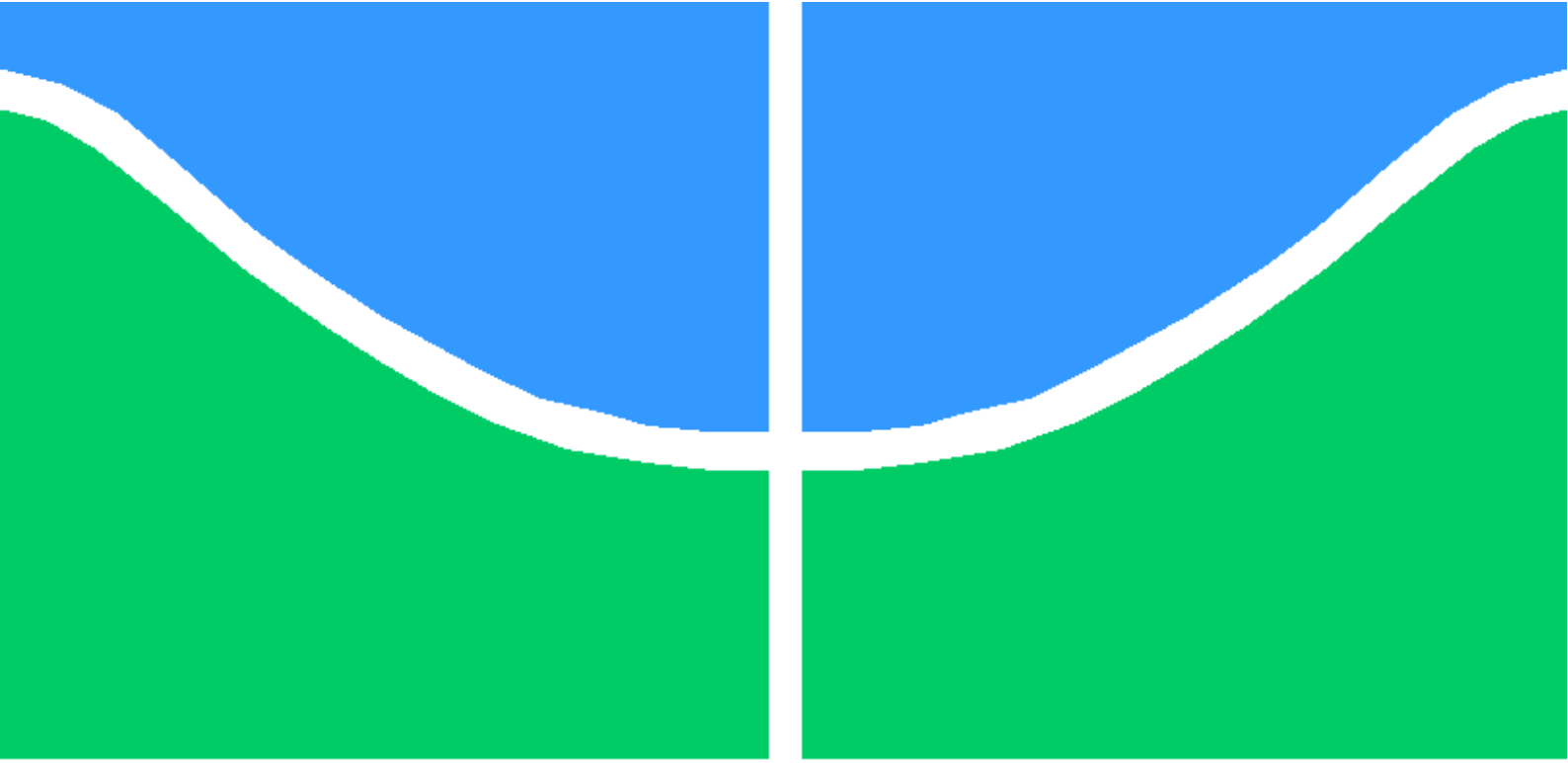

TRABALHO DE GRADUAÇÃO

Desenvolvimento de Aplicativos para Dispositivos Móveis com Reconhecimento de Voz em Português

Rafael José Alves

Vítor Schwingel Goulart

Brasília, fevereiro de 2014

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia

TRABALHO DE GRADUAÇÃO

Desenvolvimento de Aplicativos para Dispositivos Móveis com Reconhecimento de Voz em Português

Rafael José Alves

Vítor Schwingel Goulart

Relatório submetido ao Departamento de Engenharia Elétrica como requisito parcial para obtenção do grau de Engenheiro de Redes de Comunicação

Banca Examinadora

Prof. Ugo Silva Dias, ENE/UnB _____
Orientador

Prof. Flávio Elias Gomes de Deus, ENE/UnB _____
Examinador interno

Prof. Ricardo Zelenovsky, ENE/UnB _____
Examinador interno

Dedicatórias

Ao meu grande amor e grande companheira, Annita Petrocchi

Vítor Schwingel Goulart

Dedico esse trabalho a Deus e a todos aqueles que sempre acreditaram em mim.

Rafael José Alves

Agradecimentos

Agradeço primeiramente a Deus, pois sem ele não teria chegado até onde cheguei e por ter me dado força durante esses 5 anos de graduação. Agradeço aos meus pais que me apoiaram durante o curso e me incentivaram a sempre continuar nos estudos. Agradeço também a todos os professores e colegas com quem tive contato durante a graduação e por cada lição (acadêmica, ética e moral) que pude aprender com cada um. Agradeço ainda ao Professor Ugo Silva Dias por ter nos acompanhado durante todos os momentos de nosso projeto de graduação, por todas as orientações de cunho acadêmico e profissional e por toda a sua dedicação e suporte a nossa formação.

Rafael José Alves

Agradeço a todos os meu colegas e professores de graduação. Todos contribuiram de alguma forma para minha formação. Em especial, a minha companheira Annita, por todo o apoio, compreensão, carinho e paciência ao longo desta jornada. Agradeço também aos meus amigos, que sempre me apoiaram, me escutaram e ajudaram com o que puderam, mas principalmente serviram de porto seguro nos momentos difíceis. Também a minha família por todo o suporte, e o Professor Ugo Silva Dias, que dedicou muitas horas de seu tempo nesta orientação e demonstrou imensa consideração e empenho durante a evolução do trabalho.

Vítor Schwingel Goulart

RESUMO

O presente trabalho tem por objetivo apresentar os resultados do desenvolvimento de aplicações para o sistema operacional *Android*, as quais visam a utilização de comandos de voz para a ativação de circuitos ou dispositivos externos conectados a um dispositivo móvel por meio de rede *Bluetooth*. Em um primeiro momento, é descrito o funcionamento do *Android* e de sua biblioteca de reconhecimento de voz assim como aplicações que foram criadas com o intuito de demostrar o funcionamento do reconhecimento de voz e como comandos de voz podem facilitar a interação do usuário com uma aplicação. Em um segundo momento, é apresentada a plataforma código aberto de prototipagem eletrônica *Arduino* e como é possível conectá-la ao Android através de Bluetooth utilizando a biblioteca Amarino, permitido a transmissão de dados entre aplicações nos dois dispositivos. Em seguida, são apresentadas duas aplicações envolvendo todos os conceitos abordados anteriormente mostrando como uma conexão entre *Android* e *Arduino* permite que o usuário possa controlar circuitos externos por comandos de voz. Por último, é apresentado de forma resumida como o princípio apresentado de ativação de circuitos por comando de voz pode ser utilizado em outras situações, e são brevemente explicados conceitos da computação ubíqua.

ABSTRACT

This work presents the results of the applications development for the Android mobile operational System. Those applications use voice commands to activate electronic circuits or other external devices connected to the Android device by Bluetooth connection. The Android architecture and *modus-operandi* are also briefly resumed, as well as its voice recognition library. This work also shows how the voice commands make it simpler to interact with such applications. The Arduino open-source system is shown as a good way to develop prototypes and its functionalities are also presented. The Amarino system, a tool for connecting Arduino and Android systems is presented here, it enables such connection via Bluetooth networking. Two applications that have been created are shown and details about their programming and characteristics are also presented. Finally, the voice controlling principles are extended for other functionalities and ubiquitous computing are briefly explained.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE SIGLAS

Siglas

TCP	*Transmission Control Protocol*
IP	*Internet Protocol*
UDP	*User Datagram Protocol*
GPS	*Global Positioning System*
OHA	*Open Handset Alliance*
OS	*Operational System*
MIT	*Massachusetts Institute of Technology*
MCU	*Microcontroller Unit*
DVM	*Dalvik Virtual Machine*
SDK	*Software Development Kit*
ADT	*Android Development Tools*
IDE	*Integrated Development Environment*
AVD	*Android Virtual Device*
USB	*Universal Serial Bus*
APK	*Android package*
ID	*Identification* – Identificação
VM	*Virtual Machine* – Máquina Virtual
SD	*Secure Digital*
SMS	*Short Message Service*
API	*Application Programming Interface*
TTL	*Transistor-Transistor Logic*
LED	*Light Emitting Diode*
PAN	*Personal Area Network*
MAC	*Media Access Control*
iOS	*iPhone Operating System*
app	Aplicativo – Abreviação.

Capítulo 1

Introdução

1.1 Contexto e Motivação

Abrir a porta de casa pelo celular, ligar o ar condicionado do carro antes de sair de casa e controlar um semáforo em um cruzamento movimentado de uma grande cidade. Essas ações, que não parecem ter nada em comum, assemelham-se no controle computacional, muitas vezes via rede. Estes objetos controlados por computador estão cada vez mais presentes ao nosso redor. A humanidade caminha em direção a um futuro totalmente informatizado.

Em seu artigo de 1991, Weiser cria o conceito de computação ubíqua (*Ubiquous Computing* ou *ubicomp*). Para ele, a informática deveria evoluir até o ponto em que se tornasse invisível para o usuário. Um termo melhor seria "imperceptível", afinal, a ideia é que a pessoa nem sequer note que está interagindo com uma máquina. Um meio de fazer isso seria utilizar as chamadas "interfaces naturais" do ser humano. O movimento dos olhos, o toque e a fala são muito mais intuitivos do que digitar em um teclado, por exemplo.[7]

É claro que a computação ubíqua já é uma realidade nos dias atuais, ao contrário do que era em 1991, data do artigo de Weiser. Isto deve-se, em grande parte, à presença cada vez maior e mais abrangente dos aparelhos celulares no cotidiano do homem moderno.

"O mercado de celulares está crescendo cada vez mais. Estudos mostram que hoje em dia mais de 3 bilhões de pessoas possuem um aparelho celular, e isso corresponde a mais ou menos metade da população mundial.

Hoje em dia os usuários comuns estão procurando cada vez mais celulares com diversos recursos como câmeras, músicas, bluetooth, ótima interface visual, jogos, GPS, acesso a internet e e-mails, e agora ainda temos a TV Digital.

O mercado corporativo também está crescendo muito, e diversas empresas estão buscando incorporar aplicações móveis a seu dia-a-dia para agilizar seus negócios e integrar as aplicações móveis com seus sistemas de *back-end*. Empresas obviamente visam o lucro e mais lucro, e os celulares e *smartphones* podem ocupar um importante espaço em um mundo onde a palavra 'mobilidade' está cada vez mais conhecida." [1]

Com a popularização dos telefones celulares e, mais recentemente, dos *smartphones*, desenvolver soluções para estes dispositivos se tornou uma divisão específica do desenvolvimento de programas nos últimos anos. Cada vez mais vemos desenvolvedores especializados e grandes empresas entrando no mercado de criação de aplicativos para aparelhos móveis.

1.2 Sistema Operacional *Android*

Atualmente, o sistema operacional mais popular em *smartphones* no mundo é o *Google Android*. De acordo com a revista Forbes, em 2013, este sistema operacional bateu recorde de participação no mercado, atingindo 81% do *market share*, conforme mostra a Figura 1.1. A matéria também da conta de que grande parte destes 81% dos usuários faz uso do sistema através de um aparelho barato, com preço de até 215 dólares. Isto significa também que, ao se desenvolver uma aplicação cujo objetivo é atingir um grande público e todas as classes sociais, deve se pensar de forma relevante em operações com o *Android*.[2] O *Android* causou um grande impacto quando foi anunciado, atraindo a atenção de muita gente. Desenvolvido pelo Google, o sistema é ainda apoiado por várias empresas lideres do mercado de telefonia como a Motorola, LG, Samsung, Sony e muitas outras. Esse grupo, chamado de de *Open Handset Alliance (OHA)* foi criado com a intenção de padronizar uma plataforma de código aberto e livre para celulares, justamente para atender a todas as expectativas e tendências do mercado atual.[1]

Marketshare de sistemas operacionais móveis, 2013 (unidades em milhões)

Sistema Operacional	Unidades Exportadas em 2013	Marketshare em 2013	Unidades Exportadas em 2012	Marketshare em 2012	Mudança Ano-a-Ano
Android	187.4	79.3%	108	69.1%	73.5%
iOS	31.2	13.2%	26	16.6%	20.0%
Windows Phone	8.7	3.7%	4.9	3.1%	77.6%
BlackBerry OS	6.8	2.9%	7.7	4.9%	-11.7%
Linux	1.8	0.8%	2.8	1.8%	-35.7%
Symbian	0.5	0.2%	6.5	4.2%	-92.3%
Outros	N/A	0.0%	0.3	0.2%	-100.0%
Total	**236.4**	**100.0%**	**156.2**	**100.0%**	**51.3%**

Fonte: IDC Worldwide Mobile Phone Tracker, August 7, 2013

Figura 1.1: *Market Share* de Sistemas Operacionais Móveis em 2013

Tendo por base o conceito de computação ubíqua pode-se utilizar *smartphones* para controlar vários dispositivos presentes em nosso dia-a-dia. O *Android*, por ser uma plataforma de código aberto e livre, possui muita versatilidade para trabalhar com soluções de ativação e controle de outros dispositivos. Entretanto, em muitos casos, faz-se necessária a utilização de uma interface entre o *Android* e o dispositivo que se quer controlar. Uma forma de obter tal solução é por meio do *Arduino*.

1.3 A Placa Controladora *Arduino*

O *Arduino* faz parte do conceito de *hardware* e *software* livre e está aberto para uso e contribuição de toda sociedade. O conceito *Arduino* surgiu na Itália, em 2005, com o objetivo de criar um dispositivo que fosse utilizado em projetos/protótipos construídos de uma forma menos custosa do que outros sistemas disponíveis no mercado. Ele pode ser usado para desenvolver artefatos interativos *stand-alone* ou conectados ao computador, utilizando diversos aplicativos, tais como: Adobe Flash, Processing, Max/MSP, Pure Data ou SuperCollider.

O *Arduino* foi projetado com a finalidade de ser de fácil entendimento, de fácil programação e de fácil aplicação, além de ser multiplataforma, podendo ser configurado em ambientes Linux, Mac OS e Windows. Além disso, um grande diferencial deste dispositivo é ser mantido por uma comunidade que trabalha na filosofia *open-source*, desenvolvendo e divulgando gratuitamente seus projetos.

O equipamento é uma plataforma de computação física: são sistemas digitais ligados a sensores e atuadores, que permitem construir sistemas que percebem a realidade e respondem com ações físicas. Ele é baseado em uma placa microcontrolada, com acessos de Entrada/Saída (I/O), sobre a qual foi desenvolvida uma biblioteca de funções que simplifica a sua programação, por meio de uma sintaxe similar à das linguagens C e C++.

Em resumo, o *Arduino* é um kit de desenvolvimento, que pode ser visto como uma unidade de processamento capaz de medir variáveis do ambiente externo, transformadas em um sinal elétrico correspondente, através de sensores ligados aos seus terminais de entrada. De posse da informação, ele pode processá-la computacionalmente. A Figura 1.2 apresenta um diagrama de blocos de uma cadeia de processamento utilizando o *Arduino*.

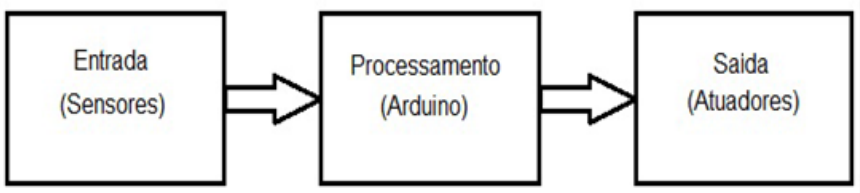

Figura 1.2: Diagrama de Blocos da Cadeia de Processamento *Arduino*

Uma vez que o *Arduino* é baseado em um microcontrolador e, portanto, é programável, torna-se possível criar diversas aplicações diferentes com certa facilidade. Além disso, o próprio equipamento pode ser reutilizado, através de uma nova programação. Por sua vez, a sua programação é simplificada pela existência de diversas funções que controlam o dispositivo, com uma sintaxe similar à de linguagens de programação comumente utilizadas (C e C++).

Assim sendo, em um ambiente profissional, as características do *Arduino* fazem dele uma boa ferramenta de prototipação rápida e de projeto simplificado. Por outro lado, em um ambiente acadêmico, ele pode ser perfeitamente utilizado como ferramenta educacional, uma vez que não requer do usuário conhecimentos profundos de eletrônica digital nem da programação de dispositivos digitais específicos. O *Arduino* e suas características importantes e utilizadas neste trabalho

serão abordados ainda no Capítulo 3.

Fazer com que o *Arduino* receba comandos de dispositivos *Android* é exatamente a principal função do Amarino, sistema criado pelo pesquisador austríaco Bonifaz Kaufmann, do MIT, Instituto de Tecnologia de Massachusetts – do inglês, *Massachusetts Institute of Technology* – Estados Unidos. Esta criação foi a principal inspiração para este trabalho e está descrita em detalhes no final do Capítulo 3.

1.4 Apresentação do manuscrito

Este trabalho busca propor uma solução com a utilização de *Arduino* e *Android* combinados. A ideia é simplificar tarefas comuns usando conceitos da computação ubíqua, eletrônica e programação, além das propriedades de aplicações com controle por voz, que estão cada vez mais presentes nos aparelhos modernos, por exemplo: Siri, Google Now, etc.

Por meio de uma placa com interface *Bluetooth*, é possível conectar um aparelho *Android* em uma placa *Arduino* e enviar comandos. Desta forma, pode-se programar o *Arduino* para gerar qualquer tipo de sinal elétrico que, por sua vez, é capaz de acionar diversos dispositivos.

Nota-se que se trata de uma ideia bastante abrangente, porém, conforme este trabalho apresenta, os mesmos princípios básicos podem ser utilizados em diversas situações específicas. Isto proporciona uma grande aplicabilidade a esta pesquisa, ainda mais em um mercado tão efervescente como o de aparelhos celulares.

Este trabalho será apresentado nos seguintes capítulos:

- Capítulo 2: Princípios Básicos do Sistema Operacional *Android*:

Apresenta os princípios básicos de funcionamento do sistema operacional *Android*, assim como a plataforma utilizada para a programação e suas linguagens. Também mostra aplicativos que foram desenvolvidos com a finalidade de entender na prática o que foi estudado sobre este sistema operacional. Tais aplicativos foram construídos com a intenção de estudar o comportamento das bibliotecas de reconhecimento de voz presentes no *Android*. Estas bibliotecas também serão mostradas em detalhes neste capítulo.

- Capítulo 3: Plataforma *Arduino* e sua conexão com o *Android* via *Bluetooth*.

A placa *Arduino* utilizada é mostrada mais detalhadamente neste capítulo. Também apresenta o modo pelo qual é realizada a conexão com um aparelho *Android* através do *Bluetooth*. A biblioteca Amarino, utilizada durante a conexão *Bluetooth* é descrita neste capítulo.

- Capítulo 4: Princípios e Soluções Propostas utilizando conceitos básicos do Amarino.

Utilizando os conceitos mostrados nos capítulos 2 e 3, são apresentados neste capítulo as soluções desenvolvidas para a ativação de dispositivos externos como lâmpadas, motores, etc. São evidenciadas também outras possíveis aplicações dos princípios demonstrados. Dessa forma, deixa-se claro a abrangência da pesquisa e de suas ramificações.

- Capítulo 5: Conclusão

Aqui é demonstrada a validade das soluções propostas, com todos os conceitos que as permeiam. Acessibilidade, inteligência e agilidade são discutidas como características de aplicativos realmente úteis. O princípio proposto por este trabalho cumpre seu objetivo de criar uma solução de automação acessível, de baixo custo e altamente integrada com o mundo atual.

Capítulo 2

Princípios Básicos do Sistema Operacional Android

2.1 Introdução

O primeiro passo para a programação em dispositivos móveis, após a escolha da plataforma, é aprender como ela funciona. Compreender os principais conceitos e como os elementos do sistema podem ser utilizados e relacionados entre sí é muito importante.

Neste capítulo serão discutidas características do sistema operacional Google *Android*, bem como os princípios utilizados neste trabalho de pesquisa. Comandos de um celular para aparelhos externos são, muitas vezes, descritos na própria documentação do sistema e também costumam ter interfaces básicas que podem ser facilmente inseridas em um código de um aplicativo.

2.2 Arquitetura do Android

Na bibliografia produzida pelo Google sobre o *Android*, o sistema é definido como uma "pilha de softwares". O *kernel* 2.6 do Linux é utilizado como base para todo o restante da pilha. Nele se encontram os diversos drivers utilizado pelo dispositivo, bem como o sistema de gerenciamento de energia.

No nível acima do *kernel*, a camada de biblioteca consiste basicamente em instruções que permitem que o dispositivo lide com diferentes tipos de dados. A camada de tempo de execução inclui um conjunto de bibliotecas do núcleo Java (*Core Libraries*). Nesta camada está a Máquina Virtual Dalvik (DVM). Esta é muito importante para o funcionamento do *Android*. Ao permitir que múltiplas instâncias rodem ao mesmo tempo, a DVM cria o isolamento de processos no Android, evitando o travamento do sistema quando há um problema em algum processo. Além disto, o gerenciamento de memória também é simplificado, devido a pouca memória requerida por cada instância. No nível acima, fica a camada de *framework* de aplicação, que contém as aplicações básicas do aparelho, como o gerenciamento de localização ou o gerenciador de atividades, por

exemplo. Finalmente, no nível mais alto da pilha, fica a camada de aplicação, que possui toda a interface com o usuário. Um resumo da arquitetura do *Android* segue na Figura 2.1.

Figura 2.1: Arquitetura Básica do *Android*

2.3 O Desenvolvimento de Aplicações

O *Android Software Development Kit* (SDK) é uma ferramenta completa para o desenvolvimento de *apps* para o *Android*. Está disponível gratuitamente no site da Google. Estão incluídas uma versão do Eclipse IDE com o *Android Development Tools* (ADT) e um emulador para testar os aplicativos através de um dispositivo virtual (*Android Virtual Device* - AVD) além de permitir o teste em um aparelho conectado via USB.

Criar aplicativos é relativamente simples e está se tornando cada vez mais popular. O mercado de movimenta bilhões de dólares por ano, e está crescendo significativamente. No entanto, para obter os resultados esperados neste trabalho, foi necessário pesquisar as características de aplicações consideradas excelentes pelo mercado. Levando em conta a acessibilidade como prioridade, bem como os conceitos de computação ubíqua.

Para criar uma aplicação é necessário entender como ela é executada pelo sistema e os principais componentes que podem constituí-la. Deve-se compreender como a aplicação se comporta dentro do sistema, os componentes que definem uma aplicação, o arquivo de manifesto no qual são declarados os componentes e os recursos do dispositivo que serão necessários para a aplicação, recursos que

não estão no código da aplicação, mas que permitem otimizar o comportamento da aplicação para a variedade de configuração de dispositivos existente.

2.3.1 A Aplicação Dentro do Sistema Android

Aplicações *Android* são escritas na linguagem de programação Java. A ferramenta Android SDK compila então o código, além de todos os dados e recursos em um *Android package* (APK), um arquivo com extensão .apk. Um arquivo APK contém todo o conteúdo da aplicação e é utilizado pelo sistema Android para instalá-la.

Vale lembrar que, apesar das aplicações *Android* serem escritas na linguagem Java o sistema não possui uma máquina virtual Java (JVM). O que existe é uma máquina virtual chamada Dalvik que é otimizada para execução em dispositivos móveis. Ao desenvolver as aplicações pode-se utilizar a linguagem Java e todos os seus recursos normalmente, mas depois que o bytecode (*.class*) é compilado ele é convertido para o formato .dex (*Dalvik Executable*), que representa a aplicação do *Android* compilada. [16]

Uma vez instalada, cada aplicação *Android* vive em sua própria zona de segurança:

- O sistema operacional *Android* é um sistema Linux multiusuário no qual cada aplicativo é um usuário diferente. [16]

- Por padrão, o sistema atribui a cada aplicativo um ID de usuário Linux único (utilizado apenas pelo sistema e é desconhecido pelo aplicativo). O sistema atribui permissões para todos os arquivos em um aplicativo tal que apenas o ID de usuário atribuído para aquela aplicação possa acessá-los. [16]

- Cada processo possui sua máquina virtual (virtual machine, VM) própria, tal que o código de um aplicativo é executado isoladamente em relação aos demais aplicativos. [16]

- Por padrão, cada aplicativo é executado no seu próprio processo Linux. *Android* inicia o processo quando qualquer um dos componentes do aplicativo necessita ser executado, então finaliza o processo quando ele não é mais necessário ou quando o sistema necessita recuperar memória para outros aplicativos. [16]

Dessa forma o sistema *Android* implementa o princípio do privilégio mínimo, ou seja, cada aplicativo, por padrão, possui acesso apenas aos componentes que são necessários para seu funcionamento e nada mais. Isso cria um ambiente muito seguro no qual um aplicativo não pode acessar outras partes do sistema se não possui permissão para tal. [16]

Existem, porém, maneiras para um aplicativo compartilhar dados com outros aplicativos e para um aplicativo acessar serviços do sistema:

- É possível fazer com que dois aplicativos compartilhem o mesmo ID de usuário Linux, em cujo caso eles estão aptos a acessar os arquivos de cada um. Para conservar recursos do sistema, aplicativos que possuem o mesmo ID de usuário podem ser executados no mesmo processo Linux e compartilhar a mesma VM (os aplicativos devem também ser assinados com o mesmo certificado). [16]

- Um aplicativo pode requisitar permissão para acessar dados do dispositivo tais como os contatos do usuário, mensagens SMS, o cartão de memória SD, a câmera, *Bluetooth*, entre outros. Todas as permissões devem ser concedidas pelo usuário no instante de instalação do aplicativo. [16]

Em resumo, é dessa forma que uma aplicação *Android* se comporta dentro do sistema.

2.3.2 Componentes de Aplicativos

Os Componentes de aplicativos são blocos essenciais que constituem uma aplicação *Android*. Cada componente é uma entidade única dentro da aplicação e que desempenha um papel específico que ajuda a definir o comportamento geral da aplicação. Existem quatro tipos diferentes de componentes. Cada um possui um propósito distinto e um ciclo de vida que define como o componente é criado e destruído durante a execução da aplicação. Os quatro componentes são: Activities, Services, Content Providers, e Broadcast receivers. A seguir tem-se uma explicação mais detalhada de cada um desses componentes. [16]

- *Activities*:

Uma *Activity* é um componente da aplicação que fornece uma tela na qual o usuário pode interagir a fim de fazer alguma coisa como discar um número, enviar um e-mail, ou visualizar um mapa. Cada uma possuiu uma janela na qual deve desenhar sua interface de usuário. Normalmente essa janela preenche toda a tela do dispositivo, mas pode ser menor que a tela e flutuar sobre outras janelas. Uma aplicação normalmente consiste de múltiplas *Activities* que estão relacionadas entre si. Tipicamente uma das *Activites* da aplicação é definida como a principal, que é apresentada ao usuário quando a aplicação é iniciada. Cada *Activity* pode então iniciar outra a fim de executar ações diferentes. Toda vez que uma nova *Activity* é iniciada a anterior é parada, mas o sistema a preserva em uma pilha. Quando uma nova *Activity* é iniciada, ela é colocada no topo da pilha e obtém foco do usuário. A pilha obedece ao mecanismo básico "ultimo a entrar, primeiro a sair" de forma que, quando o usuário termina de realizar as ações na atividade atual e pressiona o botão de voltar, a *Activity* é retirada da pilha (e destruída) e a anterior é retomada.[16]

Quando uma *Activity* é parada devido ao início de uma nova, ela é notificada sobre essa mudança de estado através dos métodos de retorno do ciclo de vida da *Activity*. Existem vários métodos de retorno que uma *Activity* pode receber devido a uma mudança em seu estado – se o sistema está criando-a, parando-a, retomando-a ou destruindo-a – e cada método fornece a oportunidade de realizar uma ação específica que é apropriada àquela mudança de estado. Por exemplo, quando parada, deve liberar quaisquer recursos tais como conexões de rede ou banco de dados. Quando a *Activity* é retomada, pode-se readquirir os recursos necessários e retomar as ações que foram interrompidas. Todas essas transições de estado são parte do ciclo de vida de uma *Activity*.[16]

- *Services*:

Um serviço é um componente de aplicação que pode executar operações de longa duração em segundo plano e não fornece uma interface de usuário. Outro componente de aplicação pode iniciar um serviço e este continuará a executando em segundo plano mesmo após o usuário alternar

para outra aplicação. Adicionalmente um componente pode ligar-se ao serviço para interagir com ele e até mesmo realizar comunicações entre processos. Por exemplo, um serviço pode lidar com transações de rede, tocar música, realizar leitura e gravação em um arquivo ou interagir com um fornecedor de conteúdo (*Content Providers*, um dos componentes de aplicação *Android*), tudo em segundo plano.[16]

- *Content Providers*:

Um *Content Provider*, ou provedor de conteúdo em uma tradução direta, gerencia um conjunto compartilhado de dados de aplicativo. Esses dados podem ser armazenados no sistema, em um banco de dados *SQLite*, na *web*, ou qualquer outra forma de armazenamento local que o aplicativo possua acesso. Por meio de um provedor de conteúdo, outros aplicativos podem consultar ou até mesmo modificar os dados (se o provedor de conteúdo permitir). Por exemplo, o sistema *Android* fornece um provedor de conteúdo que gerencia as informações de contato do usuário. Dessa forma, qualquer aplicativo com as devidas permições pode consultar parte do provedor de conteúdo para ler e gravar informações sobre uma determinada pessoa. Provedores de conteúdo também são úteis para leitura e gravação de dados que são privados para um aplicativo e não são compartilhados. Por exemplo, o aplicativo de amostra *Note Pad* disponível no *Android* SDK utiliza um provedor de conteúdo para guardar notas e arquivos simples de texto. [16]

- *Broadcast Receivers*:

Um *Brodadcast Receiver*, é um componente que responde a uma mensagem *broadcast* enviado ao sistema. Muitas mensagens deste tipo são originadas no próprio sistema, por exemplo, uma mensagem anunciando que a tela desligou, que a bateria está fraca, ou que uma imagem foi capturada. Aplicativos podem também iniciar *broadcasts*, por exemplo, para deixar outros aplicativos saberem que algum dado foi baixado para o dispositivo e está disponível para eles utilizarem. Embora um broadcast receiver não exiba uma interface de usuário, eles podem criar uma notificação na barra de status para alertar o usuário que um evento ocorreu. Mais comumente, porém, um broadcast receiver é apenas uma "entrada" para outros componentes e destina-se a fazer uma quantidade mínima de trabalho. Por exemplo, ele pode iniciar um serviço para realizar algum trabalho relacionado ao evento que disparou o *broadcast*. [16]

2.3.3 *Intents*

Intents são usados para ativar componentes de aplicações e para entregar mensagens entre aplicações diferentes ou entre componentes diferentes dentro da mesma aplicação. Consiste em uma mensagem assíncrona enviada por uma aplicação ou componente para outro. A mensagem é entregue pelo sistema operacional para o componente a qual foi endereçada. Um Intent pode ser visto como um mecanismo para disparar certas ações. Por exemplo, existe um Intent para realizar uma ligação (ACTION_DIAL) no *Android*. Quando uma aplicação necessita fazer uma ligação ela precisa enviar um Intent do tipo ACTION_DIAL com o número a ser discado anexado ao Intent. O sistema então receberá o Intent e encaminhará a mensagem para a aplicação capaz de lidar com a solicitação. Se houver mais de um componente registrado para uma determinada

ação, o sistema irá exibir uma caixa de opções ao usuário perguntando qual aplicação deve realizar a ação requerida. [1]

Uma *Intent* pode ser utilizado em três casos fundamentais:

- Para iniciar uma *Activity* (uma tela de uma aplicação);

- Para iniciar um *Service*;

- Para entregar uma mensagem *Broadcast*;.

2.3.4 O Arquivo AndroidManifest.xml

O arquivo AndroidManifest.xml é a base de uma aplicação *Android*. Ele é obrigatório e deve ficar na pasta raiz do projeto, contendo todas as configurações necessárias para executar a aplicação,como o nome do pacote utilizado, o nome das classes de cada Activity e várias outras configurações. Quando uma aplicação é instalada em um dispositivo móvel, o sistema *Android* analisa o arquivo AndroidManifist.xml para obter informações sobre os componentes embutidos, as permissões requeridas, assim como outros metadados relacionados a aplicação. [1]

Um exemplo de código de um arquivo AndroidManifest.xml é mostrado no trecho de código a seguir:

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3        package="project.voxLED"
4        android:versionCode="1"
5        android:versionName="1.0">
6      <uses-sdk android:minSdkVersion="8" />
7      <uses-permission android:name="android.permission.RECORD_AUDIO"/>
8      <application android:icon="@drawable/icon" android:label="@string/
           app_name" android:debuggable="true">
9          <activity android:name="project.voxLED.LedConect"
10                   android:label="@string/app_name">
11             <intent-filter>
12                 <action android:name="android.intent.action.MAIN" />
13                 <category android:name="android.intent.category.LAUNCHER"
                       />
14             </intent-filter>
15         </activity>
16
17     <activity android:name="Vox_LED"></activity>
18
19      </application>
20 </manifest>
```

O trecho de código acima pertence ao aplicativo VoxLed, que será explicado em detalhes no Capítulo 4. Ao analisar o arquivo AndroidManifest acima, percebe-se que existem duas Activities – LedConnect e Vox_LED – sendo que a primeira foi definida como “MAIN”, ou seja, será a Activity

que iniciará primeiro e constituirá a tela inicial da aplicação. Percebe-se também que, para que o aplicativo possa ser instalado, é necessário que o usuário forneça, no instante da instalação, a permissão "RECORD_AUDIO", que tem por função permitir o funcionamento do reconhecimento de voz.

Detalhes como a versão do aplicativo e a versão do Android para qual ele é direcionado também podem ser encontrados no AndroidManifest.xml

2.3.5 Bibliotecas de Voz

Este trabalho de pesquisa tem como alicerce a comunicação via voz com o aparelho celular. Esta será utilizada para a interação com objetos ao redor do usuário, provendo assim automação e acessibilidade.

O sistema *Android* possui em sua API o pacote *Android.speech* que disponibiliza várias classes e funcionalidades para serviços de voz.

Fazem parte desse pacote as seguintes classes:

- *Recognition Service*: fornece uma classe base para implementarções de serviço de reconhecimento;

- *RecognitionsService.Callback*: recebe retornos do serviço de reconhecimento de voz e encaminha para o usuário;

- *RecognizerIntent*: Constantes para o suporte do reconhecimento de voz através de um Intent.

- *RecognizerResultsIntent*: Constantes para *Intents* relacionadas à exibição dos resultados do reconhecimento de voz;

- *Speech Recognizer*: fornece acesso ao serviço de reconhecimento de voz;

Além disso, há também no pacote uma interface, *RecognitionListener*, usada para receber notificações da classe SpeechRecognizer quando eventos relacionados ao reconhecimento de voz ocorrem.

Para iniciar o serviço de voz dentro de um aplicativo pode-se utilizar a classe SpeechRecognizer que possuiu um listener que é chamado quando alguma palavra é pronunciada.

De forma resumida, o funcionamento desta classe pode ser visualizado neste trecho de código:

```
1 SpeechRecognizer stt = SpeechRecognizer.createSpeechRecognizer(this);
2 stt.setRecognitionListener(implents  RecognitionListener aqui)
3 Intent intent = ...;
4 stt.startListening(intent);;
```

Depois de obter a instância de *SpeechRecognizer*, é necessário informar ao listener que é a interface *RecognitionListener* que precisamos implementar para receber o resultado do reconhecimento de voz.

Em seguida deve-se criar um Intent contendo mensagens próprias para configurar o serviço de

reconhecimento de voz e que servirá de parâmetro de entrada do método *startListening(intent)* que envia o intent ao sistema e inicia o serviço de voz.

A criação do Intent pode ser feita conforme mostra o código a seguir:

```
Intent intent = getRecognizerIntent();
protected Intent getRecognizerIntent(){
            Intent intent = new Intent(RecognizerIntent.
               ACTION_RECOGNIZE_SPEECH);
 intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE_MODEL, RecognizerIntent.
    LANGUAGE_MODEL_FREE_FORM);
            intent.putExtra(RecognizerIntent.EXTRA_CALLING_PACKAGE,
               getPackageName());
            intent.putExtra(RecognizerIntent.EXTRA_PROMPT,"Fale algo");
            intent.putExtra(RecognizerIntent.EXTRA_LANGUAGE, "pt-BR");
            intent.putExtra(RecognizerIntent.EXTRA_MAX_RESULTS, 10);
            return intent;
   }
```

Além disso, como é necessário implementar a interface *RecognitionListener* e ela possui vários métodos, pode-se criar uma classe, que será chamada de *BaseRecognitionListener* que possui os métodos da interface a qual está implementando da seguinte forma:

```
public class BaseRecognitionListenet implements RecognitionListener {
      private static final String TAG = "RecognitionListener";

      public void onBeginningOfSpeech() { }

      public void onBufferReceived(byte[] buffer) {}

      public void onEndOfSpeech() { }

      public void onError(int error) {}

      public void onEvent(int eventType, Bundle params) {}

      public void onPartialResults(Bundle partialResults) {      }

      public void onReadyForSpeech(Bundle params) {}

      public void onResults(Bundle results) {}

      public void onRmsChanged(float rmsdB) {}

      public String getError(int code) {}
}
```

O método mais importante dessa interface é o *onResults(Bundle)*, que é chamado quando alguma palavra pronunciada é reconhecida. Portanto, para recuperar o que for reconhecido este

método precisa ser sobrescrevido.

Após implementar a interface *RecognitionListener* através da classe *BaseRecognitionListener*, pode-se utiliza-la como o listener da classe *SpeechRecognizer*, conforme demostra o seguinte trecho de código:

```
1 SpeechRecognizer stt = SpeechRecognizer.createSpeechRecognizer(this);
2 stt.setRecognitionListener(new BaseRecognitionListenet(){
3 public void onResults(Bundle results){
4             ArrayList<String> words = results.getStringArrayList(
                 SpeechRecognizer.RESULTS_RECOGNITION);
5
6                 }
7             });
```

Por meio do método *onResults()* é possível obter as palavras que foram reconhecidas por meio do serviço de reconhecimento de voz. Os resultados do reconhecimento de voz são adicionados ao *ArrayList* words e podem ser utilizados pelo aplicativo para realizar alguma ação com base no que foi dito pelo usuário e captado pelo serviço de reconhecimento de voz.

Porém, para que a implementação descrita funcione corretamente, é necessário que o aplicativo que esteja implementando o reconhecimento de voz possua a permissão de gravação de áudio. Para isso basta adicionar a permissão no arquivo AndroidManifest.xml (que também é responsável por gerenciar as permissões necessárias para o aplicativo funcionar corretamente).

Para adicionar a permissão basta incluir a seguinte linha de código ao AndroidManifest.xml:

```
1 <uses-permission Android:name="Android.permission.RECORD_AUDIO"/>
```

2.4 Aplicações Desenvolvidas

Com a finalidade de estudar melhor este sistema operacional, foram criadas aplicações com base nos objetivos deste trabalho, utilizando as bibliotecas de voz supracitadas e trabalhando a acessibilidade e facilidade de navegação, resultando numa interface amigável para o usuário e de simples entendimento. Estes conceitos são cruciais na computação ubíqua e tornam a experiência de utilizar um app mais agradável.

Nesta parte do trabalho foram criados dois aplicativos. Através deles é possível observar o funcionamento da API de voz oferecida pelo sistema *Android* assim como fixar os conhecimentos de programação Android estudados.

Os aplicativos criados foram o Vox_Calc e o Vox_Hub.

2.4.1 Vox_Calc, a Calculadora de Voz

O primeiro aplicativo criado para aplicar os conhecimentos de programação Android e demonstrar o funcionamento da biblioteca de voz foi o Vox_Calc, que consiste em uma calculadora na

qual as operações são inseridas por voz ao invés do tradicional teclado numérico.

São três modos de operação: Contas Simples, que realiza as quatro operações básicas, raiz quadrada e cúbica, logaritmo base 10 e neperiano, Primeiro Grau, que encontra o valor de x para uma equação de primeiro grau da forma $y = a * x + b$ e gera o gráfico correspondente quando solicitado, e Segundo Grau, que encontra as raízes da função de segundo grau $y = a * x^2 + b * x + c$ e gera o gráfico correspondente quando solicitado.

A tela inicial do aplicativo, que pode ser vista na Figura 2.2.

É nessa tela que o usuário escolhe o modo de operação desejado. Após escolher o modo de operação o usuário é direcionado para a tela observada na Figura 2.3:

Como se pode ver na figura 2.3, a tela é composta de dois botões, um para ativar o reconhecimento de voz e receber a operação a ser calculada e outro para voltar à interface de escolha do modo de operação, além de um campo no qual é exibido o resultado da operação.

Ao se pressionar o botão do reconhecimento de voz – na forma de um microfone, como visto na figura 2.3, o aplicativo inicia o serviço de reconhecimento de voz e sinaliza para o usuário através de um bip, que está pronto para receber uma operação. Após o bip, o serviço de voz aguarda por até 10 segundos o usuário começar a falar a operação desejada. Se dentro desse tempo não houver nenhuma entrada de voz o serviço de voz notifica a aplicação de que nenhuma entrada foi recebida e é encerrado. Para iniciá-lo novamente pasta apertar mais uma vez o botão de ativação na interface.

Figura 2.2: Tela Inicial do Vox_Calc

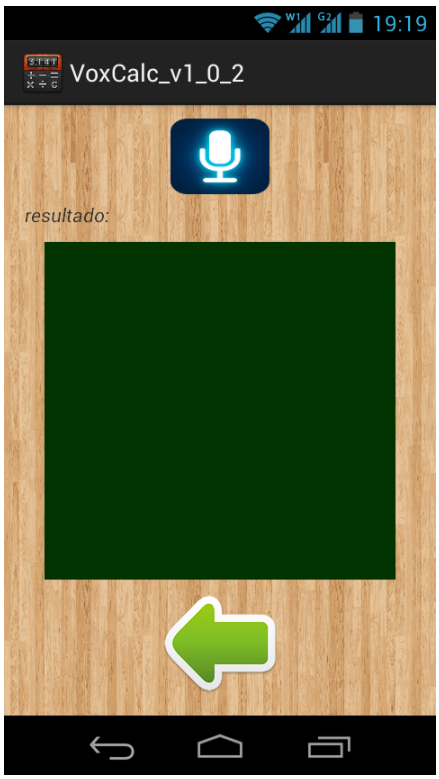

Figura 2.3: Tela de Operação do Vox_Calc

Se dentro do tempo limite o usuário falar uma operação, está é recebida pelo serviço de reconhecimento de voz que envia o que foi reconhecido para os servidores do Google através da Internet e recebe de retorno o que foi reconhecido em forma de texto que é então repassado para a aplicação.

São retornadas para aplicação dez *Strings* que mais se aproximam do que o usuário falou, das quais, geralmente, a primeira é a mais próxima. Dessa forma, para realizar o calculo, é selecionado primeiro dentre os dez resultados para a realização da operação desejada pelo usuário de acordo com o modo de operação selecionado na tela inicial da aplicação. Caso a *String* reconhecida não corresponda a uma das formas aceita pelo sistema, o usuário recebe na tela a mensagem: "Operação não reconhecida, tente novamente".

Para cada modo (Simples, Primeiro Grau e Segundo Grau) há uma forma correta de pronunciar a operação desejada. Operações pronunciadas de formas diferentes das aceitas não serão processadas e será informado ao usuário que a operação recebida não está na forma esperada.

Segue uma lista das formas apropriadas de pronuncia das operações de cada modo do aplicativo.

Para o Cálculo Simples:

– Adição: "x mais y", Ex.: "4 mais 5", "2 mais 2";

– Subtração: "x menos y". Ex.: "7 menos 2", "3 menos 1";

– Multiplicação: "x vezes y". Ex.: "2 vezes 3", "8 vezes 5";

– Divisão: "x dividido por y". Ex.: "4 dividido por 2", "6 dividido por 3";

– Raiz quadrada: “raiz quadrada de x”. Ex.: “raiz quadrada de 4”, “raiz quadrada de 9”;

– Raiz cúbica: “raiz cúbica de x”. Ex.: “raiz cúbica de 4”, “raiz cúbica de 9”;

– Logaritmo na base 10: “log de x”. Ex.: “log de 5”, “log de 7”;

– Logaritmo neperiano: “neperiano de x”. Ex.: “neperiano de 3”, “neperiano de 15”;

Após realizar o cálculo, o aplicativo retorna uma tela com o resultado exposto de forma didática como pode ser visto na Figura 2.4.

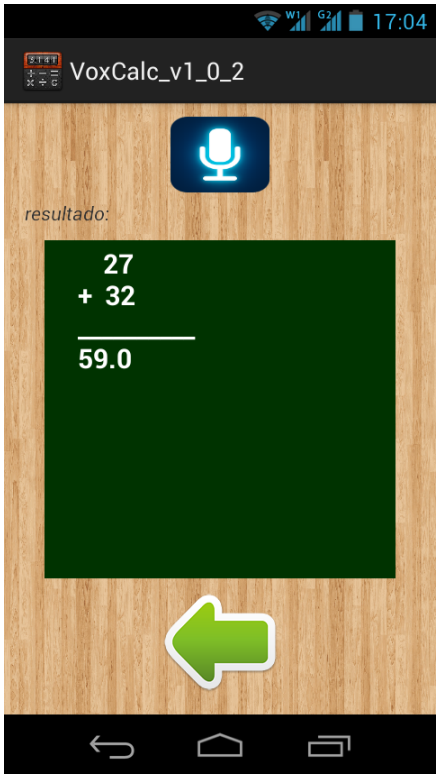

Figura 2.4: Cálculo Simples realizado com o Vox_Calc

Para Equações de Primeiro Grau:

– Equação de primeiro grau na forma “$ax+b=0$”: “ax mais b” ou “ax menos b”. Ex.: “2x mais 5”,“3x menos 7”, “menos 7x mais 8”;

– Gerar o gráfico da equação já calculada: “gráfico”.

Ao realizar o cálculo de uma equação de primeiro grau, é mostrado para o usuário o resultado de forma didática, como nos cálculos simples e pode ser observado na Figura 2.5

Ao pressionar novamente o botão e executar o comando “gráfico”, o aplicativo imprime na tela o gráfico da função obtida, como mostra a Figura 2.6

Para Equações de Segundo Grau:

– Equação de segundo grau “$ax^2+bx+c=0$”: “ax ao quadrado mais bx mais c”. Ex.: “2x ao quadrado mais 5x menos 7”;

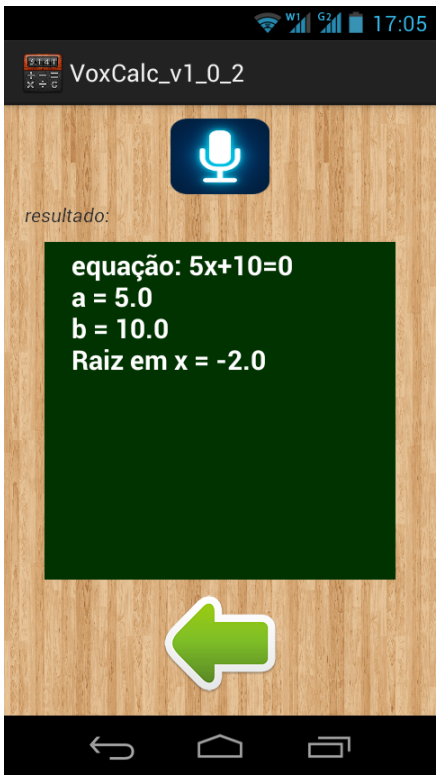

Figura 2.5: Equação de Primeiro Grau Resolvida com o Vox_Calc

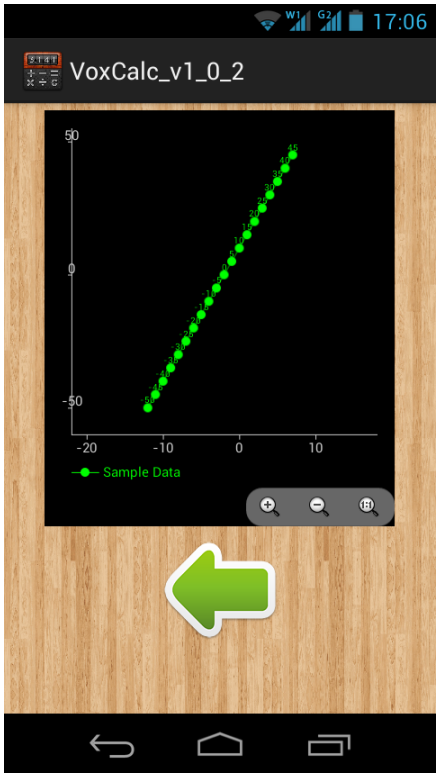

Figura 2.6: Gráfico de Equação de Primeiro Grau

– Equação de segundo grau "$ax^2 + bx = 0$": "ax ao quadrado mais bx". Ex.: "2x ao quadrado mais 5x";

– Equação de segundo grau "$ax^2 + c = 0$": "ax ao quadrado mais c". Ex.: "7x ao quadrado mais 8";

– Gráfico da equação de segundo grau já resolvida: "gráfico"

Quando uma equação é resolvida, o resultado é retornado de forma semelhante aos outros modos. A Figura 2.7 exemplifica a interface.

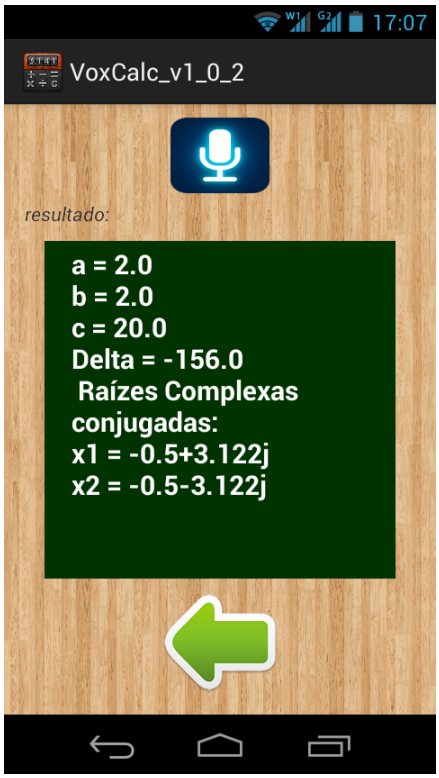

Figura 2.7: Equação de Segundo Grau Resolvida com o Vox_Calc

Pressionando novamente o botão e comandando "gráfico", obtêm-se o gráfico da função, mostrado na Figura 2.8

Um diagrama de estados do aplicativo Vox_Calc pode ser visto na Figura 2.9

A aplicação permanece na tela inicial até que o usuário escolha realizar uma operação ou ver as instruções de uso do *app*. Ao fazer a seleção, migra-se para o estado correspondente. Nestes próximos estados, o usuário tem a opção de voltar à tela inicial ou enunciar a equaçõa ou operação a ser realizada. Quando um comando de voz é executado, a resposta numérica é calculada e o resultado é exibido na tela, sem que ocorra uma transição de estado No caso do comando para a geração de gráficos, o aplicativo transiciona para um novo estado em que é gerado o gráfico da equação obtida.

Quando o usuário deseja ver as instruções de utilização do app, ao clicar em "Instruções" na tela inicial, o próximo estado será "tutorial", no qual é exibida uma tela com informações sobre

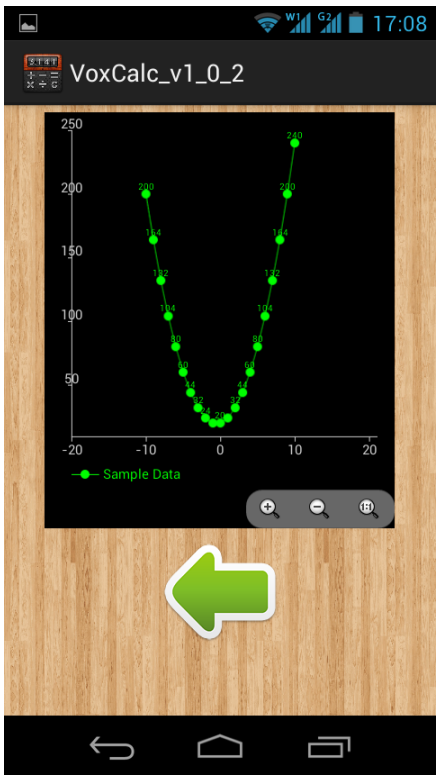

Figura 2.8: Gráfico de Equação do Segundo Grau

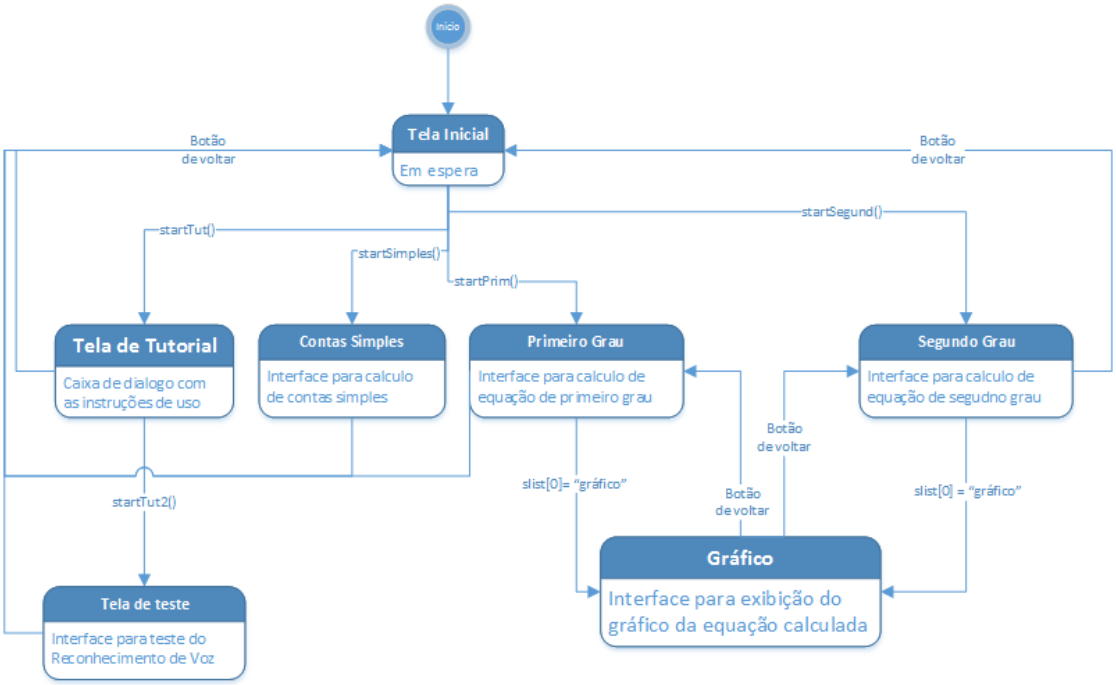

Figura 2.9: Diagrama de Estados do Vox_Calc

os comandos aceitos. É apresentada ainda ao usuário a opção de testar o comando de voz, que se aceita, encaminha-o para uma nova tela – representando um novo estado – na qual é possível realizar um teste de microfone e reconhecimento da voz em língua portuguesa.

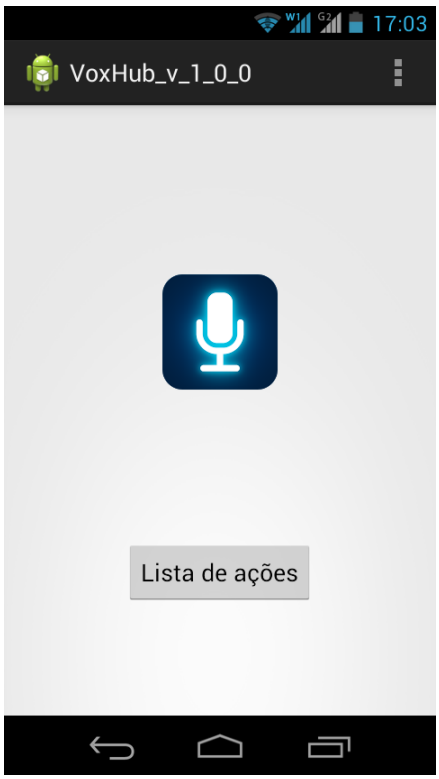

Figura 2.10: Tela Inicial do Vox_Hub

2.4.2 Vox_Hub. Uma pequena central de voz

Para demostrar a possibilidade de abrir outros aplicativos a partir de comandos de voz foi criado o Vox Hub.

O Vox Hub consiste apenas de uma interface principal que contem apenas o botão para ativar o reconhecimento de voz e um botão para listar as ações que podem ser executadas via comando de voz. Diversas opções são colocadas à disposição do usuário. A tela inicial do Vox Hub pode ser vista na Figura a 2.10.

Os comandos aceitos pelo aplicativo são os seguintes:

– "calculadora": inicia o aplicativo Vox_Calc (descrito na sessão anterior);

– "agenda": inicia o aplicativo de agenda de contatos padrão do dispositivo;

– "www.site.com": abre o endereço web "www.site.com"no navegador web padrão do dispositivo. Basta informar um endereço web válido que inicie com "www";

– "buscar" + "palavra chave": busca por "palavra chave" em www.google.com e abre o resultado da busca no navegador web do dispositivo. Ex.: "buscar UnB", realiza a busca pela palavrava chave "UnB" no Google e abre o resultado no navegador.

– "ligar para xxxx-xxxx", realiza a ligação telefônica para o número "xxxx-xxxx". Ex.: "ligar para 99999999"

Um diagrama de estados do aplicativo Vox_Hub na figura 2.11

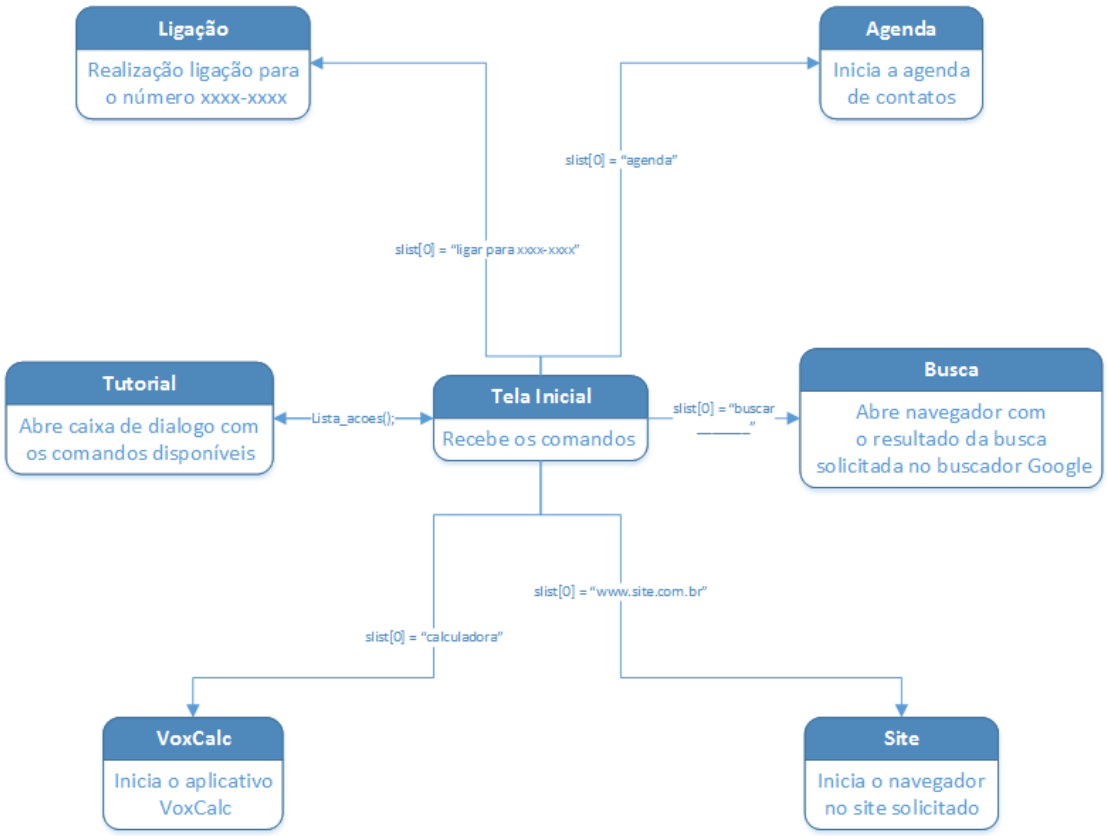

Figura 2.11: Diagrama de Estados do Vox_Hub

O VoxHub possui um estado principal que, de acordo com o comando recebido, encaminha o usuário para o destino desejado. Cada operação de transição de estado deriva diretamente do comando de voz.

2.5 Conclusão

O sistema operacional *Android* é definitivamente a melhor opção para o desenvolvimento rápido de aplicativos, especialmente quando o objetivo é valer-se de conceitos de computação ubíqua para prover soluções de automação e acessibilidade. Isto se deve a diversas características, desde a facilidade desenvolvimento até as bibliotecas de voz de fácil utilização, passando pelo grande número de usuários e facilidade de entrada no crescente mercado de apps.

Este capítulo mostrou de forma resumida a arquitetura e funcionamento interno do sistema Android, bem como suas bibliotecas de voz utilizadas para o desenvolvimento de aplicações. Os aplicativos mostrados nesse capitulo foram desenvolvidos com fins didáticos, de forma a compreender o funcionamento de aplicativos Android e principalmente demonstrar o funcionamento do reconhecimento de voz.

Ao utilizar comandos de voz para realizar cálculos e abrir outros aplicativos percebe-se a carac-

terística que essa funcionalidade possui de facilitar a vida do usuário, tornando o programa simples e intuitivo, preservando os conceitos de computação pervasiva.

Capítulo 3

Plataforma *Arduino* e sua Conexão com o *Android* via *Bluetooth*

3.1 Introdução

> "Arduino é uma plataforma open-source de prototipagem eletrônica baseada em hardware e software flexíveis e fáceis de usar. É destinado a artistas, designers, entusiastas e qualquer pessoa que tenha interesse em criar objetos e ambientes interativos."

www.arduino.cc (Tradução Livre)

Desenvolver protótipos e soluções inovadoras é um desafio. A plataforma Arduino existe justamente para facilitar este processo. Devido a sua simplicidade de montagem, facilidade de programação e baixíssimo custo, o seu *toolkit* microcontrolador é o mais popular atualmente entre projetistas. A Figura 3.1 mostra exemplos de placas Arduino famosas no mercado.

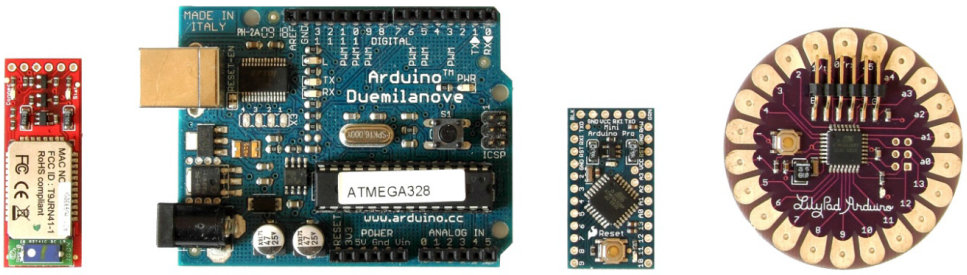

Figura 3.1: Exemplos de placas Arduino

Para começar a desenvolver, basta uma placa controladora Arduino e o Arduino IDE,e já se pode fazer o *upload* de programas para o microcontrolador da placa.

3.2 A Placa Controladora

No centro de funcionamento do Arduino está o microcontrolador. Também estão presentes na placa componentes diversos que auxiliam em seu funcionamento e portas de entrada e saída de informação. A Figura 3.1 mostra alguns tipos de placas Arduino, cada uma com suas próprias características, bem como um shield *Bluetooth*, que foi utilizado neste trabalho para transmitir informação sem fio.

Um tipo específico de placa Arduino, que também foi utilizada neste projeto, é a Arduino Duemilanove, cujo aspecto é apresentado na Figura 3.2.

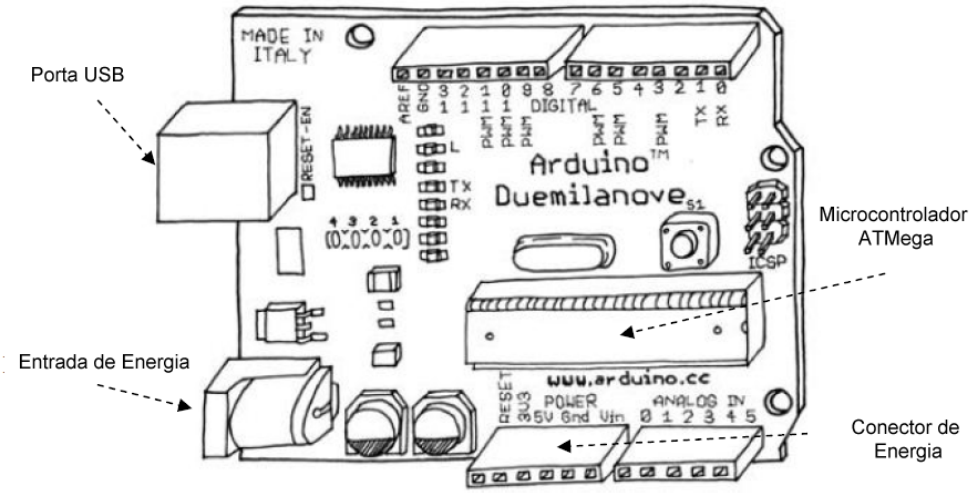

Figura 3.2: Esboço do Arduino Duemilanove

A placa Arduino Duemilanove utilizada no protótipo deste projeto possui um microcontrolador ATMega328P, 14 entradas de I-O. O *clock* tem uma velocidade de 16 Mhz e uma memória flash de armazenamento de código de 32 Kb, 2 kB de SRAM para armazenamento das variáveis e 1 kB de EEPROM que guarda os dados permanentemente mesmo após a placa ser desligada.

Cada um dos 14 pinos digitais do Duemilanove pode ser usado como entrada ou saída, usando as funções de pinMode(), digitalWrite(), e digitalRead(). Eles operam com 5V. Cada pino pode fornecer ou receber um máximo de 40 mA e tem um resistor *pull-up* interno (desconectado por padrão) de 20-50 KOhms. Além disso, alguns pinos tem funções especializadas:

– Serial: 0 (RX) e 1 (TX). Usados para receber (RX) e transmitir (TX) dados seriais TTL. Estes pinos são conectados aos pinos correspondentes do chip serial "FTDI USB *to* TTL". É através deles que o módulo *Bluetooth* transmite dados.

– PWM: 3, 5, 6, 9, 10 e 11. Fornecem uma saída analógica PWM de 8 bits com função analogWrite().

– SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). Estes pinos suportam comunicação SPI.

– LED: 13. Há um LED já montado e conectado ao pino digital 13.

O Duemilanove tem 6 entradas analógicas, cada uma delas está ligada a um conversor analógico-digital de 10 bits, ou seja, convertem o sinal analógico em um valor dentre 1024 possibilidades (exemplo: de 0 a 1023). Por padrão, elas operam na faixa de 0 a 5 V, embora seja possível mudar o limite superior usando o pino AREF e um pouco de configuração. Adicionalmente, alguns pinos têm funcionalidades especializadas:

– I2C: 3 (DAS) e 5 (SCL). Suportam comunicação I2C (TWI) usando a biblioteca Wire.

– AREF: referência de tensão para entradas analógicas. Usados com analogReference().

– *Reset*. Entrada para o sinal de *Reset*, ativado em nível baixo. Tipicamente utilizado para adicionar um botão de reiniciar aos *shields* (placas que podem ser plugadas ao *Arduino* para extender suas capacidades).

A placa pode ser alimentada pela conexão USB ou por qualquer fonte de alimentação externa. A alimentação é selecionada automaticamente.

A alimentação externa pode ser tanto de uma fonte ou de uma bateria. A fonte pode ser conectada com um plug de 2,1 mm (centro positivo) no conector de alimentação. Se for proveniente de uma bateria basta conectar os cabos da bateria nos pinos GND (terra) e Vin (entrada de tensão) do conector de alimentação.

O Arduino Duemilanove tem ainda um polifusível que protege a porta USB do computador contra curto-circuito e sobrecorrente. Apesar da maioria dos computadores possuírem proteção interna própria, o fusível proporciona uma proteção extra. Se mais de 500 mA forem aplicados na porta USB, o fusível irá automaticamente interromper a conexão até que o curto ou sobrecarga seja removida. Um resumo com as configuração da placa Duemilanove pode ser vista na Tabela 3.1:

Microcontrolador	ATmega328 / ATmega168
Pinos de entrada analógica	6
Pinos de I/O digitais	14 (dos quais 6 podem ser saídas PWM)
Tensão operacional	$5\ V$
Tensão de alimentação (recomendada)	$7 - 12\ V$
Tensão de alimentação (limites)	$6 - 20\ V$
Corrente contínua por pino de I/O	$40\ mA$
Corrente contínua para o pino 3.3 V	$50\ mA$
Memória flash	32 KB (2KB usados para o *bootloader*) / $16\ KB$
SRAM	$2\ KB$
EEPROM	$1\ KB$
Freqüência de clock	$16\ MHz$

Tabela 3.1: Características Básicas do Arduino Duemilanove

A placa Arduino Duemilanove se conecta diretamente a um computador através de um cabo USB. Dessa forma, não é necessário a adição de *hardware* externo, facilitando o processo de escrita e *upload* de código. O ambiente no qual se desenvolve os programas para o Arduino, chamado Arduino IDE, é apresentado a seguir.

3.3 O Desenvolvimento de Aplicações com Arduino IDE

Uma das vantagens do Arduino é justamente a facilidade da programação. O Arduino IDE é gratuito e simples, contendo um editor de texto, um compilador e um monitor serial.

A linguagem de programação utilizada para escrever programas Arduino é baseada em C/C++. Desta forma, quando se compila um programa, o compilador traduz o texto escrito para C e então utiliza um compilador open-source avgr-gcc. O código compilado é então enviado ao microcontrolador da placa para ser utilizado. Na Figura 3.3 é apresentado um exemplo simples de um programa Arduino.

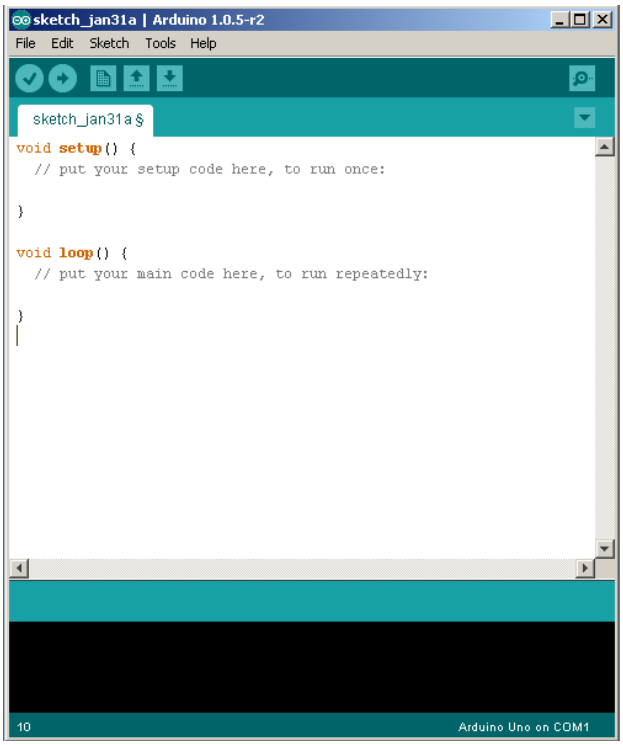

Figura 3.3: Programa Arduino Básico

Praticamente todos os programas contém uma função setup e uma função loop. A função setup é executada logo que o microcontrolador inicia o funcionamento do programa e a função loop é executada em um processo contínuo até que a energia seja cortada ou um outro código seja enviado ao microcontrolador.

O IDE Arduino possui diversas bibliotecas, funções e constantes próprias, que visam simplificar o código e deixar a programação mais intuitiva. A referência completa de toda a linguagem é encontrada em www.arduino.cc .

3.4 Conexão Bluetooth

3.4.1 Breve resumo do protocolo

O *Bluetooth*, criado pela empresa Ericsson em 1994, utiliza uma potência de curto alcance para transmitir informações via *wireless* entre dispositivos. Devido a essa característica, é chamado de uma especificação de PAN – Personal Area Network – sem fio. O "*Bluetooth Special Interest Group*"é o grupo internacional responsável por desenvolver e adaptar o protocolo.

Graças ao seu baixo consumo de energia, está presente em grande parte dos aparelhos celulares no mundo. No entanto, a potência de transmissão de diferentes dispositivos pode variar, como mostra a 3.2, existem diferentes classes de *Bluetooth*.

Classe	Potência (mW)	Potência (dBm)	Alcance (m)	Exemplos de dispositivos
1	100	20	~100	BT Access Point, dongles
2	2.5	4	~10	Teclado, mouse, smartphone
3	1	0	~1	Headsets

Tabela 3.2: Tabela de classes Bluetooth

A grande maioria dos aparelhos celulares é composta de transmissores *Bluetooth* classe 2, ou seja, transmitindo em 2,5mW e com um alcance médio de até 10m.

Embora seja citado em muitas publicações como um protocolo, o *Bluetooth* é na verdade um conjunto de protocolos, divididos em camadas, cada qual com sua função específica. As camadas são conhecidas como protocolos núcleo, protocolos de substituição de cabo, protocolos de controle de telefonia e protocolos adaptados.

A camada de núcleo tem a função de especificar o tipo de conexão entre dispositivos. Possui protocolos que estabelecem a conexão e otimizam a transmissão sem fio. A camada de substituição de cabo, que cria uma porta serial virtual no transmissor e no receptor, é, na prática constituída dos protocolos que formam o canal de comunicação, propriamente dito. A camada de controle de telefonia define o controle de chamada e estabelece a sinalização de início e fim de transmissão. Finalmente, alguns protocolos com outras funções são adaptados para o funcionamento do *Bluetooth*, como o TCP/IP e o UDP, por exemplo.

3.4.2 O Módulo *Bluetooth* JY-MCU

Para que haja comunicação entre o disposivito *Android* e a placa Arduino, é necessário um canal de comunicação sem fio. O *Bluetooth* é a opção mais adequada neste caso, principalmente por estar presente nos dois dispositivos, ter um bom alcance e ser completamente invisível ao usuário final.

Praticamente todos os aparelhos celulares e *tablets* modernos que trabalham com o sistema *Android* possuem uma interface *Bluetooth*. No caso do Arduino, no entanto, é necessário que haja um módulo conectado a placa para que possa transmitir e receber informações via *Bluetooth*.

Existem alguns modelos de módulos no mercado. O mais fácil de conectar e programar é o JY-MCU, mostrado na Figura 3.4, que foi utilizado neste trabalho justamente pela simplicidade, baixo custo e funcionamento eficaz. Quando conectado, o JY-MCU é capaz de ser localizado por outro dispositivo com *Bluetooth*, por exemplo um aparelho *Android*, e dados podem ser enviados a um alcance de até 10 m.

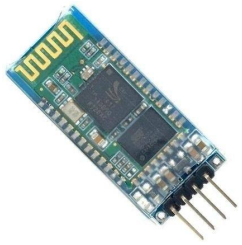

Figura 3.4: Módulo Bluetooth JY-MCU

Utilizando este módulo, é possível transmitir informações de um aparelho celular ou *tablet* para o Arduino. Assim, utilizando comandos de voz para interagir com o *Android*, indiretamente está se interagindo com o Arduino. Torna-se possível então acionar motores, lâmpadas e quaisques outros aparelhos elétricos ou eletrônicos à distância atravéz da voz.

O módulo tem seu endereço MAC como identificação para que o dispositivo *Android* o encontre. Para completar o pareamento entre os dois dispositivos, o módulo *Bluetooth* solicita ao sistema *Android* que confirme uma senha padrão, a confirmação da senha é, então, solicitada ao usuário que necessita inserir a senha apenas uma vez. Essa senha é padrão para a maioria dos módulos *Bluetooth*s e consiste na sequência 12345.

Ainda assim, o *Android* e o Arduino devem "se entender", ou seja, comunicar-se em uma linguagem comum. É justamente aí que entra a biblioteca Amarino.

3.5 Amarino: *Android* + Arduino

Amarino é um sistema que permite que dispositivos *Android* se comuniquem com o Arduino. Possui dois componentes: um aplicativo *Android* chamado "Amarino" e uma biblioteca Arduino chamada "MeetAndroid". A logomarca do Amarino é mostrada na Figura 3.5.

O aplicativo Amarino é uma interface gráfica amigável para que o usuário possa enviar e receber informações para o Arduino. Possui uma série de eventos e ações de teste que podem ser utilizados, bem como um centro de controle de gerenciamento de conexões *Bluetooth*. Para encontrar um dispositivo, no entanto, é necessário que a função *Bluetooth* do aparelho já esteja ativada.

Em uma configuração avançada, o aplicativo Amarino é capaz trabalhar em conjunto com outra

Figura 3.5: Logomarca Amarino

aplicação, fazendo com que este tercerio *app* comunique-se com o Arduino. Isto torna possível a customização e criação de soluções que utilizam *Android* e Arduino em conjunto. Neste projeto, foram desenvolvidas aplicações que utilizam o Amarino como base para interagir com sistemas externos, isto será explicado em detalhes no capítulo 4.

O outro componente do sistema Amarino é a biblioteca "MeetAndroid", que contém funções Arduino necessárias para que a comunicação ocorra sem problemas. Além de tudo isso, o Amarino provê suporte para o desenvolvimento de aplicações que utilizam suas propriedades em segundo plano. Em resumo: o Amarino funciona também como um *background* service.

Para fazer uso dessa funcionalidade, basta descrever os eventos que se deseja utilizar através de intents no *Anrdoid*, assim, o Amarino pode receber e retransmitir esses dados para o Arduino. A biblioteca MeetAndroid é amplamente utilizada para este fim. Um resumo do processo é apresentado na Figura 3.6:

Como indica a Figura acima, o primeiro passo é definir a intent no código *Android*. O aplicativo Amarino então traduz o comando recebido e trabalha em sincronia com o programa Arduino em atividade na placa conectada via *Bluetooth*.

Além de transmitir comandos do dispositivo *Android* para o Arduino, o Amarino também é capaz de realizar o inverso, recebendo dados obtidos pelo Arduino. O estabelecimento do canal de comunicação ocorre de maneira bem simples, basta que a aplicação *Android* envie a seguinte intent:

```
1 sendBroadcast (new Intent("amarino.CONNECT");
```

Este comando automaticamente inicia o Amarino como um serviço em *background* no *Android*. Da mesma maneira, pode-se usar o comando acima para desconectar os dispositivos, ao terminar a aplicação.

3.6 Conclusão

O Arduino é uma solução para protótipos muito popular no planeta. Suas origens educacionais são o grande aspecto por trás de sua simplicidade e baixo custo. Por ser versátil, é utilizado em

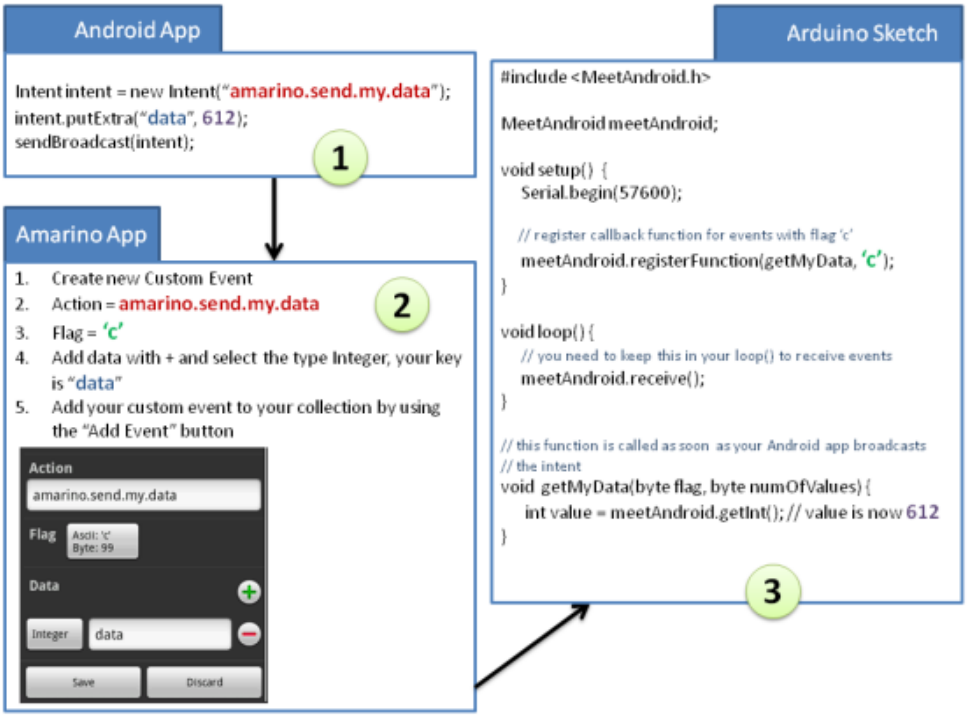

Figura 3.6: Resumo de Funcionamento do Amarino[5]

muitas soluções distintas, inclusive no Amarino, sistema de comunicação entre dispositivos *Android* e placas microcontroladoras.

O controle de mecanismos externos também é uma característica do Arduino, e através de seus pinos de entrada e saída (I/O) é possível enviar sinais para circuitos e outros aparelhos. Ao utilizar o Amarino, o dispositivo *Android* pode enviar os comandos a placa controladora que, através de seu controlador ATMega, torna-se uma intermediária, encaminhando o comando para o dispositivo externo conectado.

Tudo isto pode ser programado utilizando o Amarino como um serviço de *background* no *Android*. Embora exista uma aplicação própria e uma biblioteca Arduino (meetAndroid), desenvolvedores podem criar seus próprios apps para trabalhar com a comunicação *Bluetooth* entra *Android* e *Arduino*.

No próximo capítulo, serão apresentadas as aplicações criadas neste trabalho, que utilizam justamente este serviço do Amarino. Diversos conceitos de acessibilidade, computação pervasiva e interação com o usuário são aplicados para criar soluções úteis e aplicáveis. Além disso, os apps são fáceis de usar e instalar, permitindo, dentre diversas funções, controlar LEDs e circuitos elétricos através da voz em língua portuguesa.

Capítulo 4

Princípios Soluções Propostas Utilizando Conceitos do Amarino

4.1 Introdução

Proposto e desenvolvido por Kaufmann, o Amarino possui uma infinidade de aplicações, principalmente devido a enorme gama de aparelhos Android existentes no mercado, bem como os diferentes tipos de Arduino com suas diferentes finalidades. Diversas experiências e protótipos foram montados.

Em suas publicações, Kaufmann cita diversas vezes as possíveis aplicações do Amarino em soluções de automação (algumas já existentes) e cita também, mesmo que separadamente, a vantagem da sua pesquisa para prover mais acessibilidade em grandes cidades, prédios públicos e até mesmo em residências. No entanto, há pouca referência às vantagens de juntar as duas ideias, para obter uma solução mais completa. [5]

Buscando unir automação, acessibilidade, e os conceitos da computação pervasiva, foram criadas duas aplicações que utilizam como base o Amarino, a junção de *Android* e *Arduino* através da conexão *Bluetooth*. O VoxLed, que controla 3 LEDs apenas com o uso da voz em português, e o VoxLamp, que ativa um relé para ligar ou desligar um aparelho elétrico também através da voz em língua portuguesa.

4.2 VoxLed

O VoxLed tem por finalidade tornar intuitivo o comando de LEDs através da voz. Possui inteface simples, consistindo apenas de um botão para receber os comandos do usuário. Antes de efetuar os comandos, porém, é preciso se inicie uma conexão *Bluetooth* entre o aparelho e o módulo conectado à placa *Arduino*. Desta forma, ao iniciar o aplicativo, o usuário depara-se com a interface da Figura 4.1:

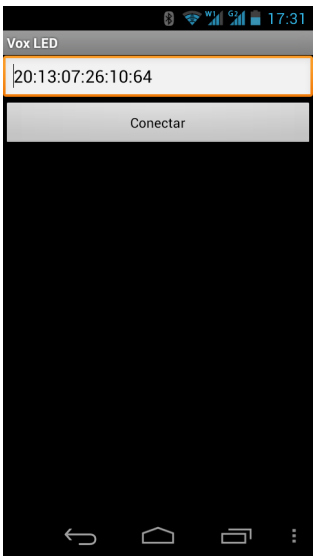

Figura 4.1: Tela Inicial do VoxLed

Esta primeira tela tem o objetico de efetuar a conexão via *Bluetooth* do aparelho com o módulo *Arduino*. A ideia aqui é permitir que o usuário mude o endereço do dispositivo com o qual queira se conectar, permitindo conectar com dispositivos diferentes. Esta tela está presente nos dois aplicativos deste trabalho. Ao pressionar o botão, a seguinte função é chamada:

```
1    public void onClick(View v)
2    {
3        DEVICE\_ADDRESS = idField.getText().toString();
4        PreferenceManager.getDefaultSharedPreferences(this)
5            .edit()
6                .putString("device", DEVICE\_ADDRESS)
7            .commit();
8        Amarino.connect(this, DEVICE\_ADDRESS);
9        Intent i = new Intent(this, Vox_LED.class);
10       startActivity(i);
11   }
```

Ela recebe o valor "DEVICE_ADDRESS", digitado pelo usuário e realiza a conexão por meio da função "Amarino.connect". Logo após, inicia a Activity Vox_Led.class, que consiste na tela para receber os comandos de voz.

A conexão poderia ser efetuada sem a necessidade do botão, ou mesmo da interface, mas neste caso o usuário ficaria retido a conectar-se apenas com um módulo *Bluetooth* específico. Após a comunicação ser iniciada, o usuário é então encaminhado para a tela principal do aplicativo, mostrada na Figura 4.2.

Tudo que o usuário precisa fazer é pressionar o botão para falar. Os comandos são palavras-chave previamente definidas, no caso "vermelho", "azul" e "verde", e servem para ligar o LED da respectiva cor. Caso o comando seja executado com o LED ligado, ele será desligado. Os comandos de voz servem para colocar os pinos do Arduino nos quais os LEDs estão conectados em HIGH

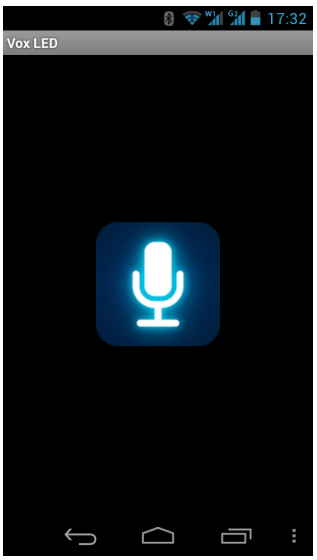

Figura 4.2: Tela Principal do VoxLed

(5V) ou LOW (0V), dependendo apenas da condição anterior do pino. Isto significa que o conceito pode ser aplicado com outros dispositivos, não exclusivamente com LEDs.

Para que a aplicação funcione é necessário conectar o *Arduino* a um pequeno circuito, ligando os LEDs vermelho, verde e azul aos pinos 9, 10 e 11, respectivamente. As instruções são intuitivas e derivam de pré-definições estabelecidas no código do aplicativo *Android*:

```
1 private void updateRed(){
2         Toast.makeText(Vox_LED.this, Integer.toString(red), Toast.
              LENGTH_SHORT).show();
3         Amarino.sendDataToArduino(this, LedConect.DEVICE_ADDRESS, 'r',
              red);
4     }
```

No trecho de código acima está o exemplo da função "updateRed()" que é chamada quando o comando "vermelho" é dito pelo usuário. A função envia ao *Arduino* o comando 'r', que faz com que a placa inverta a posição lógica do pino 9, conforme pode ser visto no trecho do código *Arduino* presente na placa:

```
1 void red(byte flag, byte numOfValues)
2 {
3      analogWrite(redLed, meetAndroid.getInt());
4 }
```

A função *Arduino* "red", previamente registrada como "r", faz a operação *Write*, que neste caso é a função que envia ao pino seu novo valor lógico. Este valor é um inteiro que varia de 0 a 255, sendo 0 desligado e 255 totalmente ligado. Este valor inteiro é obtido do *Android*, através da variável também chamada "red", conforme mostra o código do aplicativo *Android* acima.

As outras cores realizam a mesma operação, mudando apenas os nomes das funções e variáveis. A grande realização deste aplicativo é a junção da biblioteca de voz (explicada na seção 2.4) com

o Amarino.

Um circuito montado com 3 LEDs conectados ao *Arduino* para uso do VoxLED é apresentado na Figura 4.3.

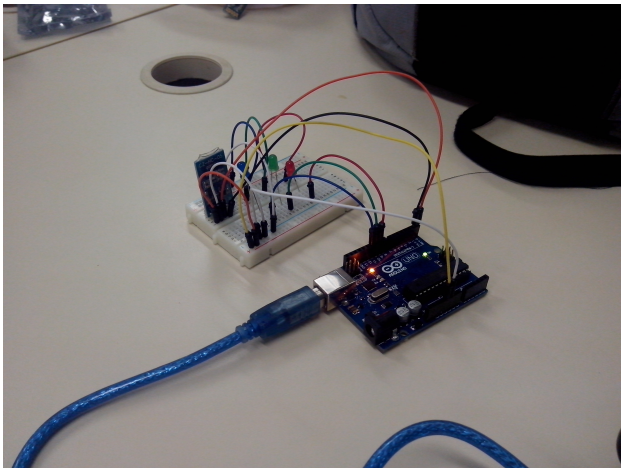

Figura 4.3: Circuito de 3 LEDs para teste do VoxLED

Um diagrama de estados do aplicativo VoxLed na figura 4.4

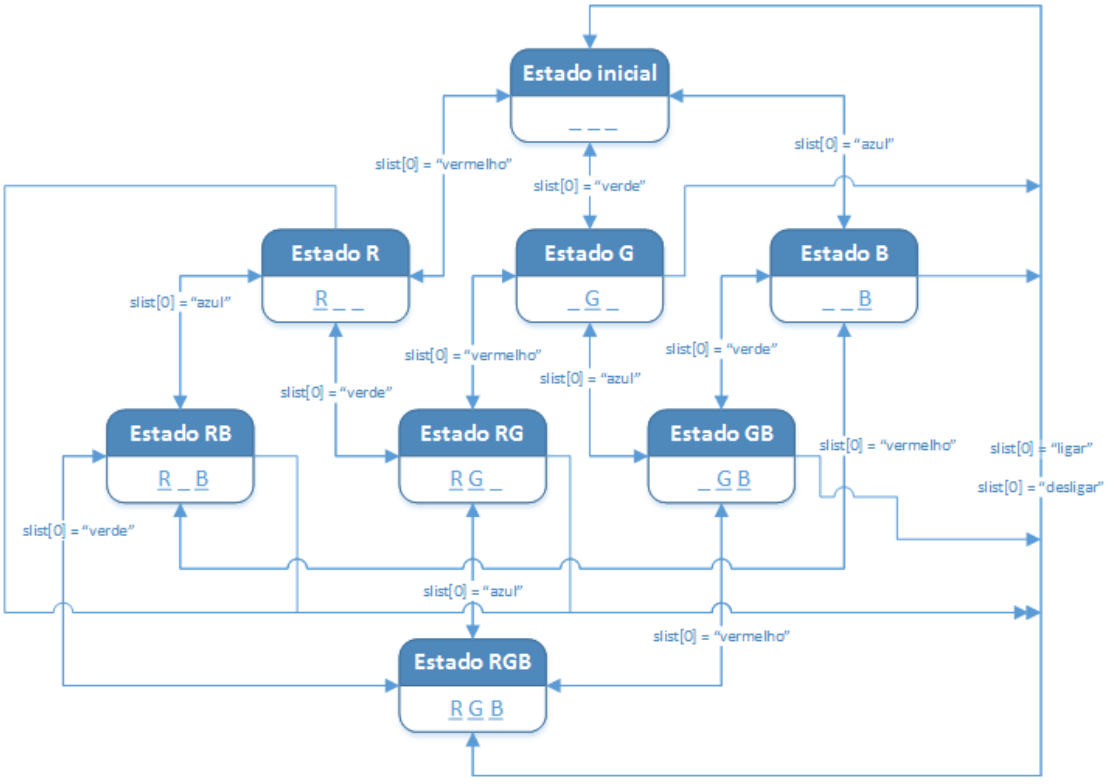

Figura 4.4: Diagrama de Estados do VoxLed

4.3 VoxLamp

O VoxLamp recebe seu nome da ideia inicial de usar os comandos de voz para ligar lâmpadas em uma sala, no entanto, a aplicação tem como característica a ativação de um relé, que pode ligar ou desligar qualquer tipo de aparelho elétrico.

Sua interface, assim como no VoxLed, também é simples. A tela inicial contém apenas a interface de conexão bluetooth, sendo que os comandos de voz são recebidos na próxima tela. A tela inicial do VoxLamp é apresentada na Figura 4.5.

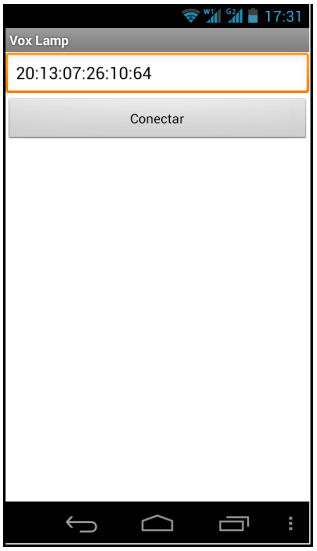

Figura 4.5: Tela Inicial do VoxLamp

Após realizada a conexão *Bluetooth* através do botão "Conectar", o usuário é encaminhado para a tela principal do aplicativo, que contém apenas o botão para receber os comandos. Os comandos resumem-se a ativar ou desativar determinado circuito. A Figura 4.6 mostra a interface do VoxLamp.

Por meio do comando de voz, que pode ser enviado por uma palavra-chave previamente definida, um sinal de 5 volts é enviado do *Arduino* para um relé, que é então chaveado, permitindo ou não a passagem de uma corrente externa até um dispositivo elétrico. O circuto da Figura 4.7 apresenta o circuito.

Quando um sinal de 5V é enviado a partir do pino 4 da placa *Arduino*, este sinal passa por um resistor e então polariza um transistor. Este tem a função de liberar a passagem da corrente proveniente da fonte 5V do *Arduino*, que ainda passa por um diodo para evitar que uma corrente reversa cause danos à placa. O relé então é ativado e fecha ou abre o circuito entre a lâmpada, neste caso, e a tomada.

O pino 4 do *Arduino* só troca de estado quando as palavras-chave são ditas ao dispositivo *Android*. Por exemplo, a palavra "ligar" coloca o pino no estado 255 (Recorda-se, do Capítulo 3 que este é um pino PWM) e a palavra "desligar" o coloca na posição lógica 0. O método de envio de dados do *Android* para o *Arduino* é semelhante ao do aplicativo VoxLed, mas ao invés de trocar

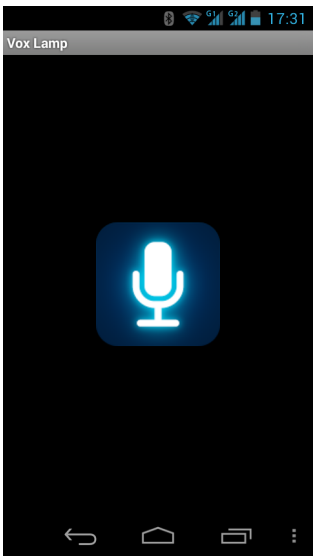

Figura 4.6: Tela Principal do VoxLamp

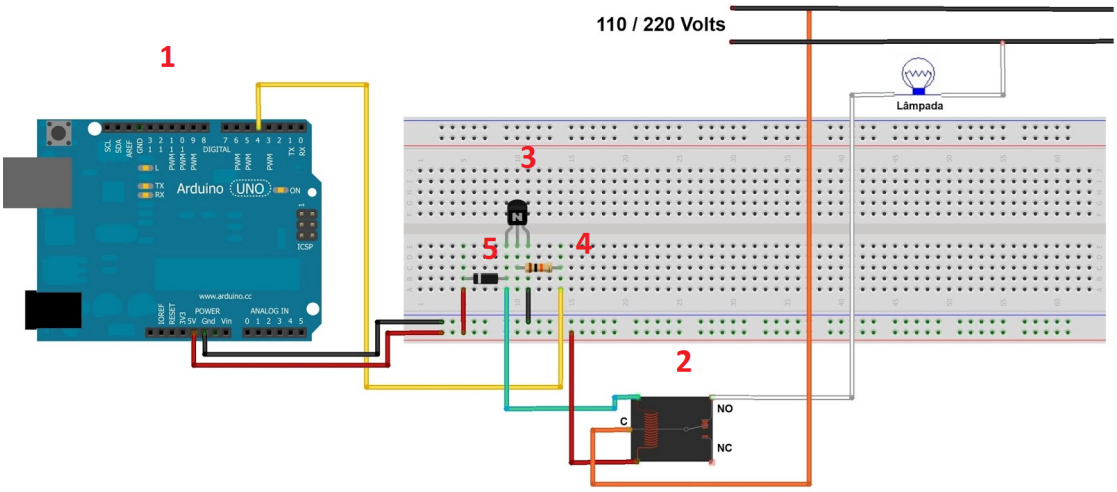

Figura 4.7: Circuito para Controlar uma Lâmpada. Conectado ao VoxLamp.

a posição de determinado pino, define sua posição em 0 ou 255.

```
if(slist[0].equals("desligar")){
    led = 0;
    updateLamp(); }
if(slist[0].equals("ligar")){
    led = 255;
    updateLamp(); }
```

Os comandos possíveis estão especificados no trecho de código acima. Como se pode ver, as palavras "ligar" e "desligar" estão pré-definidas como comandos, podendo ser alteradas no programa *Android*, caso se deseje. Após executado um comando, o valor da variável, aqui chamada "led", é atualizado e a função "updateLamp()", que se assemelha a "updateRed()" ou qualquer outra das funções "update()", é chamada. Esta função apenas faz a operação "analogWrite", mandando via

bluetooth o valor ao *Arduino*, que então é responsável por atualizar o valor de seu pino 4, que chaveia o relé, ligando ou desligando o circuito a ele conectado.

O circuto do VoxLamp pode ser instalado facilmente em uma sala, podendo ficar totalmente escondido do usuário, que apenas usaria seu celular e sua voz para ativar o circuito, permitindo assim uma interação completamente intuitiva. A ideia de ativar um relé através da voz pode ser estendida para ativar qualquer tipo de aparelho elétrico, criando assim diversas opções de aplicação para o VoxLamp. As Figuras 4.8, 4.9 e 4.10 apresentam um circuito de teste que foi montado para o VoxLamp. O relé foi ativado pelo comando de voz, permitindo a passagem de corrente pela lâmpada.

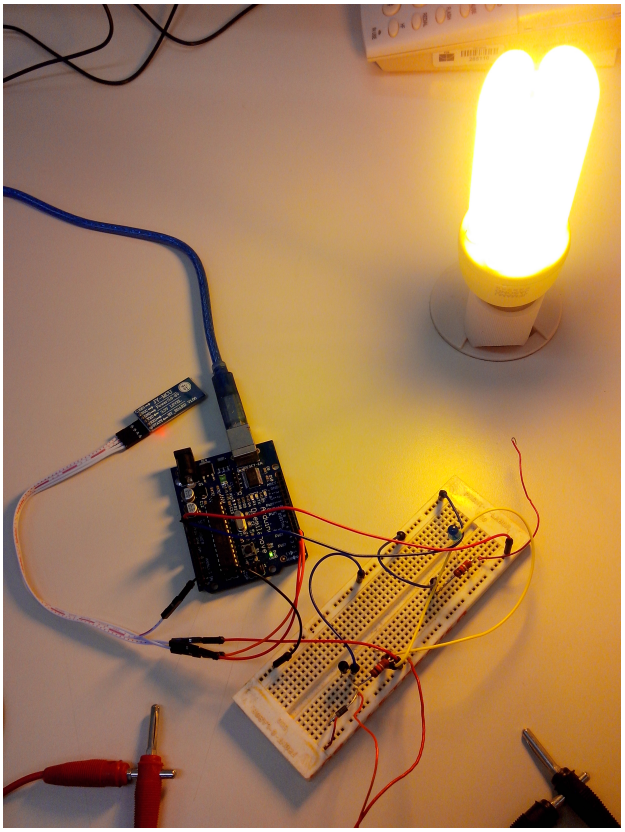

Figura 4.8: Circuito de teste do VoxLamp. Lâmpada foi ligada através da voz.

Um diagrama de estados do aplicativo VoxLamp é apresentado na Figura 4.11

4.4 Conceitos que Motivaram o VoxLed e o VoxLamp

Quando Weiser criou o conceito de computação ubíqua em 1991, não existia Android nem sequer Arduino, no entanto, já se falava da combinação de aparelhos *handheld*, portáteis, que teriam a função de controlar circuitos externos. Na época, não se imaginou que um telefone celular seria capaz de tal façanha. [7]

Atualmente, os dispositivos portáteis de alta capacidade de processamento estão ficando cada vez mais inteligentes, rápidos, indispensáveis. Então Kaufmann criou o Amarino, para fazer com

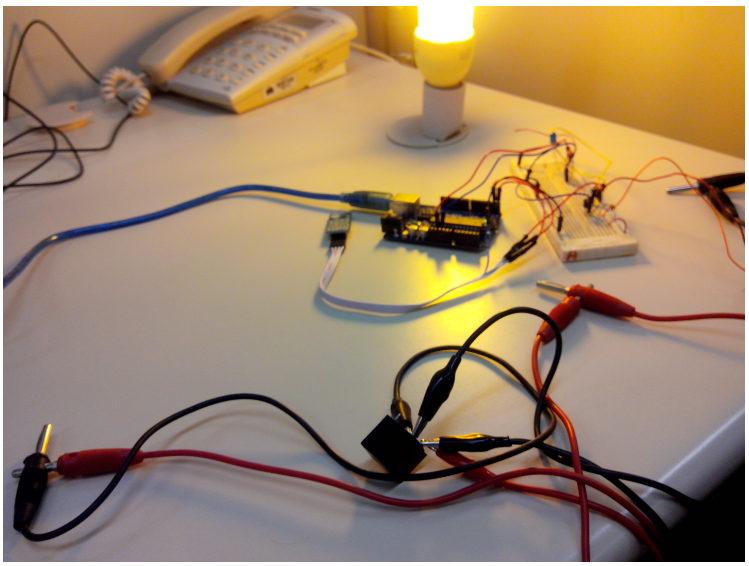

Figura 4.9: Detalhe em primeiro plano do relé conectado ao circuito

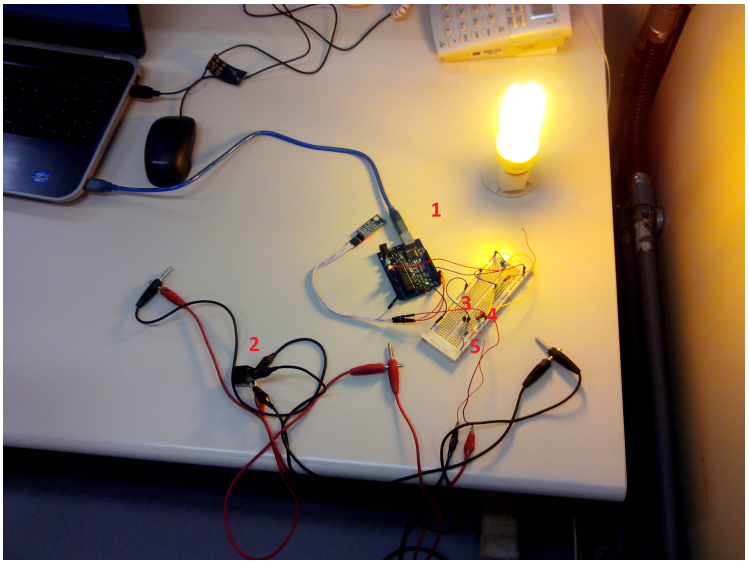

Figura 4.10: Visão Geral do Circuito

que dispositivos Android, o sistema operacional de aparelhos móveis mais popular do mundo, fossem capaz de trabalhar em conjunto com o Arduino para controlar circuitos externos e executar as mais diversas tarefas. [6]

Este trabalho propôe o uso de duas aplicações que empregam conceitos de acessibilidade de simplicidade de interação com o usuário para controlar circuitos elétricos externos ou LEDs, criando assim uma solução de automação com alta aplicabilidade. Os apps VoxLed e VoxLamp são simples e muito eficazes.

Com a habilidade de controlar LEDs através da voz, pode-se criar efeitos de luzes bastante complexos, e com a possibilidade de ativar ou desativar circuitos através da voz a uma distância considerável, as possibilidades são virtualmente infinitas.

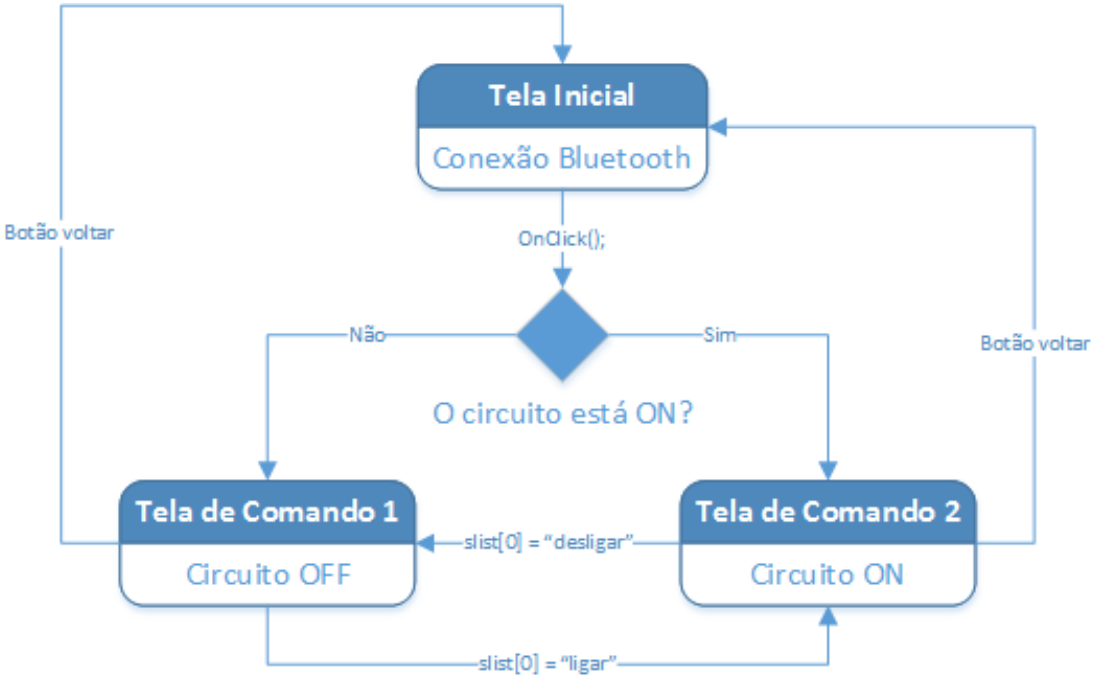

Figura 4.11: Diagrama de Estados do VoxLamp

4.5 Outras Possibilidades de Utilização dos Aplicativos e dos Conceitos Envolvidos

Em seu artigo "*Amarino: a toolkit for the rapid prototyping of mobile ubiquitous computing.*" de 2013, Kaufmann cita alguns exemplos de aplicação prática do sistema Amarino. Dentre eles, destaca-se o "Call my Shirt" que ativa LEDs em uma camiseta equipada com uma pequena placa Arduino quando o usuário está em uma ligação. Este tipo de vestimenta já pode até mesmo ser comprada online. [5]

As aplicações propostas no capítulo 4 possuem uma gama tão vasta de utilização que mereceram um capítulo a parte para discutí-las. A ideia aqui é mostrar o tamanho do impacto que estes simples apps podem ter no cotidiano, levando-se em conta todos os seus aspectos.

Com o poder de controlar LEDs via *Bluetooth*, diversos sinais luminosos comuns poderiam ter uma vida útil aumentada, bem como uma utilização mais inteligente. Um semáforo pode ser controlado através da voz de um agente de trânsito que observa o tráfego em uma rodovia movimentada. Artistas podem controlar as luzes de um show sem precisar de um funcionário externo. Quando uma pessoa precisar de uma luz de emergência, como a da Figura 4.12, em casa ou dentro de seu carro, basta falar ao celular para ligá-las.

Acionar circuitos elétricos pela voz através de um relé possibilita o uso de qualquer aparelho elétrico à distância. Como o *Bluetooth* aqui utilizado tem um alcance de aproximadamente 10 m, seria possível ligar um eletrodoméstico, como um liquidificador, antes mesmo de entrar na cozinha. Bastaria reclamar que uma sala está muito fria para o ar condicionado desligar. Uma pessoa com deficiência física poderia trancar a porta de casa com uma fechadura eletrônica sem precisar da

Figura 4.12: Luzes de Emergência de LEDs de 5V

ajuda de terceiros. São muitas as aplicações. Os conceitos por trás da ideia são simples, no entanto. Sempre com o objetivo de facilitar a vida cotidianda do homem moderno.

No próximo capítulo serão apresentadas as considerações finais deste trabalho, juntamente com um pequeno resumo dos objetivos que estas aplicações pretendem cumprir e seus princípios relacionados.

Capítulo 5

Consideraões Finais

Conceitos como o da computação ubíqua auxiliam no desenvolvimento de soluções intuitivas de forma a diminuir a distância entre os usuários e a tecnologia permitindo que cada vez mais pessoas possam usufruir dos benefícios da inovação sem excluir aqueles que não possuem conhecimentos computacionais profundos. [18]

O sistema *Android* segue essa linha ao possibilitar ao usuário a utilização de diversos aplicativos em seu dia-a-dia de forma bastante acessível e prática, auxiliando-os em diversas atividades. Além disso, por ser um sistema aberto e muito difundido, não faltam materiais de boa qualidade para aqueles interessados em apresentar suas próprias soluções em forma de aplicativos. E para aqueles que preferem deixar o desenvolvimento para os especialistas, basta uma passagem rápida pela loja de aplicativos do *Android*, a *Google Play Store*, para encontrar diversas opções de aplicativos. No entanto, apps que criam soluções em automação residencial, por exemplo, são bastante limitadas, especialmente em língua portuguesa.

5.1 Assistentes de Voz

Com base nos aplicativos mostrados no Capítulo 2 (Vox Calc e Vox Hub), pode-se ver que a interação entre o usuário e o dispositivo *Android* através de comandos de voz abre inúmeras possibilidades para novos aplicativos criando uma nova forma de controlar um aparelho celular.

Há três assistentes de voz bastante conhecidos no mercado: Siri, para o sistema móvel iOS , *S Voice* nos aparelhos *Android* da Samsung e o *Google Now*, que vem integrado no aplicativo de buscas nas versões do *Android* superiores a 4.0. Eles possuem várias funções ativadas por meio da voz e cada uma possui suas vantagens e desvantagens. Porém, todos possuem uma desvantagem em comum: o reconhecimento de voz não fica ativo permanentemente de forma que para ativa-lo é necessário que o usuário interaja com o dispositivo pressionando algum botão na tela ou realizando algum gesto. Dessa forma, já que é preciso, por assim dizer, usar as mãos, para ativar o serviço de voz, às vezes é mais prático e rápido utilizar as funções do modo tradicional diminuindo assim a utilização do reconhecimento de voz.

Essa limitação muitas vezes está relacionada ao *hardware* dos dispositivos. Deixar o reconhecimento de voz constantemente ativo pode gastar rapidamente a bateria do celular além de aumentar consideravelmente o consumo da rede de dados, pois o processamento dos dados do serviço de voz normalmente é feito em servidores de forma que é necessária uma conexão de *internet* ativa. Embora já seja possível no *Android* baixar para o dispositivo versões de um idioma para o reconhecimento de voz *off-line*, o reconhecimento *online* ainda produz resultados melhores e mais precisos.

Essa situação pode mudar ao passo que os fabricantes passem produzir aparelhos celulares com *hardware* dedicado ao reconhecimento de voz. O primeiro exemplo disso é o smartphone lançando em agosto de 2013 pela Motorola, o *Moto X*. Criado tendo em mente uma interação maior do usuário através da voz ele possui uma arquitetura diferenciada de outros aparelhos, além do processador principal de dois núcleos e a GPU de quadro núcleos, ele possui ainda dois outros processadores (de um núcleo cada) de baixíssimo consumo para manter funções de voz e gestos ativos o tempo todo. Além disso, o serviço de voz já é completamente integrado com o assistente de voz do Google, o *Google Now*, de forma que é possível até mesmo desbloquear a tela do aparelho por comando de voz dizendo "*Ok Google Now*".

Ainda não se sabe se arquiteturas de processamento dedicadas à voz como a do Motorola *Moto X* irão se popularizar e se serão adotadas por outras fabricantes. Mas, já é o primeiro passo para tornar a utilização de voz cada vez melhor e para facilitar ainda mais a utilização da tecnologia.

Novos avanços tecnológicos trazem mais opções de acessibilidade. A automação em diversos setores possibilita a realização de diversas atividades, tanto por pessoas que possuam algum tipo de deficiência como por pessoas que buscam alguma forma de comodidade e praticidade.

Porém, muitas vezes as tecnologias de acessibilidade e automação não foram feitas de forma que possam se conectar ou trocar informações entre si. Nesses casos, cada tecnologia possui uma interface diferente, formas variadas de operações entre outros fatores limitantes.

5.2 *Arduino* e *Android* Integrados para Simplificar

Dispositivos como *Android* e *Arduino* surgem então como formas de unificar tecnologias e permitir a operação simplificada e acessível de diversos equipamentos, outros dispositivos, e outras tecnologias gerenciadas, muitas vezes, por apenas um aplicativo.

O *Arduino*, com o microcontrolador ATMega, é incrivelmente versátil. Sendo utilizado para controlar sistemas e circuitos elétricos em diversos tipos de solução e protótipos ao redor do mundo. Já o *Android*, com sua facilidade de interconexão com outros sitemas, também apresenta grande versatilidade, até mesmo quando se deseja utilizar o aparelho celular ou *tablet* para controlar e enviar comandos a dispositivos externos que não necessariamente realizam funções de telefone ou câmera fotográfica. Os sistemas mostrados no Capítulo 4 dão uma boa ideia do papel do *Android* o do *Arduino* para simplificar tarefas através de comandos de voz e de uma conexão simples.

Os princípios demostrados nesses sistemas são simples e podem ser facilmente implementados

ou adaptados para situações diversas, como foi abordado, também, no Capítulo 4. Os materiais são acessíveis e de custo razoável, incluindo o sistema *Android* que está disponível em aparelhos de baixo custo enquanto que uma placa *Arduino* pode ser facilmente encontrada em sites de eletrônica confiáveis. Além disso, a conexão entre *Android* e *Arduino* via *Bluetooth* é incrivelmente facilitada pelo Amarino, conforme descrito no capítulo 3.

5.3 Aplicação dos Conceitos de Computação Ubíqua

> "Não estamos na era da informação. Não estamos na era da Internet. Nós estamos na era das conexões. Ser conectado está no cerne da nossa democracia e nossa economia. Quanto maior e melhor forem essas conexões, mais forte serão nossos governos, negócios, ciência, cultura, educação..."
> *David Weinberger*

Conforme explicado ao longo deste trabalho, a proposta era valer-se de conceitos básicos da computação ubíqua para gerar uma solução de automação simples, acessível e confiável. Por meio das aplicações desenvolvidas, VoxLed e VoxLamp, um usuário tem a possibilidade de controlar, através da voz em língua portuguesa, sinais luminosos de LEDs e circuitos elétricos acionados por relé. O leitor pode encontrar todos os códigos dos aplicativos desenvolvidos, bem como vídeos de testes com o *Arduino* conectado a um celular *Android*, no link:
https://www.dropbox.com/sh/knzpf2nruc22gc6/dk8Vyi7Pa8

Ao descrever o Amarino, Kaufmann explica que os conceitos de computação ubíqua sempre foram levados em conta em seu trabalho, talvez por isso o Amarino seja tão voltado a conexão sem fio e facilidade de interação com o usuário. A utilização do sistema como um serviço em *background* é essencial para o funcionamento das aplicações aqui propostas. [6]

Tornar a computação onipresente, como escreveu Weiser, está se tornando uma realidade. Em uma residência comum por exemplo, em que uma pessoa interage o tempo inteiro com interruptores, botões, cliques, aparelhos elétricos e eletrônicos, é uma excelente ideia tornar algumas destas tarefas realizáveis apenas com a voz. Daí os conceitos tão citados neste trabalho e o motivo de serem a fundação dos aplicativos propostos. [7]

REFERÊNCIAS BIBLIOGRÁFICAS

[1] LECHETA, R. R. Google Android: aprenda a criar aplicações para dispositivos móveis com o Android SDK. São Paulo: Novatec, 2010. 2ł Edição. 608 p.

[2] DARREL ETHERINGTON. Android Nears 80% Market Share In Global Smartphone Shipments, As iOS And BlackBerry Share Slides, Per IDC. Disponível em: http://techcrunch.com/2013/08/07/Android-nears-80-market-share-in-global-smartphone-shipments-as-ios-and-blackberry-share-slides-per-idc/ .Acesso em: 06 de fev de 2014.

[3] BONIFAZ KAUFMANN. Amarino: Android Meets Arduino. Disponível em: http://www.amarino-toolkit.net/ .Acesso em: 06 de fev de 2014

[4] ROBERTO DI RENNA, RODRIGO BRASIL, THIAGO CUNHA, MATHYAN BEPPU, ERIKA FONSECA, ALEXANDRE VEGA. Introdução ao kit de desenvolvimento Arduino. Disponível em: http://www.telecom.uff.br/pet/petws/downloads/tutoriais/arduino/Tut_Arduino.pdf Acesso em 06 de fev de 2014.

[5] Bonifaz Kaufmann and Leah Buechley. (2010). Amarino: a toolkit for the rapid prototyping of mobile ubiquitous computing. In Proceedings of the 12th international conference on Human computer interaction with mobile devices and services (MobileHCI '10). ACM, New York, NY, USA, 291-298.

[6] Bonifaz Kaufmann. (2010). Design and implementation of a toolkit for the rapid prototyping of mobile ubiquitous computing. Master's thesis. Alpen-Adria-Universität Klagenfurt, Klagenfurt, Austria.

[7] Mark Weiser. (1991) The Computer for the 21st Century. Scientific American. Disponível em: http://wiki.daimi.au.dk/pca/_files/weiser-orig.pdf Acesso em: 06 de fev de 2014.

[8] TONY BRADLEY. Android dominates market share but Apple makes all the money. Disponível em: http://www.forbes.com/sites/tonybradley/2013/11/15/android-dominates-market-share-but-apple-makes-all-the-money/ Acesso em: 06 de fev de 2014.

[9] ARDUINO, Arduino. Disponível em: http://www.arduino.cc Acesso em 06 de fev de 2014.

[10] ATMEL CORPORATION. Atmega328P Datasheet. Disponível em: http://www.atmel.com/devices/atmega328p.aspx Acesso em 06 de fev de 2014.

[11] ARDUINO E CIA. Ligando uma Lâmpada com relé. Disponível em: http://www.arduinoecia.com.br/2013/05/ligando-uma-lampada-com-rele.html Acesso em 06 de fev

de 2014.

[12] SAGAR SAPKOTA. MultiColorLamp using Amarino, Android and Arduino. Disponível em:
http://www.buildcircuit.com/multi-color-lamp-using-amarino-Android-and-arduino/ Acesso em 06 de fev de 2014.

[13] SUE SMITH. Android SDK: Build a Speak and Repeat App. Disponível em:
http://mobile.tutsplus.com/tutorials/android/android-sdk-build-a-speak-and-repeat-app/ Acesso em: 06 de fev de 2014.

[14] FABIO CIMINO. Arduino acionar portas ou equipamentos 12V. Disponível em:
http://labdegaragem.com/forum/topics/arduino-acionar-portas-ou-equipamentos-12v?xg_source=activity Acesso em: 06 de fev de 2014.

[15] GOOGLE. Google Developer for Android. Disponível em:
http://developer.android.com/develop/index.html Acesso em: 06 de fev de 2014.

[16] GOOGLE. Fundamentals and Components for Android Development. Disponível em: http://developer.android.com/guide/components/fundamentals.html Acesso em: 06 de fev de 2014.

[17] GOOGLE: SpeechRecognizer Library Reference and Documentation. Disponível em:
http://developer.android.com/reference/android/speech/SpeechRecognizer.html Acesso em: 06 de fev de 2014.

[18] ANDRE LEMOS. Cibercultura e Mobilidade, a era da conexão. Disponívem em:
https://www.razonypalabra.org.mx/anteriores/n41/alemos.html Acesso em 06 de fev de 2014.

[19] PASQUALE STIRPARO, IAN LÖSCHNER. Scientific Research. Secure Bluetooth for Trusted m-Commerce. Disponível em: http://file.scirp.org/Html/1-9701762_33221.htm Acesso em: 06 de fev de 2014.

[20] BLUETOOTH.ORG. Bluetooth Techinical Descripition. Disponível em:
https://www.bluetooth.org/en-us/specification/adopted-specifications Acesso em: 06 de fev de 2014.