permite uma melhor seleção de dispositivos antes do estabelecimento de uma conexão;
- Bluetooth 3.0: a principal melhoria dessa versão está em sua alta taxa de transmissão de dados quando comparado com as demais versões. Com ela pode-se atingir até 24Mb/s, isso graças a adição de transmissões do protocolo 802.11. Além da velocidade, a economia de energia também foi melhorada através de um gerenciamento mais efetivo;

- Bluetooth 4.0: até o momento é a versão mais recente. Tem como foco a economia de energia, fazendo com que os dispositivos consumam muito menos energia que as demais versões, porém sua taxa de transmissão de dados é limitada a 1Mb/s.

Mesmo havendo diferenças quanto a alguns aspectos entre as versões aqui apresentadas, estas não devem oferecer problemas de conexão quando ocorre uma tentativa entre duas versões distintas, o que de fato deve ocorrer é a limitação de taxa de transferência de dados, sempre adequando o enlace ao limite da versão de menor velocidade.

3.2.4 Protocolos do Bluetooth

O Bluetooth possui uma grande variedade de protocolos, sendo estes definidos pela organização Bluetooth SIG (*Special Interest Group*). Basicamente a pilha de protocolos está dividida em três grupos lógicos, protocolos de transporte, protocolos de middleware e aplicações (QUEIROZ, 2008).

3.2.4.1 *Protocolos de Transporte*

No grupo de protocolos de transporte é efetuado o descobrimento de dispositivos Bluetooth e o gerenciamento de ligações físicas e lógicas para protocolos e aplicações em camadas mais elevadas. Entre as camadas envolvidas nesse grupo encontram-se as seguintes (BLUETOOTH SIG, 2012):

- Camada RF (*Radio Frequency*): trata-se da camada base da pilha que analogamente corresponde à camada física do modelo OSI (*Open Systems Interconnection*), onde tem-se a transmissão de dados por radio frequência, bem como sua modulação.
- Camada *Baseband:* nessa camada tem-se a especificação do controlador de enlace do Bluetooth, sendo este responsável pelo protocolo de controle, rotina de enlace nas camadas mais baixas e gerenciamento de links síncronos SCO (*Synchronous Connection Oriented*) e assíncronos ACL (*Asynchronous Connection Oriented*).

O SCO basicamente é um link usado para dados de voz e consiste em um conjunto de timeslots reservados em um link ACL existente. Nesse padrão se estabelece um link sincronizado entre os dispositivos onde é feita uma alocação de slots para cada aparelho. Isto torna o SCO um padrão muito utilizado para aplicações onde se deseja transmitir dados de maneira contínua, como voz, por exemplo.

A taxa de transmissão de transmissão nesse tipo de enlace é de 432Kb/s para dados e 64Kb/s pra voz. Como fator negativo pode-se citar que o SCO não permite uma retransmissão de pacotes de dados, o que acarreta no dispositivo receptor uma reprodução de som com ruído.

No caso do ACL tem-se um tipo usual de link de rádio utilizado para pacote de dados usando a tecnologia *Time Division Multiple Access* (TDMA) para determinar o acesso. Ele estabelece um link assíncrono com slots inicialmente livres entre um dispositivo denominado mestre e outros dispositivos definidos como escravos.

O ACL é indicado para aplicações onde se deseja garantir a integridade dos dados durante as comunicações entre os dispositivos, pois permite a retransmissão deles caso ocorra perda de pacotes e sua velocidade de transmissão é de 721Kb/s.

- Camada *Link Manager*: trata-se de uma camada de gerenciamento de enlace que trata das configurações e controle destas. Ela é responsável pela implementação do *Link Management Protocol* (LMP), usado para controle do link de rádio entre dois dispositivos.
- L2CAP (*Logical Link Control and Adaptation Protocol):* a utilização do L2CAP provê a multiplexação, segmentação e remontagem de pacotes, além de implementar garantia de Qualidade de Serviço (QoS) no lado do host. Ela ainda oferece serviços de dados para camadas mais elevadas, tendo sua comunicação com a camada *Baseband* efetuada através de links ACL.
- HCI (*Host Controller Interface*): trata-se de uma interface que padroniza a comunicação entre a pilha do lado do host com o lado do controlador. Seu principal objetivo é possibilitar a operação entre os dispositivos e o uso de protocolos de aplicações, sendo possível através da implementação de suas três partes, o *driver* (localizado na parte referente ao host), *firmware* (presente no *hardware* do controlador) e o módulo de transporte que fica entre ambos.

3.2.4.2 *Protocolos de Middleware*

Entre o grupo de protocolos de middleware encontra-se uma seleção de protocolos de padrões industriais e também desenvolvidos pelo Bluetooth SIG. Nele estão incluídos (BLUETOOTH SIG, 2012):

- TCS (*Telephony Control protocol Specification*): utilizado para configurar e controlar chamadas de voz e dados entre dispositivos Bluetooth;
- SDP (*Service Discovery Protocol*): o SDP é usado para permitir aos dispositivos descobrir quais serviços cada qual suporta e quais parâmetros usar para conexão baseando-se em atributos e classes;
- RFCOMM (*Radio Frequency Communication*): trata-se de um conjunto de protocolos de transporte criado sobre o L2CAP que emula portas seriais RS-232. Pode efetuar 60 conexões simultâneas para um mesmo dispositivo Bluetooth de cada vez e transmite um simples fluxo de dados byte à byte, sem formatação, como em um cabo serial;
- OBEX (*OBject EXchange*): promove a troca de objetos binários entre dispositivos. É muito usado em comunicações do tipo infra-vermelho;
- PPP (*Point-to-Point Protocol*): trata-se de um perfil para conexões do tipo ponto-a-ponto;
- TCP/IP (*Transmission Control Protocol / Internet Protocol*): permite a comunicação dos dispositivos com outros conectados a internet.

O Bluetooth tem suporte ao protocolo TCP/IP para ser utilizado sobre o L2CAP, além da disponibilidade de perfil para utilização do protocolo PPP sobre o RFCOMM para desenvolvimento de aplicativos.

3.2.4.3 Aplicações

No grupo de aplicações basicamente encontram-se aplicativos desenvolvidos que utilizam os links Bluetooth. A figura 4 ilustra de maneira mais clara alguns dos protocolos envolvidos na tecnologia e abordados até aqui.

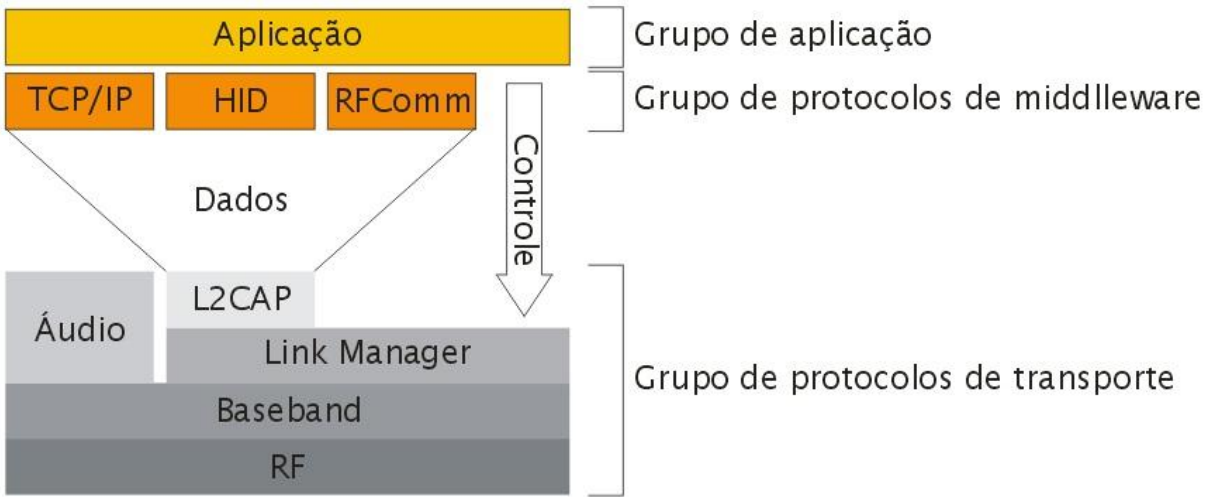

Figura 4 – Pilha de protocolos no Bluetooth
Fonte: Ericsson History (1998).

Existem ainda outros elementos pertencentes aos protocolos que compõem os chamados perfis de Bluetooth e são definidos na especificação. Esses perfis em sumo servem como soluções padronizadas a serem utilizadas na interface para comunicação entre os dispositivos, sendo que cada um destes pode suportar um ou mais perfis. (CREATIVE COMMONS, 2012)

3.3 BLUETOOTH NO ANDROID

A plataforma Android tem incluída em sua arquitetura suporte à pilha de protocolos Bluetooth para envio de dados para outros dispositivos que possuam essa tecnologia embarcada. A camada de aplicação *framework* oferece essa funcionalidade através da Bluetooth API, onde são permitidas conexões ponto-a-ponto ou multiponto entre os dispositivos (ANDROID DEVELOPERS, 2012b).

Utilizando essa API pode-se fazer com que uma aplicação Android realize tarefas que incluem checagem de outros dispositivos próximos, estabelecimento de canais de comunicação através do protocolo RFCOMM, transferência de dados em modo *full-duplex*, além do gerenciamento de múltiplas conexões.

Para implementar essas interfaces em uma aplicação Android, tem-se basicamente que cumprir com quatro etapas:

- Configuração do Bluetooth;

- Checagem de dispositivos disponíveis e pareados na área de alcance;
- Conexão dos dispositivos;
- Transferência de dados entre os dispositivos.

Felizmente todas essas etapas estão disponibilizadas em um pacote chamado *android.bluetooth*. Para tanto o programador deve utilizar os métodos disponíveis na API Bluetooth. Algumas das classes abaixo são pertencentes ao pacote e cada uma exerce um papel específico dentro da API (ANDROID DEVELOPERS, 2012b):

- *BluetoothAdapter*: essa classe representa o ponto de partida para toda a interação com o Bluetooth e através dela pode-se realizar a busca por outros dispositivos Bluetooth, consultas a dispositivos pareados e possibilidade de escuta de comunicação advindas de outros aparelhos;
- *BluetoothDevice*: descreve dispositivos conhecidos ou encontrados onde se podem consultar informações como nome, endereço ou estado, além de iniciar uma conexão através de sockets;
- *BluetoothSocket*: representa a interface entre dois dispositivos para um socket Bluetooth permitindo a uma aplicação a troca de dados entre estes;
- *BluetoothServerSocket*: trata-se de um servidor de socket que basicamente permanece em escuta para requisições de entrada. Este socket é essencial para estabelecer a conexão entre dois dispositivos, pois o dispositivo solicitante precisa abrir um servidor socket com essa classe para que posteriormente seja retornado um *BluetoothSocket* conectado quando esta é aceita.

Para utilização do pacote *android.bluetooth*, a plataforma Android possui em sua arquitetura classes de permissões que devem ser declaradas, a fim de que uma aplicação só execute determinadas ações mediante autorização do usuário. A inclusão dessas permissões tem basicamente o intuito de tornar uma aplicação mais segura e também mais eficiente. No caso do Bluetooth, deve-se declarar pelo menos uma das duas permissões abaixo:

- *android.permission.BLUETOOTH*: é responsável pela solicitação da permissão de Bluetooth para realizar a comunicação com outro dispositivo (requisição, aceitação, transferência de dados na conexão);

- *android.permission.BLUETOOTH_ADMIN*: necessária para realizar a busca de dispositivos e também responsável pelas configurações do Bluetooth. Quando se usa essa permissão, obrigatoriamente deve-se usar a primeira citada.

A figura 5 mostra um exemplo de permissão implementada em uma aplicação.

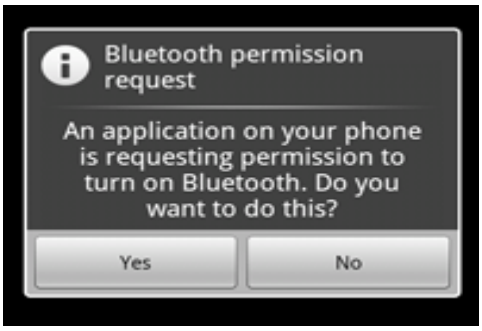

Figura 5 – Caixa de diálogo para habilitar o Bluetooth
Fonte: Android Developers (2012b).

O exemplo de requisição de permissão para habilitar o Bluetooth mostrado na figura 5 demonstra como uma aplicação deve se comportar em um primeiro momento para uso do serviço. Isto sempre é desejável, uma vez que caso o Bluetooth não seja suportado no dispositivo, recursos deste podem ser desativados a fim de economizar energia (ANDROID DEVELOPERS, 2012b).

Entre as demais permissões passíveis de implementação estão as já descritas anteriormente em classes envolvendo o rastreamento de dispositivos, consulta de dispositivos pareados, habilitação de descobrimento, além de permissões relacionadas com conexões e seu gerenciamento.

Por fim, convém lembrar que os recursos da Bluetooth API aqui citados não estão presentes em todas as versões do Android. Para utilização desta, o dispositivo em questão deve conter no mínimo a versão Android 2.0 (API nível 5) ou superior.

3.4 EQUALIZADORES DE ÁUDIO

Hoje é possível encontrar no mercado diversas soluções comerciais em termos de circuitos integrados (CI) que se propõem a atender as mais variadas

funções em projetos eletrônicos. Entre os circuitos disponíveis, existem os chamados equalizadores de áudio, que de maneira simplória podem ser definidos como filtros eletrônicos que permitem controlar a resposta em frequência de um sistema de som. Existem alguns tipos de equalizadores disponíveis hoje para uso, entre eles destacam-se (SOUND INSTITUTE, 1996):

- Equalizador filtro Shelving: é o tipo de equalizador mais simples, muito usado em sistemas de som doméstico. Seu uso é normalmente efetuado quando é desejado o controle de frequências de tom grave abaixo de 100Hz ou frequências agudas igual/superiores à 10kHz;
- Equalizador paramétrico: trata-se de um equalizador com opções mais sofisticadas, que permitem, por exemplo, controlar o ganho e determinar em qual frequência é mais afetado, além de possibilitar o ajuste da largura de banda;
- Equalizador gráfico: construído de maneira a permitir a definição dos filtros de maneira individual e graficamente, esses equalizadores são divididos em três tipos, que se diferem devido ao número de filtros de cada um:
    - Banda de Oitava: tem esse nome devido à sua constituição entre 9 e 10 filtros, com suas frequências separadas por um conjunto de uma oitava;
    - Banda de oitava dois terços: nesse equalizador tem-se normalmente 15 controles aplicáveis, sendo estes separados por dois terços de uma oitava;
    - Banda de oitava um terço: é o equalizador que permite o maior grau de controle, sendo ele constituído de 27 a 30 controles que propiciam uma precisão maior que os demais. Nele as frequências centrais estão separadas por uma distância menor, apenas um terço de oitava.

A figura 6 mostra um típico equalizador do tipo gráfico.

Figura 6 – Equalizador de áudio do tipo gráfico
Fonte: Sound Institute (2012).

Esses equalizadores mencionados atendem uma gama de aplicações relativas a sistemas de áudio e pode-se, através deles, por exemplo, efetuar a equalização de um sistema de auto-falantes produzindo uma suavização da resposta e obtendo um som de mais qualidade. Outra aplicação importante dos equalizadores está relacionada à compensação do áudio em ambientes propensos a fenômenos acústicos como reverberação e eco. O equalizador nesses casos consegue amenizar os efeitos negativos de maneira a melhorar a clareza do som ambiente. Seja para uso na inteligibilidade ou para aumento de ganho/volume de um sistema de áudio, os equalizadores se mostram importantes circuitos eletrônicos que quando ajustados de maneira correta, propiciam uma excelente solução para projetos que visam entregar a um ouvinte uma experiência de som mais agradável e adequada (SOUND INSTITUTE, 1996).

4 DESENVOLVIMENTO

4.1 DESENVOLVIMENTO DE APLICATIVOS NO AUXÍLIO A DEFICIENTES

Com o avanço da tecnologia dos dispositivos móveis, estes cada vez mais agregam novas funcionalidades e se tornam mais acessíveis, com isso acabou-se criando um novo nicho de mercado que cresce ano após ano, o de desenvolvimento de aplicativos.

Hoje se tem a disposição algumas plataformas de desenvolvimento de *softwares* que oferecem diversos recursos para criação de aplicativos nos mais variados segmentos, entre estes, o de auxilio a pessoas com algum nível de deficiência.

Independente do tipo de deficiência, seja ela visual, auditiva, mental ou mesmo motora, existem diversas maneiras de se desenvolver aplicativos que venham a interagir e provocar diferentes estímulos ao usuário. Segundo Oliveira (2012), diretor de negócios da Pró-Ativa Soluções, empresa que está desenvolvendo um *software* que traduz o Português para a Língua Brasileira de Sinais (LIBRAS), existem diversos aplicativos desenvolvidos para portadores de deficiências, porém a ampla maioria deles provém do mercado americano, o que nem sempre se adequa a usuários de línguas diferentes.

Com o intuito de desenvolver um aplicativo direcionado para deficientes auditivos e que este possa ser utilizado em aparelhos que tenham maior acessibilidade em termos de custo, foi acolhido o Android, uma plataforma que possui milhares de adeptos no mundo todo e que é conhecida pela facilidade na criação de códigos e obtenção de recursos nativos de dispositivos móveis.

Para o desenvolvimento deste trabalho, foi primeiramente efetuada uma pesquisa visando um levantamento de dados o qual se refere aos objetivos específicos do projeto. Foram utilizadas basicamente duas formas de obtenção de informações. A primeira delas baseada em buscas de artigos correlacionados ao tema na internet e a segunda através de questionamentos junto aos professores da

Universidade a fim de sanar dúvidas técnicas, assim como obter sugestões de como tornar o projeto mais eficiente.

Dentre as duas formas utilizadas, o uso da internet como fonte de informações foi a que proveu a maior parte do conhecimento necessário, uma vez que hoje se tem um vasto acervo disponível para consultas *online*. Nesse âmbito, foram tomadas algumas precauções quanto à obtenção de dados com relação à sua confiabilidade, pois existem diversas publicações que apresentam um conteúdo duvidoso em seu contexto, por isso foram considerados obras e artigos provindos de fontes reconhecidas.

O complemento de informações relevantes ao projeto foi obtido através de consultas junto ao corpo docente da Universidade em reuniões realizadas com o Professor orientador e dessa maneira conseguiu-se discutir o andamento da teoria aplicada ao trabalho, assim como obter considerações do que seria mais relevante para relato.

Para cumprimento dos objetivos específicos descritos no projeto, foi efetuada primeiramente a pesquisa de faixas de frequência no espectro eletromagnético aptas para aplicação em equipamentos de auxílio auditivo. Para levantamento de tais informações foi efetuada uma consulta à resolução nº 506, de 1º de Julho de 2008 da Agência Nacional de Telecomunicações, a qual regulamenta o uso de equipamentos de radiocomunicação de radiação restrita.

Segundo esta resolução, equipamentos de auxílio auditivo possuem frequências de operação estabelecidas nas faixas de 72,0 a 73,0 MHz, 74,6 a 74,8 MHz e 75,2 a 76,0 MHz, o que a princípio restringe o uso de determinados circuitos de áudio. Felizmente esta também regulamenta o uso de equipamentos que utilizam tecnologia de modulação digital, como por exemplo, o Bluetooth que opera na faixa de 2,4 GHz e apresenta os requisitos necessários para uso, independente do tipo de aplicação a que se destina.

Levando em consideração essa primeira informação, foram avaliadas as opções de transmissão de dados em formato digital para uso no projeto. Em um primeiro momento a melhor solução para uso consistiu no desenvolvimento de uma placa de circuito impresso contendo circuitos integrados dedicados à transmissão e amplificação de determinadas frequências de áudio. Uma segunda alternativa levantada surgiu da ideia se se usar dispositivos móveis como celulares ou *tablets* para transmissão de dados através de tecnologia embarcada.

Devido à maior facilidade de implementação, menor custo, facilidade de aquisição e popularização de dispositivos móveis, foi adotado a ideia de se desenvolver um aplicativo em plataforma Android que venha a realizar o gerenciamento de todo o sistema de auxílio proposto, compreendendo a coleta, tratamento, transmissão, recepção e entrega dos dados. A tecnologia de transmissão digital utilizada nesse caso é a Bluetooth pela sua ampla disseminação entre os aparelhos móveis comercializados no momento, além de sua segurança e qualidade.

O uso de dispositivos móveis dotados de sistema Android facilitou a realização do projeto, visto que a tecnologia Bluetooth embarcada nos aparelhos em conjunto com seus recursos nativos conduziu à conclusão da etapa referente a pesquisa em soluções de radiofrequência que fossem capazes de realizar a transmissão e recepção de áudio de maneira compatível, exigindo apenas a realização do desenvolvimento de um aplicativo que realizasse todo o gerenciamento da informação entre os dispositivos.

4.2 APLICATIVO AUDIOLINK

O AudioLink é um aplicativo desenvolvido em plataforma Android que tem como objetivo oferecer às pessoas com deficiência auditiva uma solução para escuta a distância em ambientes ruidosos. Sua utilização se dá através de sua instalação em dois dispositivos dotados de sistema Android, formando assim um enlace de comunicação onde um deles desempenha o papel de locutor, enquanto o outro se comporta como ouvinte.

Neste trabalho, o dispositivo tido como locutor representa o transmissor e permanece em poder do palestrante. Uma vez que o aplicativo é iniciado, o som da voz do palestrante é captado pelo microfone do aparelho, sendo em uma próxima etapa amostrado pelo aplicativo que efetua a digitalização e transmissão do áudio através da tecnologia Bluetooth.

Do outro lado do enlace, tem-se o usuário com deficiência auditiva de posse do dispositivo que se comporta como receptor desses sinais que posteriormente são

entregues na saída de áudio nativa do aparelho. A figura 7 demostra a topologia descrita:

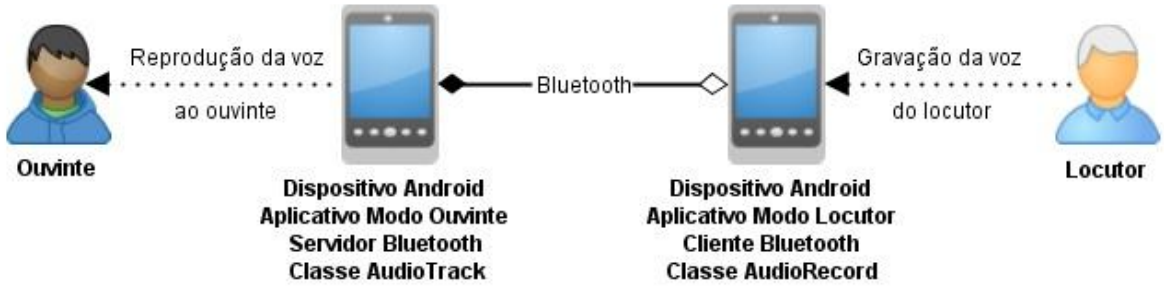

Figura 7 – Diagrama geral de funcionamento
Fonte: Autoria própria.

O aplicativo foi desenvolvido em módulos distribuídos através de cinco arquivos fonte e um arquivo de configuração dos quais suas funções principais estão destacadas a seguir:

- Módulo “Tela Principal”: Contém todas as interações básicas com o usuário controlando o modo de operação do aplicativo e ativação de recursos Bluetooth junto ao Android (apêndice A);
- Módulo “Selecionar Ouvinte”: Se refere à tela de seleção do dispositivo ouvinte quando o aplicativo está funcionando em modo locutor (apêndice B);
- Módulo “Locutor”: Compreende as etapas de conexão ao ouvinte, gravação e transmissão do áudio além do controle de erros de transmissão na conexão Bluetooth (apêndice C);
- Módulo “Ouvinte”: Engloba as etapas de espera pelo locutor, recepção e regeneração do áudio e também o controle de possíveis erros da conexão Bluetooth (apêndice D);
- Módulo “Constantes do Aplicativo”: Se refere as constantes globais do aplicativo utilizadas na sinalização de eventos entre os objetos deste (apêndice E);
- Arquivo de permissões “*AndroidManifest*”: Provê informações básicas ao SDK Android como versão mínima de API requerida pelo aplicativo além das permissões necessárias para seu funcionamento (apêndice F).

A interface do aplicativo foi elaborada de maneira simplificada visando tornar o seu uso fácil e intuitivo. A tela principal possui dois botões que representam as funções de locutor ou ouvinte.

A fim de ilustrar todo o fluxo de atividades no aplicativo foi elaborado o fluxograma contido na figura 8, com o intuito de prover uma melhor interpretação de quais processos ocorrem e seu papel nos dispositivos envolvidos.

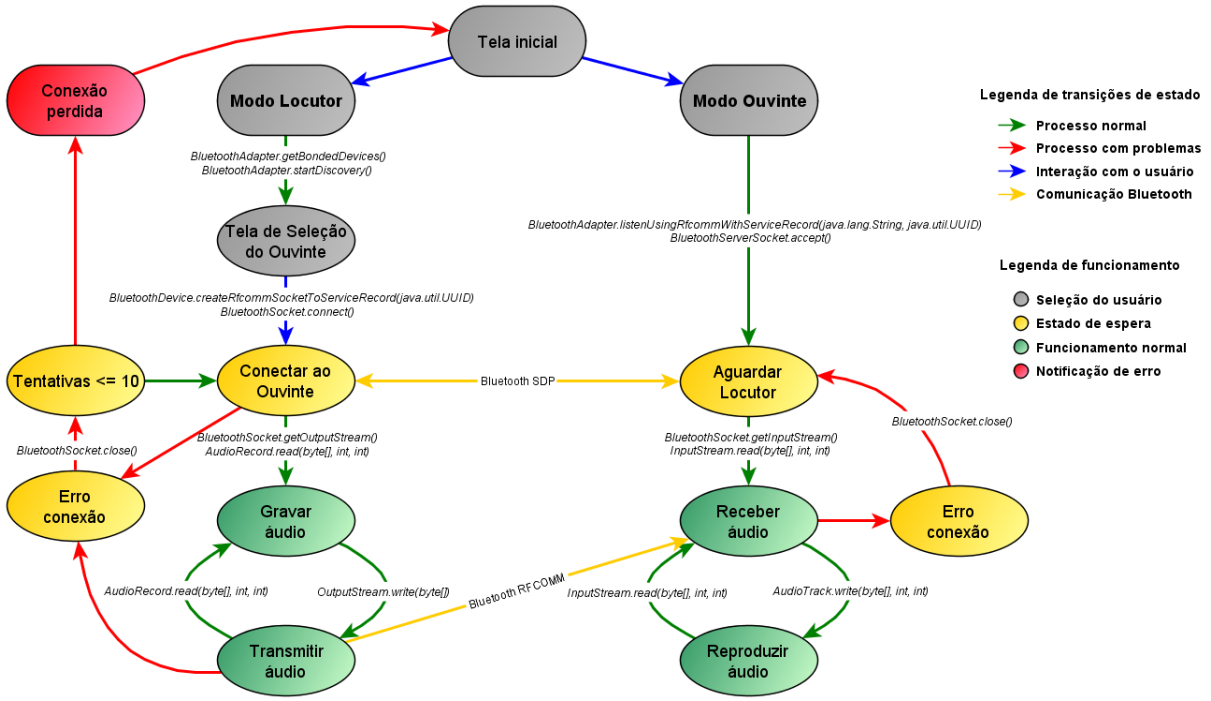

Figura 8 – Fluxograma de funcionamento do aplicativo audiolink
Fonte: Autoria própria.

A forma com que foram relatados os processos envolvendo o aplicativo mescla uma simulação de uso juntamente com uma explicação mais técnica e detalhada. Tendo em mente os processos observados na figura anterior, têm-se melhores condições de compreensão desses processos descritos na próxima seção.

4.2.1 Conexão

Uma vez que o aplicativo está instalado em dois dispositivos, o procedimento para sua utilização começa através do toque ao ícone disponibilizado na página de interação do aparelho, conforme mostra a figura 9.

Figura 9 – Ícone do aplicativo na página de interação
Fonte: Autoria própria.

Assim que o aplicativo é inicializado, ocorre a verificação da disponibilidade do serviço Bluetooth no aparelho através do método *BluetoothAdapter.getDefaultAdapter()*. Caso este esteja disponível, mas não habilitado, uma mensagem de solicitação do tipo *BluetoothAdapter.ACTION_REQUEST_ENABLE* é exibida para ativação do serviço. A figura 10 ilustra a tela mostrada ao usuário.

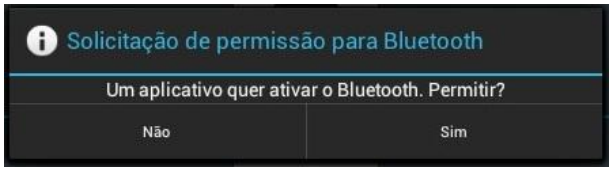

Figura 10 – Tela de solicitação para ativação do Bluetooth
Fonte: Autoria própria.

Caso o serviço já esteja habilitado no dispositivo a tela de solicitação não se apresenta ao usuário e dá lugar a ela a tela principal disponibilizando dois botões referentes às opções do aplicativo, locutor e ouvinte. A opção locutor é representada pelo desenho de um microfone, enquanto que a opção ouvinte é representada pelo desenho de um fone de ouvido. A figura 11 ilustra as seleções disponíveis.

Figura 11 – Tela de opções do aplicativo
Fonte: Autoria própria.

4.2.1.1 *Modo Ouvinte*

O modo ouvinte foi desenvolvido para realizar duas tarefas dentro do sistema. A primeira é a criação de um servidor para conexão Bluetooth, e a segunda é a regeneração das amostras de áudio recebidas pelo servidor Bluetooth assim que esteja conectado.

Ao ser selecionado este modo, uma mensagem de solicitação no Bluetooth do tipo *BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE* é apresentada, de forma a permitir a detecção deste pelo dispositivo locutor para efetuar o pareamento. Esse período de visibilidade do dispositivo ouvinte tem a duração de 600 segundos (10 minutos), conforme mostra a figura 12.

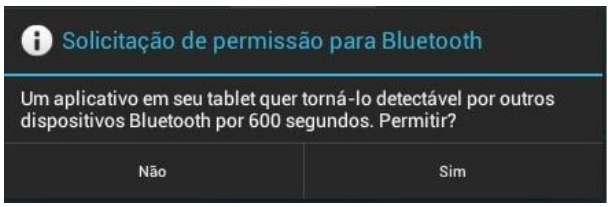

Figura 12 – Tela de solicitação para detecção de dispositivos
Fonte: Autoria própria.

Uma vez autorizado, o dispositivo ouvinte permanece em espera até o encerramento do tempo de visibilidade ou a conclusão do pareamento. Durante

esse período o botão ouvinte permanece ativo na cor amarela, indicando a tentativa de conexão, enquanto que o botão locutor é desabilitado. A figura 13 exibe essa situação.

Figura 13 – Tela de conexão ao locutor
Fonte: Autoria própria.

Este procedimento de conexão é possível através do método *BluetoothAdapter.listenUsingRfcommWithServiceRecord(java.lang.String, java.util.UUID)* que é chamado para criar um servidor Bluetooth indicando um identificador único global UUID (*Universally Unique Identifier*) do serviço SDP. Este identificador é exatamente o mesmo que será utilizado na outra ponta do enlace no dispositivo locutor para que este possa se conectar ao dispositivo ouvinte e identificar a aplicação entre outras em atividade no aparelho.

Todo esse processo acaba por criar um objeto servidor *BluetoothServerSocket*, que identifica o serviço de onde é possível aceitar conexões de entrada através do método *BluetoothServerSocket.accept()*. Enquanto aguarda por conexões de entrada, o *software* fica bloqueado até que o dispositivo locutor, identificando-se com a mesma UUID do servidor, inicie a conexão.

Após o dispositivo locutor ter iniciado a conexão, o método *BluetoothServerSocket.accept()* retorna o objeto *BluetoothSocket* que identifica a conexão entre os dois dispositivos. A figura 14 mostra a tela de conexão efetuada com sucesso.

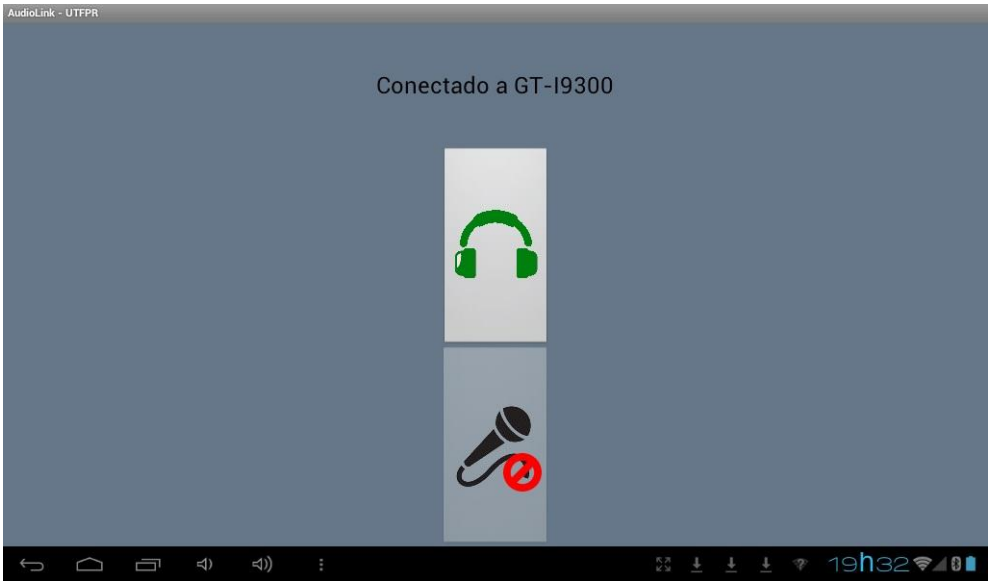

Figura 14 – Conexão estabelecida entre ouvinte e locutor
Fonte: Autoria própria.

Através do *BluetoothSocket* deve-se solicitar um objeto do tipo *InputStream,* pelo método *BluetoothSocket.getInputStream()*. O papel da classe *InputStream* é o de representar um fluxo de entrada de bytes para a classe que o implementa. Portanto, dentro da classe *BluetoothSocket,* ela representa a recepção de dados via socket Bluetooth.

De maneira contínua o dispositivo ouvinte lê dados enviados pelo dispositivo locutor através da função *InputStream.read(byte[], int, int)*, onde, após a leitura, as amostras de áudio são regeneradas através da classe *AudioTrack* pelo método *AudioTrack.write(byte[], int, int)*.

Esse método copia as amostras para o *mixer* de reprodução do aparelho e são reproduzidas de acordo com a taxa de amostragem selecionada, respeitando o método *First-In First-Out* (FIFO), controlada diretamente pelo sistema Android.

4.2.1.2 *Modo Locutor*

Neste modo o aplicativo é responsável pelas etapas de amostragem de áudio e transmissão das amostras através da conexão Bluetooth estabelecida com o ouvinte. Uma vez selecionada a opção, o usuário se depara com uma tela contendo

um histórico de dispositivos que já foram pareados em algum momento e que podem estar aptos a uma nova conexão. A lista de dispositivos já pareados é obtida através do método *BluetoothAdapter.getBondedDevices().*

O usuário pode ainda realizar a procura por outros dispositivos próximos. Isto é possível através da utilização do método *BluetoothAdapter.startDiscovery*(). A figura 15 mostra a tela exibida para seleção.

Figura 15 – Tela de seleção de dispositivos
Fonte: Autoria própria.

Cada dispositivo listado é representado internamente por um objeto da classe *BluetoothDevice* que concentra as informações de cada dispositivo. Assim que o dispositivo desejado é selecionado para comunicação, a cor do botão locutor se altera para a cor amarela, indicando a tentativa de conexão, enquanto que o botão ouvinte é desabilitado. Durante esse processo, o aplicativo realiza a criação da conexão através do método *BluetoothDevice.createRfcommSocketToServiceRecord(java.util.UUID)* utilizando a mesma UUID obtida do dispositivo ouvinte, já que esta funciona como seleção do serviço no dispositivo de destino através de uma consulta SDP realizada pela própria API Bluetooth Android. A figura 16 mostra a situação descrita.

Figura 16 – Tela de conexão ao ouvinte
Fonte: Autoria própria.

Assim que a conexão é estabelecida através do método *BluetoothSocket.connect(,)* o aplicativo locutor continuamente grava amostras de áudio através da chamada *AudioRecord.read(byte[], int, int)* solicitando o valor mínimo destas requerido pela classe *AudioRecord*, sendo este valor dependente do aparelho utilizado. Nesse processo, o aplicativo permanece coletando amostras até que o volume total requerido para gravação esteja completo. Somente quando a amostragem estiver finalizada, o conjunto de dados pode ser enviado através da conexão Bluetooth.

Para o envio das amostras de áudio é necessário obter através da classe *BluetoothSocket* o objeto *OutputStream* pelo método *BluetoothSocket.getOutputStream()*. O objetivo da classe *OutputStream* é de representar um fluxo de saída de bytes para a classe que o implementa, portanto dentro da classe *BluetoothSocket* ela representa o envio de dados pelo socket Bluetooth. Através da chamada *OutputStream.write(byte[]) são enviadas as amostras de áudio* ao ouvinte conectado. A figura 17 mostra a tela da conexão efetuada no dispositivo locutor.

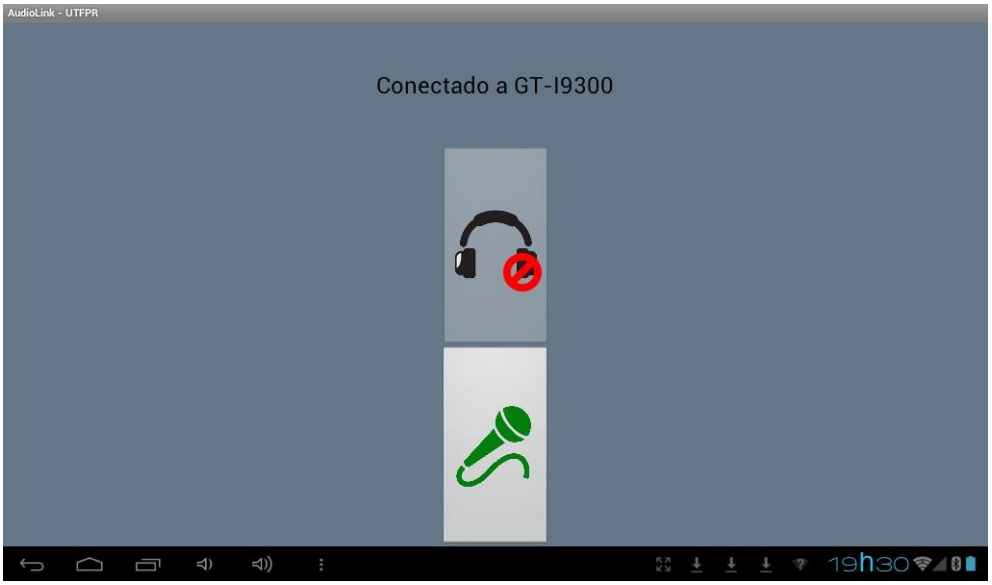

Figura 17 – Conexão estabelecida entre locutor e ouvinte
Fonte: Autoria própria.

O modo locutor possui um número máximo de tentativas de conexão ou reconexão com o dispositivo selecionado, sendo estes num total de 10. Caso a conexão com o dispositivo ouvinte seja perdida durante a transmissão, o botão locutor se torna amarelo e a mensagem de reconexão é exibida ao usuário.

Somente quando as 10 tentativas se esgotarem é que o aplicativo locutor irá considerar a conexão como perdida, tornando o botão locutor vermelho e exibindo a mensagem de conexão perdida ao usuário, conforme mostra a figura 18.

Figura 18 – Tela de conexão perdida
Fonte: Autoria própria.

4.2.2 Requisitos de *Hardware* e *Software*

O aplicativo AudioLink é bastante flexível quanto aos requisitos para sua utilização, ficando somente obrigatória a utilização de aparelhos com sistema Android 2.0 (API 5) ou superior e que sejam dotados de comunicação Bluetooth. Ambos os itens são tidos como básicos e estão disponíveis mesmo nos aparelhos mais simples do mercado de *smartphones*.

O seu uso não fica limitado apenas a aparelhos celulares, podem também ser utilizados em *tablets* ou qualquer outro dispositivo, desde que atendam aos requisitos já mencionados.

4.3 CIRCUITO DE EQUALIZAÇÃO DE ÁUDIO

Para realização das etapas de filtragem de ruídos e equalização do áudio, utilizou-se como base o circuito integrado BA3812L. Este componente consiste em um equalizador gráfico de áudio com capacidade para controle de tons até cinco canais, sendo ideal para uso em aplicações que requerem qualidade de som estéreo.

Dentre as suas principais características estão baixa distorção e ruído, larga faixa de operação em frequências e alimentação com uma banda de operação dinâmica que compreende desde 3.5 até 16 Volts, o que proporciona uma fácil adaptação à maioria dos equipamentos de áudio.

Os cinco canais que correspondem às frequências passíveis de serem usadas para equalização, podem ser definidas de forma independente de acordo com valores de capacitores externos utilizados.

A figura 19 mostra o módulo referente a um dos canais disponíveis no CI.

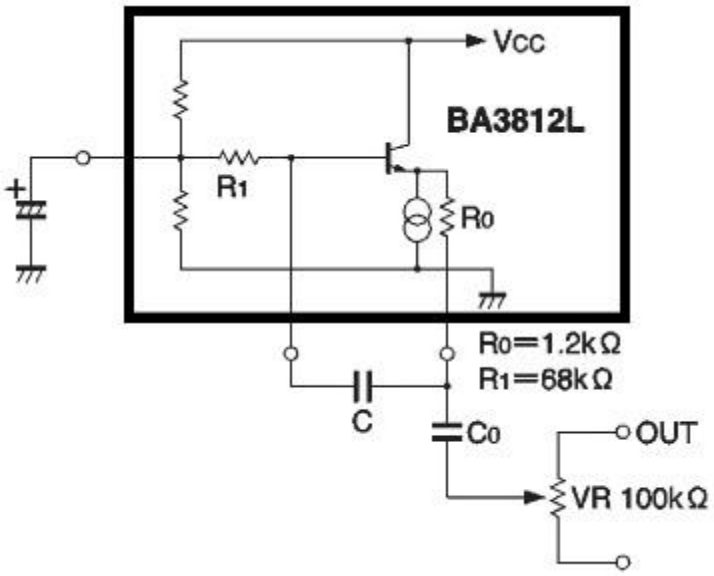

Figura 19 – Circuito interno usado no chip ba3812l
Fonte: Rohm semiconductor (2012).

Basicamente se vê dentro do retângulo um indutor de estado sólido que utiliza um elemento ativo, sendo estes diretamente integrados ao chip. Os demais componentes externos utilizados se resumem a capacitores e um potenciômetro.

O diagrama completo do CI com os cinco canais empregados e seus respectivos módulos para cada frequência podem ser observado na figura 20.

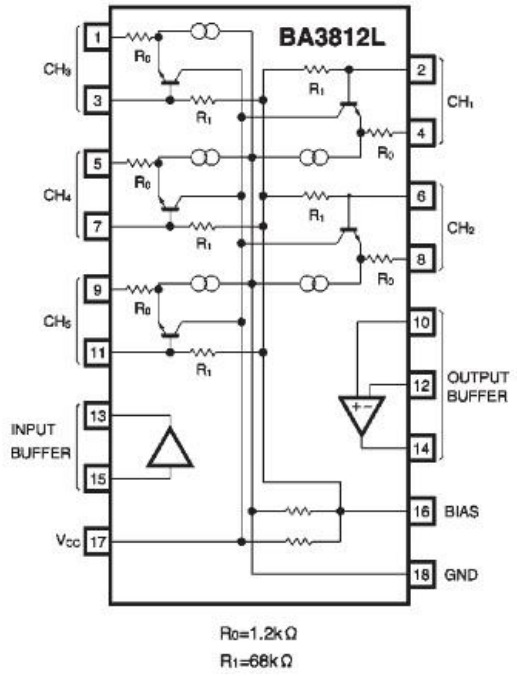

Figura 20 – Diagrama em bloco do chip BA3812L
Fonte: Rohm semiconductor (2012).

O circuito projetado para uso no trabalho desenvolvido se compõe basicamente do CI já relatado e capacitores externos que são definidos de acordo com as frequências de ressonância calculadas através da seguinte fórmula:

$$fo\ (Hz) = \frac{1}{2\pi\sqrt{R0.R1.C.C0}}\ (R = \Omega,\ C = F)$$

Além da frequência de ressonância, tem-se ainda que calcular o fator Q, o qual determina a largura de banda da frequência, sendo que, quanto maior o valor de Q, mais estreita é a faixa de frequência de ressonância. A fórmula de cálculo desse fator é dada por:

$$Q = \sqrt{\frac{C.R1}{C0.R0}}$$

Vale lembrar que para o projeto foram levadas em consideração apenas três faixas de frequência correspondentes à correção de deficiência auditiva do tipo “colher”, situadas em 600Hz, 2KHz e 3KHz, mas nada impede que outras faixas sejam utilizadas para correção da deficiência.

O valor do fator Q de cada banda possui um faixa de valores que devem ser obtidos com o uso de capacitores para efetiva correção desse tipo de deficiência. Os filtros devem respeitar às seguintes faixas:

- Filtro de 600Hz: valor de Q entre 0,2 à 0,7;
- Filtro de 2kHz: valor de Q entre 1 à 1,8;
- Filtro de 3kHz: valor de Q entre 0.9 e 1,4.

Tendo essas informações, foram efetuados os cálculos utilizando as fórmulas anteriormente descritas, sendo obtidos os valores da tabela 3.

Tabela 2 – Valores de Q utilizados para cada filtro

Frequência Nominal (Hz)	fo	Q	C (F)	C0 (F)
600	557,15	0,23	1n	1u
2000	2149,26	1,10	1,2n	56n
3000	3386,96	1,19	820p	33n

Fonte: Autoria própria.

Os outros dois canais sobressalentes no CI permanecem sem utilização. Quanto à alimentação, optou-se pela utilização de uma bateria alcalina de 9 Volts acoplada ao circuito, o que lhe confere uma boa autonomia de uso.

Tendo definidos os parâmetros já relatados, foi efetuado o seguinte desenho esquemático do circuito, mostrado na figura 21.

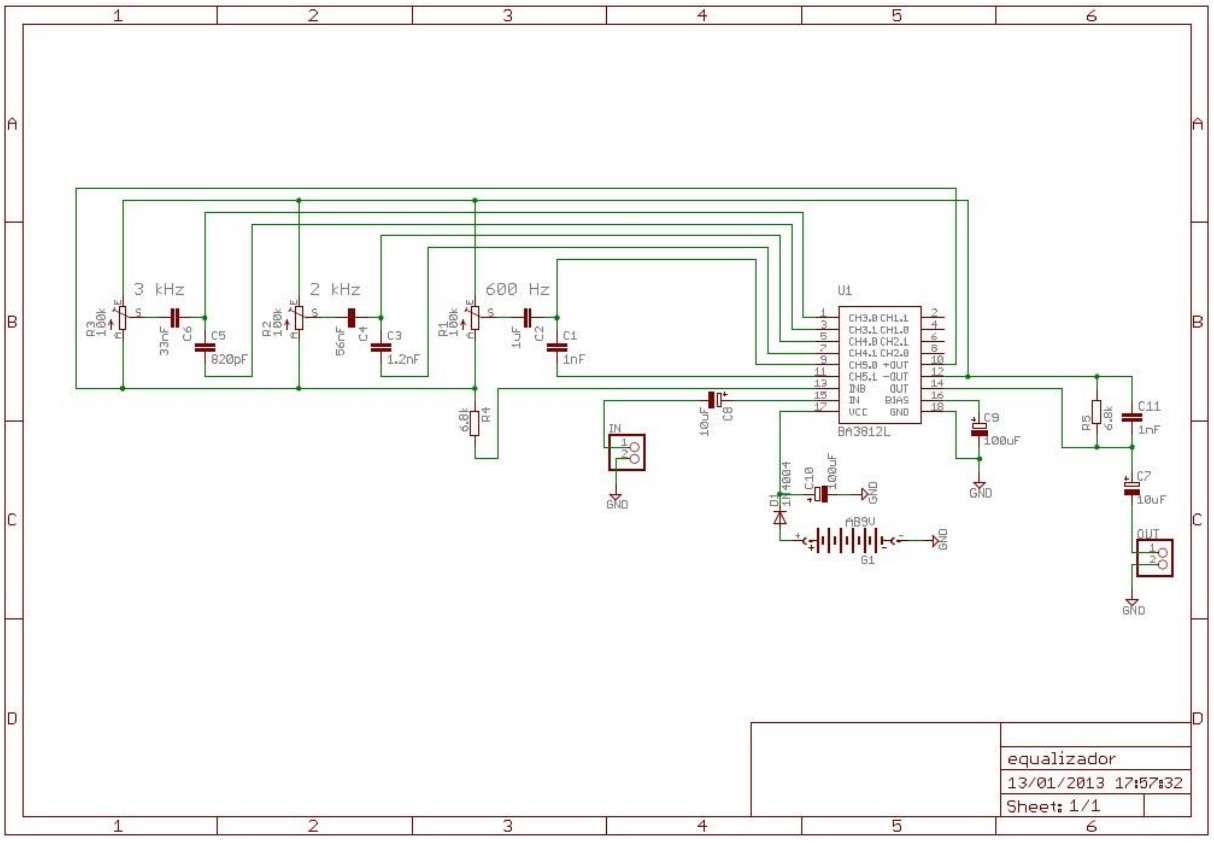

Figura 21 – Esquemático do circuito
Fonte: Autoria própria.

Com base nesse esquemático, o circuito foi montado em uma *protoboard* a fim de efetuarmos testes de maneira isolada e também acoplado ao dispositivo móvel dotado do aplicativo AudioLink a fim de comprovarmos seu propósito.

Para ajuste do controle do ganho em cada filtro implementado, o usuário dispõe de três potenciômetros (referentes a cada frequência), que controlam a potencia aplicada ao circuito. O ajuste do volume do áudio é realizado no próprio dispositivo móvel através de seu controle nativo.

O capítulo a seguir relata os testes realizados em laboratórios da UTFPR e seus resultados.

5 RESULTADOS E ANÁLISE

Após o desenvolvimento do aplicativo AudioLink e montagem do circuito eletrônico responsável pela equalização do áudio, foram efetuados testes para determinação do seu funcionamento.

Em um primeiro momento foram realizados testes do aplicativo não acoplado ao circuito equalizador para verificação de aspectos de alcance da comunicação em diferentes distâncias, estabilidade da conexão e inteligibilidade do áudio transmitido sem nenhum tipo de correção. Foram constatadas as seguintes situações:

- A qualidade do áudio notada no dispositivo ouvinte se mostrou satisfatória. A informação transmitida foi compreendida sem problemas;
- A comunicação entre os dispositivos realizada em diferentes distâncias demonstrou que o raio de alcance do Bluetooth é suficiente para utilização em ambientes de pequenas ou grandes dimensões, sem obstáculos. O enlace de comunicação foi estabelecido a mais de 30 metros de distância entre os aparelhos;
- Os dispositivos permaneceram conectados durante longos períodos para verificação da estabilidade da comunicação e não houve quedas no enlace.

Em uma segunda fase de testes, foram levantadas as curvas de ganho nas frequências pré-determinadas no circuito de equalização. Esta etapa foi realizada em laboratórios da instituição com o uso de equipamentos próprios, entre eles, fonte de alimentação destinada ao circuito, gerador de sinais para injeção de frequências relevantes no estudo e osciloscópio para análise das formas de ondas e ganho obtidos. Os seguintes gráficos, figuras 22 à 24, demonstram os resultados constatados em cada filtro implementado:

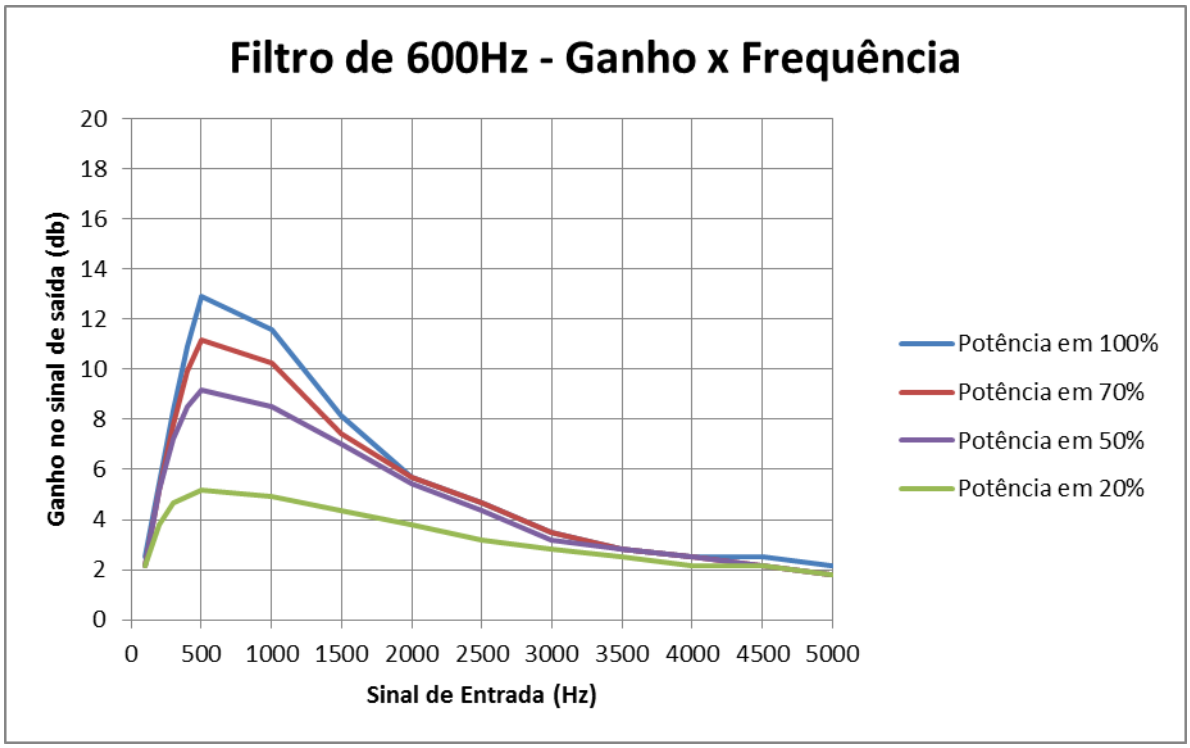

Figura 22 – Gráfico de ganho para filtro de 600Hz
Fonte: Autoria própria.

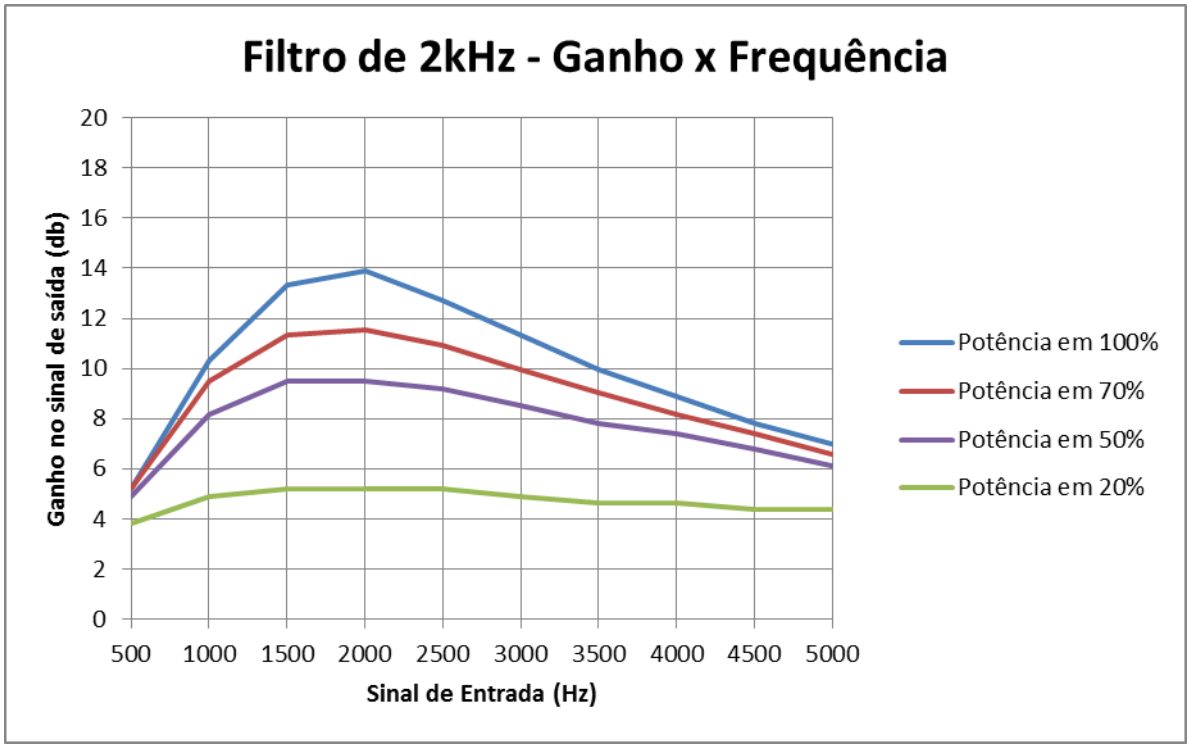

Figura 23 – Gráfico de ganho para filtro de 2KHz
Fonte: Autoria própria.

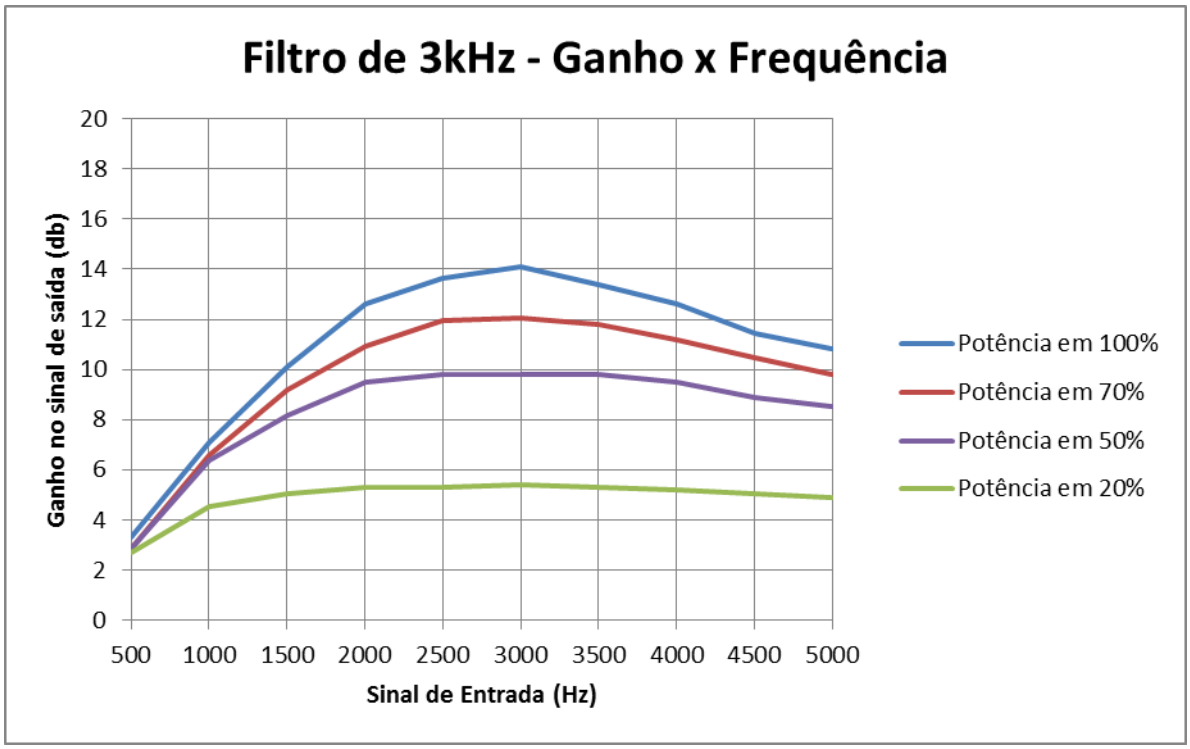

Figura 24 – Gráfico de ganho para filtro de 3KHz
Fonte: Autoria própria.

Como teste final, foram realizadas comunicações entre dispositivos dotados do aplicativo desenvolvido considerando o acoplamento do módulo ouvinte ao circuito equalizador. O ganho obtido com a presença do equalizador foi notório, mesmo para pessoas que não possuem perda auditiva, o que comprovou um real ganho e a eficácia do projeto.

6 CONCLUSÃO

É de extrema importância em uma sociedade que todos os indivíduos contidos nela tenham direitos e condições iguais em todos os aspectos, inclusive no que se refere ao aprendizado. O projeto desenvolvido tem entre outros aspectos, o intuito de agregar dispositivos móveis de uso cotidiano a um simples módulo externo, de maneira a oferecer uma solução atual e moderna para deficientes auditivos.

Esse trabalho proposto e demonstrado foi de extrema valia não somente pelo aspecto de auxílio a pessoas que venham a se beneficiar em sua utilização, mas também pela demonstração de como o desenvolvimento de aplicativos em plataformas do tipo *Open Source* está em plena expansão e acabam por gerar uma grande variedade de soluções nas mais diversas áreas.

Devemos destacar também que o aprendizado adquirido durante o curso foi determinante para se concluir o trabalho com êxito, tanto as disciplinas relacionadas à área de exatas quanto às de humanas se mostraram fundamentais para a idealização do projeto e sua construção efetiva.

Por fim, é relevante mencionar que é de interesse de todos que cada vez mais novas soluções sejam apresentadas em benefício da sociedade através de abordagens que contribuam ou levem ao desenvolvimento de novos estudos, sendo este o maior propósito do projeto aqui desenvolvido.

6.1 CONTRIBUIÇÕES E TRABALHOS FUTUROS

Dentre as contribuições do trabalho está o próprio desenvolvimento do aplicativo em si que pode servir de referência para outros programadores em futuras implementações de novas funcionalidades, entre elas, possibilitar a comunicação entre vários dispositivos ouvintes com um mesmo locutor, sendo isso perfeitamente possível visto o que a tecnologia Bluetooth disponibiliza.

Referente ao módulo equalizador de áudio, pode-se projetar uma placa de circuito impresso para soldagem dos componentes e montagem em uma caixa de

pequenas dimensões para utilização junto ao dispositivo ouvinte, visando a manipulação deste de maneira mais prática.

No intuito de deixar o projeto ainda mais compacto, fica aqui a ideia de utilizar filtros digitais no Android em conjunto ao aplicativo AudioLink, o que resultaria na eliminação do *hardware* equalizador.

Por último, é importante mencionar que a saída de áudio tratado disponibilizada atualmente prevê o uso juntamente de fones de ouvido do tipo *headset*, o que por si só reduz a exposição do ouvido a ruídos externos, porém o projeto também idealiza o uso em conjunto com o próprio aparelho auditivo do deficiente, o que pode acarretar em uma maior eficiência por parte deste.

REFERÊNCIAS

A. R. Gomes e Silva; J. M. M. da Silva; H. M.Oliveira. **Equalizadores perceptuais para melhoria de sinais de voz, baseado em resultados de exames de audiometria**, Recife, jul. 2008. Disponível em: <www2.ee.ufpe.br/codec/CBEB2008.pdf>. Acesso em: 24 fev. 2013.

ALECRIM, Emerson. **Tecnologia Bluetooth**. Disponível em: <http://www.infowester.com/bluetooth.php>. Acesso em: 05 nov. 2012.

AGÊNCIA NACIONAL DE TELECOMUNICAÇÕES. **RESOLUÇÃO Nº506:** Regulamento sobre Equipamentos de Radiocomunicação de Radiação Restrita**.** Brasília, 2008.

ANDROID DEVELOPERS, INC (a). ***Android, the world's most popular mobile platform***. 2012a. Disponível em: <http://developer.android.com/about/index.html>. Acesso em: 04 nov. 2012.

ANDROID DEVELOPERS, INC (b). ***android.bluetooth****.* Disponível em: <http://developer.android.com/reference/android/bluetooth/package-summary.html>. Acesso em: 03 dez. 2012.

ANDROID DEVELOPERS, INC (c). ***android.media***. Disponível em: <http://developer.android.com/reference/android/media/package-summary.html>. Acesso em: 02 dez. 201.

ANDROID DEVELOPERS, INC (d). ***App Framework***. Disponível em: <http://developer.android.com/guide/topics/manifest/uses-sdk-element.html#ApiLevels>. Acesso em: 03 jan. 2012.

ANDROID DEVELOPERS, INC (e). ***What is API level?***. Disponível em: <http://developer.android.com/about/versions/index.html>. Acesso em: 10 out. 2012.

BLUETOOTH SIG. **Bluetooth**. Disponível em: <http://www.bluetooth.com/Pages/Bluetooth-Home.aspx>. Acesso em: 16 nov. 2012.

CREATIVE COMMONS, **Funcionamento do Bluetooth**. Kioskea.net. Disponível em <http://pt.kioskea.net/contents/bluetooth/bluetooth-fonctionnement.php3>. Acesso em: 25 nov. 2012.

DREOSSI, Raquel Cecilia Fischer. **O ruído e sua interferência sobre estudantes em uma sala de aula:** revisão de literatura. Pró-Fono Revista de Atualização Científica, Barueri, v. 17, n. 2, p. 251-258, maio-ago. 2005.

ERICSSON HISTORY. **Ericsson Review:** the telecommunications technology journal. Disponível em: <http://ericssonhistory.com/Global/Ericsson%20review/Ericsson%20Review.%20199 8.%20V.75/Ericsson_Review_Vol_75_1998_3.pdf>. Acesso em: 04 out. 2012.

FERNANDES, João Candido. **Padronização das condições acústicas para salas de aula**. XIII SIMPEP - Universidade Estadual Paulista Júlio de Mesquita Filho, Bauru, 2006.

INÁCIO, Wederson Honorato. **A inclusão escolar do deficiente auditivo:** contribuições para o debate educacional**.** Universidade Federal de Uberlândia, Instituto de Psicologia, 2009.

INSTITUTO BRASILEIRO DE GEOGRAFIA E ESTATÍSTICA: **Censo Demográfico.** Brasil, 2000. Disponível em: <http://www.ibge.gov.br/home/presidencia/noticias/noticia_visualiza.php?id_noticia=4 38&id_pagina=1>. Acesso em: 15 nov. 2012.

LECHETA, Ricardo R. **Google Android: aprenda a criar com aplicações para dispositivos móveis com Android SDK** / Ricardo R. Lecheta. – 2. Ed. Ver. E ampl. – São Paulo: Novatec Editora, 2010.

MORIMOTO, Carlos Eduardo. **Redes, guia prático**. Porto Alegre: Sul Editores, 2008. 555p.

NETO, Fernando; LIMA FILHO, Jader. **Projeto de um filtro equalizador CMOS de baixa tensão para dispositivos de auxílio auditivo** (Trabalho de iniciação científica) – Curso de Engenharia Elétrica, Universidade Estadual Paulista, Guaratinguetá, 2003.

OLIVEIRA, João Paulo dos Santos. **Acessibilidade no Android**. Disponível em: <http://www.universosmart.com.br/tecnologia/acessibilidade-no-android-103.html>. Acesso em: 12 jan. 2012.

PETRAGLIA, Antônio; BARÚQUI, Fernando Antônio Pinto. **Correção de perdas auditivas induzidas por ruído**, Universidade Federal do Rio de Janeiro, Escola de Engenharia, Departamento de Eletrônica e Computação, 2002.

PHILIPS ELECTRONICS. **Fone estéreo Bluetooth**. Disponível em: <http://www.p4c.philips.com/cgi-bin/dcbint/cpindex.pl?scy=BR&mid=Link_FAQs&view=aa12_view_partial.html&sessi on=20110802021914_66.249.72.73&list=aa12_list_partial.html&slg=POR&ctn=SHB9 001WT/00&dct=FAQ&refnr=0067633&faqview=>. Acesso em: 18 nov. 2012.

PROJECT, A. O. S. (2012). **Android open source**. Disponível em: <http://source.android.com>. Acesso em: 15 nov. 2012.

QUEIROZ, L. E. C. **Protocolo de Redes Bluetooth**. Distribuição e Integração de Sistemas. Barcarena, 2008.

RAPOPORT, Eduardo; PETRAGLIA, Antônio. **Estudo e correção de perdas auditivas induzidas por ruídos usando equalizadores programáveis**. In: CONGRESSO NACIONAL BRASILEIRO DE AUTOMÁTICA, 14. , 2002, Natal. Disponível em: <http://www.sigma.ufrj.br/UFRJ/SIGMA/jornadaIC/publicacao_foco/trabalhos/consulta/relatorio.stm?app=JIC_PUBLICACAO_TRABALHO&ano=2001&codigo=83&buscas_cruzadas=ON>. Acesso em 21 dez. 2011.

ROHM SEMICONDUCTOR, **BA3812L Datasheet 5 channel graphic equalizer**. Disponível em: <http://www.alldatasheet.com/datasheetpdf/pdf/35974/ROHM/BA3812L.html>. Acesso em: 24 fev. 2013.

SOUND INSTITUTE, **Equalizer**. Disponível em: <http://www.soundinstitute.com/article_detail.cfm/ID/110>. Acesso em: 24 nov. 2012.

TANENBAUM, A. S. **Sistemas Operacionais Modernos**. Editora Pearson, 2. Ed.. São Paulo (2003).

TECHNOLOGY of Hearing Aids: Analog vs. Digital. **HearingCentral**. Disponível em: <http://www.hearingcentral.com/hearingAidTech.asp> Acesso em: 09 nov. 2011.

TURNER, D., WILHELM, R.; LEMBERG, W. (2006). **The free type project freetype 1.** Disponível em: <http://freetype.sourceforge.net/freetype1/index.html> Acesso em: 25 nov. 2012.

WHO: WORLD HEALTH ORGANIZATION. **World Report on Disability**, New York, 2011. 325p. Disponível em: <http://whqlibdoc.who.int/publications/2011/9789240685215_eng.pdf>. Acesso em: 12 nov. 2011.

APÊNDICE A – CÓDIGO FONTE DO MÓDULO “TELA PRINCIPAL”

Principal.java

```
00001 /** Tela principal do aplicativo.
00002  * @author Everton Anjos, Mauricio Fiszt
00003  * @file Principal.java
00004  */
00005 package com.utfpr.audiolink;
00006
00007 import android.os.Bundle;
00008 import android.os.Handler;
00009 import android.os.Message;
00010 import android.app.Activity;
00011 import android.view.View;
00012 import android.view.View.OnClickListener;
00013 import android.widget.ImageButton;
00014 import android.widget.TextView;
00015 import android.widget.Toast;
00016 import android.content.Intent;
00017 import android.bluetooth.BluetoothAdapter;
00018 import android.bluetooth.BluetoothDevice;
00019 import android.util.Log;
00020
00021 /** Tela principal de utilização do aplicativo */
00022 public class Principal extends Activity {
00023   private static final String LOGTAG = "AudioLink"; ///< Tag para debug usada no Log.d.
00024   private static final int REQ_ENABLE_BT = 1; ///< Código de requisição para habilitação do bluetooth.
00025   private static final int REQ_DISCOVERABLE_BT = 2; ///< Código de requisição para habilitação da procura de outros
equipamentos via bluetooth.
```

```
00026   private static final int REQ_DEVICE_ADDRESS = 3; ///< Código de requisição para a procura e seleção de um dispositivo
para conexão.
00027   private BluetoothAdapter btAdapter = null; ///< Objeto de sistema para acesso ao adaptador bluetooth.
00028   private ImageButton btnStreamer = null; ///< Botão para entrar em modo streamer(locutor)
00029   private ImageButton btnListener = null; ///< Botão para entrar em modo listener(ouvinte)
00030   private BtListener listener = null; ///< Objeto ouvinte.
00031   private BtStreamer streamer = null; ///< Objeto locutor.
00032   private TextView intrucoes = null; ///< Texto para instruções no inicio da tela
00033   private ArrayAdapter<String> newDevices = null; ///< Lista de dispositivos novos encontrados através de pesquisa na
rede.
00034   /** Handler de recebimento de mensagens de outras threads. */
00035   private final Handler threadHandler = new Handler() {
00036     /** Controle de mensagens de outras threads através de Handle
00037      * @param msg mensagem recebida de outra thread
00038      */
00039     public void handleMessage(Message msg) {
00040       if (msg.what == AudioLinkConstantes.IPC.HEADER_MESSAGE) /// mensagem tipo alteração de instruções
00041         intrucoes.setText( msg.obj.toString() ); /// altera texto para instruções
00042       if (msg.what == AudioLinkConstantes.IPC.CLOSE_STREAMER) /// mensagem tipo fechar streamer
00043         streamer = null;
00044       if (msg.what == AudioLinkConstantes.IPC.CLOSE_LISTENER) /// mensagem tipo fechar listener
00045         listener = null;
00046       if (msg.what == AudioLinkConstantes.IPC.CHANGE_UI) { /// mensagem tipo Interface ao Usuário ( U_ser I_nterface)
00047         switch (msg.arg2) { /// de acordo com a ação desejada, faça
00048           case AudioLinkConstantes.UI.Action.ENABLE: /// habilitar objeto
00049             if (msg.arg1==AudioLinkConstantes.UI.LISTENER) { /// se botão ouvinte, faça
00050               btnListener.setEnabled(true);
00051               btnListener.setImageResource (R.drawable.btn_head); /// alterar imagem do botão ouvinte para .
00052             }
00053             else { /// senão, botão locutor, faça
00054               btnStreamer.setEnabled(true);
```

```
00055             btnStreamer.setImageResource (R.drawable.btn_mic); /// alterar imagem do botão locutor para .
00056           }
00057         break;
00058         case AudioLinkConstantes.UI.Action.DISABLE: /// desabilitar objeto
00059           if (msg.arg1==AudioLinkConstantes.UI.LISTENER) { /// se botão ouvinte, faça
00060             btnListener.setEnabled(false); /// desabilitar botão ouvinte.
00061             btnListener.setImageResource (R.drawable.btn_head_dis); /// alterar imagem do botão ouvinte para notificar que
está desabilitado.
00062           }
00063           else { /// senão, botão locutor, faça
00064             btnStreamer.setEnabled(false); /// desabilitar botão locutor.
00065             btnStreamer.setImageResource (R.drawable.btn_mic_dis); /// alterar imagem do botão locutor para notificar que
está desabilitado.
00066           }
00067         break;
00068         case AudioLinkConstantes.UI.Action.OK: /// mostrar funcionamento OK
00069           if (msg.arg1==AudioLinkConstantes.UI.LISTENER) /// se botão ouvinte, faça
00070             btnListener.setImageResource (R.drawable.btn_head_on); /// alterar imagem do botão ouvinte para demonstrar
funcionamento correto.
00071           else /// senão, botão locutor, faça
00072             btnStreamer.setImageResource (R.drawable.btn_mic_on); /// alterar imagem do botão locutor para demonstrar
funcionamento correto.
00073         break;
00074         case AudioLinkConstantes.UI.Action.WARNING: /// mostar funcionamento com problemas/indeterminado
00075           if (msg.arg1==AudioLinkConstantes.UI.LISTENER) /// se botão ouvinte, faça
00076             btnListener.setImageResource (R.drawable.btn_head_warn); /// alterar imagem do botão ouvinte para demonstrar
problemas.
00077           else /// senão, botão locutor, faça
00078             btnStreamer.setImageResource (R.drawable.btn_mic_warn); /// alterar imagem do botão locutor para demonstrar
problemas.
00079         break;
```

```
00080         case AudioLinkConstantes.UI.Action.ERROR: /// mostrar mau funcinamento ou erros
00081           if (msg.arg1==AudioLinkConstantes.UI.LISTENER) /// se botão ouvinte, faça
00082             btnListener.setImageResource (R.drawable.btn_head_error); /// alterar imagem do botão ouvinte para demostrar
erro.
00083           else /// senão, botão locutor, faça
00084             btnStreamer.setImageResource (R.drawable.btn_mic_error); /// alterar imagem do botão locutor para demostar
erro.
00085         break;
00086         default: /// opção padrão
00087           if (msg.arg1==AudioLinkConstantes.UI.LISTENER) /// se botão ouvinte, faça
00088             btnListener.setImageResource (R.drawable.btn_head); /// alterar imagem do botão ouvinte para desenho padrão.
00089           else /// senão, botão locutor, faça
00090             btnStreamer.setImageResource (R.drawable.btn_mic); /// alterar imagem do botão locutor para desenho padrão.
00091         break;
00092       }
00093     }
00094   }
00095 };
00096
00097 @Override
00098 /** Evento chamado quando a tela será criada */
00099 public void onCreate(Bundle savedInstanceState) {
00100   super.onCreate(savedInstanceState);
00101   setContentView(R.layout.activity_principal); /// aplica layout desta tela a partir de res/layout/activity_principal.xml
00102   Log.d(LOGTAG, "onCreate");
00103   if ((btAdapter = BluetoothAdapter.getDefaultAdapter()) == null) { /// Checar disponibilidade bluetooth no aparelho
00104     Toast.makeText(this, "Bluetooth não está disponível neste aparelho!", Toast.LENGTH_LONG).show(); /// mostra ao
usuário que o recurso não está disponível neste aparelho
00105     finish(); /// sai do aplicativo
00106
00107   }
```

```
00108    intrucoes = (TextView) findViewById(R.id.textMessageTop); /// obtém objeto de texto para intruções ao usuário
00109    btnListener = (ImageButton) findViewById(R.id.btn_listener); /// obetém objeto do botão listener(ouvinte)
00110    /** Confiugura ouvinte para o evento onClick no botão ouvinte */
00111    btnListener.setOnClickListener(new OnClickListener() {
00112      public void onClick(View v) {
00113        if ( listener == null ) { /// se serviço ouvinte estiver desligado, faça
00114          if (btAdapter.getScanMode() != BluetoothAdapter.SCAN_MODE_CONNECTABLE_DISCOVERABLE) { /// se
visibilidade bluetooth estiver desligada, faça
00115            Intent discoveryBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE); /// criar intent para
a ativação da visibilidade Bluetooth
00116            discoveryBtIntent.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION, 600); /// adicionar tempo
desejado para permanecer visível
00117            startActivityForResult(discoveryBtIntent, REQ_DISCOVERABLE_BT); /// inicia atividade para autorização do
usuário
00118          }
00119          else {
00120            listener = new BtListener (threadHandler); /// criar novo processo ouvinte
00121            listener.start(); /// iniciar processo ouvinte
00122          }
00123        }
00124        else /// se serviço ouvinte estiver ligado, faça
00125          listener.cancel(); /// cancelar processo ouvinte
00126      }
00127    });
00128    btnStreamer = (ImageButton) findViewById(R.id.btn_streamer); /// obetém objeto do botão streamer(locutor)
00129    /** Confiugura ouvinte para o evento onClick no botão locutor */
00130    btnStreamer.setOnClickListener(new OnClickListener() {
00131      public void onClick(View v)  {
00132        if (streamer == null) { /// se o serviço locutor estiver desligado, faça
00133          Intent selOuvinteIntent = new Intent(getBaseContext(), SelOuvinte.class); /// cria intent para seleção de um
dispositivo para conectar-se
```

```
00134           startActivityForResult(selOuvinteIntent, REQ_DEVICE_ADDRESS); /// inicia atividade para solicitar um dispositivo
para conexão
00135         }
00136         else
00137           streamer.cancel(); /// cancelar processo locutor
00138       }
00139     });
00140   }
00141
00142   @Override
00143   /** Evento chamado quando a tela será disponibilizada ao usuário */
00144   public void onStart() {
00145     super.onStart();
00146     Log.d(LOGTAG, "onStart");
00147     if (!btAdapter.isEnabled()) { /// se bluetooth está desligado, faça
00148       Intent enBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE); /// cria Intent para ativação de
bluetooth
00149       startActivityForResult(enBtIntent, REQ_ENABLE_BT); /// inicia tela de solicitação para ativação de bluetooth
00150     }
00151   }
00152
00153   @Override
00154   /** Evento chamado quando a tela será finalizada */
00155   public void onDestroy() {
00156     super.onDestroy();
00157     Log.e(LOGTAG, "onDestroy");
00158     if (listener != null) { /// se ouvinte estiver em funcionamento, faça
00159       listener.cancel(); /// finalizar processo ouvinte
00160       listener=null; /// zerar processo ouvinte
00161     }
00162     if (streamer != null) { /// se locutor estiver em funcionamento, faça
```

```
00163       streamer.cancel(); /// finalizar processo locutor
00164       streamer=null; /// zerar objeto locutor
00165     }
00166   }
00167
00168   @Override
00169   /** Receptor de resultados de outras telas (Activity) */
00170   public void onActivityResult(int requestCode, int resultCode, Intent data) {
00171     Log.d(LOGTAG, "onActivityResult > requestCode:"+requestCode+" , resultCode:"+resultCode);
00172     switch (requestCode) {
00173       case REQ_ENABLE_BT: /// solicitação para habilitação de bluetooth
00174         if (resultCode != Activity.RESULT_OK) { /// se não foi autorizado, faça
00175           Toast.makeText(this, "Habilitação do Bluetooth não permitida pelo usuário, saindo...",
Toast.LENGTH_SHORT).show(); /// mostra mensagem ao usuário
00176           finish(); /// finaliza aplicação
00177         }
00178       break;
00179       case REQ_DISCOVERABLE_BT: /// solicitação para habilitar visibilidade bluetooth
00180         if (resultCode < 600) { /// se não foi autorizado, faça
00181           Toast.makeText(this, "Procura remota não permitida pelo usuário, saindo...", Toast.LENGTH_SHORT).show(); ///
mostrar mensagem ao usuário
00182           finish(); /// finalizar aplicação
00183         }
00184         else { /// se visibilidade foi autorizada, faça
00185           listener = new BtListener (threadHandler); /// criar serviço ouvinte
00186           listener.start(); /// iniciar ouvinte
00187         }
00188       break;
00189       case REQ_DEVICE_ADDRESS: /// solicitação de seleção de dispositivo para conexão
00190         if (resultCode == Activity.RESULT_OK) { /// se foi selecionado algum dispositivo, faça
```

```
00191           String address = data.getExtras().getString(SelOuvinte.EXTRA_DEVICE_ADDRESS); /// obter endereço MAC do
dispositivo selecionado
00192           BluetoothDevice device = btAdapter.getRemoteDevice(address); /// criar objeto de dispositivo bluetooth
00193           streamer = new BtStreamer (threadHandler, device); /// criar serviço locutor
00194           streamer.start(); /// iniciar locutor
00195         }
00196       break;
00197     }
00198   }
00199 }
```

APÊNDICE B – CÓDIGO FONTE DO MÓDULO “SELECIONAR OUVINTE”

SelOuvinte.java

```
00001 /** Seleção de dispositivo para conexão no modo Locutor.
00002  * @author Everton Anjos, Mauricio Fiszt
00003  * @file SelOuvinte.java
00004  */
00005 package com.utfpr.audiolink;
00006
00007 import android.app.Activity;
00008 import android.bluetooth.BluetoothAdapter;
00009 import android.bluetooth.BluetoothDevice;
00010 import android.content.BroadcastReceiver;
00011 import android.content.Context;
00012 import android.content.Intent;
00013 import android.content.IntentFilter;
00014 import android.os.Bundle;
00015 import android.util.Log;
00016 import android.view.View;
00017 import android.view.Window;
00018 import android.view.View.OnClickListener;
00019 import android.widget.AdapterView;
00020 import android.widget.ArrayAdapter;
00021 import android.widget.Button;
00022 import android.widget.ListView;
00023 import android.widget.TextView;
00024 import android.widget.AdapterView.OnItemClickListener;
00025 import java.util.Set;
00026
00027 /** Tela para seleção do dispositivo Bluetooth onde se encontra o ouvinte. */
```

```
00028 public class SelOuvinte extends Activity {
00029   private static final String LOGTAG = "SelOuvinte"; ///< Tag para debug usada no Log.d
00030   public static String EXTRA_DEVICE_ADDRESS = "device_address"; ///< endereço selecionado pelo usuário
00031   private BluetoothAdapter btAdapter = null; ///< Objeto de sistema para acesso ao adaptador bluetooth.
00032   private ArrayAdapter<String> pairedDevices = null; ///< Lista de dispositivos já pareados com este celular.
00033   private ArrayAdapter<String> newDevices = null; ///< Lista de dispositivos novos encontrados através de pesquisa na
rede.
00034
00035   private OnItemClickListener mDeviceClickListener = new OnItemClickListener() {
00036     /** Ouvinte do evento onClick da lista de dispositivos encontrados e/ou pareados */
00037     public void onItemClick(AdapterView<?> av, View v, int arg2, long arg3) {
00038       btAdapter.cancelDiscovery(); /// cancelar procura por novos dispositivos
00039       /** Obter MAC address do item selecionado pelo usuário */
00040       String info = ((TextView) v).getText().toString();
00041       String address = info.substring(info.length() - 17);
00042       Intent intent = new Intent(); /// criando objeto Intent para adicionar MAC selecionado
00043       intent.putExtra(EXTRA_DEVICE_ADDRESS, address); /// adicionando MAC ao Intent que será repassado á tela
anterior
00044       setResult(Activity.RESULT_OK, intent); /// setar resultado desta tela que contém o mac selecionado pelo usuário
00045       finish(); /// finalizar esta atividade(tela)
00046     }
00047   };
00048
00049   private final BroadcastReceiver mReceiver = new BroadcastReceiver() {
00050     @Override
00051     /** Ouvinte dos eventos de broadcast solicitados por esta tela */
00052     public void onReceive(Context context, Intent intent) {
00053       String action = intent.getAction(); ///< obtém nome da ação notificada
00054       if (BluetoothDevice.ACTION_FOUND.equals(action)) { /// evento de notificação de um novo dispositivo encontrado
00055         BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE); /// obtém detalhes do
dispositivo encontrado
```

```
00056         if (device.getBondState() != BluetoothDevice.BOND_BONDED) /// se o dispositivo encontrado não está na lista de dispositivos pareados, faça
00057           newDevices.add(device.getName() + "\n" + device.getAddress()); /// adiciona a lista de novos dispositivos encontrados
00058       } else if (BluetoothAdapter.ACTION_DISCOVERY_FINISHED.equals(action)) { /// evento de notificação do fim de procura por novos dispositivos
00059         setProgressBarIndeterminateVisibility(false); /// desabilitar barra de progresso
00060         setTitle(R.string.msg_sel_device); /// alterar título da janela
00061         if (newDevices.getCount() == 0) { /// se não foram encontrados novos dispositivos, faça
00062           String noDevices = getResources().getText(R.string.msg_none).toString(); /// cria texto de nehnum dispositivo encontrado
00063           newDevices.add(noDevices); /// adiciona mensagem que não foi encontrado nenhum dispositivo
00064         }
00065       }
00066     }
00067   };
00068
00069   @Override
00070   /** Evento chamado quando a tela será criada */
00071   protected void onCreate(Bundle savedInstanceState) {
00072     super.onCreate(savedInstanceState);
00073     requestWindowFeature(Window.FEATURE_INDETERMINATE_PROGRESS); /// habilita recurso da janela para processo indeterminado
00074     setContentView(R.layout.activity_selouvinte);  /// aplica layout desta tela a partir de res/layout/activity_selouvinte.xml
00075     Log.d(LOGTAG, "onCreate");
00076     setResult(Activity.RESULT_CANCELED); /// seta resultado desta atividade como RESULT_CANCELED por padrão caso o usuário saia da tela com o botão voltar
00077     Button scanButton = (Button) findViewById(R.id.btn_scanear); /// obtém objeto botão de scanner e instancia método onClick
00078     scanButton.setOnClickListener(new OnClickListener() {
00079       public void onClick(View v) {
```

```
00080         setProgressBarIndeterminateVisibility(true); /// mostra/ativa barra de progresso indeterminado
00081         setTitle(R.string.msg_scaneando); /// altera título da janela
00082         findViewById(R.id.msg_new_devices).setVisibility(View.VISIBLE); /// mostra sub-menu de dispositivos encontrados
00083         if (btAdapter.isDiscovering()) /// caso já se esteja procurando por dispositivos cancelar procura
00084           btAdapter.cancelDiscovery();
00085         btAdapter.startDiscovery(); /// iniciar pesquisa por novos dispositivos nos arredores
00086         v.setVisibility(View.GONE); /// remover visibilidade do botão de procura
00087       }
00088     });
00089     btAdapter = BluetoothAdapter.getDefaultAdapter(); /// obtém objeto do adaptador bluetooth no sistema
00090     pairedDevices = new ArrayAdapter<String>(this, R.layout.activity_item_device_name); /// inicializa array que contém os
dispositivos bluetooth já pareados com este aparelho
00091     newDevices = new ArrayAdapter<String>(this, R.layout.activity_item_device_name); /// inicializa array que conterá os
dispositivos novos encontrados nos arredores
00092     ListView pairedListView = (ListView) findViewById(R.id.paired_devices); /// busca objeto ListView no layout para
dispositivos já pareados
00093     pairedListView.setAdapter(pairedDevices); /// define origem dos dados que serão apresentados neste ListView
00094     pairedListView.setOnItemClickListener(mDeviceClickListener); /// configura onClickListener para receber a notificação de
item selecionado
00095     ListView newDevicesListView = (ListView) findViewById(R.id.new_devices); /// busca objeto ListView no layout para
novos dispositivos
00096     newDevicesListView.setAdapter(newDevices); /// define origem dos dados que serão apresentados neste ListView
00097     newDevicesListView.setOnItemClickListener(mDeviceClickListener); /// configura onClickListener para receber a
notificação de item selecionado
00098     IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND); /// configura filtro de mensagens para o evento
BluetoothDevice.ACTION_FOUND
00099     registerReceiver(mReceiver, filter); /// configura listener para receber notificações da API Bluetooth o evento
BluetoothDevice.ACTION_FOUND
00100     filter = new IntentFilter(BluetoothAdapter.ACTION_DISCOVERY_FINISHED); /// configura filtro de mensagens para o
evento BluetoothAdapter.ACTION_DISCOVERY_FINISHED
```

```
00101     registerReceiver(mReceiver, filter); /// configura listener para receber notificações da API Bluetooth o evento BluetoothAdapter.ACTION_DISCOVERY_FINISHED
00102     Set<BluetoothDevice> btPairedDevices = btAdapter.getBondedDevices(); /// obtém lista de dispositivos já pareados com este aparelho
00103     if (btPairedDevices.size() > 0) { /// se existirem dispositivos pareados
00104       findViewById(R.id.msg_paired_devices).setVisibility(View.VISIBLE); /// mostrar lista de dispositivos pareados
00105       for (BluetoothDevice device : btPairedDevices) { /// para cada dispositivo pareado, faça
00106         pairedDevices.add(device.getName() + "\n" + device.getAddress()); /// adiciona na lista de dispositivos pareados o o nome e endereço MAC do dispositivo
00107       }
00108     } else {
00109       String noDevices = getResources().getText(R.string.msg_none).toString(); /// cria texto de dispositivo não encontrado
00110       pairedDevices.add(noDevices); /// adiciona texto informando que não há dispositivos pareados
00111     }
00112   }
00113
00114   @Override
00115   /** Evento chamado quando a tela será finalizada */
00116   protected void onDestroy() {
00117     super.onDestroy();
00118     Log.e(LOGTAG, "onDestroy");
00119     if (btAdapter != null)
00120       btAdapter.cancelDiscovery(); /// cancelar procura por dispositivos
00121     unregisterReceiver(mReceiver); /// desregistrar listener's de eventos BluetoothDevice.ACTION_FOUND e BluetoothAdapter.ACTION_DISCOVERY_FINISHED
00122   }
00123 }
```

APÊNDICE C – CÓDIGO FONTE DO MÓDULO “LOCUTOR”

BtStreamer.java

```
00001 /** Serviço Locutor
00002  * @author Everton Anjos, Mauricio Fiszt
00003  * @file BtStreamer.java
00004  */
00005 package com.utfpr.audiolink;
00006
00007 import android.bluetooth.BluetoothDevice;
00008 import android.bluetooth.BluetoothSocket;
00009 import android.os.Handler;
00010 import android.os.Message;
00011 import android.util.Log;
00012 import java.io.IOException;
00013 import java.io.OutputStream;
00014 import android.media.AudioRecord;
00015 import android.media.MediaRecorder;
00016 import android.media.AudioFormat;
00017
00018 /** Thread de envio de stream de áudio para cliente Bluetooth. */
00019 public class BtStreamer extends Thread {
00020   private static final String LOGTAG = "BtStreamer"; ///< Tag para debug usada no Log.d
00021   private static final int MAXIMO_RECONEXOES = 10; ///< Define quantas tentativas serão realizadas para reconexão com
o ouvinte de áudio.
00022   private BluetoothDevice btDevice = null; ///< Objeto de sistema para as propriedades de um dispositivo.
00023   private BluetoothSocket btSocket = null; ///< Objeto socket para comunicação com o dispositivo conectado
00024   private OutputStream streamOut = null; ///< Output stream do socket do cliente conectado.
00025   private Handler activity = null; ///< Handler da Activity para notificação de eventos e atualização de dados.
00026   private int attempts = 0; ///< contador do número de tentativas de reconexão com o ouvinte de áudio.
```

```
00027   private boolean run = true; ///< controle do loop principal
00028   private AudioRecord input = null; ///< Objeto de sistema para amostragem de áudio.
00029   private int minBufferSize = 0; ///< Tamanho mínimo do buffer suportado pelo hardware para gravação de áudio.
00030   private byte[] buffer = null; ///< Buffer interno de áudio com mesmo tamanho que minBufferSize.
00031
00032   /** Contrutor do cliente Bluetooth e fonte de áudio
00033    * @param hdl Hanlder da activity que será utilizado para notificação de eventos
00034    * @param device descritor de dispositivo bluetooth encontrado
00035    * @param audio objeto gravador de audio
00036    */
00037   public BtStreamer(Handler hdl, BluetoothDevice device) {
00038     activity = hdl; /// copia handler para notificação de mudanças na activity
00039     btDevice = device; /// copia dispositivo encontrado que se deve conectar
00040     try {
00041       minBufferSize = AudioRecord.getMinBufferSize(AudioLinkConstantes.SAMPLING_RATE,
AudioFormat.CHANNEL_IN_MONO, AudioFormat.ENCODING_PCM_16BIT ); /// Obtendo tamanho mínimo do buffer suportado
pelo hardware.
00042       if (minBufferSize > 0) { /// se buffer mínimo retornado é um valor válido, faça
00043         buffer = new byte[minBufferSize]; /// criação do buffer local das amostras de áudio gravadas do microfone
00044         input = new AudioRecord(MediaRecorder.AudioSource.DEFAULT, AudioLinkConstantes.SAMPLING_RATE,
AudioFormat.CHANNEL_IN_MONO, AudioFormat.ENCODING_PCM_16BIT, minBufferSize); /// Construindo objeto de acesso ao
hardware para gravação de audio.
00045         if (input.getState() != AudioRecord.STATE_INITIALIZED) { /// Verifica se objeto está inicializado.
00046           input.release(); /// destruir objeto que não está inicializado!
00047           input = null; /// zerar objeto
00048         }
00049       }
00050     } catch (Exception e) {
00051       Log.d(LOGTAG, "ERRO! > "+e.getMessage()); /// mostra mensagem de exceção
00052     }
00053   }
```

```
00054
00055   /** Loop principal do cliente Bluetooth */
00056   public void run () {
00057     Log.e(LOGTAG, "run:"+run+" "+btDevice.getAddress());
00058     mensagemIpc ( AudioLinkConstantes.IPC.CHANGE_UI, AudioLinkConstantes.UI.LISTENER,
AudioLinkConstantes.UI.Action.DISABLE, null ); /// desabilitar icone do ouvinte
00059     if ( input != null ) /// se objeto gravador estiver instanciado
00060       input.startRecording(); /// inicia gravação do áudio
00061     while ( run && attempts < MAXIMO_RECONEXOES ) { /// se controle de loop for verdadeiro e tentativas de reconexão
não exceder MAXIMO_RECONEXOES, faça
00062       try {
00063         btSocket = btDevice.createRfcommSocketToServiceRecord(AudioLinkConstantes.AUDIO_SDP_UUID); /// obtém
socket RFCOMM do dispositivo indicado na UUID do serivço de áudio
00064         if (btSocket != null) { /// se socket criado com sucesso
00065           mensagemIpc ( AudioLinkConstantes.IPC.HEADER_MESSAGE, 0, 0, "Conectado-se a "+btDevice.getName()+",
tentativa "+(attempts+1)+"/"+MAXIMO_RECONEXOES ); /// instrução de processo de conexão
00066           mensagemIpc ( AudioLinkConstantes.IPC.CHANGE_UI, AudioLinkConstantes.UI.STREAMER,
AudioLinkConstantes.UI.Action.WARNING, null); /// icone de alerta
00067           attempts++; /// incrementa contador de tentativas de reconexão
00068           try {
00069             btSocket.connect(); /// inicia conexão RFCOMM com o socket criado (dispositivo + UUID serviço)
00070             try {
00071               if ((streamOut = btSocket.getOutputStream()) != null) { /// obtém OutputStream para transmissão de dados
00072                 attempts = 0; /// zera contador de tentativas de reconexão
00073                 mensagemIpc ( AudioLinkConstantes.IPC.HEADER_MESSAGE, 0, 0, "Conectado a "+btDevice.getName() ); ///
instruçoes de funcinamento ok
00074                 mensagemIpc ( AudioLinkConstantes.IPC.CHANGE_UI, AudioLinkConstantes.UI.STREAMER,
AudioLinkConstantes.UI.Action.OK, null); /// icone demonstrando funcionamento ok
00075                 while ( run ) {
00076                   if ( input != null ) {
00077                     input.read(buffer, 0, minBufferSize); /// lê buffer com dados da gravação do áudio
```

```
00078                 try {
00079                   streamOut.write(buffer); /// envia para a conexão RFCOMM estabelecida
00080                   streamOut.flush(); /// força a limpeza do buffer local de envio
00081                 } catch (IOException e) {
00082                   Log.d(LOGTAG, btDevice.getAddress()+" > ERRO! > write() > "+e.getMessage());
00083                   try {
00084                     btSocket.close(); /// fecha conexão
00085                     btSocket = null; /// zera objeto
00086                     mensagemIpc ( AudioLinkConstantes.IPC.HEADER_MESSAGE, 0, 0, "Reconectando a
"+btDevice.getName() ); /// instruções de reconexão
00087                     mensagemIpc ( AudioLinkConstantes.IPC.CHANGE_UI, AudioLinkConstantes.UI.STREAMER,
AudioLinkConstantes.UI.Action.WARNING, null); /// ícone indicando problemas
00088                     break; /// escapa do while interno
00089                   } catch (IOException e1) {
00090                     Log.d(LOGTAG, btDevice.getAddress()+" > close() > "+e1.getMessage());
00091                   }
00092                 }
00093               }
00094             }
00095           }
00096         } catch (IOException e) {
00097           Log.d(LOGTAG, btDevice.getAddress()+" > ERRO! > getOutputStream() > "+e.getMessage());
00098         }
00099       } catch (IOException e) {
00100         Log.d(LOGTAG, btDevice.getAddress()+" > connect() > "+e.getMessage());
00101       }
00102       try {
00103         synchronized (this) {
00104           wait(1000); /// espera mais 1 segundo entre tentativas
00105         }
00106       } catch (InterruptedException e1) {
```

```
00107           Log.d(LOGTAG, btDevice.getAddress()+" > ERRO! > wait() > "+e1.getMessage());
00108         }
00109         if (btSocket != null) { /// se socket existe
00110           try {
00111             btSocket.close(); /// fecha conexão
00112             btSocket = null; /// zera objeto
00113           } catch (IOException closeException) {
00114             Log.d(LOGTAG, btDevice.getAddress()+" > close() > "+closeException.getMessage());
00115           }
00116         }
00117       }
00118     } catch (IOException e) {
00119       Log.d(LOGTAG, btDevice.getAddress()+" > ERRO! > createRfcommSocketToServiceRecord() > "+e.getMessage());
00120     }
00121   }
00122   mensagemIpc ( AudioLinkConstantes.IPC.CHANGE_UI, AudioLinkConstantes.UI.LISTENER,
AudioLinkConstantes.UI.Action.ENABLE, null ); /// habilitar icone do ouvinte
00123   if (run == true) { /// se conexão foi perdida
00124     mensagemIpc ( AudioLinkConstantes.IPC.HEADER_MESSAGE, 0, 0, "Conexão perdida com "+btDevice.getName());
/// mostra instrução conexão perdida
00125     mensagemIpc ( AudioLinkConstantes.IPC.CHANGE_UI, AudioLinkConstantes.UI.STREAMER,
AudioLinkConstantes.UI.Action.ERROR, null); /// sinaliza com ícone de erro
00126   }
00127   else { /// se conexão foi finalizada pelo usuário
00128     mensagemIpc ( AudioLinkConstantes.IPC.HEADER_MESSAGE, 0, 0, "Selecione uma opção" ); /// mostra instrução
padrão
00129     mensagemIpc ( AudioLinkConstantes.IPC.CHANGE_UI, AudioLinkConstantes.UI.STREAMER,
AudioLinkConstantes.UI.Action.UNKNOW, null); /// sinaliza com ícone padrão
00130   }
00131   Log.e(LOGTAG, "run:"+run+" "+btDevice.getAddress());
00132   if (input != null) { /// se gravador existe
```

```
00133     input.stop(); /// interrompe gravação de áudio
00134     input.release(); /// libera recursos internos do gravador
00135     input = null; /// zera objeto
00136   }
00137   if (btSocket != null) { /// se socket existe
00138     try {
00139       btSocket.close(); /// fecha conexão
00140       btSocket = null; /// zera objeto
00141     } catch (IOException closeException) {
00142       Log.d(LOGTAG, btDevice.getAddress()+" > close() > "+closeException.getMessage());
00143     }
00144   }
00145   mensagemIpc ( AudioLinkConstantes.IPC.CLOSE_STREAMER, 0, 0, null); /// sinaliza para zerar objeto streamer
00146 }
00147
00148 /** Para thread cliente para tranmissão de áudio */
00149 public void cancel() {
00150   Log.e(LOGTAG, "cancel");
00151   run = false; /// interrompe laço de execução principal
00152 }
00153
00154 /** Envia para Activity uma mensagem via Handler
00155  * @param ipcCode Código IPC de acordo com a classe AudioLinkConstantes.
00156  * @param arg1 argumento 1 da mensagem
00157  * @param arg2 argumento 2 da mensagem
00158  * @param data Dados para serem enviados junto á mensagem.
00159  */
00160 private void mensagemIpc ( int ipcCode, int arg1, int arg2, Object data ) {
00161   if (activity != null) { /// se foi informado o handler da activity principal
00162     Message msg = activity.obtainMessage(); /// criar objeto de mensagem para a activity principal
00163     msg.what = ipcCode; /// informar código de diálogo
```

```
00164       msg.obj = data; /// adicionar dados á mensagem
00165       msg.arg1 = arg1; /// adicionar opção 1
00166       msg.arg2 = arg2; /// adicionar opção 2
00167       activity.sendMessage(msg); // envia mensagem para thread
00168     }
00169   }
00170 }
```

APÊNDICE D – CÓDIGO FONTE DO MÓDULO “OUVINTE”

BtListener.java

```
00001 /** Serviço Ouvinte
00002  * @author Everton Anjos, Mauricio Fiszt
00003  * @file BtListener.java
00004  */
00005 package com.utfpr.audiolink;
00006
00007 import android.bluetooth.BluetoothAdapter;
00008 import android.bluetooth.BluetoothServerSocket;
00009 import android.bluetooth.BluetoothSocket;
00010 import android.os.Handler;
00011 import android.os.Message;
00012 import android.util.Log;
00013 import java.io.IOException;
00014 import java.io.InputStream;
00015 import android.media.AudioTrack;
00016 import android.media.AudioTrack.OnPlaybackPositionUpdateListener;
00017 import android.media.AudioFormat;
00018 import android.media.AudioManager;
00019
00020 /** Thread de recebimento de stream de áudio via servidor Bluetooth. */
00021 public class BtListener extends Thread {
00022   private static final String LOGTAG = "BtListener"; ///< Tag para debug usada no Log.d
00023   private static final String AUDIO_SDP_NAME = "AudioLink"; ///< Nome do serviço SDP.
00024   private BluetoothServerSocket serverSocket = null; ///< Socket para a recepção de conexões.
00025   private BluetoothSocket socket = null; ///< Socket criado a partir da conexão do cliente.
00026   private InputStream streamIn = null; ///< Input stream do socket do cliente conectado.
00027   private byte[] buffer = new byte[100]; ///< Buffer de recebimento/leitura dos dados.
```

```
00028   private boolean run = true; ///< controle de funcionamento do loop de recepção de dados.
00029   private Handler activity = null; ///< Handler da Activity para notificação de eventos e atualização de dados.
00030   private AudioTrack output = null; ///< Objeto de sistema para regeneração de áudio.
00031
00032   /** Contrutor do servidor Bluetooth e ouvinte do áudio.
00033    * @param hdl Hanlder da activity que será utilizado para notificação de eventos
00034    */
00035   public BtListener(Handler hdl) {
00036     activity = hdl; /// copia handler para notificação de mudanças na activity
00037     try {
00038       int minBufferSize = 0;
00039       minBufferSize = AudioTrack.getMinBufferSize(AudioLinkConstantes.SAMPLING_RATE,
AudioFormat.CHANNEL_CONFIGURATION_MONO, AudioFormat.ENCODING_PCM_16BIT); /// Obtendo tamanho mínimo do
buffer suportado pelo hardware.
00040       if (minBufferSize > 0) { /// se tamanho mínimo do buffer for válido
00041         output = new AudioTrack(AudioManager.STREAM_MUSIC, AudioLinkConstantes.SAMPLING_RATE,
AudioFormat.CHANNEL_CONFIGURATION_MONO, AudioFormat.ENCODING_PCM_16BIT, minBufferSize,
AudioTrack.MODE_STREAM); /// Construindo objeto de acesso ao hardware reprodução de audio.
00042         if (output.getState() != AudioTrack.STATE_INITIALIZED) { /// Verifica se objeto está inicializado.
00043           Log.d ( LOGTAG, "ERRO! > output.getState():"+output.getState()+" != STATE_INITIALIZED" );
00044           output.release(); /// destruir objeto que não está inicializado!
00045           output = null; /// zerar objeto
00046         }
00047         /** Criando listener para a notificação de fim de buffer interno atingido. */
00048         output.setPlaybackPositionUpdateListener( new OnPlaybackPositionUpdateListener () {
00049             /** Metodo da recepção de notificação de final do buffer interno.
00050              * @param track objeto que gerou o sinal.
00051              */
00052             public void onMarkerReached(AudioTrack track) {
00053               if (track.getPlayState() == AudioTrack.PLAYSTATE_PLAYING) /// verifica se o áudio está sendo reproduzido
00054                 track.pause(); /// parar/interromper reprodução de áudio pois buffer interno de áudio chegou no fim
```

```
00055           }
00056           public void onPeriodicNotification(AudioTrack arg0) { /* implementação sem uso */ }
00057         } );
00058       }
00059       try {
00060         serverSocket = BluetoothAdapter.getDefaultAdapter().listenUsingRfcommWithServiceRecord(AUDIO_SDP_NAME,
AudioLinkConstantes.AUDIO_SDP_UUID); /// criar servidor RFCOMM para o serviço de áudio indicado pela UUID
00061       } catch (IOException e) {
00062         Log.e(LOGTAG, " > ERRO! > listenUsingRfcommWithServiceRecord:"+e.getMessage());
00063       }
00064     } catch (Exception e) {
00065       Log.d(LOGTAG, "ERRO! > "+e.getMessage()); /// mostra mensagem de exceção
00066     }
00067   }
00068
00069   /** Loop principal do servidor Bluetooth */
00070   public void run () {
00071     Log.e(LOGTAG, "run");
00072     if (run == true)
00073       mensagemIpc ( AudioLinkConstantes.IPC.CHANGE_UI, AudioLinkConstantes.UI.STREAMER,
AudioLinkConstantes.UI.Action.DISABLE, null ); /// desabilitar icone do locutor
00074     while (run) { /// enquanto controle de funcionamento do ouvinte for verdadeiro
00075       if ( serverSocket != null ) { /// se criação do servidor ocorreu sem problemas
00076         mensagemIpc ( AudioLinkConstantes.IPC.HEADER_MESSAGE, 0, 0, "Aguardando conexão do locutor" ); ///
instruções para aguardar locutor
00077         mensagemIpc ( AudioLinkConstantes.IPC.CHANGE_UI, AudioLinkConstantes.UI.LISTENER,
AudioLinkConstantes.UI.Action.WARNING, null ); /// mostrar ícone de atenção
00078         try {
00079           socket = serverSocket.accept(); /// aguardar conexão do cliente, é bloqueante até que o cliente bluetooth RFCOMM
se conecte ou o processo seja interrompido na thread via cancel
00080           if (socket != null) { /// se conexão foi criada com sucesso
```

```
00081           try {
00082             if ((streamIn = socket.getInputStream()) != null) { /// obtém InputStream para recepção de dados
00083               mensagemIpc ( AudioLinkConstantes.IPC.HEADER_MESSAGE, 0, 0, "Conectado a
"+socket.getRemoteDevice().getName() ); /// instruções de conexão estabelecida
00084               mensagemIpc ( AudioLinkConstantes.IPC.CHANGE_UI, AudioLinkConstantes.UI.LISTENER,
AudioLinkConstantes.UI.Action.OK, null ); /// ícone de notificação para funcionamento ok
00085               while (run) { /// enquanto controle de funcionamento do ouvinte for verdadeiro
00086                 int val=0; /// volume de dados lidos do socket bluetooth
00087                 try {
00088                   if ((val=streamIn.read(buffer)) > 0) { /// lê da conexão RFCOMM amostras de áudio
00089                     if (output != null) { /// se objeto de reprodução foi instanciado
00090                       output.setNotificationMarkerPosition (output.getNotificationMarkerPosition ()+ ( val/2 )); /// atualiza
notificação de fim de buffer
00091                       output.write ( buffer, 0, val ); /// escreve amostras no buffer interno do AudioTrack
00092                       if (output.getPlayState() == AudioTrack.PLAYSTATE_STOPPED || output.getPlayState() ==
AudioTrack.PLAYSTATE_PAUSED) /// verifica se a reprodução de áudio está parada ou interrompida.
00093                         output.play(); /// inicia/reinicia reprodução do áudio.
00094                     }
00095                   }
00096                 } catch (IOException e) { /// erro , conexão perdida/cancelada
00097                   Log.e(LOGTAG, "ERRO! > read:"+e.getMessage());
00098                   break;
00099                 }
00100               }
00101               mensagemIpc ( AudioLinkConstantes.IPC.HEADER_MESSAGE, 0, 0, "Reconectando a
"+socket.getRemoteDevice().getName() ); /// instruções indicam que conexão está com problemas
00102               mensagemIpc ( AudioLinkConstantes.IPC.CHANGE_UI, AudioLinkConstantes.UI.LISTENER,
AudioLinkConstantes.UI.Action.WARNING, null ); /// indica conexão com problemas através de ícone
00103             }
00104           } catch (IOException e) {
00105             Log.e(LOGTAG, "ERRO! > getInputStream:"+e.getMessage());
```

```
00106           }
00107         }
00108         else {
00109           Log.e(LOGTAG, "ERRO! > socket:"+socket+" ?");
00110         }
00111       } catch (IOException e) {
00112         Log.e(LOGTAG, "ERRO! > accept:"+e.getMessage());
00113       }
00114       try {
00115         if ( socket != null ) { /// se socket existe
00116           socket.close(); /// fechar socket conexão
00117           socket = null; /// zerar objeto socket
00118         }
00119       } catch (IOException e) {
00120         Log.e(LOGTAG, "ERRO! > close:"+e.getMessage());
00121       }
00122     }
00123   }
00124   mensagemIpc ( AudioLinkConstantes.IPC.CHANGE_UI, AudioLinkConstantes.UI.STREAMER,
AudioLinkConstantes.UI.Action.ENABLE, null ); /// habilitar icone do locutor
00125   mensagemIpc ( AudioLinkConstantes.IPC.CHANGE_UI, AudioLinkConstantes.UI.LISTENER,
AudioLinkConstantes.UI.Action.UNKNOW, null ); /// botão padrão
00126   mensagemIpc ( AudioLinkConstantes.IPC.HEADER_MESSAGE, 0, 0, "Selecione uma opção"); /// instruçoes padrão
00127   if ( output != null ) { /// se reprodutor de áudio existe
00128     output.stop(); /// parar reprodução de áudio
00129     output.release(); /// liberar recursos do reprodutor de áudio
00130     output = null; /// zerar objeto
00131   }
00132   try {
00133     if ( socket != null ) { /// se socket existe
00134       socket.close(); /// fechar socket conexão
```

```
00135         socket = null; /// zerar objeto socket
00136       }
00137     } catch (IOException e) {
00138       Log.e(LOGTAG, "ERRO! > close:"+e.getMessage());
00139     }
00140     mensagemIpc ( AudioLinkConstantes.IPC.CLOSE_LISTENER, 0, 0, null); /// sinaliza para zerar objeto listener
00141   }
00142
00143   /** cancela, para processo de recepção do stream de áudio */
00144   public void cancel() {
00145     Log.e(LOGTAG, "cancel");
00146     run = false; /// parar processo principal
00147     try {
00148       if (serverSocket != null ) {
00149         serverSocket.close();
00150       }
00151     } catch (IOException e) {
00152       Log.e(LOGTAG, "ERRO! > close:"+e.getMessage());
00153     }
00154   }
00155
00156   /** Envia para Activity uma mensagem via Handler
00157    * @param ipcCode Código IPC de acordo com a classe AudioLinkConstantes.
00158    * @param arg1 argumento 1 da mensagem
00159    * @param arg2 argumento 2 da mensagem
00160    * @param data Dados para serem enviados junto á mensagem.
00161    */
00162   private void mensagemIpc ( int ipcCode, int arg1, int arg2, Object data ) {
00163     if (activity != null) { /// se foi informado o handler da activity principal
00164       Message msg = activity.obtainMessage(); /// criar objeto de mensagem para a activity principal
00165       msg.what = ipcCode; /// informar código de diálogo
```

```
00166       msg.obj = data; /// adicionar dados á mensagem
00167       msg.arg1 = arg1; /// adicionar opção 1
00168       msg.arg2 = arg2; /// adicionar opção 2
00169       activity.sendMessage(msg); // envia mensagem para thread
00170     }
00171   }
00172 }
```

APÊNDICE E – CÓDIGO FONTE DO MÓDULO “CONSTANTES DO APLICATIVO”

AudioLinkConstantes.java

```
00001 /** Constantes do aplicativo
00002  * @author Everton Anjos, Mauricio Fiszt
00003  * @file AudioLinkConstantes.java
00004  */
00005 package com.utfpr.audiolink;
00006
00007 import java.util.UUID;
00008
00009 /** Definições padrão do projeto */
00010 public final class AudioLinkConstantes {
00011   public static final int SAMPLING_RATE = 44100; ///< Sampling rate de 44.1 kHz
00012   public static final UUID AUDIO_SDP_UUID = UUID.fromString("b4727ed0-067c-11e2-892e-0800200c9a66"); ///< Define a
UUID do serviço de áudio.
00013   /** Códigos de mensagem entre processos */
00014   public class IPC {
00015     public static final int HEADER_MESSAGE = 1; ///< alterar mensagem na tela principal
00016     public static final int CHANGE_UI = 2; ///< alterar propriedades dos botões
00017     public static final int CLOSE_STREAMER = 3; ///< remover referência ao objeto BtStreamer
00018     public static final int CLOSE_LISTENER = 4; ///< remover referência ao objeto BtListener
00019   }
00020   /** Códigos para mudanças na inrterface ao usuário */
00021   public class UI {
00022     public static final int STREAMER = 1; ///< objeto na interface do locutor
00023     public static final int LISTENER = 2; ///< objeto na interface do ouvinte
00024     /** Ações possíveis de serem acionadas nos objetos ouvinte e locutor na interface */
00025     public class Action {
00026       public static final int UNKNOW = 1; ///< utilizar opçõe padrão de imagem
```

```
00027       public static final int DISABLE = 2; ///< desabilitar objeto
00028       public static final int ENABLE = 3; ///< habilitar objeto
00029       public static final int OK = 4; ///< sinalizar através de imagem funcionamento ok
00030       public static final int WARNING = 5; ///< sinalizar através de imagem funcionamento com problemas
00031       public static final int ERROR = 6;  ///< sinalizar através de imagem mau funcionamento ou erro
00032     }
00033   }
00034 }
```

APÊNDICE F – CÓDIGO FONTE DE PERMISSÕES “*ANDROIDMANIFEST*”

AndroidManifest.xml

```
00001 <manifest
00002   xmlns:android="http://schemas.android.com/apk/res/android"
00003   package="com.utfpr.audiolink"
00004   android:versionCode="1"
00005   android:versionName="1.0" >
00006   <uses-sdk
00007     android:minSdkVersion="7"
00008     android:targetSdkVersion="7" />
00009   <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
00010   <uses-permission android:name="android.permission.BLUETOOTH" />
00011   <uses-permission android:name="android.permission.RECORD_AUDIO" />
00012   <application
00013     android:icon="@drawable/ic_launcher"
00014     android:label="@string/app_name"
00015     android:theme="@style/AppTheme" >
00016     <activity
00017       android:name=".Principal"
00018       android:label="@string/title_activity_principal"
00019       android:screenOrientation="portrait"
00020       android:configChanges="keyboardHidden" >
00021       <intent-filter>
00022         <action android:name="android.intent.action.MAIN" />
00023         <category android:name="android.intent.category.LAUNCHER" />
```

```
00024       </intent-filter>
00025     </activity>
00026     <activity
00027       android:name=".SelOuvinte"
00028       android:label="@string/msg_sel_device"
00029       android:theme="@android:style/Theme.Dialog"
00030       android:screenOrientation="portrait"
00031       android:configChanges="keyboardHidden" />
00032   </application>
00033 </manifest>
```