**Título do Trabalho**

**Bruno Lima Cardoso**

Projeto Final de Curso submetido ao Departamento de Ciência da Computação do Instituto de Matemática da Universidade Federal do Rio de Janeiro como parte dos requisitos necessários para obtenção do grau de Bacharel em Informática.

Apresentado por: Bruno Lima Cardoso

---

Autor 1

Aprovado por:

---

Prof. Nome do Professor Orientador
(Presidente)

---

Prof. Nome do Professor 1

---

Prof. Nome do Professor 2

RIO DE JANEIRO, RJ - BRASIL
MAIO DE 2011

# RESUMO

Título do Trabalho

Autor(es) do Trabalho

Orientador: Nome do Orientador

Texto que apresenta um resumo do trabalho dando uma descrição geral.

# ABSTRACT

Project Title

Author(s) of Project

Supervisor: Supervisor´s Name

Text that presents an abstract of the work giving a general description.

# Siglas

HMM - Hidden Markov Models

ASR – Automatic Speech Recognition

CSR – Continuous Speech Recognition

MFCC – Mel-Frequency Cepstral Coefficients

CMN – Cepstral Mean Normalization

FDP – Função Densidade de Probabilidade

HTTP - Hypertext Transfer Protocol

Web - World Wide Web

REST - Representational State Transfer

XML - Extensible Markup Language

YAML - Yet Another Markup Language

JSON - Javascript Object Notation

SOAP - Simple Object Access Protocol

RASSF - Rede de Atuadores e Sensores Sem Fio

IP - Internet Protocol

WoT - Web of Things

PCD - Pessoa Com Deficiência

PSD - Pessoa Sem Deficiência

# Lista de Variáveis

t - unidade de tempo

$x_t$ - vetor de parâmetros calculado a partir de um segmento de fala

$c_t$- vetor de MFCCs calculado a partir de um segmento de fala

$c_t$ - vetor de MFFCs delta de primeira ordem computados a partir dos coeficientes $c_t$

$\Delta\Delta c_t$ - vetor de MFFCs delta de segunda ordem computados a partir dos coeficientes $\Delta c_t$

$\Theta$ - tamanho da janela usada para cálculo dos coeficientes delta de primeira e segunda ordens

M - um modelo oculto de Markov

X - sequência de vetores acústicos

$A = a_{ij}$ - matriz de todas as probabilidades de transição entre um estado i e outro j

$B = b_i$ - matriz de todas as probabilidades de emissão de saída em um determinado estado i

$\prod = \prod_i$ - matriz com as probabilidades de um modelo ser iniciado a partir de um estado i

S - sequência de estados

W - sequência de palavras

# Sumário

**1 Introdução** **9**

1.1 Objetivos . . . . . 10

1.2 Organização do Trabalho . . . . . 11

**2 Fundamentos** **12**

2.1 *Web of Things* - WoT . . . . . 13

2.1.1 Nós como Recursos RESTful . . . . . 13

2.2 Android . . . . . 16

2.3 Sistema para Reconhecimento de Palavras Isoladas . . . . . 19

2.3.1 Extração de Parâmetros do Sinal . . . . . 20

2.3.2 Modelagem Acústica . . . . . 22

2.3.3 Modelo de Linguagem . . . . . 24

2.3.4 Decodificação . . . . . 25

2.3.5 Ferramenta Sphinx . . . . . 27

2.3.6 Sphinx-2 . . . . . 28

2.3.7 Sphinx-3 . . . . . 28

2.3.8 Sphinx-4 . . . . . 28

2.3.9 Pocket Sphinx . . . . . 29

2.4 Arduino . . . . . 29

**3 Funcionamento do Sistema** **32**

3.1 Rede de Atuadores e Sensores Sem Fio (RASSF) . . . . . 33

3.2 Smart Gateway (SG) . . . . . 36

3.2.1 Camada de Apresentação . . . . . 38

3.2.2 Camada de Controle . . . . . 38

3.2.3 Camada de Abstração . . . . . 39

3.3 Aplicação . . . . . 40

3.3.1 Sistema de Reconhecimento de Fala para Android . . . 41

3.3.2 Cliente REST para Android . . . . . 42

**4 Trabalhos Correlatos** **45**

**5 Conclusão e Trabalhos Futuros** **47**

**Referências Bibliográficas** **48**

**A Configuração Galax 5** **51**

# Lista de Figuras


2.1 Integração direta dos dispositivos do mundo real através do IP. 15
2.2 Arquitetura de integração utilizando *Smart Gateway*. . . . . 15
2.3 Diagrama das etapas envolvidas em um sistema CSR. . . . . 20
2.4 Diagrama do processo de extração de parâmetros de um sinal de voz. . . . . 22
2.5 Exemplo de HMM com três estados emissores . . . . . 23
2.6 Arduino UNO utilizado neste trabalho. . . . . 30

3.1 Arquitetura (a) Autônoma e (b) Semi-Autônoma. . . . . 34
3.2 Componentes dos (a) sensores e (b) atuadores. . . . . 34
3.3 Visualização em alto nível de um *Smart Gateway*. . . . . 37
3.4 Arquitetura REST sugerida pela Google 10. . . . . 43

# Lista de Tabelas

A.1 Especificações técnicas do *Samsung Galaxy 5*. . . . . 51

# Capítulo 1

# Introdução

Hoje em dia, a informática como um todo, tem estado cada vez mais presente no cotidiano das pessoas. Isto se dá pela baratização dos diversos dispositivos eletrônicos e computacionais assim como as facilidades proporcionadas pelos mesmo. Tais facilitações, além de promover comodidade e praticidade ao público geral, se bem empregados, propiciam uma melhor qualidade de vida e independência funcional a portadores de necessidades especiais (Pessoas com Deficiência – PCD). Como por exemplo, leitores de tela para cegos, sistemas que permitem a integração de tetraplégicos com computadores (Motrix) e muitos outros.

O objetivo do sistema proposto neste trabalho visa promover independência funcional à portadores de lesão severas, como tetraplégicos, portadores de ELA (Esclerose Lateral Amiotrófica) e afins, quanto a utilização de determinados aparelhos. A interação dos artefatos digitais com o mundo físico é crucial para tal propósito. Em particular, a *Internet of Things*[21] tem explorado o desenvolvimento de aplicativos para dispositivos, os quais passam a ser chamados de *smart things*, com o intuito de integrá-los ao mundo real (internet). São eles, sensores e atuadores de redes, dispositivos embarcados,

rádios, televisores, aparelhos de DVD, enfim, objetos que possuam reforço digital. Tais dispositivos estão, em sua grande maioria, desconectado da rede formando assim diversas pequenas ilhas de incompatibilidade e, apesar de alguns possuírem acesso a internet, o fato de não poderem ser controlados ou monitorados com a presença de programas não licenciados tornam ainda mais difícil a integração com outros dispositivos.

A *Web of Things* [3] é posto como um refinamento da *Internet of Things* que tem como objetivo, não apenas posicionar *smart things* na internet, mas dentro da web (na camada de aplicação), desta forma os dispositivos tornam-se parte de um ambiente em que cada um fornece múltiplas aplicações/serviços diversificados à outros diferentes em hardware e software igualmente integrados. Desta forma, torna-se de extrema relevância o desenvolvimento de um mecanismo de comunicação independente de plataforma de modo a promover interoperabilidade entre os mesmos.

A seguir serão apresentados os objetivos deste trabalho assim como sua organização.

## 1.1 Objetivos

O objetivo central deste trabalho é propor um sistema que confira independência funcional à PCD's fazendo uso da arquitetura *Web of Things* para o desenvolvimento de um ambiente inteligente - *Smart Building*. Seguindo esses preceitos e sem perdas de generalidade, objetivou-se desenvolver uma estrutura que permita ao usuário controlar um aparelho televisor utilizando apenas a sua voz. Para tal foi necessário:

- A implementação de um sistema de reconhecimento da fala para um *Smartphone Android 2.1* para o reconhecimento dos comandos feitos

pelo usuário.

- A implementação de um cliente HTTP em um *Smartphone Android 2.1* utilizado para fazer as requisição dos recursos disponíveis.[1]
- A implementação de um *Smart Gateway (SG)* responsável por disponibilizar os recursos existentes através da Web.
- A implementação do atuador responsável por executar as requisições feitas pelo usuário.

## 1.2 Organização do Trabalho

Este trabalho esta organizado da seguinte maneira: o Capítulo 2 apresenta todos os fundamentos necessários para o desenvolvimento de cada estrutura integrante de um *Smart Building* com arquitetura *Web of Things*, além dos fundamentos que envolvem um sistema de reconhecimento de fala, incluindo as etapas de treinamento dos modelos - acústicos e linguísticos - e a etapa de decodificação. O Capítulo 3 detalha a implementação de cada componente desenvolvido. O Capítulo 4 apresenta os trabalhos que serviram de base e inspiração para este projeto. E por fim, o Capítulo 5 conclui o projeto explicitado os resultados alcançados e propondo modificações futuras e possíveis expansões.

[1] Recursos disponíveis - referem-se as ações que podem ser executadas pelos atuadores disponíveis

# Capítulo 2

# Fundamentos

Neste capítulo serão apresentados os principais fundamentos que envolvem a implementação de um *Smart Building* que possui *Web of Things* como arquitetura base, assim como uma breve descrição das ferramentas utilizadas, como forma de facilitar o entendimento dos próximos capítulos.

Na Seção 2.1, o conceito de *Web of Things* será brevemente apresentado e as estruturas possíveis para implantação do mesmo.

Nas Seções 2.2 e 2.3, os fundamentos referentes a camada de interação com o usuário serão vistos e distribuídos da seguinte forma:

- Na Seção 2.2, o sistema operacional Android será apresentado de modo à justificar sua escolha, ressaltando pontos positivos e negativos que foram ponderados.

- Na Seção 2.3, as etapas que envolvem um Sistema de Reconhecimento de Palavras Isoladas são brevemente descritas e apresentadas em forma de diagrama. Nas subseções que seguem, os conceitos que envolvem cada etapa do sistema são apresentados de forma mais detalhada.

E, por fim, na Seção 2.4, o microcontrolador Arduino será visto, pois este

foi utilizado na implementação da interface entre o sistema e o mundo físico.

## 2.1 *Web of Things* - WoT

A realização da *Web of Things* requer a extensão da Web existente para que os objetos do mundo real e dispositivos embarcados possam interagir com ela. Entretanto, muitos destes dispositivos, apesar de possuírem microcontroladores, não conseguem conectar-se à Web, seja por limitações de *hardware* (ex.: não possui uma placa Ethernet) ou de *software*. Para contornar este problema utilizam-se sensores e atuadores específicos para cada aparelho, os quais suprem as deficiências já citadas.

Ao invés de utilizar os protocolos Web apenas para transporte de dados, comumente utilizado em *Web Services*, pretende-se fazer os dispositivos uma parte integral da mesma, usando o protocolo HTTP na camada de aplicação. Desta forma, as funcionalidades dos dispositivos embarcados do mundo real são disponíveis através da *API RESTful* [7] sobre HTTP, como apresentado a seguir.

### 2.1.1 Nós como Recursos RESTful

O termo REST se referia, originalmente, a um conjunto de princípios de arquitetura. Na atualidade se usa no sentido mais amplo para descrever qualquer interface web simples que utiliza XML e HTTP(ou YAML, JSON, ou texto puro), sem as abstrações adicionais dos protocolos baseados em padrões de trocas de mensagem, como o protocolo de serviços web SOAP.

Em particular, REST usa a Web como uma plataforma de aplicação e, desta forma utiliza todos os recursos agregados ao protocolo HTTP como autenticação, encriptação, compressão, e cache. E assim, recursos podem ser

acessados e seus resultados são visíveis por qualquer aplicação que possua um cliente HTTP (navegadores Web, por exemplo) sem a necessidade de gerar códigos com auto grau de complexidade para interagir com o serviço.

Para tanto, REST possui duas regras básicas:

- Todo modelo de aplicação deixa de centrar-se em operações e passa a centrar-se em dados, i.e., tudo que oferece serviço torna-se uma fonte que pode ser identificada, sem ambiguidade, por uma URI.

- As quatro principais operações provenientes do HTTP (GET, POST, PUT, DELETE) são as únicas operações possíveis nas fontes, definindo assim, uma interface uniforme conhecida e difundida.

Tradicionalmente, REST é usado para integrar sistemas Web, como dito anteriormente. Entretanto, seu aspecto leve o faz um candidato ideal para dispositivos embarcados proverem serviços, sabendo que estes são limitados em recursos.

Apesar de tudo, serviços REST também possuem certas limitações. Sua inerente simplicidade, paradoxalmente implica em uma grande dificuldade de desenvolvimento de serviços complexos.

O trabalho [3] apresente duas possíveis abordagens para construir um sistema funcional com *Web of Things*. No primeiro, dispositivos fazem parte diretamente da Web possuindo um servidor HTTP cada e, desta forma, "construindo uma nuvem de serviços, como mostrado na Figura 2.1. No segundo os dispositivos conectam-se à Web através de um *Smart Gateway*, o qual traduz as requisições sobre os protocolos, com pode ser visto na Figura 2.2

Para este trabalho, foi escolhido o segundo modelo, pois o microcontrolador utilizado no protótipo não possui nenhum *HTTP Server* já implementado ou qualquer biblioteca disponível, portanto, para o primeiro modelo,

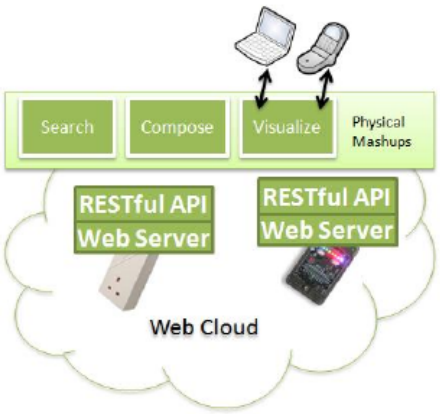
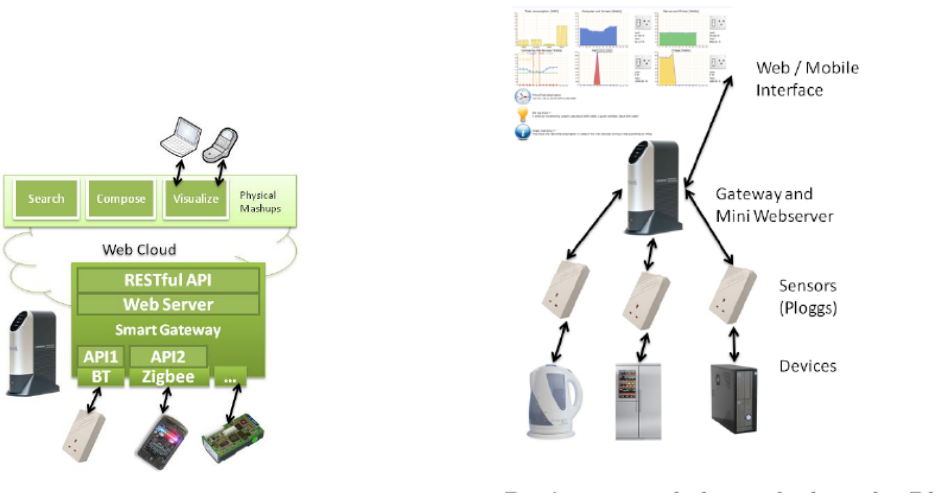



Figura 2.1: Integração direta dos dispositivos do mundo real através do IP.




(a) (b)

Figura 2.2: Arquitetura de integração utilizando *Smart Gateway*.

haveria a necessidade de desenvolver tal estrutura o que não é o propósito deste trabalho.

Por fins didáticos, a estrutura escolhida será dividida em três níveis:

- **Rede de Atuadores e Sensores Sem Fio (RASSF)** - constitui-se dos dispositivos que interagem com o ambiente e aparelhos de modo a prover serviços. Os papéis dos sensores e atuadores é coletar dados do ambiente e executar apropriadas ações baseadas nos mesmos e/ou nas requisições das camadas a cima. No protótipo deste trabalho tais dispositivos serão integrados utilizado um microcontrolador chamado Arduino, como dito no início deste capítulo;

- ***Smart Gateway* (SG)** - constitui-se da estrutura responsável por permitir o acesso dos serviços providos pela RASSF através da Web. O seu objetivo central é abstrair os protocolos de comunicação proprietários ou APIs dos dispositivos e prover acesso à suas funcionalidades via RESTful;

- **Aplicação** - constitui-se da parte visível ao usuário final onde este fará suas requisições. A aplicação é o componente responsável por emitir um conjunto de consultas ou interesses, que descrevem as características dos fenômenos físicos que o usuário deseja analisar. Como este trabalho prima por acessibilidade, nesta camada foi desenvolvido um sistema de reconhecimento de palavras isoladas para o português brasileiro no Sistema Operacional Android na versão 2.1 de sua API, formando um estrutura *command-control* acessível aos usuários.

## 2.2 Android

Todo o acionamento do sistema é feito através de comandos que são falados num microfone.

*"The first truly open and comprehensive platform for mobile devices, all the software to run a mobile phone but without the proprietary obstacle that have bindered mobile innovation."*

*Andy Rubin*[11]

Em dias anteriores ao Twitter e Facebook, quando a Google não passava de um vislumbre nos olhos de seus fundadores, telefones móveis eram apenas isto - aparelhos portáteis tão pequenos que cabiam em pastas e munidos de baterias que podiam durar várias horas. E alem disso ofereciam a liberdade de fazer ligações sem estarem fisicamente conectados a uma linha.

Incrivelmente pequenos, estilosos e dotados de funcionalidades os telefones móveis hoje são indispensáveis para vida cotidiana. Os avanços em *hardware* os fizeram menores e mais eficientes enquanto eram inclusos inúmeros periféricos.

Apos as câmeras e os tocadores de música, estes possuem GPS, acelerômetros e telas *touch screens*. Alem disso, estas inovações em *hardware* proveu um campo fértil para o desenvolvimento em *software*.

O *Android* é uma nova estrutura de sistema operacional desenvolvido para dispositivos móveis com grande poder de *hardware*. É originário de um grupo de companhias conhecido como *Open Handset Allience - OHA* liderada pela *Google*. Hoje muitas companhias, tanto membros originais da OHA e outras, tem investido intensamente no *Android*, de modo à alocar significativos recursos de engenharia para aperfeiçoar o sistema operacional e trazer dispositivos que o portam ao mercado.

*Windows Mobile*, o *iPhone* da *Apple* e o *Palm Pre* proveem um rico e simples ambiente de desenvolvimento para aplicações móveis. Entretanto, diferente da *Android*, estes são construídos em sistemas operacionais proprietários que, em certos casos priorizam aplicações nativas em detrimento das

feitas por terceiros, assim como, em certos casos, restringem e/ou controlam a veiculação das mesmas, alem de se limitarem a comunicação entre aplicações e dados nativas.

Em oposição as estruturas supracitadas, o *Android* oferece uma nova possibilidade para aplicações móveis disponibilizando um ambiente de desenvolvimento baseado no *kernel Linux* o qual é *open-source*. O acesso à *hardware* é permitido à todas aplicações através de uma série da bibliotecas e interações entre as aplicações nativas. Alem disso, todas as aplicações possuem o mesmo grau de importância, sejam elas nativas ou de terceiros, pois ambos os grupos são desenvolvidos com a mesma API, são executadas no mesmo tempo de execução e o usuário pode remover/substituir qualquer aplicação nativa, até mesmo o discador ou a interface do painel principal.

Outro ponto que merece ser ressaltado no Android é a existência do Android NDK, o qual é uma ferramenta que possibilita embarcar componentes que fazem uso de códigos não nativos em aplicações *Android*. Desta forma, é permitido o desenvolvimento de partes desta aplicação usando linguagens como C e C++, provendo assim benefícios à mesma como o reuso de existentes códigos ou, em certos casos, aumentar sua velocidade.

O NDK possui:

- Um conjunto de ferramentas e arquivos de construção usados para gerar bibliotecas de códigos em C e C++;
- Uma forma de embutir a biblioteca nativa em uma aplicação que pode ser executado em um dispositivo com Android.
- Um conjunto de sistemas nativos de cabeçalhos e bibliotecas que são suportadas em todas as futuras versões da plataforma Android, a partir da Android 1.5.

- Documentação, exemplos e tutoriais.

O NDK foi de extrema relevância para este projeto, pois o *framework* utilizado para implantar o sistema de reconhecimento de voz na plataforma Android foi desenvolvido em linguagem C. Portanto, o NDK permitiu a criação de uma biblioteca que disponibilizava funções referentes ao reconhecimento da fala.

## 2.3 Sistema para Reconhecimento de Palavras Isoladas

A implementação de um Sistema de Reconhecimento de Fala pode ser vista como uma tarefa de aprendizado de máquina e pode ser dividida basicamente em duas etapas: treinamento e teste. Previamente à etapa de treinamento é realizada a extração de parâmetros do sinal, ou seja, o áudio presente na base é transformado em coeficientes que servirão de entrada para a etapa de treinamento.

Na etapa de treinamento são realizados o treinamento do modelo acústico e do modelo de linguagem. O primeiro tem como propósito calcular a verossimilhança de uma sequência de vetores acústicos dado um modelo, enquanto o segundo tem por objetivo mapear os relacionamentos entre palavras, estimando a probabilidade de ocorrência de uma palavra em função das anteriores.

Entretanto, para este trabalho será desenvolvido um sistema de reconhecimento para palavras isoladas que possui uma abordagem um pouco diferente, o que será descrito mais a frente. Já a fase de testes, compreende o reconhecimento em si, também conhecido como decodificação. Esta fase representa não apenas um problema de reconhecimento de padrões como um problema

de busca em um grafo, tendo por objetivo buscar a sequência de palavras que melhor se adapta aos vetores acústicos de entrada, dados os modelos. Então, realiza-se uma etapa de análise dos resultados, retirando-se então as medidas de interesse. O diagrama apresentado na Figura 2.3 ilustra, de forma resumida, as etapas mencionadas.

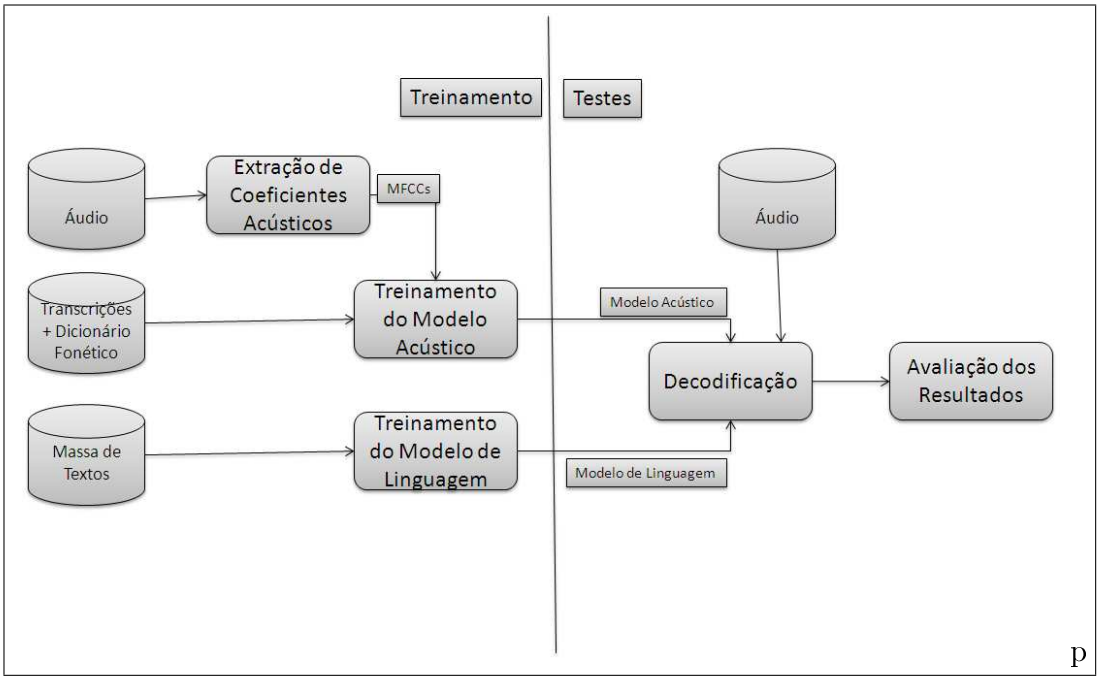



Figura 2.3: Diagrama das etapas envolvidas em um sistema CSR.

### 2.3.1 Extração de Parâmetros do Sinal

Para que seja possível processar o sinal de fala para geração do sistema de reconhecimento, torna-se necessária, em uma primeira etapa, a conversão da onda sonora em um sinal digital, que pode ser compreendido pelo computador. O processamento do sinal de fala consiste na amostragem do sinal, janelamento e extração de parâmetros que serão relevantes para o processo de reconhecimento. A extração dos parâmetros é um dos assuntos mais

importantes na área de reconhecimento de fala.

A função principal desta etapa de extração de parâmetros é a da divisão do sinal em blocos - supondo que o sinal de fala pode ser considerado estacionário durante um período de tempo muito pequeno, de alguns milissegundos - e derivação de uma estimativa suavizada do espectro a partir de cada bloco. Em um configuração considerada padrão, esses blocos - usualmente chamados de janelas - possuem duração de 25ms e são obtidos a cada 10ms (o que é conhecido por *frame shift*). Ainda, o sistema usa blocos sobrepostos de forma a capturar a informação contida nos seus limites.

Após essa fase, é aplicada uma transforma de Fourier a cada bloco, passando os valores obtidos do domínio do tempo para o domínio da frequência. Depois é feita uma filtragem, com o objetivo de extrair os parâmetros que permitam mais facilmente o reconhecimento da fala. A saída é um vetor de valores filtrados, comumente chamados de mel-spectrum [13], cada um correspondendo ao resultado da filtragem do espectro de frequências da entrada por um filtro individual. Então, o comprimento do vetor de saída é igual ao número de filtros criados. As constantes que definem essa filtragem são o número de filtros (cujas frequências centrais estão em uma escala logarítimica, conforme o ouvido humano), a frequência mínima e a frequência máxima. As frequências máxima e mínima dependem da origem do sinal de voz. Para uma fala em um sistema de telefonia, com frequências de corte de 300 e 3700 Hz, o uso de limites fora desses valores significa apenas perda de banda.

Para uma fala clara, a frequência mínima deve estar acima de 100 Hz, para livrar-se de possíveis interferências da corrente alternada (50/60 Hz) e também porque normalmente não informação útil abaixo desta frequência.

A frequência máxima deve ser menor que a frequência de Nyquist, ou seja, menor que metade da frequência de amostragem. Além disso, acima

de 6800 Hz não há muito que possa ser usado para melhorar a separação entre os modelos. Para canais muito ruidosos, uma frequência máxima em torno de 5000 Hz deve ajudar a diminuir o ruído. Davis e Mermelstein [12] mostraram que os coeficientes de frequência Mel-cepstrais apresentam características robustas para um bom reconhecimento de fala.

A Figura 2.4 ilustra de forma resumida o processo de extração de parâmetros do sinal.

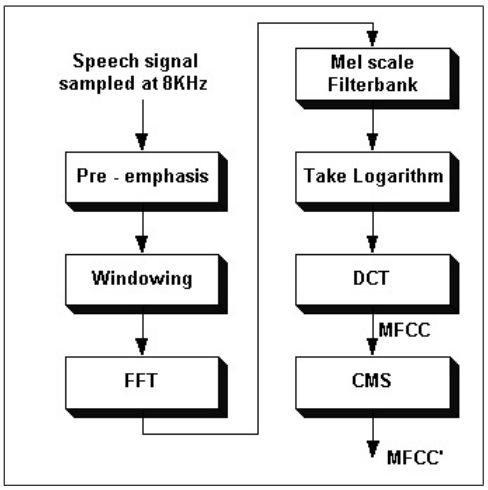



Figura 2.4: Diagrama do processo de extração de parâmetros de um sinal de voz.

### 2.3.2 Modelagem Acústica

A modelagem acústica consiste em um método para calcular a verossimilhança de uma sequência de vetores X, dado um modelo M. Os Modelos Ocultos de Markov ou *Hidden Markov Models (HMMs)* são considerados o estado da arte para modelagem acústica.

Os HMMs podem modelar uma unidade menor da palavra (como os fonemas ou mesmo fonemas inseridos em contextos diferentes, como difones ou

trifones), uma palavra, ou ainda uma frase inteira.

Os HMMs podem ser vistos como máquinas de estado finitas, onde a cada unidade de tempo ocorre uma transição entre estados e cada estado emite um vetor acústico com uma função densidade de probabilidade associada. Um modelo M pode ser escrito como na Equação 2.1.

$$M = A, B, \Pi = a_{ij}, bi, \pi_i, i, j = 1, ..., N \tag{2.1}$$

A Figura 2.5 representa um HMM com três estados emissores, onde são adicionados dois estados não emissores no início e no fim do modelo para fins de facilitar a união entre modelos. Na figura, $x_t$ representa um vetor acústico emitido em uma unidade de tempo $t$. Nesta figura, a probabilidade de emissão de um vetor acústico por um estado é representada por uma mistura de gaussianas.

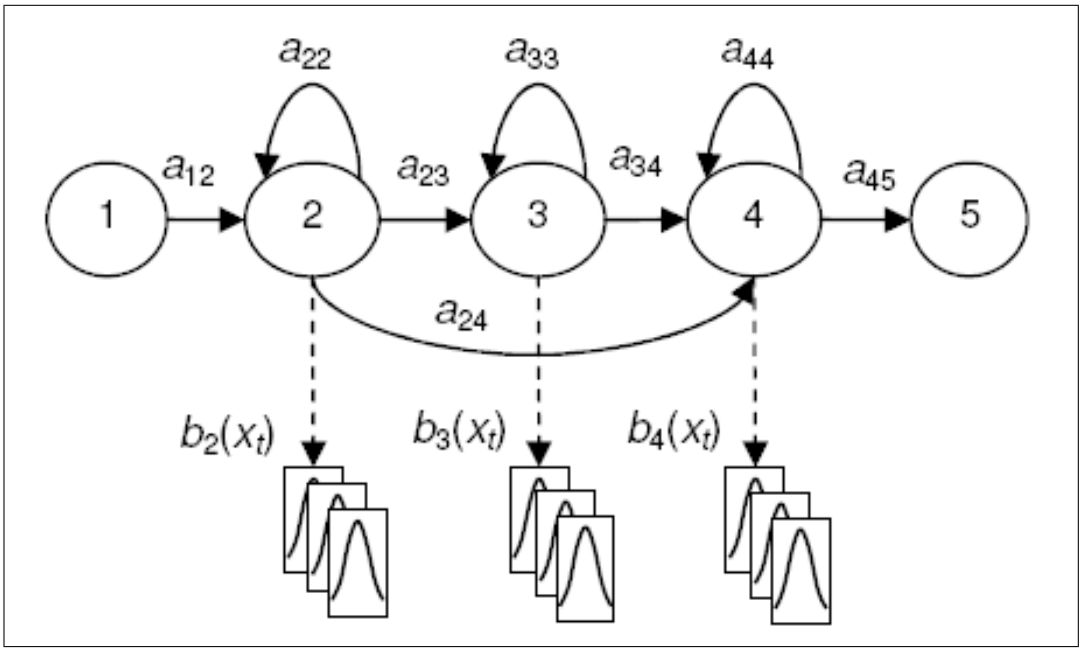



Figura 2.5: Exemplo de HMM com três estados emissores

O objetivo da modelagem é encontrar o ajuste dos parâmetros do modelo

que maximize a verossimilhança. Isto é feito através de um algoritmo de reestimação de parâmetros. Dentre eles, o mais usado é o algoritmo de Baum-Welch, também conhecido como *Forward-Backward Algorithm* ou algoritmo de avanço-retorno. Além de todos os parâmetros presentes na mistura de gaussianas associada a cada estado, como médias e variâncias, as matrizes de transição entre estados também são consideradas parâmetros do modelo.

Ainda, de modo a otimizar esses modelos, pode-se utilizar uma técnica de compartilhamento de estados na ocasião do treinamento de fonemas inseridos em contextos diferentes, como no caso dos trifones. Esta técnica pressupõe que alguns fonemas inseridos em diferentes contextos podem não produzir uma variabilidade acústica suficientemente grande para que sejam modelados por HMMs diferentes. Assim, esses trifones podem ser agrupados dentro de um mesmo modelo, através de uma técnica de agrupamento de estados. Geralmente, utilizam-se algoritmos de árvores de decisão. Através dessa técnica, constrói-se uma árvore para cada fonema, onde o nó pai corresponde a todos os estados com todos os diferentes contextos em que aquele fonema pode ser inserido (no caso de trifones, contexto à direita e contexto à esquerda), ou seja, mesmo fonema central. Os arcos dessa árvore correspondem a perguntas fonéticas que servirão para agrupar estados dentro de categorias diferentes e, por fim, os estados que caírem dentro de um mesmo nó folha serão agrupados como um único estado.

### 2.3.3 Modelo de Linguagem

O modelo de linguagem provê uma estrutura que define uma inter-relação entre as palavras. De forma geral, estas estruturas são classificadas entre duas categorias: gramáticas de grafos dirigidos ou modelos estocásticos N-Gram. As gramáticas de grafos dirigidos representam um grafo dirigido de

palavras onde cada palavra encontra-se em um vértice e cada arco representa a probabilidade de transição entre estas duas palavras (vértices). Já o modelo estocástico gera probabilidades para as palavras baseado na observação das n-1 palavras anteriores (ver [14]). Como falado anteriormente, neste trabalho será utilizado um sistema de reconhecimento de palavras isoladas onde o modelo de linguagem é classificado como uma gramática de grafo dirigido onde cada aresta tem peso constante, ou seja, a probabilidade de transição entre palavras é constante para qualquer palavra.

### 2.3.4 Decodificação

A etapa de decodificação ocorre durante a fase de testes e consiste em uma busca pela sequência de palavras que melhor se adapta aos vetores acústicos, dados os modelos, ou seja, realiza-se uma busca pela sequência de estados que maximiza a chamada probabilidade a posteriori, que pode ser calculada através da fórmula de Bayes, como mostrado na Equação 2.2.

$$P(W|X) = \frac{P(X|W)P(W)}{P(X)} \tag{2.2}$$

Uma vez que, para fins de reconhecimento de fala, o elemento observado não corresponde a um único elemento e sim a uma sequência de vetores acústicos, então, para uma sequência de X vetores, considerando a ocorrência de cada observação como um evento independente, pode-se construir uma regra de decisão $\widehat{W}$ para o problema através da maximização da probabilidade a posteriori, como mostrado na Equação 2.3. Nesta equação, ainda é possível notar a eliminação do termo presente no denominador da Equação 2.3. A ausência deste termo é justificável, uma vez que ele será o mesmo para todas as classes testadas.

$$\begin{aligned} \widehat{W} &= arg\max_{W} P(W|X) \\ &= arg\max_{W} \frac{P(X|W)P(W)}{P(X)} \\ &= arg\max_{W} P(X|W)P(W) \end{aligned} \tag{2.3}$$

É fácil verificar que a chamada probabilidade a priori P(W) da Equação 2.3 será fornecida pelo modelo de linguagem e a probabilidade condicional P(X|W) será fornecida pelo modelo acústico. Logo, onde se lê probabilidade a priori, leremos a probabilidade de ocorrer uma determinada sequência de palavras e onde se lê probabilidade condicional, leremos a probabilidade de observar uma determinada sequência de vetores dada uma sequência de palavras.

O algoritmo mais utilizado nesta etapa de decodificação é conhecido como algoritmo de Viterbi. Ele consiste em um algoritmo de busca síncrono, que procura pelo estado mais provável a cada unidade de tempo. Este algoritmo considera uma aproximação, conhecida por aproximação de Viterbi. Como mostrado na Equação 2.4, esta aproximação considera que, como o objetivo da decodificação é encontrar a melhor sequência de palavras, então, o somatório da Equação 2.3 pode ser substituído pelo máximo, de forma a encontrar a melhor sequência de estados.

$$\widehat{W} = arg\max_{W} P(W|X) \cong arg\max_{W}[P(W)\max_{s_0^T} P(X, s_0^T|W)] \tag{2.4}$$

De forma a melhorar o desempenho da busca, o decodificador usa duas estratégias: guardar alguns caminhos ótimos intermediários, de forma que alguns resultados possam ser usados por outros caminhos sem que haja necessidade de computá-los novamente, além de realizar poda nos nós que se encontram abaixo de um limiar pré-estabelecido. Desta forma, a cada passo, o

decodificador elimina caminhos que são muito provavelmente desnecessários, percorrendo apenas um feixe do espaço de busca. Este tipo de busca é conhecido como busca em feixe.

### 2.3.5 Ferramenta Sphinx

O Sphinx é uma ferramenta que oferece diversos componentes para implementar as etapas de um sistema de reconhecimento de fala, provendo flexibilidade na alteração de vários parâmetros. Desenvolvido na Universidade de Carnegie Mellon, o Sphinx possui uma documentação bastante escassa apresentando apenas alguns manuais e tutoriais ligeiramente desatualizados em sua página, o que faz com que o uso desta ferramenta não seja tão difundido. Porém, isso não diminui o interesse pela mesma dado que se trata de um software totalmente livre. Além disso, o Sphinx-4 - versão do decodificador do Sphinx utilizada neste trabalho - é escrita na linguagem de programação JAVA - trazendo o benefício de ser multiplataforma - e apresenta desempenho bastante similar àquele de seu antecessor, o Sphinx-3, que é escrito na linguagem C. Os decodificadores disponíveis pelo Sphinx e suas propriedades são apresentados com mais detalhes adiante nesta seção.

O Sphinx Train é o componente usado para o treinamento do modelo acústico. E é composto por uma série de programas escritos na linguagem de programação C que servem para o propósito de geração de coeficientes acústicos e treinamento dos HMMs. O Sphinx Train permite o uso tanto de HMMs contínuos como semi-contínuos no treinamento. Neste trabalho, no entanto, apenas foram utilizados HMMs contínuos.

Para geração do modelo de linguagem, o pacote de ferramentas cmuclmtk (CMU-Cambrigde Language Model Toolkit) v2 é utilizado. Este *toolkit*, também disponibilizado pelo grupo Sphinx da Universidade de Carnegie Mellon,

consiste em uma série de ferramentas de software para UNIX, que facilitam a construção e teste de modelos de linguagem. Já para a etapa de reconhecimento, existem várias versões de decodificadores do Sphinx disponíveis. A escolha de uma delas está diretamente relacionada ao tipo de aplicação que se deseja. Seguem abaixo algumas informações pertinentes a respeito de cada uma dessas versões.

### 2.3.6 Sphinx-2

O Sphinx-2 é o sistema antecessor ao Sphinx-3 e, embora não tão preciso quanto ele, o sistema é rápido e pode rodar em tempo real, o que faz com que ele seja uma boa escolha para aplicações livres. O Sphinx-2 utiliza modelos de HMMs com FDPs de saída semi-contínuas. Ele é considerado obsoleto, não sendo mais utilizado na Universidade de Carnegie Mellon.

### 2.3.7 Sphinx-3

O Sphinx-3 utiliza modelos HMMs com FDPs de saídas contínuas e é considerado o estado da arte da Universidade de Carnegie Melon. Ele é o sucessor do Sphinx-2 apresentado adiante - e sua última versão disponibilizada foi a 0.7.

### 2.3.8 Sphinx-4

O Sphinx-4 [5] corresponde a uma versão do Sphinx-3 escrita totalmente na linguagem de programação Java. No entanto, ele foi desenvolvido para ser um sistema muito mais flexível que o Sphinx-3 e possui desempenho semelhante, segundo [5].

### 2.3.9 Pocket Sphinx

O Pocket Sphinx é uma versão do Sphinx-2, que pode rodar em sistemas embarcados. Suporta atualmente o Embedded Linux e o Windows CE. No mais, apresenta as mesmas vantagens e desvantagens do Sphinx-2, seu antecessor.

É válido mencionar que o Sphinx-4 é o único entre os decodificadores apresentados acima que continuam recebendo atualizações na Carnegie Mellon. Entretanto, o Pocket Sphinx ainda consegue suprir todas as necessidades deste projeto não influenciando na eficiência do mesmo. Contudo, é possível que o Sphinx-4 possa ser embarcado no Android já que ele foi totalmente escrito em Java, mas, para tal, seria gasto um tempo de averiguação podendo resultar em um trabalho muito extenso de adequação ou, até mesmo, em falha. Logo, como não é propósito deste trabalho adaptar novas estruturas e, nem tão pouco, esta iniciativa é encorajada pelos desenvolvedores do Sphinx, foi mantida a posição original de uso do decodificador Pocket Sphinx, uma vez que ele é próprio para dispositivos embarcados, sendo então apropriado para uso em sistemas de reconhecimento de fala contínua no Sistema Operacional Android, mediante alguns ajustes, como será mostrado no Capítulo 3.

## 2.4 Arduino

Arduino [15] é uma plataforma de prototipagem eletrônica *opem-source* baseada em flexibilidade, fácil acesso à *hardware* e *software* destinada à artistas, *designers*, curiosos ou qualquer pessoa interessada em criar objetos ou ambientes interativos.

Originalmente, este microcontrolador, foi criado como uma plataforma

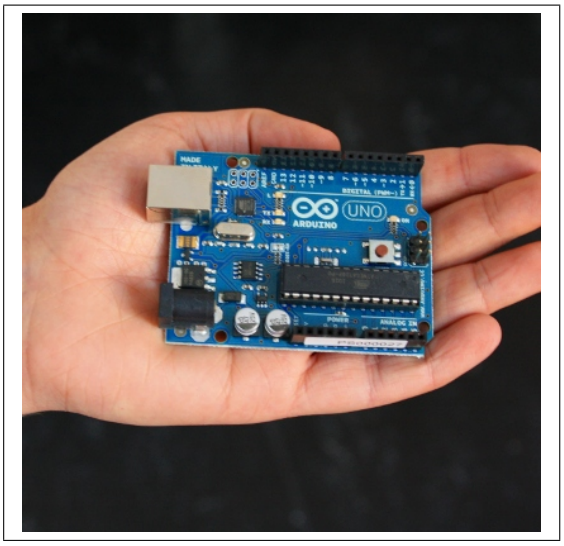



Figura 2.6: Arduino UNO utilizado neste trabalho.

educacional para uma classe de projetos da *Interaction Desing Institute Ivrea* em 2005 com o intuito de suportar mentes artísticas e baseadas a *design*. Focado em simplicidade, busca uma estrutura voltada para um público sem muita base tecnológica.

O Arduino pode perceber o ambiente através de entradas oriundas de uma variedade de sensores e á capaz de executar ações ao seu redor usando controles de luz, motores e outros atuadores. Seu microcontrolador é programado usando a linguagem de programação Arduino [16] (baseado em *Wiring* [17]) e o ambiente de desenvolvimento Arduino (desenvolvido em Java e baseado em *Processing* [18]). Projetos em Arduino podem ser auto-suficientes (quando executam sem interferências externas) ou comunicando-se com *softwares* em um computador usando um cabo USB ou combinado com algum componente que permita tráfego de dados sem fio (Ex.: *Shield Bluetooth*).

Oferecendo tudo que é necessário para uma comunicação ubíqua [19], sua utilização cresceu de tal maneira que ultrapassou o domínio para o qual foi inicialmente proposto. Hoje é usado nas mais diferentes gamas onde

este trabalho é um grande exemplo, pois, sua facilidade de manuseio e a flexibilidade em agregar componentes otimizando a integração com outros dispositivos o torna candidato ideal, posto que, como dito inicialmente, a integração é o ponto chave na implementação de uma *Web of Things*.

# Capítulo 3

# Funcionamento do Sistema

Neste capítulo será apresentado o protótipo do sistema proposto no início deste documento. Para tal, como foi feito no capitulo 2, o sistema será dividido em níveis:

- **RASSF** - será apresentada na primeira seção, de modo a explicitar quais componentes físicos foram utilizados e a arquitetura do programa interno;
- **SG** - será apresentado na segunda seção, e como o item anterior, explicita quais componentes físicos foram utilizados e a arquitetura do programa interno, além de introduzir uma abordagem relativamente diferente da apresentada no trabalho referenciado por [5], dadas as diferenças de Linguagens de Programação utilizadas na base do sistema;
- **Aplicação** - será apresentada na terceira seção, expondo como o sistema de reconhecimento de fala para Android foi desenvolvido, assim como todo o processo de preparação dos dados, extração de coeficientes

e treinamento dos modelos acústico e linguístico. Também será vista a arquitetura implementada para o Cliente REST.

## 3.1 Rede de Atuadores e Sensores Sem Fio (RASSF)

Na RASSF, os papéis dos sensores e atuadores é coletar dados do ambiente e executar apropriadas ações baseadas nos mesmos e/ou nas requisições das camadas a cima.

Os sensores detectam os fenômenos do ambiente e transmitem esses dados para os atuadores para que eles executem as devidas ações. Entretanto, estes podem ser enviados para o SG, o qual pode delega aos atuadores a execução das ações cabíveis. Esta estrutura de ação é chamada Arquitetura Autônoma (Figura 3.1 (a)), pois não há existência de intervenção humana. Quando o dado, ao invés de ser processado e executado na própria rede, é enviado para a camada de aplicação através do SG solicitando uma intervenção humana, a estrutura é chamada de Arquitetura Semi-Autônoma (Figura 3.1 (b)), desde que a ação seja coordenada pelo SG.

Dependendo da funcionalidade a ser agregada, uma das duas arquiteturas deve ser usada, ou até mesmo as duas. Como esta sendo proposto um sistema *command-control*, a arquitetura semi-autônoma é a mais propícia, entretanto, para acessibilidade, quanto mais autônoma a arquitetura for, mediante configurações iniciais, maior a independência do usuário. De modo à resolver este dilema o sistema foi implementado utilizando a arquitetura semi-autônoma como base, mas com flexibilidade suficiente para expandi-la e/ou combina-la com uma arquitetura autônoma dependendo da demanda.

Os componentes dos nós sensores e atuadores de uma RASSF podem ser vistos em Figura 3.2 (a) e (b), respectivamente.

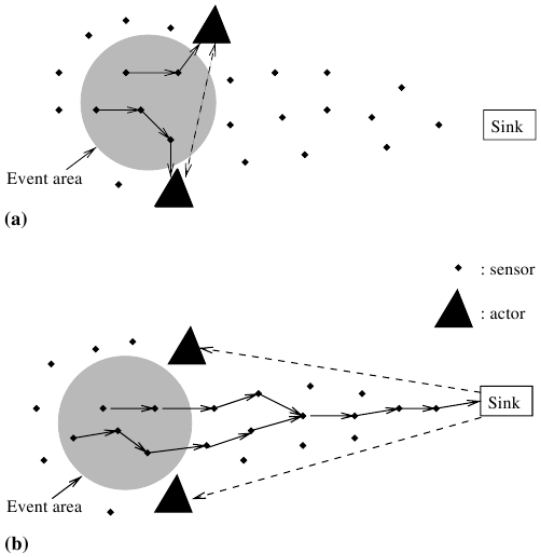



Figura 3.1: Arquitetura (a) Autônoma e (b) Semi-Autônoma.

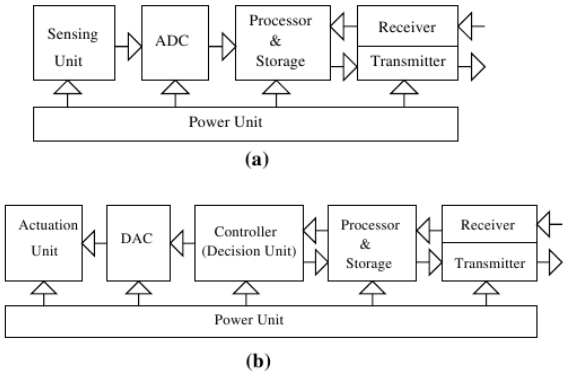



Figura 3.2: Componentes dos (a) sensores e (b) atuadores.

Os nós sensores são equipamentos possuidores de uma unidade de força, subsistemas de comunicação (transmissor e receptor), recursos de armazenagem e processamento, Conversores Analógicos para Digitais (*Analog to Digital Converter - ADC*) e unidade de sensoriamento. A unidade de Sensoriamento possui a tarefa de observar fenômenos como: eventos térmicos, óticos ou acústicos. O dado é coletado de forma analógica e convertido para digital através do ADC, daí é analisado pelo processador e então transmitido para os atuadores cabíveis.

A função da unidade de decisão (controlador) é receber a leitura do sensor como entrada e gerar um comando de ação como saída. Tais comandos são convertidos em dado analógico por um Conversor Digital para Analógico (*Digital to Analog Converter - DAC*) e são transformados em ações pelas unidades de atuação.

Todas as necessidades citadas anteriormente são supridas pelo Arduino UNO, o que combinado com sua facilidade e flexibilidade de desenvolvimento, o tornaram candidato ideal para ser escolhido de modo a compor a rede de atuação e sensoriamento.

Alem disso, uma RASSF possui duas características básicas [4]:

- **Requerimentos em tempo real**: Em RASSF, dependendo da situação, há a necessidade de uma rápida resposta para em determinado evento. Por exemplo, em incêndio, as ações devem ser iniciadas na área do evento o quanto antes. Além disso os dados coletados devem continuar válidos até o momento da ação.

- **Coordenação** - Diferentemente das redes de sensores onde a entidade central é responsável pela coleta e processamento dos dados, em uma RASSF ocorre um outro fenômenos chamado sensor-atuadores e atuadores-atuadores, os quais pode ser vistos no trabalho referenciado

por [4].

Neste trabalho a característica coordenação supracitada não foi implementada, pois foi utilizado apenas um componente atuador que possui a função de controlar um televisor através de um emissor *Infra Red*. Desta forma, somente a característica de requerimento em tempo real se fez necessária, pois, apesar de não ser utilizada neste protótipo, objetivou-se a possível expansão do sistema para uma arquitetura autônoma, como dito anteriormente.

## 3.2 Smart Gateway (SG)

Hoje, como a integração direta de dispositivos com a Web ainda é extremamente difícil, dado que muitos destes não suportam protocolos para esses fins, como *Internet Protocol (IP)* ou HTTP que são comumente utilizados em RASSF, uma abordagem diferenciada se faz necessário. O trabalho [3] propõe o uso do conceito SG como elemento intermediário, propiciando o acesso aos dispositivos pela Web. O objetivo central do SG é abstrair os protocolos de comunicação proprietários ou APIs dos dispositivos e prover acesso à suas funcionalidades via RESTful.

Segundo o trabalho referenciado por [5], a arquitetura de um SG deve ser projetada com base em três princípios: simplicidade, extensibilidade e modularidade. Simplicidade e extensibilidade para permitir a extensão e a customização para atender as necessidades dos usuários. E, Modularidade, para que os componentes internos possam interagir através de pequenas interfaces, permitindo assim, a evolução e troca das partes individuais do sistema.

Para tal, sua arquitetura deve ser composta por três camadas principais: camada de apresentação, camada de controle e camada de abstração. A Figura 3.3 mostra, de forma geral, esta estrutura.

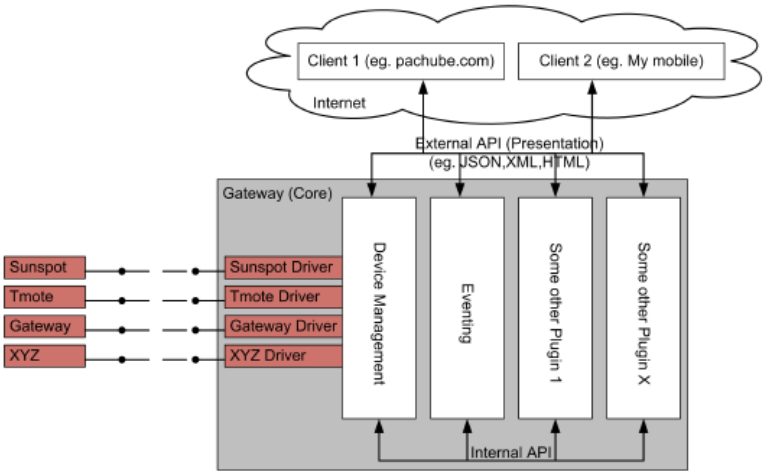



Figura 3.3: Visualização em alto nível de um *Smart Gateway*.

Diferente da linguagem utilizada no último trabalho referenciado, uma abordagem diferente foi tomada, pois a linguagem Python oferece uma unidade organizacional que tem por base os três princípios de um SG. Esta arquitetura é chamada de módulo.

Mark Lutz [20] define módulo como a unidade organizacional de programação de auto-nível que empacota códigos de programas e dados para reuso. Em termos concretos, módulos geralmente correspondem à arquivos de programas em Python ou extensões de código em linguagens externas como C, Java, C# ou etc, onde cada arquivo é um módulo e um módulo importa outros módulos para usar os nomes definidos por estes. Desta forma, os módulos proveem um modo mais fácil de organizar componentes em um sistema. O que é objetivado por um SG.

A seguir será apresentado o funcionamento de cada estrutura do SG (Figura 3.3) e como foi adaptada para o Python quando esta se fez necessário.

### 3.2.1 Camada de Apresentação

A camada de Apresentação torna os componentes do SG acessíveis para a Web. Sendo a fina camada acima da camada de controle, gerencia as requisições dos clientes através de uma interface REST.

Com o intuito de tornar os dispositivos acessíveis através da Web, um mapeamento dos nomes dos dispositivos em *Uniform Resource Identifier (URI)* [8] é executado pela camada de apresentação. O dispositivo nomeado "sensor1" será mapeado por "/sensor1". Isto permite aos usuários procurar dinamicamente a lista de dispositivos.

Requisições da Web utilizando este mapeamento serão redirecionadas para o *driver* referente ao dispositivo informado, o qual, tratará a solicitação e gerará uma resposta.

Na implementação desta camada nenhuma adaptação foi exigida. A biblioteca/módulo BaseHTTPServer supriu as necessidades do servidor HTTP gerando apenas a obrigatoriedade da definição do mapeamento anteriormente citado.

### 3.2.2 Camada de Controle

A camada de controle é composta por diversos componentes independentes chamados plugins.

Um plugin é um componente de software que é carregado na inicialização do SG . Sua principal funcionalidade é permitir a acoplagem eficiente dos drivers e, assim, possibilitar o direcionamento, sem ambiguidade, das requisições da camada acima (Camada de Apresentação) para os seus respectivos drivers.

É permitido à um plugin depender de outros, Entretanto, para manter

uma fraca dependência entre os mesmos, deve ser utilizado uma mecanismo de sincronização de modo que, ao ocorrer uma mudança no estado de qualquer dispositivo, o sistema consiga percebe-la e reorganizar-se ao ponto de não permitir as requisições quando não houver recurso disponível (exemplo: o plugin de um determinado dispositivo chama seu método independente do dispositivo estar ausente ou não).

Como este trabalho consiste de um sistema extremamente simples existindo apenas um único driver, como será explicitado à seguir, a única funcionalidade desenvolvida nesta camada consiste no mecanismo de sincronização acima definido, de modo a impedir as requisições inexistentes por falta de recursos.

### 3.2.3 Camada de Abstração

Como na maioria dos sistemas operacionais modernos, o SG prove uma camada de abstração para os dispositivos. Para as aplicações no nível acima, todos os dispositivos são vistos da mesma forma, mesmos aqueles com implementações completamente diferentes.

A camada de abstração é ilustrada ao lado esquerdo da Figura 3.3. Drivers especializados são usados para se comunicar através de seus protocolos proprietários com os respectivos dispositivos físicos (SunSpotDriver utiliza ZigBee com SunSpot).

Para dispositivos que já suportam protocolos Web, a implementação do driver trabalha apenas repassando as requisições da camada de apresentação para os dispositivos. Entretanto, quando o dispositivo não suporta protocolos Web, o driver é responsável por traduzir a requisição para seu protocolo de modo à fazer o dispositivo compreende-lo.

Como não foi encontrado no período de desenvolvimento deste trabalho

um componente para o Arduino que o permitisse suporta protocolos Web, o uso de qualquer protocolo de comunicação entre ele e o SG não trouxe grande variação, pois, para qualquer padrão utilizado haveria a necessidade da implementação de um driver nesta camada. Logo, sem perdas de generalidade, o protocolo escolhido foi o 802.15 *Bluetooth*. O que consequentemente demandou o desenvolvimento do seu respectivo driver que foi facilitado graças à utilização da biblioteca/módulo Bleuz.

## 3.3 Aplicação

A aplicação é o componente responsável por emitir um conjunto de consultas ou interesses, que descrevem as características dos fenômenos físicos que o usuário deseja analisar. Os interesses da aplicação podem indicar os tipos de dados desejados; a frequência com que esses dados devem ser coletados; a necessidade ou não dos dados sofrerem algum tipo de agregação; os limiares a partir dos quais os dados devem ser transmitidos; ou ainda, eventos que podem disparar algum comportamento peculiar da rede, como a ativação de sensores específicos, a alteração na taxa de sensoriamento ou o início de uma ação para os atuadores. Em particular, este último é o de maior relevância, posto que o protótipo proposto consiste de um sistema *command-control*.

De modo geral, a ideia central da REST reside no conceito de recurso como algum componente de uma aplicação que é valorada por uma identificação única. Na Web as identificações dos recursos é feita pela URI, como dito anteriormente, desta forma os clientes de serviços RESTful podem seguir esta estrutura para encontrar recursos de modo a interagir com os mesmos, como em navegadores Web. Isto permite aos clientes explorar serviço simplesmente navegando, e, em muitos casos, estes vão usar uma variedade de tipos de links

para estabelecer diferentes relações entre os recursos disponíveis. Entretanto, neste trabalho, a geração de tais links não se fez necessário, pois o sistema seria acionado por voz conferindo maior independência ao usuário portador, mas, como o SG seguiu a arquitetura REST, ainda é possível acessar os recursos através de qualquer cliente REST, como por exemplo um navegador WEB, posto que se conheça o endereço do servidor.

Para este componente foi desenvolvido um sistema de reconhecimento de fala e um Cliente REST em um *Smartphone Sansung Galaxy 5* (especificações no Apêndice A), os quais combinados conferem independência ao usuário.

A seguir serão apresentados o sistema de reconhecimento de voz e o cliente REST para Android.

### 3.3.1 Sistema de Reconhecimento de Fala para Android

O sistema de reconhecimento de fala para Android foi desenvolvido com base na versão de demostração disponibilizada na página virtual do Sphinx [9], a qual possui uma interface com uma tela de exposição dos textos reconhecidos e um botão utilizado para disparar o início da gravação do áudio, enquanto é mantido pressionado, e o fim da gravação dando início ao reconhecimento, quando deixa de ser pressionado.

Este sistema demostrativo usa o pacote Pocketsphinx que é uma redução do Sphinx 3 desenvolvido em linguagem de programação C. Portanto, para portá-lo para Android, o qual utiliza Java, foram utilizadas duas ferramentas: o SWIG [22] e o pacote Android NDK [23].

Como o objetivo deste trabalho é gerar independência, o sistema supracitado foi alterado ao ponto de não necessitar do botão. Portanto, ao ser iniciado, este se comporta de modo a interpretar (reconhecer) o que é dito em tempo real sem a necessidade de qualquer tipo de acionamento.

Objetivando a simplicidade do protótipo, foram definidos 6 comandos representativos do controle remoto. São eles:

- Ligar;
- Desligar;
- Canal +;
- Canal -;
- Volume +;
- Volume -.

Desta forma foi possível manejar as funções básicas oferecidas pelo aparelho televisor sem perdas de generalidade.

### 3.3.2 Cliente REST para Android

Para acessar os recursos disponibilizados pelo SG, é necessário a implementação de um cliente REST. Entretanto, para potencializar sua utilização em um dispositivo cujo seu sistema operacional é Android, a arquitetura montada seguiu os padrões apresentados na Googlo I/O 2010 [10], a qual pode ser vista na Figura 3.4.

Cada entidade terá suas funções apresentadas brevemente:

- **Método REST** - Concentra a arquitetura do cliente REST
  - Prepara o HTTP URL e o corpo da requisição HTTP;
  - Executa a transação HTTP;
  - Processa a resposta HTTP;

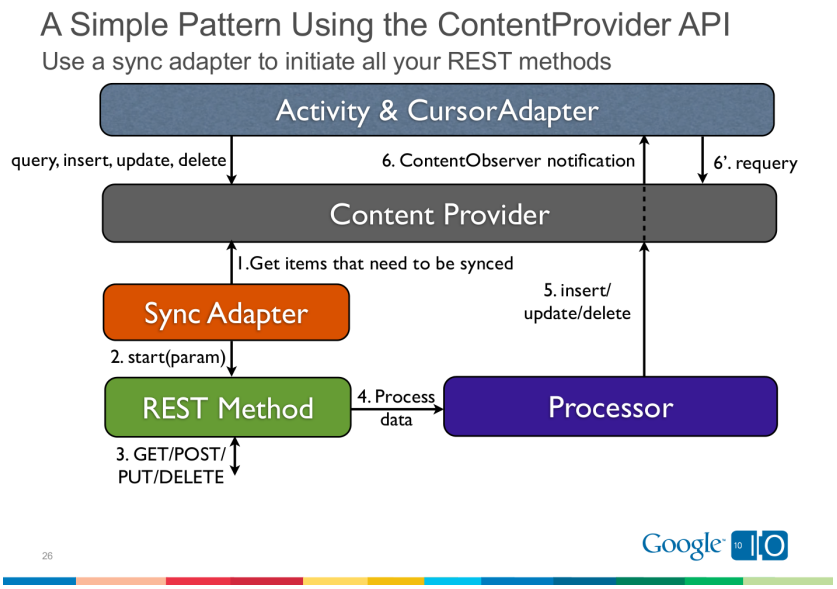



Figura 3.4: Arquitetura REST sugerida pela Google 10.

  - Seleciona o tipo de conteúdo opcional para resposta (XML, JSON, Binary);
  - Habilita o codificador de conteúdo gzip quando possível;
  - Roda o método REST em uma *thread* separada;
  - Usa o cliente Apache HTTP.

- ***Sync Adapter*** - Utilizado para executar as operações em *background* de maneira assíncrona;
- ***Processor*** - Concentra a lógica do sistema.
- ***Content Provider*** - Atua como interface entre a *Activity* e o *Processor*/*Sync Adapter* provendo dados e permitindo que a *Activity* possa operar sem interrupção.
- ***Activity & CursorAdapter*** - Uma *Activity* é o componente da apli-

cação que prove uma tela com a qual o usuário possa interagir de modo a fazer algo como, discar e efetuar uma ligação, digitar o conteúdo de uma mensagem e envia-lo, ver um mapa e etc. Já o *CursorAdapter* é um adaptador para expor dados oriundos de uma base.

1. Adiciona a operação de ouvir em *onResume* e remove-lo em *onPouse*
2. Trata as notificações do *ContentProvider*

Neste trabalho o componente *Content Provider* supracitado não foi necessário, pois do REST o método GET foi o único implementado, consequentemente, não gerando transição de dados de um banco.

# Capítulo 4

# Trabalhos Correlatos

Nos dias atuais muito se fala sobre acessibilidade, entretanto, poucos sabem o que de fato este termo significa. De maneira objetiva, acessibilidade significa não apenas permitir que pessoas com deficiências ou mobilidade reduzida participem de atividades que incluem o uso de produtos, serviços e informação, mas a inclusão e extensão do uso destes por todas as parcelas presentes em uma determinada população.

Para que a acessibilidade atinja níveis funcionais à PCDs com lesões severas, torna-se indispensável a utilização de aparatos eletrotécnicos. Um exemplo muito claro é o Motrix [1]. Este programa permite que pessoas com deficiências motoras graves, em especial tetraplegia e distrofia muscular, possam ter acesso à microcomputadores, permitindo assim, em especial com a intermediação da Internet, um acesso amplo à escrita, leitura e comunicação utilizando apenas a voz.

O uso do Motrix torna viável a execução pelo PCD de quase todas as operações que são realizadas por PSD (Pessoa Sem Deficiência), mesmo as que possuem acionamento físico complexo, tais como jogos. Através de um mecanismo inteligente, o computador realiza a parte motora mais difícil

destas tarefas possibilitando assim as ações desejadas.

O Projeto Motrix vem sendo desenvolvido no Núcleo de Computação Eletrônica da Universidade Federal do Rio de Janeiro (UFRJ) desde março/2002 sob coordenação do Professor José Antônio Borges.

Este programa foi a grande inspiração para o sistema desenvolvido neste trabalho. Posto que o objetivo central consiste em expandir as funcionalidades de controle do mundo virtual ao mundo físico, ao ponto do PCD controlar objetos eletrônicos sem o auxílio de PSDs.

Entretanto, controlar objetos físicos ao invés de virtuais é relativamente mais complexo e requer uma abordagem diferenciada. O trabalho *Towards Web of Things* [3] mostra como tornar objetos do mundo físico acessíveis no mundo virtal. E mais, transforma-os parte integrante da Web fazendo uso de protocolos abertos. Desta forma, qualquer aplicação que conheçam estes padrões pode acessá-los.

O trabalho apresenta dois modos diferentes de integrar os dispositivos via REST:

- Integração direta baseado nos avanços da computação embarcada;
- A utilização de SGs para dispositivos com recursos limitados.

Ambas metodologias foram ilustradas implementando-as em duas plataformas diferentes. Foi mostrado como um eco-sistema de dispositivos RESTful pode facilitar significativamente a criação de *mashups* físico-virtuáis. E neste processo foi constatada a grande flexibilidade e poderoso mecanismo de prototipagem de aplicações que é oferecido pela combinação do REST e a Web para conectar dispositivos.

# Capítulo 5

# Conclusão e Trabalhos Futuros

# Referências Bibliográficas


[1] Projeto Motrix - http://intervox.nce.ufrj.br/motrix/

[2] OLIVEIRA, VF, *Reconhecimento de Fala Contínua Para O Português Brasileiro Baseado Em HTK e SPHINX*, Projeto Final, Escola Politécnica/COPPE/UFRJ, março 2010

[3] GUINARD, D, *Towards the Web of Things: Web Mashups for Embedded Devices*

[4] AKYILDIZ, IF, KASIMOGLU, IH, *Wireless Sensor and actor networks: research challenges*, article, Georgia Institute of Technology/Broadband and Wireless Network Laboratory, maio 2004

[5] TRIFA V, WIELAND S, GUINARD D, BOHNERT T, *Design and Implementatio of a Gateway for Web-based Interection and Management of Embedded Devices*, article, Intitut for Pervasive Computing, ETH Zurich, SAP Research CEC Zurich

[6] DELICATO, FC, *Middleware Baseado em Serviços para Redes de Sensores Sem Fio*, Dissertação de Doutorado, Engenharia Elétrica/COPPE/UFRJ, julho 2005

[7] LEONARDO RICHARDISON, SR, *RESTFul Web Serices*, 2007, O'Reilly

[8] URI - http://www.w3.org/TR/uri-clarification/

[9] CMU Sphinx - http://cmusphinx.sourceforge.net

[10] Google I/O 2010 - http://www.google.com/events/io/2010/sessions/developing-RESTful-android-apps.html

[11] http://googleblog.blogspot.com/2007/11/wheres-my-gphone.html

[12] Davis and Mermelstein, *Comparison of Parametric Representations for Monosyllable Word Recognition in Continuously Spoken Sentences*, IEEE Transactions on Acoustic, Speech and Signal Processing, 1980 .

[13] Donald G. Childers, David P Skinner, Robert C. Kemerait, *The Cepstrum: A Guide to Processing*, Proceedings of the IEEE, VOL. 65, NO. 10, October 1977

[14] Lawrence Rabiner, Biing-Hwang Juang, *Fundamentals of Speech Recognition*, Pentice-Hall International, Inc, 1993

[15] Arduino - http://arduino.cc

[16] Linguagem de Programação Arduino - http://arduino.cc/en/Reference/HomePage

[17] *Wiring* - http://wiring.org.co/

[18] *Processing* - http://www.processing.org/

[19] Weiser M., *The Computer of the Twenty-First Century*, Scientifc America, Vol. 256, No. 3, Sept. 1991, pp. 94-104.

[20] *Learning Python*,

[21] Dieter Uckelmann, Mark Harrison, Florian Michahelles, *Architecting the Internet of Things* Springer Science, 2011

[22] *SWIG* - http://www.swig.org/

[23] *Android NDK* - http://developer.android.com/sdk/ndk/index.html

# Apêndice A

# Configuração Galax 5

| | | | |
|---|---|---|---|
| Tecnologia | Frequência | GSM | Quad Band (850 + 900 + 1800 + 1900 MHz) |
| | | 3G | UMTS (850/1900/2100MHz) |
| | Rede e Dados | GPRS | Sim |
| | | EDGE | Sim |
| | | 3G | Sim |
| | Sistema Operacional | Android 2.1 | |
| | Navegador | WAP 2.0/xHTML,HTML | |
| Bateria | Padrão | Capacidade | 120mAh |
| | | Tempo de Conversa | 2G: até 9h |
| | | Tempo de Standby | 2G: até 20 dias |
| Conectividade | Bluetooth | Sim | |
| | WAP | Sim | |
| | USB | Sim | |
| | Navegador HTML | Sim | |
| | Wi-Fi | Sim | |
| | GPS | Sim | |
| | AGPS | Sim | |
| | Aplicações PC Sync | Sim | |
| | USB Mass Storage | Sim | |
| | Infravermelho | Sim | |

Tabela A.1: Especificações técnicas do *Samsung Galaxy 5*.