UNIVERSIDADE FEDERAL FLUMINENSE
INSTITUTO DE COMPUTAÇÃO
GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

MARCELO DE CAMPOS NIERO

SISTEMA DE SEGMENTAÇÃO DE ÁUDIO DE REUNIÕES PRESENCIAIS UTILIZANDO RECONHECIMENTO DE LOCUTOR

NITERÓI
2010

MARCELO DE CAMPOS NIERO

SISTEMA DE SEGMENTAÇÃO DE ÁUDIO DE REUNIÕES PRESENCIAIS UTILIZANDO RECONHECIMENTO DE LOCUTOR

Dissertação apresentada ao Curso de Graduação em Ciência da Computação da Universidade Federal Fluminense, como requisito parcial para obtenção do Grau de Bacharel.

Orientador: Prof.ª Ana Cristina Bicharra Garcia

NITERÓI
2010

MARCELO DE CAMPOS NIERO

SISTEMA DE SEGMENTAÇÃO DE ÁUDIO DE REUNIÕES PRESENCIAIS UTILIZANDO RECONHECIMENTO DE LOCUTOR

Dissertação apresentada ao Curso de Graduação em Ciência da Computação da Universidade Federal Fluminense, como requisito parcial para obtenção do Grau de Bacharel.

Aprovada em dezembro de 2010.

BANCA EXAMINADORA

Prof.ª Drª. Ana Cristina Bicharra Garcia - Orientador
UFF

Prof. Dr. Inhaúma Neves Ferraz
UFF

Prof.ª Dr. Anselmo Antunes Montenegro
UFF

NITERÓI
2010

Ao Verbo vivo, Príncipe da paz,
Rei dos reis, único digno de
toda honra, glória e louvor

AGRADECIMENTOS

A Deus que me direcionou e me deu forças para perseverar.

A minha família que me apoiou durante a graduação.

À Universidade Federal Fluminense,

À Professora Ana Cristina Bicharra Garcia,

Ao Professor Dirceu Gonzaga da Silva que me orientou na área de processamento de voz,

A todos os professores com quem convivi e de quem tive o privilégio de ser aluno no Curso de Graduação em Ciência da Computação.

A todos os amigos com quem convivi nessa jornada acadêmica.

RESUMO

A tecnologia *groupware* em questão consiste em uma ferramenta para facilitar a manipulação das informações de um projeto, as quais serão provenientes da gravação de uma reunião presencial.

Esse projeto é parte de um projeto maior de construção de ferramentas de suporte a aquisição, análise e modelagem de conhecimento para posterior construção de sistemas. Aqui trataremos do embasamento dos temas e conceitos escolhidos para a construção da ontologia de um domínio e sua fonte. Isso auxiliará a justificar o modelo de conhecimento final.

Esta ferramenta é composta, basicamente, por dois componentes. O primeiro é um *player* de vídeo que exibe trechos de vídeo, indexados previamente, com acesso direto a qualquer trecho. A busca pode ser feita pelo assunto do período da reunião, identificação de locutor ou por um tempo especifico do vídeo. A busca é facilitada por processamento de áudio que identifica os locutores.

Para o funcionamento correto do sistema, é necessário cadastrar todos os locutores em uma base de dados, o qual será feito a partir de uma pequena gravação da voz do locutor. Após o processamento do áudio do vídeo obteremos a identificação de todos os falantes pré-cadastrados e o tempo de inicio e fim de suas respectivas falas. Essas informações serão armazenadas em uma tabela durante a execução do sistema.

Feito o processamento, o usuário poderá navegar pela tabela e escolher, pelo tempo ou pelo locutor ou pelo texto o trecho de vídeo que deve ser exibido.

Palavras-chave: Identificação locutor, Processamento Áudio, Reunião presencial, *Player*

ABSTRACT

The groupware technology in question consists on a tool to facilitate the project information handling and this information will come from recording of a face meeting.

This project is a part of a larger project of building tools to support the acquisition, analysis and knowledge modeling for future system building. The treated part in this project consists on a basis of themes (or concepts) chosen to build the ontology of a domain and its font. This will help to justify the final knowledge model.

This tool is basically composed of two components. The first one is a video player that displays video clips, previously indexed, with direct access to any stretch. The search can be made through the theme of the stretch, through speaker identification or through a specific time of video. The search is facilitated by audio processing that identifies the speakers.

For the correct functioning of the system, it is necessary to register all speakers in a database. The registration is done from a small voice recording of the speaker. After the audio processing of the video we get the identification of all pre-registered speakers, the time of beginning and end of their speeches, respectively. These informations will be stored in a table during the execution of the system.

As soon as the processing is finished, the user can browse the table and chose by the time or by the speaker, or by the text what video stretch should be played.

Key-words: Speaker identification, Audio processing, Face meeting, *Player*

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Programming Interface
AVI	Audio Video Interleave
DLL	Dynamic Link Library
DNA	Deoxyribonucleic Acid
DTW	Dynamic Time Warping
GMM	Gaussian Mixture Models
GUI	Graphical User Interface
HMM	Hidden Markov Model
HTK	Hidden Markov Model Toolkit
JNI	Java Native Interface
MPEG	Movie Picture Experts Group
MP3	MPEG-½ Audio Layer 3
NIST	National Institute of Standards and Technology
RN	Redes Neurais
WAVE	Waveform Audio File

SUMÁRIO

1. INTRODUÇÃO

1.1. VISÃO GERAL

O presente trabalho monográfico tem a finalidade de demonstrar como foi feito um sistema de reconhecimento de locutor e segmentação do áudio de uma reunião presencial.

Para tal, serão apresentadas as motivações existentes para o desenvolvimento do trabalho e os objetivos do mesmo. Diversos temas serão abordados, tais como o porquê da linguagem de programação adotada; a finalidade das bibliotecas utilizadas e como se dá o processo de identificação de locutor. Serão expostas as ferramentas que foram utilizadas para a identificação acompanhadas de uma explicação com detalhes do modo de sua execução.

Destarte, será definido e apresentado o projeto de aplicação, levantamento de requisitos, implementação do código e seu modo de organização.

Além disso, será visto a configuração para o bom funcionamento do *software* com os propósitos abordados, e por fim, expostas sugestões para trabalhos futuros assim como melhorias no projeto.

1.2. MOTIVAÇÃO

O processamento de áudio é uma área de pesquisa que vem crescendo cada vez mais. Dentro deste ramo há várias vertentes, sendo algumas delas o reconhecimento de fala, reconhecimento de idioma, verificação de locutor, identificação de locutor, sintetizador de sons, sintetizador de voz, classificador de música, e outras.

Com o crescente aumento do número de empresas e mercados, cada vez mais aumenta a demanda por *softwares* para uso específico. Consequentemente aumenta o número de reuniões de projetos e também o volume de informações a serem manipuladas. Desta forma perde-se muito tempo para encontrar e extrair dados importantes de longas reuniões de projetos.

A partir destes dois fatos relevantes, surge à motivação deste trabalho, que corresponde em utilizar um ponto favorável do avanço das tecnologias de processamento de áudio para solucionar um problema causado pela prosperidade do mercado.

1.3. OBJETIVOS

O objetivo deste projeto é desenvolver um *player* de vídeo atrelado a uma ferramenta de identificação de locutor, cuja finalidade é identificar todos os locutores presentes no áudio do vídeo e especificar o tempo de início e fim de suas respectivas falas. Sua construção foi feita a partir de uma ferramenta de reconhecimento de fala, chamada HTK.

1.4. ORGANIZAÇÃO DO TRABALHO

O capítulo 1 traz uma breve introdução e uma visão geral sobre o documento. Já no capítulo 2, serão expostas as bibliotecas utilizadas; em que consiste a ferramenta HTK; e como se dá o processo de reconhecimento de locutor. O terceiro capítulo, por sua vez, apresenta o aplicativo proposto, que visa segmentar o áudio de acordo com as falas dos respectivos locutores presentes utilizando a ferramenta HTK. Em seguida, no capítulo 4, será feita uma análise do sistema desenvolvido. Na conclusão, serão apresentadas idéias para trabalhos futuros.

2. AS FERRAMENTAS UTILIZADAS

2.1. AS BIBLIOTECAS

O sistema em questão foi desenvolvido na linguagem de programação Java. Esta escolha se deu devido a alta portabilidade e eficiência de codificação. Assim sendo, as bibliotecas utilizadas foram a JMF, Fobs4JMF e a JAVE.

A Java Media Framework (JMF) é uma API que permite criar aplicativos Java, que capturem, reproduzam e façam *streaming* de áudio e vídeo [1].

Para utilizar o *player* desta API foi necessário entender como os arquivos de mídia são tratados. As informações sobre este tratamento foram encontradas na documentação da JMF, mais especificamente na seção *Interface Controller*, a qual prove informações sobre a alocação de recursos dos estados, geração de eventos e mecanismos para a obtenção de objetos que fornecem controles adicionais sobre o *Controller*. Assim sendo, é importante especificar os estados e sub-estados pelos quais o *player* passa, a fim de entender o processo de execução. São eles:

- ***Started*:** quando a mídia está sendo reproduzida.
- ***Stopped*:** quando a mídia não está sendo reproduzida, que por sua vez é dividido nos seguintes sub-estados:
    - ***Unrealized*:** estado inicial, onde o *Controller* é instanciado. Neste estado não é possível obter informações sobre o tempo da mídia.
    - ***Realizing*:** responsável por fazer a comunicação para alocar todos os recursos necessários, por exemplo, comunicação com o servidor ou com um arquivo local, entre outros.
    - ***Realized*:** nesta fase os recursos acima citados já foram alocados, exceto aqueles que necessitem de uso exclusivo, como um dispositivo de áudio ou um *hardware* de decodificação MPEG.
    - ***Prefetching*:** etapa em que a aquisição dos recursos ainda não alocados deve ser executada, por exemplo, os recursos não alocados pelos estados anteriores.
    - ***Prefetched*:** iniciado quando o estado anterior é concluído. Entretanto é necessário somente quando alguns métodos

específicos são chamados pelo programador, como por exemplo, *setMediaTime*.

A criação do *player* é feita através de três comandos. Primeiro deve-se chamar um métodos estático da classe **Manager**, passando, como parâmetro, o diretório absoluto da mídia a ser reproduzida. Esta chamada retorna uma instância da classe **Player** que deve ser atribuída a uma variável do tipo Player, a qual possui controle sobre o tocador.

```
player = Manager.createPlayer(new URL(strPath));
```

Feito isso, é necessário adicionar um controlador de eventos no *player* a fim de exercer maior domínio sobre os estados citados anteriormente. O controlador em questão consiste na criação de uma classe abstrata que sobrescreve alguns métodos da API.

```
player.addControllerListener(new PlayerEventHandler());
```

Antes de ir para o terceiro comando, serão mostrados os métodos sobrescritos na classe **PlayerEventHandler**. O primeiro deles é invocado quando o estado ***Realized*** é alcançado e apenas é instruído que é possível seguir para o estado ***Prefetching***.

```
@Override
public void realizeComplete(RealizeCompleteEvent realizeDoneEvent)
{
    player.prefetch();
}
```

Já o segundo comando sobrescrito é chamado quando o estado ***Prefetched*** é alcançado. Neste momento o componente visual, que contem a imagem da mídia, e o

componente de controle, os controladores da mídia, são capturados e adicionados a um painel.

```
@Override
public void prefetchComplete(PrefetchCompleteEvent prefetchDoneEvent)
{
    visualComponent = player.getVisualComponent();

    if (visualComponent != null )
    {
         panelVideo.add(visualComponent,BorderLayout.CENTER);
    }

    controlComponent = player.getControlPanelComponent();

    if (controlComponent != null)
    {
        panelVideo.add(controlComponent,BorderLayout.SOUTH);
    }
    validate();
}
```

O terceiro comando da criação do *player* consiste em dar início a alocação de recursos.

```
player.realize();
```

Apesar de permitir a criação de um *player* de forma rápida e simples, ela é compatível com poucos tipos de mídias, o que implicou no uso da segunda biblioteca. A Fobs4JMF é uma vertente, para Java, do projeto FOBS. Esta consiste na criação de um pacote orientado a objeto, em diversas linguagens, para o uso de uma API, capaz de manipular uma enorme quantidade de tipos de mídias [2], chamada FFMPEG [3]. A figura 1 ilustra a organização do FOBS.

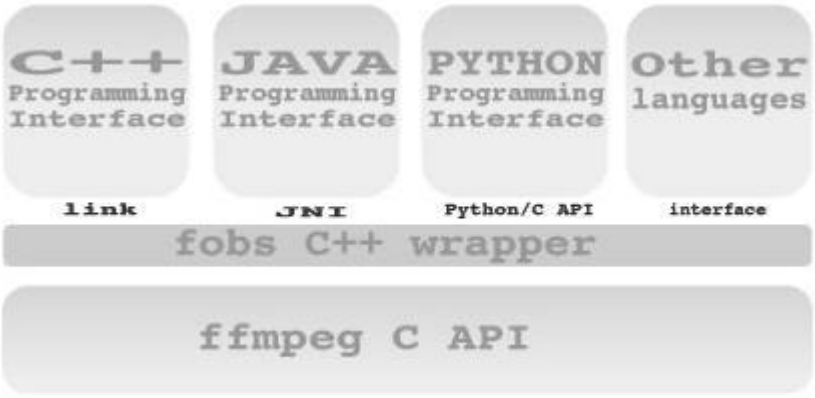

Figura 1 - Organização do projeto FOBS

O fato da Fobs4JMF funcionar como um *plugin* para a JMF permite que ela seja inserida, a qualquer momento, em um projeto que esteja usando somente a JMF sem qualquer alteração de código. Para isso basta substituir os arquivos da JMF pelos seus equivalentes da Fobs4JMF, adicionar a nova biblioteca no projeto e inserir uma DLL na pasta de arquivos do sistema operacional.

Já a biblioteca Java Audio Video Encoder (JAVE) também é baseada em um empacotamento da API FFMPEG. Suas principais funcionalidades são: transcodificar áudio e vídeo de um formato para outro, por exemplo, de AVI para MPEG ou de WAVE para MP3; separar faixas de áudio e vídeo; redimensionar um vídeo alterando seu tamanho e suas proporções entre outras. De todas as funções oferecidas, a única usada neste projeto é a extração da faixa de áudio de um vídeo, cuja utilização é feita da seguinte forma: primeiro são especificados o vídeo e o áudio; depois os atributos da extração que consistem no *codec* e no formato do áudio, e por fim o comando da extração.

```
File source = new File(filePath);
File target = new File(strTarget);

AudioAttributes audio = new AudioAttributes();
audio.setCodec("pcm_s16le");
EncodingAttributes attrs = new EncodingAttributes();
attrs.setFormat("wav");
attrs.setAudioAttributes(audio);

Encoder encoder = new Encoder();
encoder.encode(source, target, attrs);
```

2.2. O HTK

O Hidden Markov Model ToolKit (HTK) [5][6] é um conjunto de ferramentas portáteis criado para a construir e manipular Hidden Markov Models (HMM)[13], ou Modelos Ocultos de Markov. O HTK é usado, principalmente, em pesquisas de reconhecimento de fala, embora tenha sido usado em inúmeros outros aplicativos, incluindo pesquisas em síntese de fala, reconhecimento de caracteres e seqüenciamento de DNA.

O HTK consiste em um conjunto de bibliotecas e ferramentas disponíveis na linguagem C. Estas por sua vez, fornecem sofisticados recursos para a análise de fala, treinamento HMM, testes e análise de resultados.

2.3. UM BREVE HISTÓRICO

A primeira versão do HTK foi desenvolvida pelo Steve Young no Speech Vision and Robotics Group do Departamento de Engenharia da Universidade de Cambridge (DEUC) em 1989. Após seu lançamento, o interesse do mercado pelo *software* cresceu rapidamente, o que implicou na venda de seu código fonte a preço de custo de mídia. Isto ocorreu até a versão 1.2.

No início de 1992, Phil Woodland se uniu a Steve como co-desenvolvedor do HTK e as próximas versões, a partir da 1.3, passaram a ser vendidas pela Universidade, através da Lynxvale Company, pelo preço de 450 por licença para sites acadêmicos e de 950 por licença para empresas. Nesta época Steve e Phil passaram a oferecer manutenção no código, o que aumentou o número de usuários, culminando um acordo com a Entropic Research Laboratories (ERL) para assumir a manutenção e distribuição do HTK no início de 1993.

A primeira versão lançada pela ERL foi a 1.4, que incluiu gerenciamento de licença e a opção da compra de suporte, o que aumentou significativamente o preço do produto. Mesmo com a distribuição e manutenção no controle da ERL Steve e Phil continuaram empenhados no desenvolvimento de atualizações.

Em 1995 houve a fusão entre a ERL e a Universidade de Cambridge, dando origem a Entropic Cambridge Research Laboratory (ECRL), tendo Steve e Phil como diretores técnicos. Este fato representou a maior re-estruturação de muitas bibliotecas e

ferramentas, trazendo melhorias para o *software*. Neste momento os maiores contribuintes para a segunda versão foram Julian, Odell, Valtcho Valtchey e Dave Ollason. E os negócios da Entropic foram mudando gradualmente para o fornecimento de ferramentas de pesquisa, como o HTK, voltadas principalmente para a área de voz.

Em 1999 foi lançada a versão 2.2, já sem gerenciamento de licença. Esta por sua vez, foi a última lançada pela Entropic, pois em novembro do mesmo ano a companhia foi comprada pela Microsoft. Porém pouco tempo depois das negociações a Microsoft decidiu devolver o desenvolvimento e a licença do *software* para a DEUC, que em 2000 disponibilizou, gratuitamente em seu próprio site, o código fonte do projeto.

Um pouco antes do código se tornar livre, foi lançado o HTK3, baseado na última versão liberada pela Entropic. Desde então muitos tem sido os incentivos para o desenvolvimento do HTK, com o intuito de prover um suporte à infra-estrutura para investigar o estado da arte do reconhecimento de fala.

2.4. COMPONENTES DO HTK

A versão do HTK utilizada neste projeto é composta por 34 componentes. Para utilizá-los foi necessário fazer o *download* do código fonte do HTK, em seu próprio site, compilá-lo, assim gerando os executáveis, e utilizá-los através de uma chamada em Java, que excuta aplicativos externos. Esta chamada recebe como parâmetro a mesma *string* usada no Prompt de comando do Windows. A seguir serão citados somente os componentes utilizados neste projeto, seguido de uma breve explicação de suas funcionalidades e seus dados de entrada e saída, respectivamente, e o código referente a suas chamadas.

HCopy: usado para transformar um sinal de áudio em uma variedade de características acústicas de acordo com um conjunto de configurações. Recebe como entrada um sinal de áudio, um arquivo de configurações e tem como saída, um arquivo contendo as características do áudio. Este último está no formato do próprio HTK o qual tem como extensão ".mfcc". A figura 2 ilustra os dados de entrada e a saída do HCopy.

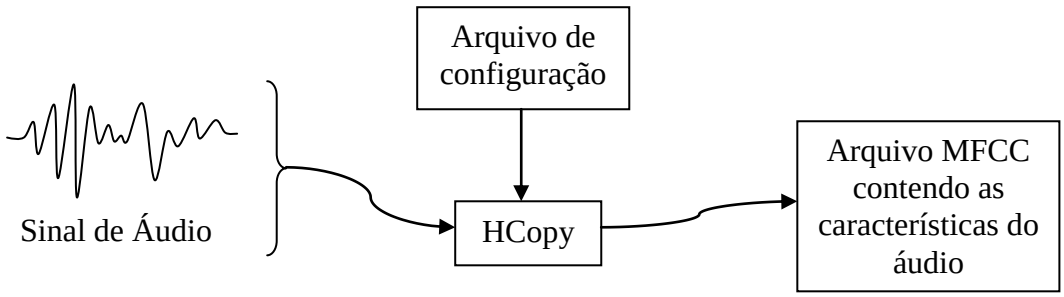

Figura 2 - Fluxograma HCopy

A *string* de chamada do HCopy é constituída pelas seguintes sub-strings: o diretório absoluto do HCopy, seguido dos caracteres "-C", indicando que a próxima sub-string é o diretório absoluto do arquivo de configuração, depois o diretório absoluto do arquivo WAV e o diretório absoluto do arquivo "mfcc".

```
Process p = Runtime.getRuntime().exec(
    "\"" + strHCopyPath +
    "\" -C \"" + strConfigFilePath +
    "\" \"" + wavFile.getAbsolutePath() +
    "\" \"" + strMFCCfilePath + "\"");
```

HInit: é um componente que prepara os dados que serão usados por outro componente, o HRest, para a geração de um modelo do sinal de áudio. A principal função do HInit é criar uma primeira versão do modelo do locutor a partir do carregamento de outro arquivo de configuração, chamado ProtoGMM. O ProtoGMM define a topologia do HMM, significando especificar o valor da variância, número de misturas e o peso das misturas a serem ignoradas. A figura 3 faz alusão dos parâmetros de entrada e a saída do HInit.

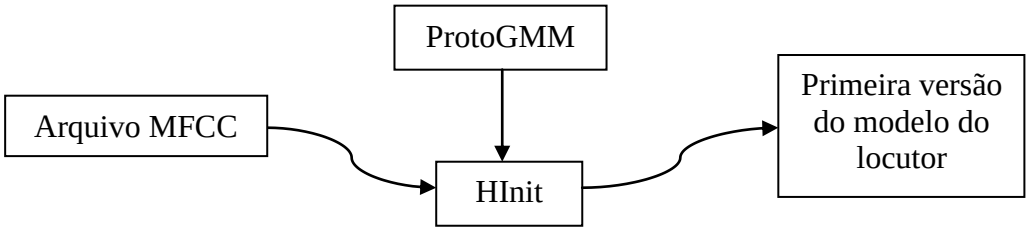

Figura 3 - Fluxograma HInit

O comando do HInit é formado pelas seguintes sub-strings: o diretório absoluto do HInit, seguido do caractere “-m 1” indicando que o número mínimo de exemplos de treinamento é um, “-v” sinalizando que o valor mínimo da variância é 0.00001, “-M” apontando que a próxima sub-string é o diretório do modelo, “-o” definindo que na sequência vem o nome do modelo a ser gerado, depois o diretório absoluto do ProtoGMM e o diretório absoluto do arquivo MFCC.

```
Process p = Runtime.getRuntime().exec(
    "\"" + strHInitPath +
    "\" -m 1 -v 0.00001 -M \"" + strModelPath +
    "\" -o \"" + strModelName +
    "\" \"" + strProtoGMMfilePath +
    "\" \"" + strMFCCfilePath + "\"");
```

HRest: este componente é responsável por aperfeiçoar o modelo gerado pelo HInit. Para realizar esta tarefa o HRest necessita de duas entradas. A primeira delas é o modelo criado pelo HInit. Já a segunda é o arquivo mfcc, gerado pelo HCopy e processado pelo HInit, como ilustrado na figura 4.

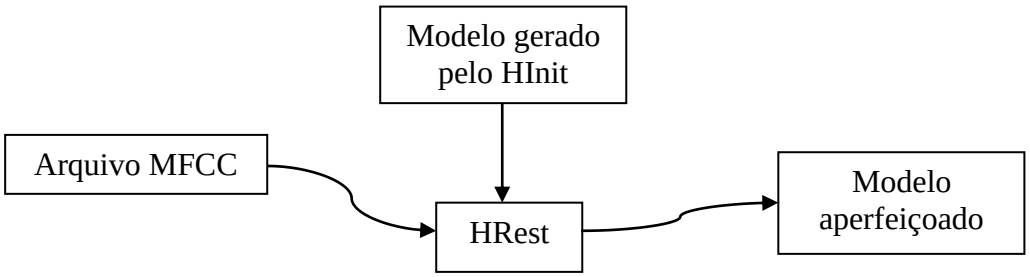

Figura 4 - Fluxograma HRest

Já a chamada do HRest é composta pelas respectivas sub-strings: o diretório absoluto do HRest, os valores “-m 1 –v 0.00001” já citado no HInit, seguido dos caracteres “-M” sinalizando que a próxima sub-string indica o diretório que contém todos os modelos cadastrados, na sequência é inserido o diretório absoluto do modelo aperfeiçoado e o diretório absoluto do arquivo “mfcc”.

```
Process p = Runtime.getRuntime().exec(
    "\"" + strHRestPath +
    "\" -m 1 -v 0.00001 -M \"" + strModelPath +
    "\" \"" + strModelFilePath +
    "\" \"" + strMFCCfilePath + "\"");
```

HParse: usado para criar a rede de reconhecimento, a qual será operada pelo próximo componente, o HVite. Esta rede consiste em um grafo, no qual cada nó possui um nome único que representa um modelo de um e somente um áudio. Para o funcionamento desta ferramenta é necessário alimentá-la com um arquivo contendo os dados da rede, elucidado na figura 5. Este arquivo equivale a uma lista que contém o nome dos modelos que participarão do reconhecimento.

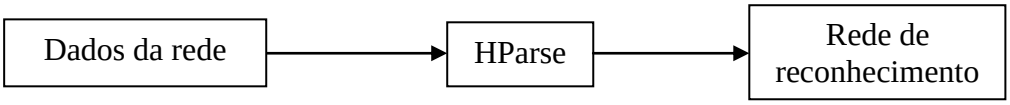

Figura 5 - Fluxograma HParse

O HParse, por sua vez, é constituído somente por três sub-strings, são elas: o diretório absoluto do HParse, diretório absoluto do arquivo que contém os dados da rede, e o diretório absoluto do arquivo da rede.

```
Process p = Runtime.getRuntime().exec(
    "\"" + strHParsePath +
    "\" \"" + strDataNetFilePath +
    "\" \"" + strNetFilePath + "\"");
```

HVite: é o componente que reconhece um locutor. Este reconhecimento se dá pelo cálculo da verossimilhança entre um modelo e um conjunto de características, e o maior valor é eleito o reconhecido. No entanto, para que a tarefa seja executada com sucesso é necessário fornecer diversos parâmetros, sendo eles: a rede de reconhecimento criada pelo HParse; o diretório dos possíveis modelos eleitos; o arquivo contendo dicionário de reconhecimento, o qual informa o que será reconhecido; a lista dos nomes dos possíveis modelos eleitos e o arquivo de características do áudio que está sendo reconhecido. A

saída deste componente é um arquivo do tipo “.mlf” contendo o nome do modelo eleito e a sua probabilidade. Os dados de entrada e saída são exibidos através da figura 6.

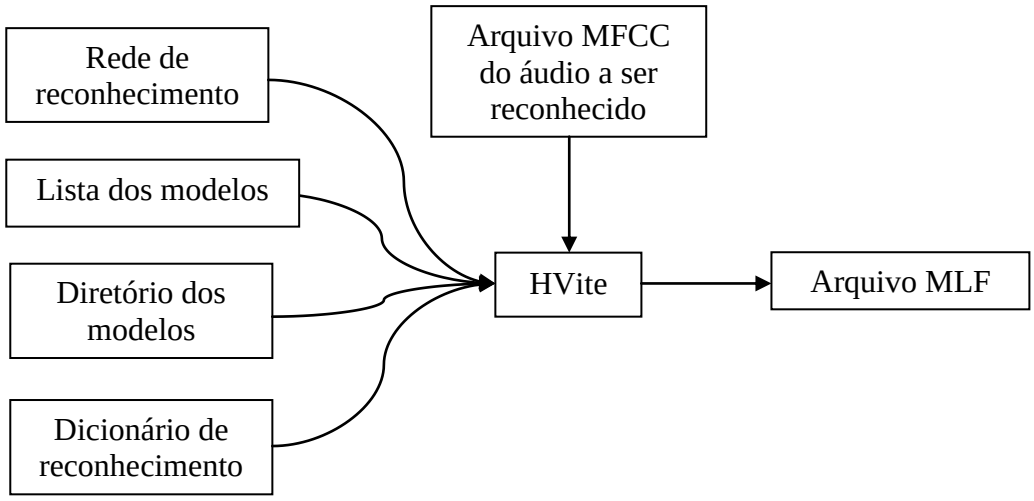

Figura 6 - Fluxograma HVite

O comando deste último componente é formado pelas sub-strings: diretório absoluto do HVite, os caracteres “-n” indicando o valor do parâmetro interno chamado nBest, “-o N” sinalizando que a saída deve ser normalizada, “-w” indicando que a informação a seguir é o diretório absoluto da rede de reconhecimento, “-d” apontando o diretório contendo todos os modelos cadastrados, “-i” mostrando que na sequência é dado o diretório absoluto do arquivo MLF, e os diretórios absoluto do dicionário, da lista dos modelos selecionados para o reconhecimento e o arquivo MFCC em processamento.

```
Process p = Runtime.getRuntime().exec(
    "\"" + strHVitePath +
    " -n " + nBest +
    " -o N -w \"" + strNetFilePath +
    "\" -d \"" + strModelPath +
    "\" -i \"" + strResultHViteFilePath +
    "\" \"" + strDicFilePath +
    "\" \"" + strSessionListFilePath +
    "\" \"" + strTempMFCCfilePath + "\"");
```

3. RECONHECIMENTO DE LOCUTOR

3.1. CONCEITOS BÁSICOS

A voz é resultado de transformações aglutinadas, operando-se em diversos níveis, tais como: semântico, linguístico, articulatório e acústico. Em primeira instância, o sinal da voz constitui-se de palavras que compõem a mensagem transmitida e, em nível secundário, abriga informações acerca do locutor, cujo reconhecimento baseia-se na análise das informações concernentes a sua identidade, enquanto que, o reconhecimento da voz é baseado na lingüística do sinal.

Variações na voz de um indivíduo se dão pelas diferenças anatômicas do aparelho fonador, e pela característica comportamental ou estrutural referente a uma pessoa ou a um grupo em questão (idiossincrasia).

As variações e diferenças observadas na voz, no que concerne ao reconhecimento do locutor, afetando seu desempenho, são:

- Intra-locutor (diz respeito ao locutor em si, ou pessoa falante): estado de saúde, estado emocional e ambiente.
- Inter-locutor (diferentes locutores): fisiológica, sotaque ou dialeto.
- Modo narrativo: leitura ou espontâneo, formal ou casual.
- Distorções acústicas:
    - Meio de gravação das locuções (mídias).
    - Meio de transmissão (canal telefônico, radiofônico, etc.).
    - Ruídos aditivos (vozes de fundo, ruídos sonoros em geral).

O processamento de voz manifesta-se em diversas áreas, possuindo empregos diversos, algumas já citadas na introdução deste documento, A figura 7 faz alusão a algumas destas áreas, com maior detalhamento para o reconhecimento de locutor [7].

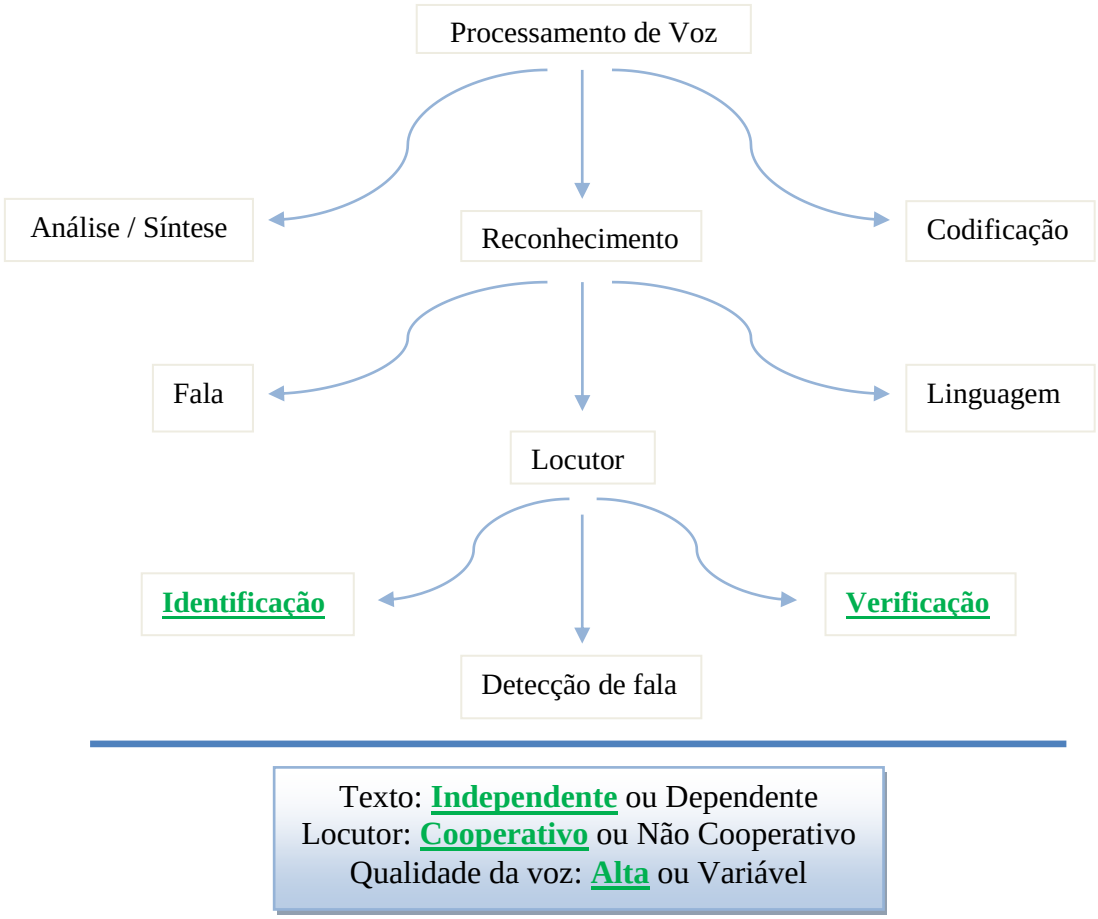

Figura 7 - Algumas áreas do processamento de voz, destacando-se o reconhecimento de locutor

Dentre as áreas do reconhecimento de locutor, exibido na figura acima, destaca-se duas delas:

- **Identificação:** dado um conjunto conhecido de N locutores, esta tarefa visa determinar qual desses locutores proferiu a amostra de voz a ser avaliada. Este conjunto possui duas divisões:
    - **Fechado:** quando a voz avaliada pertencer, necessariamente, a um dos locutores conhecidos, assim, possibilitando N decisões.
    - **Aberto:** quando a voz avaliada não pertencer a nenhum dos locutores conhecidos, desta forma, permitindo (N+1) decisões, indicando que a voz avaliada pode não ser identificada.

  Sendo assim, o desempenho da identificação diminui na medida em que se aumenta o número de locutores conhecidos.
- **Verificação:** consiste em verificar se uma dada amostra de voz foi pronunciada ou não por um locutor específico, o qual pertence a um conjunto aberto. Neste

caso, diferentemente da identificação, o desempenho independe do número de locutores.

Em relação dependência do texto, o reconhecimento pode ser dividido em duas áreas:

- **Dependente ao texto:** nos sistemas dependentes ao texto, o locutor cuja voz será analisada é induzido a falar palavras específicas, as quais pertencem aos padrões de referência da análise.
- **Independente ao texto:** os sistemas independentes ao texto têm como objetivo verificar ou identificar um locutor a partir de qualquer amostra de sua voz, ou seja, não importando as palavras pronunciadas. A implementação desses sistemas apresentam um nível de dificuldade maior quando comparado ao tipo anterior.

Quanto aos locutores, podem ser [10]:

- **Cooperativos:** sabem que estão sendo reconhecidos, é o caso quando pronunciam palavras específicas ou não, mas falam de forma clara para ajudar o sistema de reconhecimento.
- **Não-cooperativos:** não sabem que estão sendo reconhecidos, sua fala não é condicionada para ajudar o sistema.

Quanto à qualidade, podem ser:

- **Alta:** sinal de voz, gravado em ambiente com pouco ou nenhum ruído, considerado limpo; a voz pronunciada sem erros e também não é afetada pelo estado de saúde de quem fala.
- **Variável:** sinal de voz, gravado em ambiente com razão/sinal variável, podendo ser corrompido por ruído aditivo e/ou convolucional (canal telefônico, meio de transmissão, cápsula do microfone, etc.), como também por problemas de saúde do locutor.

Geralmente os sistemas de reconhecimento de locutor são compostos basicamente por três etapas: a aquisição do sinal de voz, que consiste em converter o sinal analógico em digital através de um microfone; a extração das características do

sinal de voz pertinentes ao reconhecimento e o sistema classificador. Este último atua baseado em modelos de referência provenientes do treinamento, podendo utilizar modelos para simular condições de ruído, chamado *background*, ou falsos locutores, denominados impostores [9]. Ruído é considerado todo sinal sonoro que causa uma perturbação no sinal a ser reconhecido.

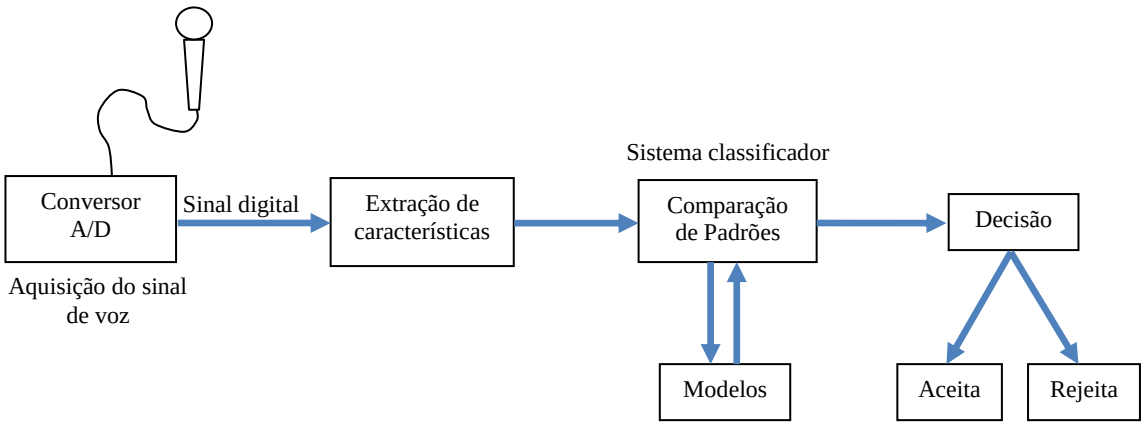

Figura 8 - Sistema genérico de reconhecimento de locutor

Na fase de aquisição, um microfone transforma a voz em um sinal elétrico (analógico) e depois um conversor analógico/digital o transforma em um sinal digital. Na etapa seguinte as características desejadas são extraídas gerando vetores de características que fornecerão os padrões para o classificador. Este por sua vez separa os modelos, provenientes do treinamento, em classes distintas. Na identificação de locutor, todos os modelos treinados são avaliados com uma locução de testes e avaliados. O modelo que obtiver o melhor resultado é aceito como verdadeiro, ou seja, a locução de teste pertence ao locutor cujo modelo foi vitorioso. Na verificação, o modelo de um suposto locutor determinará se a locução de teste pertence ou não ao locutor. Em seguida, na fase final, a decisão é baseada em limiares pré-estimados, de acordo com os valores apresentados pelo classificador [9]. A seguir serão dados mais detalhes sobre algumas etapas do processo de reconhecimento.

3.2. EXTRAÇÃO DE CARACTERÍSTICAS

O termo análise do sinal de voz consiste em extrair as características relevantes da voz, realizando uma compressão do sinal para uma posterior transmissão,

armazenamento reconhecimento ou síntese. Estas características devem atender, na medida do possível, as seguintes condições [12]:

- Eficiência da representação da informação do locutor.
- Devem ser facilmente determinadas e estáveis ao longo do tempo.
- Devem ocorrer frequentemente e naturalmente na voz.
- Não devem ter grandes variações quando gravado em outros ambientes.
- Não deve ser facilmente imitadas por outra pessoa.

No entanto, tais quesitos, na prática são quase impossíveis de serem satisfeitos pelas características. Porém para alguns tipos de aplicações é aceitável o não cumprimento de algumas delas, como por exemplo, o reconhecimento de locutor.

Partindo disto, o processo de extração funciona da seguinte forma. Primeiramente especifica-se o tamanho da janela que, geralmente, varia entre 10 a 40ms e depois o tamanho do incremento. Na segunda etapa deve-se iniciar o "janelamento", que consiste em, a partir do início do sinal sonoro, capturar um trecho do áudio do tamanho da janela, extrair as características deste segmento, avançar no tempo o tamanho do incremento e assim capturar a próxima janela até percorrer todo o áudio. A figura abaixo ilustra este processo.

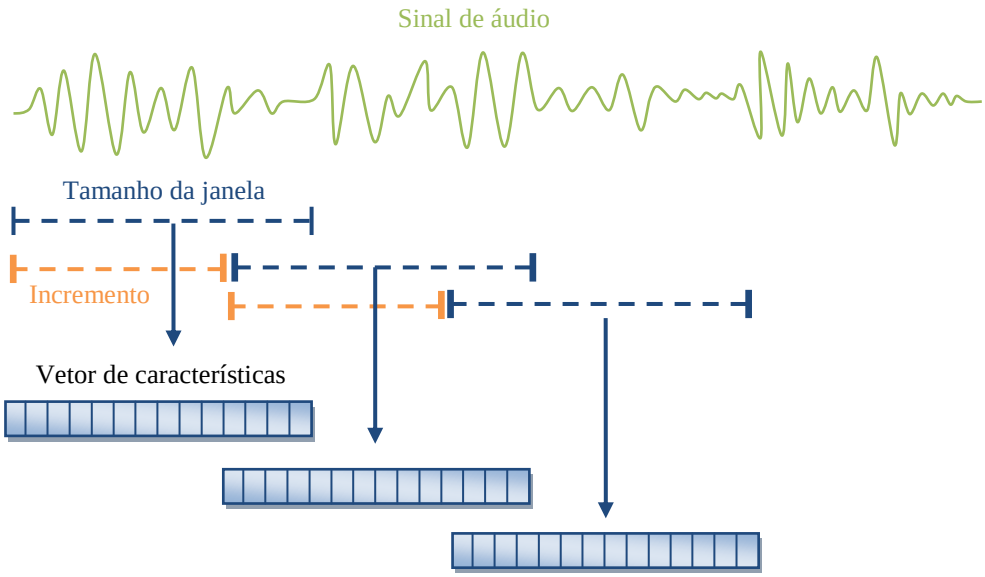

Figura 9 - Ilustração do "janelamento" do áudio

3.3. SISTEMAS DE CLASSIFICAÇÃO

A partir das características extraídas do sinal de voz de um locutor gera-se e armazena-se seu modelo. Este processo é repetido para cada novo locutor que for cadastrado no sistema. Após esta etapa, o sistema de classificação trabalha de duas maneiras, a depender da aplicação. Para a verificação de locutor o classificador compara os valores do sinal de entrada com o modelo do propenso locutor, assim podendo aceita-lo ou rejeitá-lo. Já na identificação o modelo que apresentar maior semelhança indica a quem pertence o sinal de entrada [9].

Os dois tipos de modelos mais usados são chamados de modelos estatísticos e modelo baseado no casamento de padrões característicos. No primeiro tipo o sistema de classificação é probabilístico que se propõe medir o grau de semelhança mediante observação do modelo, tomando-se como fato que a observação é uma cópia imperfeita do modelo armazenado. No segundo tipo, modelos baseado no casamento de padrões característicos, o sistema de classificação faz comparações. Outro modelo é fornecido pelas redes neurais, consideradas “aproximadoras” universais de funções, porém não se enquadram adequadamente nos tipos acima. A seguir serão citados alguns modelos.

- Modelos estatísticos: Modelo Escondido de Markov (Hidden Markov Model - HMM) e Modelo de Misturas Gaussianas (Gaussian Mixture Models - GMM).
- Modelos Baseados no Casamento de Padrões Característicos: Média de Longo Termo, Alinhamento Temporal Dinâmico (Dynamic Time Warping - DTW), Quantização Vetorial (Vector Quantization) e Modelo Autoregressivo Vetorial.
- Redes Neurais (RN)

4. O APLICATIVO PROPOSTO

4.1. DESCRIÇÃO DO APLICATIVO

O sistema proposto foi criado para reconhecer os participantes de uma conversa ou reunião através do reconhecimento de locutor e segmentação do sinal de áudio a fim de descobrir em que momento cada participante fala. No entanto, este foi desenvolvido principalmente para ser utilizado em reuniões presenciais de projetos.

O *software* em questão pode ser dividido em quatro módulos. O *player* de áudio e vídeo; a ferramenta que reconhece os locutores e segmenta o áudio; uma base de dados que armazena dados dos locutores, e as interfaces, as quais são responsáveis pela comunicação entre os módulos e por facilitar o uso do aplicativo. Os dois primeiros são considerados os componentes principais. Outro item que merece atenção é o configurador do sistema, pois é através dele que são definidos parâmetros referentes à ferramenta de reconhecimento de locutor, aos componentes do HTK e ao segmentador de áudio. Também são passados alguns diretórios de arquivos criados pelo sistema.

Sendo assim há quatro requisitos básicos para seu funcionamento. Ter os componentes do HTK, citados na seção 2.5, em um diretório local; criar um arquivo de texto que armazenará as configurações do sistema através da definição de algumas variáveis; criar um banco de dados para armazenar as informações dos locutores e por fim inserir a DLL da biblioteca Fobs4JMF na pasta de arquivos do Windows.

Cumprido os requisitos acima, o sistema está pronto para ser iniciado. O primeiro passo após abrir o programa é informar o caminho do arquivo de configuração para que as ferramentas estejam prontas para serem utilizadas. Feito isso o usuário poderá cadastrar os locutores no sistema ou carregar um áudio ou vídeo, de uma reunião, a ser processado. No caso do arquivo carregado ser um vídeo, seu áudio é automaticamente extraído para ser reconhecido e segmentado. Após o cadastramento dos locutores e carregamento da mídia, o usuário poderá iniciar uma sessão de reconhecimento informando ao sistema quais locutores, da base de dados, estão presentes na gravação. O passo seguinte consiste apenas em dar a ordem para o *software* processar o arquivo de entrada. Ao concluir esta fase os tempos de início e fim da fala de cada locutor são exibidos em uma tabela. Esta por sua vez permite ao usuário navegar na mídia através dos dados obtidos.

4.2. ANÁLISE DE REQUISITOS

Ao estudar os HTK e o processo de reconhecimento de locutor foi feito um levantamento de requisitos necessários para a estruturação das características que o *software* deve possuir. Dentre estas características podemos destacar que o usuário deve ser responsável por:

- Abrir mídia
- Configurar as ferramentas do sistema.
- Determinar os locutores a serem reconhecidos.
- Iniciar o reconhecimento do áudio de uma reunião.

A partir destas considerações iniciais e da descrição do aplicativo foi feito o diagrama de casos de uso (Figura 10), visto que o mesmo tem a principal finalidade de documentar as funcionalidades do sistema e seus relacionamentos com o ambiente externo, composto pelo usuário.

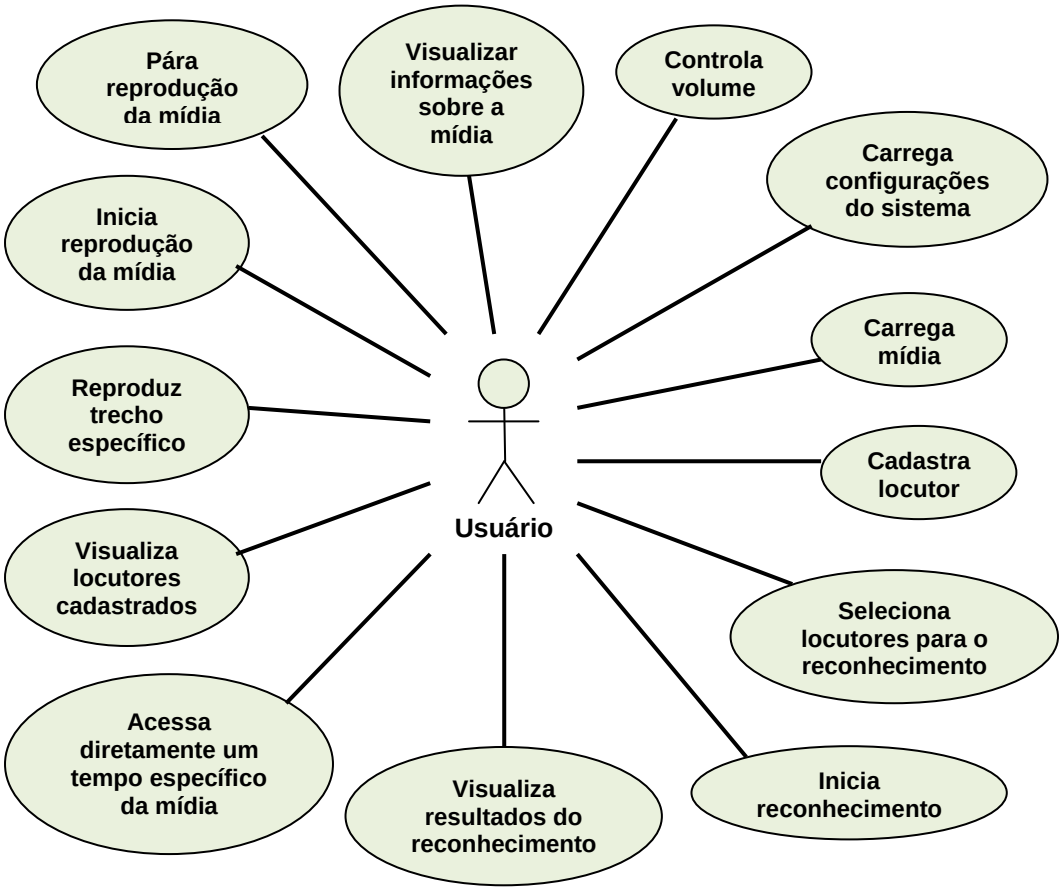

Figura 10 - Diagrama de Casos de Uso

A partir desta análise primária foi criada uma lista com os requisitos do sistema, para facilitar a concepção do projeto:

- **Requisitos Funcionais:**

[RF001] – O usuário deve carregar um arquivo de configuração.

[RF002] – O sistema deve ser capaz de reproduzir áudio e vídeo.

[RF003] – O usuário deve ser capaz de abrir arquivos de áudio e vídeo no sistema.

[RF004] – O usuário deve ser capaz de cadastrar locutores.

[RF005] – O usuário deve ser capaz de visualizar os locutores cadastrados.

[RF006] – O usuário deve ser capaz de selecionar os locutores a serem reconhecidos.

[RF007] – O usuário deve ser capaz de iniciar o reconhecimento de locutor e a segmentação do áudio.

- **Requisitos Não-Funcionais:**

[RNF001] – O áudio a ser processado deve estar em formato WAV.

[RNF002] – Para cadastrar um locutor é necessário ter um arquivo de áudio contendo amostras de sua voz.

[RNF003] – O arquivo de áudio do cadastro do locutor deve estar em formato WAV.

[RNF004] – Para reconhecer os locutores e segmentar o áudio deve ser necessário selecionar os locutores da base de dados que participarão do reconhecimento.

[RNF005] – O *software* deve ser operacionalizado em sistema Windows.

- **Requisitos Inversos:**

[RI001] – O processo de gravação da reunião não deve estar vinculado ao sistema.

[RI002] – A criação do arquivo de configuração não deve estar vinculado ao sistema.

4.3. PLANEJAMENTO E IMPLEMENTAÇÃO

A partir da análise realizada dos requisitos para o projeto, faz-se necessário detalhar todos os passos a serem realizados durante a implementação. Para iniciar este detalhamento é necessário um diagrama que dê a clara noção de como está organizado o projeto em questão. Podemos então representar esta organização através do diagrama de pacotes ilustrado pela figura 11.

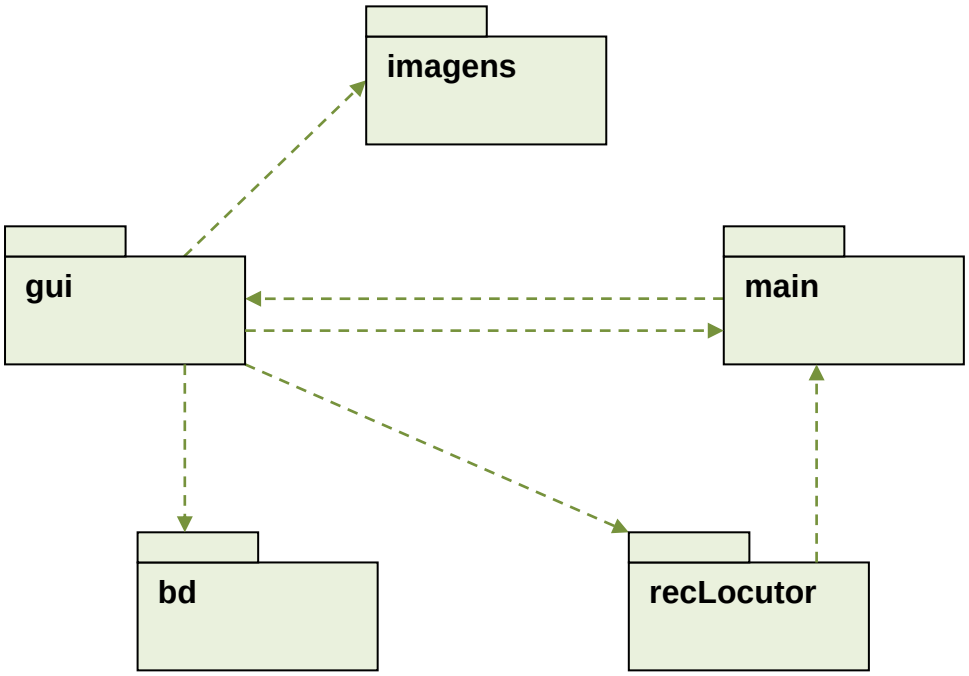

Figura 11 - Diagrama de Pacotes

O pacote definido por **main** possui a classe que dá início ao sistema e outras classes que consistem em estruturas para armazenar dados utilizados em outros pacotes. Armazena as classes:

- **Main.java:** somente dá início ao sistema chamando a classe a seguir.
- **SegmentoTableModel.java:** controla os dados da tabela que exibe os resultados do reconhecimento e da segmentação.
- **Segmento.java:** objeto usado pela classe anterior. Contém as variáveis que armazenam o tempo inicial e final de uma fala específica, o nome do locutor que a pronunciou, o texto que foi dito e o índice deste segmento.
- **ReadConfigFileSystem:** classe responsável por ler o arquivo de configuração do sistema.

- **LocutoresTableModel.java:** classe que controla os dados da tabela Cadastrados.
- **MyTableCellRender.java** e **MyTableMultilneCellRender:** criadas somente para personalizar a tabela.

O pacote **reclocutor** pode ser considerado o mais importante do projeto, pois é responsável por todo o processo de reconhecimento de locutor e segmentação de áudio. Neste pacote encontram-se as seguintes classes:

- **HTK.java:** desenvolvida no padrão *Singleton*, o qual garante a existência de apenas uma instância da classe. Responsável por controlar e armazenar todos os componentes e informações referentes ao HTK.
- **Session.java:** controla a sessões de reconhecimento através dos métodos que executam o reconhecimento e a segmentação do áudio, e os que criam os modelos dos locutores e a listas dos locutores a serem reconhecidos. Para a construção desta classe também foi adotado o padrão *Singleton*.
- **ReadHTK.java:** efetua a leitura dos arquivos "mfcc" gerados pelo HCopy.
- **HTKheader.java:** objeto que armazena o cabeçalho de um arquivo "mfcc".
- **MyMatrix.java:** objeto que armazena a matriz de dados de um arquivo "mfcc". Além disso, prove métodos para manipular esta matriz.
- **ReadMLF.java:** responsável por ler os arquivos MLF.
- **TempMFCC.java:** cria um arquivo MFCC temporário contendo um trecho específico do arquivo MFCC do áudio da reunião.
- **Result.java:** objeto que armazena o ID de um locutor e sua probabilidade. Dados estes lidos do arquivo MLF.
- **ResultVector.java:** vetor, do tipo Result, usado para armazenar todos os dados provenientes da segmentação do áudio.
- **ResultVectorFile.java:** classe usada na fase de testes, que tem como função criar um arquivo texto com todo conteúdo de um ResulVector.

O pacote denominado **gui**, abreviação de *Graphical User Interface*, simplesmente armazena as interfaces do sistema. São elas:

- **VideoPlayer.java:** é a classe principal do projeto, sendo a única a utilizar as bibliotecas citadas no item 2.1 deste trabalho. Além disso, controla o *player* do

aplicativo; as chamadas das interfaces dos métodos de reconhecimento e segmentação, e os resultados finais.

- **Cadastrar.java:** interface que trata o cadastro de locutores.
- **Cadastrados.java:** interface usada para mostrar os locutores cadastrados no banco de dados.
- **Configuracao.java:** interface que exibe as configurações do sistema.
- **SelecaoLocutores.java:** interface que permite o usuário selecionar os locutores a serem reconhecidos.

O quarto pacote, chamado **bd**, é responsável pelas transações entre a interface e o banco de dados. Contém as classes:

- **ConnectionFactory.java:** responsável por criar uma conexão com o banco de dados.
- **PessoaDAO.java:** contém algumas funcionalidades de banco, tais como, obter conexão através da fabrica de conexões; inserir, selecionar, remover dados; contar linhas e gerar ID.
- **Pessoa.java:** objeto usado para armazenar os seguintes dados de um locutor: ID, nome, nome da empresa em que trabalha, e diretório absoluto de seu áudio, arquivo MFCC e modelo.

O banco de dados tratado por este pacote, foi criado no MySQL e consiste em apenas uma tabela denominada PESSOA, com o as seguintes colunas:

- ID – INT – Chave primária - (ID do locutor iniciando em 101).
- NOME – VARCHAR(45) – NOT NULL – (nome do locutor)
- EMPRESA – VARCHAR(45) – NOT NULL – (nome da empresa representada)
- AUDIO – VARCHAR(100) – NOT NULL – (diretório absoluto do áudio da amostra de voz do locutor)
- MFCCFILE – VACHAR(100) – NOT NULL – (diretório absoluto do arquivo MFCC do áudio do locutor)
- MODELO – VARCHAR(45) – NOT NULL – (nome do modelo do locutor, formado por: “mod_<ID>”)

Por fim há o último pacote, nomeado **imagens**, criado apenas para armazenar os ícones dos botões do sistema.

Para melhor entendimento sobre o processo de reconhecimento de locutor e segmentação aplicados ao projeto, serão apresentados três métodos, da classe **Session.java**, invocados ao longo dos procedimentos em questão. Sempre que um locutor for cadastrado este método é chamado com a finalidade de gerar o arquivo MFCC, o modelo do locutor, e inserir o nome do modelo criado na lista de modelos e adicionar no dicionário. Este método recebe dois parâmetros, o diretório absoluto do áudio e o nome dos arquivos criados, como é mostrado no código abaixo, já comentado.

```
public void makeModelSpeaker(String strAudioFilePath, String
strOutputName)
{
    ...
    htk.runHCopy(new File(strAudioFilePath), strOutputName);
    htk.runHinit();
    htk.runHRest();

    // salva na lista o nome do modelo do locutor cadastrado.
    // A lista é criada se não existir
    htk.createMainList(htk.getModelName());

    // salva o nome do modelo no dicionário. O dicionário é
    // criado se não existir
    htk.createDictionary();
    ...
}
```

O próximo método é chamado quando o usuário ordena o reconhecimento, que por sua vez é a tarefa mais complexa da aplicação e é dividida em duas etapas, o pré-processamento e o processamento em si, que consiste em várias iterações de reconhecimento: na primeira etapa deve-se criar a lista contendo o nome dos modelos a serem reconhecidos; especificar o diretório absoluto dos arquivos que contêm, respectivamente, os dados da rede, a rede e o resultado de uma iteração; calcular o número de iterações de reconhecimento, e finalizando esta etapa criando o vetor que armazenará o resultado de todas as iterações.

```
public void reconhecer()
{
    TempMFCC tempMFCC = new TempMFCC();
    ReadMLF readMLF = new ReadMLF();

    createSessionList();

    String strOutputPath = htk.getOutputPath();
    htk.setDataNetFilePath(strOutputPath + "\\DataNet");
    htk.setNetFilePath(strOutputPath + "\\Net");
    htk.setResultHViteFilePath(strOutputPath+"\\Result.mlf");

    tempMFCC.readMFCCsessionFile();
    int nPassos = tempMFCC.getNumPassos();
    resultVector = new ResultVector(nPassos);
    ...
```

Na segunda etapa deve-se criar a rede e iniciar as iterações. Cada repetição é dada pelas seguintes instruções: criar um arquivo MFCC temporário, contendo uma pequena parte dos dados do arquivo MFCC da reunião; executar o HVite; ler o arquivo MLF e salvar seus dados no vetor de resultados.

```
// continuação do método reconhecer
...
htk.createNet();
String strTemMFCCfilePth = "";
String strMLFfilePath = "";
for(int i=0; i<nPassos; i++)
{
    tempMFCC.createTempMFCC();
    strTemMFCCfilePth = tempMFCC.getTempMFCCfilePath();
    htk.setTempMFCCfilePath(strTemMFCCfilePth);
    htk.runHVite();

    strMLFfilePath = htk.getResultHViteFilePath();
    readMLF.readFile(strMLFfilePath);
    resultVector.setElement( readMLF.getIdLocutor(),
        readMLF.getLikelihood(), i );
}
...
```

O terceiro método é responsável pela segmentação dos locutores, obtendo o tempo de inicio e fim da fala de cada locutor reconhecido, a partir do vetor de resultados. Para isso é necessário percorrer o vetor de resultados identificando os

índices inicias e finais de uma sequência de ID's repetidos, como será mostrado no trecho de código abaixo.

```
...
for(int i=0; i<resultVector.lenght()-1; i++)
{
    ID = resultVector.getID(i);
    strModelo = htk.MODEL_BEGIN_NAME + ID;

    // captura o tempo de inicio de uma segmento
    timeBegin = i * unidadeTempo;

    // percorre o vetor até encontrar um ID diferente
    while( (i+1<resultVector.lenght()) && (ID ==
    resultVector.getID(i+1)) )
        i++;

    // captura o tempo de termino de uma segmento
    timeEnd = i*unidadeTempo;

    // salva o segmento
    segmento = new Segmento();
    segmento.setIndice(ind++);
    segmento.setNomeLocutor( arrayLocutoresSelecionados.get(
        arrayModelosSelecionados.indexOf(strModelo)) );
    segmento.setTempoInicio(timeSecondsToString(timeBegin) );
    segmento.setTempoFim( timeSecondsToString(timeEnd) );
    segmento.setTexto("Texto falado...");

    segmentos.add(segmento);
}
...
```

Antes de iniciar o detalhamento do programa é necessário apresentar como arquivo de configuração do sistema é elaborado e as variáveis que são passadas através dele, o qual há dois tipos de dados, numéricos e diretórios, que deve estar no seguinte formato:

<NOME DA VARIÁVEL> = < "DIRETÓRIO" >

<NOME DA VARIÁVEL> = <VALOR NUMÉRICO>

As variáveis são:

- **HTK:** indica o diretório que contém os componentes do HTK, portanto é obrigatória.

- **CONFIG:** indica o diretório absoluto do arquivo de configuração usado pelo HCopy. É obrigatória.
- **PROTOGMM:** indica o diretório absoluto do arquivo PROTOGMM usado pelo HInit. Obrigatória.
- **LIST:** indica o diretório absoluto da lista que armazena o nome de todos os modelos cadastrados. Seu uso é opcional.
- **DIC: i**ndica o diretório absoluto do dicionário usado no reconhecimento. Obrigatória.
- **MFCC:** indica o diretório onde serão armazenados os arquivos MFCC's. É opcional, entretanto, se não for definida o arquivo MFCC será criado no mesmo diretório do áudio.
- **MODEL:** indica o diretório onde os modelos gerados serão armazenados. Obrigatório
- **OUTPUT:** indica o diretório onde serão armazenados os arquivos MLF, os arquivos que contém os dados da rede, a rede, o vetor de resultados usado nos testes e o MFCC temporário.
- **NBEST: i**ndica o valor numérico inteiro positivo da variável de reconhecimento N-best.
- **TARGETRATE:** indica o tamanho da janela do "janelamento" na extração de características em unidade de 100 milissegundos.
- **FRAME:** tamanho da janela de reconhecimento, em unidade de 100 milissegundos.
- **STEP:** tamanho do passo entre o início de duas janelas de reconhecimento, em unidade de 100 milissegundos.

4.4. DO PRODUTO FINALIZADO

O produto finalizado apresenta uma divisão em cinco telas. A primeira a ser exibida é a tela principal (figura 12), a qual contém o player e exibe os resultados do reconhecimento, além de fornece ao usuário acesso a todas as principais funcionalidades do sistema, tais como: abrir mídia, cadastrar locutor, selecionar locutores para

reconhecimento, visualização da base de dados, visualização das configurações e exibição de um trecho do vídeo.

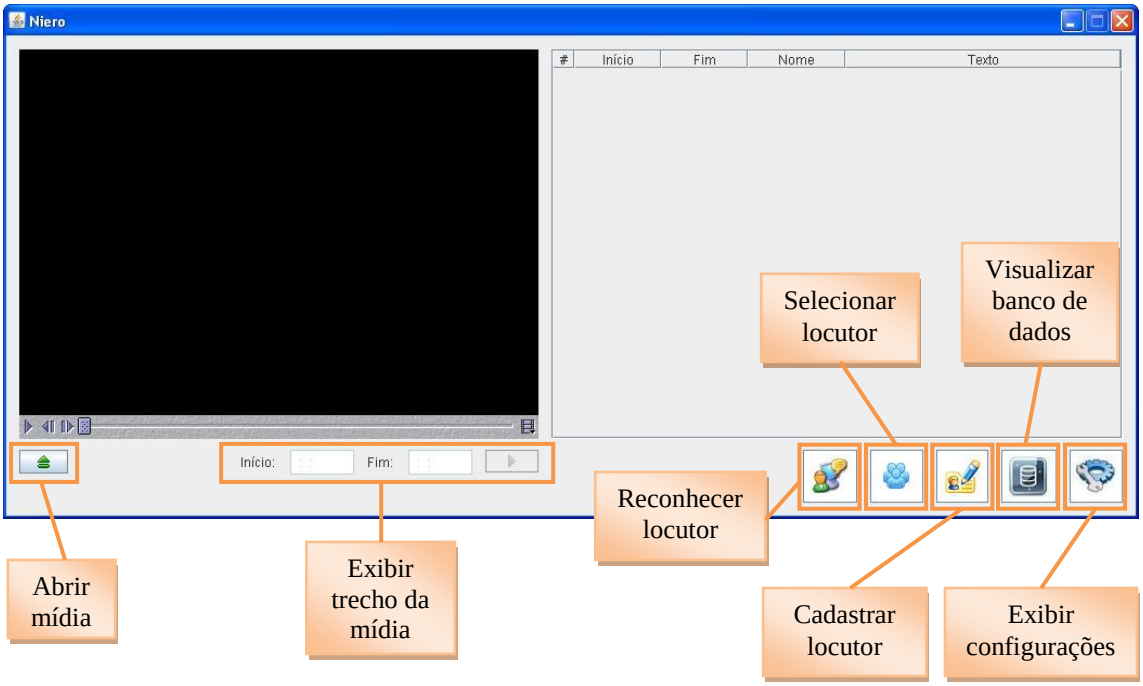

Figura 12 - Tela principal do sistema

A segunda tela (figura 13) permite ao usuário visualizar e importar um arquivo texto contendo as configurações do sistema.

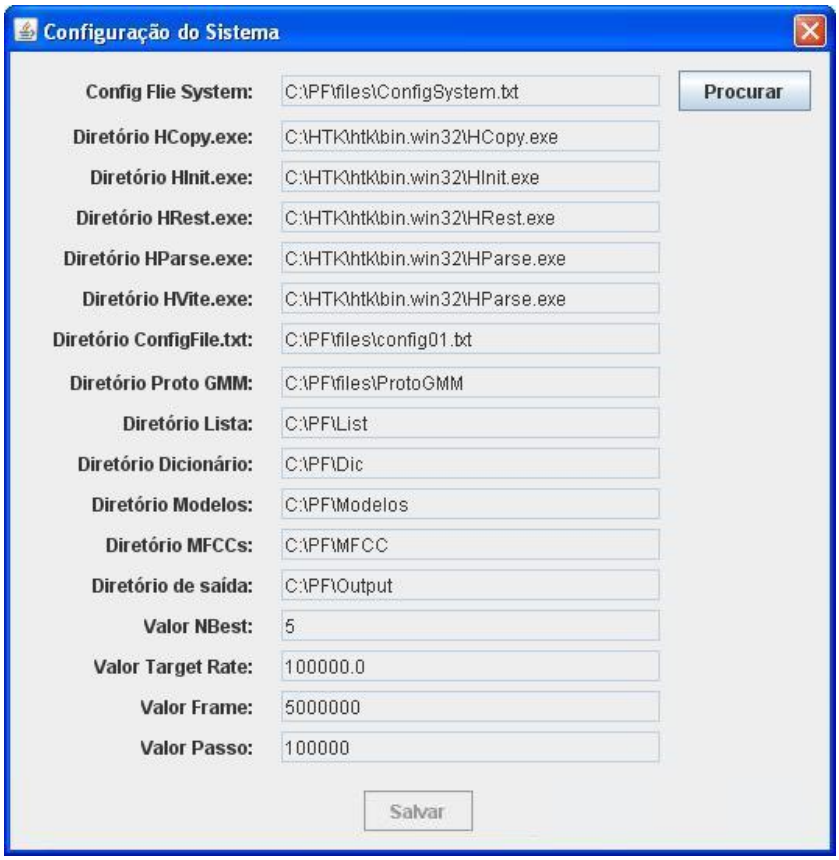

Figura 13 - Visualização e carregamento das configurações do sistema

Para cadastrar um locutor é necessário fornecer seu nome, a empresa em que ele trabalha, e um áudio contendo somente a voz do novo locutor. Estes campos estão contidos na terceira janela do aplicativo (figura 14).

Figura 14 - Cadastro de locutor

A quarta tela do programa desenvolvido permite o usuário consultar os dados de todos os locutores cadastrados. Estes dados podem ser visualizados através da figura 15.

Figura 15 - Visualização dos locutores cadastrados

A seleção de locutores para o reconhecimento é realizado através da quinta janela (figura 16). Esta permite o usuário visualizar os nomes dos locutores cadastrados e selecionar aqueles que estão presentes em uma reunião a ser segmentada.

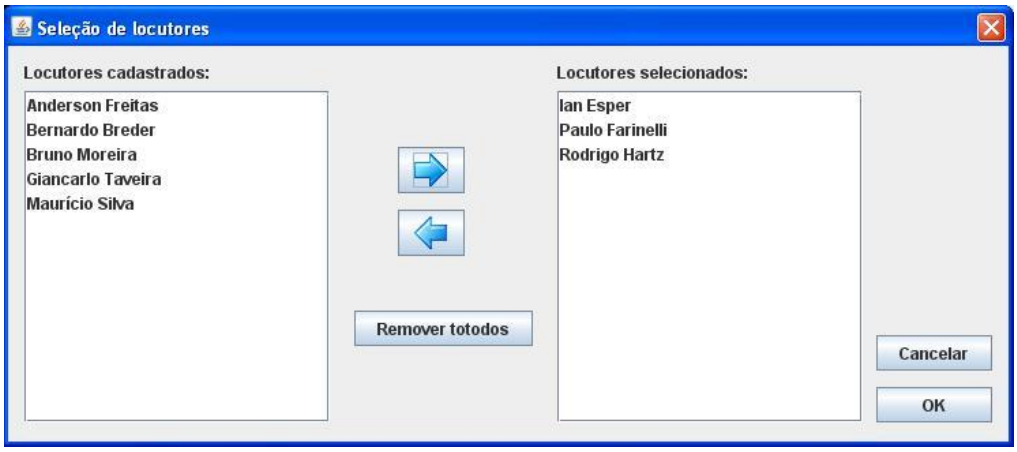

Figura 16 - Seleção de locutores para o reconhecimento

4.5. RESULTADOS

Após o término da parte de implementação do aplicativo proposto, foi possível submetê-lo ao processo completo de testes, desde a configuração do sistema até a etapa de reconhecimento e segmentação do áudio de uma reunião, a fim de avaliar seu desempenho.

Na fase de reconhecimento foram submetidos áudios de 10 e 57 segundos de duração. Ambas as gravações possuíam vozes de dois locutores cadastrados no sistema, sendo que, o áudio de menor duração consistiu em um dialogo dinâmico, ou seja, com curtos períodos de silêncio. Já o áudio de maior duração consistiu em um dialogo mais pausado, isto é, com longos períodos de silêncio. Além disso, é importante ressaltar que a duração da janela de extração de características era de 10ms (variável **target rate**), a janela de reconhecimento era de 500ms (variável **frame**), e o incremento de 10 ms (variável **step**).

Sendo assim, a figura 17 exibe o resultado obtido após o processamento do áudio de 10 segundos de duração, destacando com um circulo vermelho o segmento incorreto.

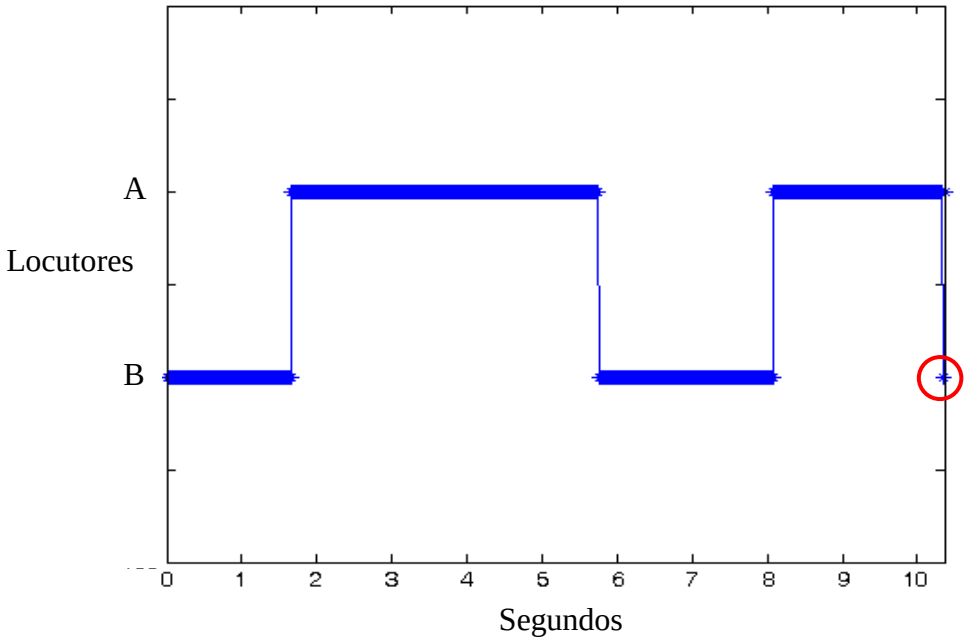

Figura 17 - Reconhecimento e segmentação do áudio de 10s

Já a figura 18 exibe o resultado obtido após o processamento do áudio de 57 segundos de duração, destacando em vermelho os segmentos incorretos e em verde o início de cada fala.

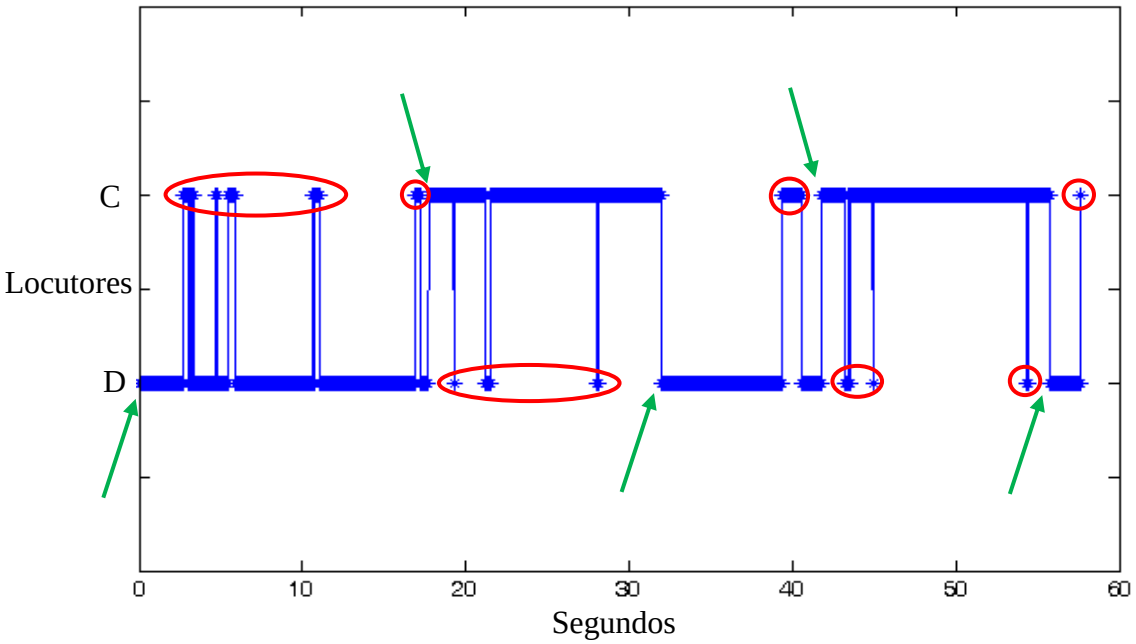

Figura 18 - Reconhecimento e segmentação do áudio de 57s

De acordo com os resultados obtidos foi possível perceber que a tecnologia de reconhecimento de locutor ainda não é perfeita, pois em alguns casos ela não consegue identificar corretamente um locutor. Este caso pode ser encontrado quando duas ou mais pessoas falam durante o mesmo tempo, quando há excesso de ruído ou longos períodos de silêncio, prejudicando o desempenho do reconhecimento. Porém, para algumas destas situações, já existem algoritmos que compensam estas falhas, como, por exemplo, métodos que detectam e removem períodos de silêncio e outros que amenizam os ruídos, todos eles usados antes do início do reconhecimento.

Outro fato observado nos testes refere-se à relação entre às variáveis STEP e FRAME, do arquivo de configuração do sistema, a duração dos períodos de silêncio e a probabilidade de acerto da segmentação. Quanto maior a duração desses períodos, maior deve ser o valor destas variáveis, ou, quando menor a duração dos períodos de silêncio, menor devem ser o valor das variáveis para que a probabilidade de acerto da segmentação seja elevada. Desta forma, é possível afirmar que o valor das variáveis e da duração dos períodos de silêncio tem uma relação diretamente proporcional com a probabilidade de acerto.

Por fim, é importante destacar a relação entre o áudio da reunião a ser processada com a duração do processo de reconhecimento executado pelo sistema. O *software* foi testado com áudio de diversos tamanhos. Para todos notou-se que o tempo de reconhecimento e segmentação do áudio demora, aproximadamente, o triplo do tempo de duração do áudio da reunião.

5. CONCLUSÕES E TRABALHOS FUTUROS

5.1. CONCLUSÕES

Foi possível visualizar como a ferramenta HTK pode ser usada para criar uma aplicação de verificação de locutor e segmentação de áudio, além de perceber como conhecimentos de processamento de áudio podem ser utilizados na construção de ferramentas que auxiliem a manipulação de informações provenientes de reuniões presenciais.

Portanto, pode-se concluir que mesmo sem o uso de atividade de pré-processamento, como por exemplo, a detecção e eliminação de silêncio e a redução de ruídos, foi possível obter resultados satisfatórios em ambientes limpos e áudios com curtos períodos de silêncio.

5.2. TRABALHOS FUTUROS

Os trabalhos futuros têm dois focos principais em relação ao aplicativo apresentado. O primeiro está na inserção de atividades de pré-processamento, a fim de aumentar a probabilidade de acerto. Já o segundo foco é diminuir o tempo de processamento substituindo o HTK por alguma outra ferramenta mais eficiente seja ela desenvolvida em um trabalho futuro ou já existente.

6. REFERÊNCIAS

[1] Java SE Desktop Technologies
http://www.oracle.com/technetwork/java/javase/tech/index-jsp-140239.html
Acessado em: 11 de Novembro de 2010

[2] Open Source Projects
http://fobs.sourceforge.net/index.html
Acessado em: 11 de Novembro de 2010

[3] FFMPEG
http://www.ffmpeg.org/index.html
Acessado em: 12 de Novembro de 2010

[4] Sauron Software Projects Repository
http://www.sauronsoftware.it/projects/jave/
Acessado em: 12 de Novembro de 2010

[5] HTK3
http://htk.eng.cam.ac.uk/
Acessado em: 12 de Novembro de 2010

[6] The HTK Book (for HTK Version 3.4)

[7] CAMPBELL, Joseph P., Jr. Biometrics Personal Identification in Networked Society. Springer US. Abril 2006.

[8] SOTOMAYOR, César Augusto Medina. **Realce de Voz á Verificação Automática de Locutor.** Dissertação (Mestrado em Ciências) – Instituto Militar de Engenharia, 2003.

[9] LIMA, Charles Borges de Lima. **Sistemas de Verificação de Locutor ndependente do Texto Baseados em GMM e AR-Vetorial Utilizando PCA.** Dissertação (Mestrado em Ciências) – Instituto Militar de Engenharia, 2001.

[10] SOUZA, Ricardo Honório Guedes de. **Estudo de Características Relevantes do Sinal de Voz para o Reconhecimento Automático de Locutor Desprevenido, Independente ao Texto**. Dissertação (Mestrado em Ciências) – Instituto Militar de Engenharia, 1996.

[11] Interface Controller – JMF API Docs
http://download.oracle.com/docs/cd/E17802_01/j2se/javase/technologies/desktop/media/jmf/2.1.1/apidocs/javax/media/Controller.html
Acessado em 25 de Novembro de 2010.

[12] ATAL, B. S. Automatic Recognition of Speakers from Their Voices. Proceedings of the IEEE, 1976.

[13] RABINER L. R. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. Proceedings of the IEEE. Fevereiro 1989.