Instituto de Ciências Matemáticas e de Computação

ISSN - 0103 - 2569

ESTUDO E DESENVOLVIMENTO DE APLICAÇÕES JAVA COM RECONHECIMENTO E SÍNTESE DE VOZ

José Fernando Rodrigues Júnior
Prof. Dr. Dilvan de Abreu Moreira

RELATÓRIOS TÉCNICOS DO ICMC

São Carlos
Março/2001

ESTUDO E DESENVOLVIMENTO DE APLICAÇÕES JAVA COM RECONHECIMENTO E SÍNTESE DE VOZ*

José Fernando Rodrigues Júnior
Prof. Dr. Dilvan de Abreu Moreira

Universidade de São Paulo
Instituto de Ciências Matemáticas e de Computação
Departamento de Ciência de Computação e Estatística
e-mail: {junio, dilvan}@icmc.sc.usp.br

Resumo

Justifica-se o estudo e a prática de técnicas de Reconhecimento/Síntese de Voz pelo fato desta área ser o marco principal das novas interfaces computacionais que atualmente se apresentam como a vanguarda da computação. Graças ao enorme aumento da capacidade de processamento juntamente com o aperfeiçoamento de técnicas de Inteligência Artificial, este novo meio de interação computador-usuário tornou-se possível e tende a ser a base dos sistemas computacionais da próxima geração.

Hoje, a possibilidade de se comunicar verbalmente com a máquina já não é mais um sonho, é uma realidade que pode influenciar violentamente a produtividade e revolucionar a interação homem-máquina. O estudo e domínio desta tecnologia devem ser efetuados de forma precoce, possibilitando a implantação de sistemas desta natureza sistematicamente e antecipadamente.

O projeto é centrado no Reconhecimento e Síntese de Voz baseado na utilização de uma nova tecnologia denominada ViaVoice, pesquisada e desenvolvida pela IBM durante os últimos 30 anos. As funcionalidades de Reconhecimento/Síntese de voz são acessadas através de uma API denominada SDK composta de três pacotes Java que possibilitam o desenvolvimento de aplicações que oferecem interface baseada neste novo paradigma: a voz. O projeto prevê uma análise aprofundada deste software seguida do desenvolvimento de aplicações baseadas na tecnologia.

Março 2001

***Trabalho realizado com auxílio da Fapesp**

1 Índice dos tópicos

1. Índice dos Tópicos....2
2. Familiarização com o software ViaVoice....3
2.1 Instalação....4
2.2 Configuração....6
2.3 Interface....7
2.4 Performance....7
2.5 Suporte....8
3. Análise dos pacotes Java para reconhecimento de voz....8
3.1 A Central de processamento e a aplicação Engine....9
3.2 Síntese....10
3.2.1 O papel da JSML....10
3.2.2 Propriedades do sintetizador....11
3.2.3 Redefinir as propriedades ou utilizar JSML....12
3.2.4 Interatividade....12
3.2.5 A interface Speakable....13
3.3 Reconhecimento....13
3.3.1 DictationGrammar....14
3.3.2 RuleGrammar....15
3.3.3 Comparando as gramáticas....16
3.3.4 Os Resultados e os Ouvidores (Listeners) de Resultados....16
3.3.5 Eventos de áudio....18
3.3.6 Eventos do motor (Engine)....18
3.4 Visão geral....19
4 Desenvolvimento de aplicativos com a tecnologia....22
4.1 O projeto Voice mp3 Player....22
4.1.1 A interface gráfica....23
4.1.2 A interface via voz....24
5. Elaboração e desenvolvimento do tutorial....31
5.1 Conteúdo....32
5.2 Design....37
6. Revisão dos pacotes Java para reconhecimento de voz....40
6.1. O nível de confiabilidade do reconhecedor....40
6.2. O nível de confiabilidade do reconhecedor....40
6.3. A captura de áudio do reconhecedor....41
6.4. Estatísticas de reconhecimento....42
7. A Aplicação “Verificador de Pronúncia”....43
7.1. O Funcionamento....43
7.2. A abrangência da aplicação....44
8. Participação do VIII Simpósio Internacional de Iniciação Científica da USP....47
9. Apresentação final do “Tutorial Speech Development Kit Java Technology ”….....48

Apêndice A - Java Speech Markup Language……..…49
Apêndice B - Java Speech Grammar Format.....50
Apêndice C - Dados utilizados para a coleta de estatísticas do reconhecimento.....52

2 Familiarização com o software ViaVoice

Esta primeira tarefa consistiu em adquirir o software necessário para a realização do projeto: o pacote de software Java e o software base do ViaVoice. Com o material disponibilizado, passamos para a fase de instalação, o que demandou a realização de alguns testes já que a documentação oferecida pela IBM tanto em seu site http://www.ibm.com/software/speech como na documentação que acompanha a distribuição não especificam os detalhes do processo de instalação como um todo. Não há um esclarecimento sobre como o pacote de software Java utiliza-se do ViaVoice, nem como deve ser feita a configuração para estabelecer a interação entre eles.

Com o software instalado foi realizada a análise da interface, no caso do pacote do ViaVoice foi necessário experimentar todos os executáveis que vêm junto com a distribuição já que não há documentação alguma sobre isto. A avaliação da performance foi realizada ministrando-se um conjunto de palavras para o reconhecimento, anotando-se os erros e os acertos. Para o teste da síntese simplesmente definiram-se algumas frases para que o computador reproduzisse.

2.1 Instalação

A API de desenvolvimento IBM Speech Development Kit Java Technology, objeto de estudo da presente pesquisa, pode ser conseguida em : http://www.alphaworks.ibm.com/aw.nsf/frame?ReadForm&/aw.nsf/techmain/AF74C5941F557778882566F300703F5B na versão Linux e em http://www6.software.ibm.com/dl/vvsdk/vvsdk-p para a versão Windows. Ambas distribuídas totalmente grátis (para fins acadêmicos) em um único arquivo comprimido. A instalação é realizada através da descompressão dos arquivos da distribuição em um único diretório. Após o descarregamento dos pacotes teremos acrescentado a seguinte subárvore ao diretório escolhido:

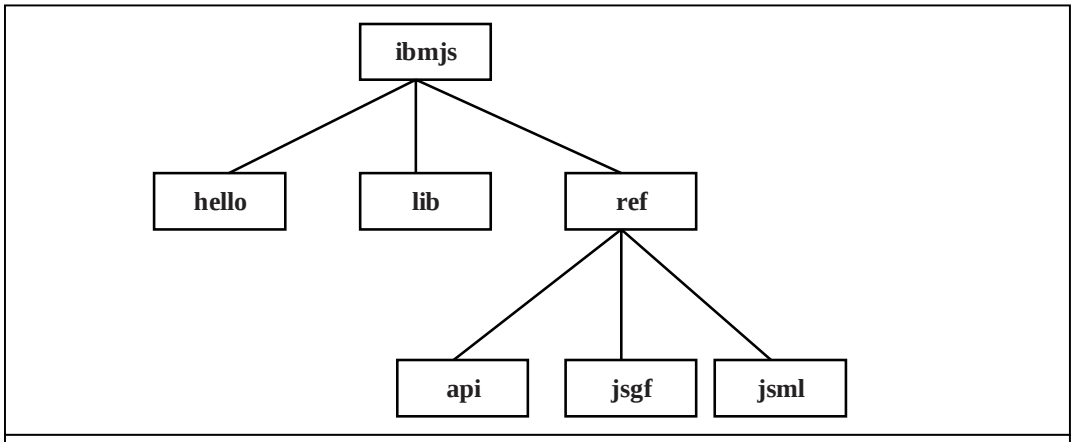

Figura 1: árvore de diretórios da distribuição do IBMSDK Java Technology.

Que correspondem a:

Diretório	Conteúdo Windows	Conteúdo Linux
../ibmjs	Subdiretórios segundo a figura + arquivo de lote (install.bat) para configuração das variáveis de sistema PATH e CLASSPATH e registro do software junto ao Sistema Operacional	Idem, no entanto o arquivo de lote trata-se de um script shell (install.sh) e a variável PATH corresponde à variável LD_LIBRARY_PATH
../ibmjs/hello	Exemplo de aplicação incluindo código fonte	Idem
../ibmjs/lib	Bibliotecas de interface IBMSDK/ViaVoice: ibmreco.dll, ibmreco_g.dll, ibmsynth.dll, ibmsynth_g.dll e vtbnfcjs.dll + classes da API Java (ibmjs.jar)	Bibliotecas de interface IBMSDK/ViaVoice: audjs.so e libibmreco.so + Classes da API Java (ibmjs.jar)
../ibmjs/ref	Subdiretórios segundo a figura, contendo informação de referência para programação	Idem
../ibmjs/ref/api	Documentação da API gerada no padrão Javadoc	Idem
../ibmjs/ref/jsgj	Documentação da linguagem de marcação Java Speech Grammar Format	Idem
../ibmjs/ref/jsml	Documentação da linguagem de marcação Java Speech Markup Language	Idem

Tabela 1: conteúdo do arquivo de distribuição do IBMSDK Java Technology

As bibliotecas utilizam-se de um software base para realização do reconhecimento e síntese de voz. Este software é o mesmo utilizado na distribuição do software comercial ViaVoice Millenium e denomina-se ViaVoice Runtime Dictation. Podemos encontrá-lo em http://www6.software.ibm.com/dl/viavoice/runtime-p na versão Windows e em http://www6.software.ibm.com/dl/viavoice/linux-p para a versão Linux.

A instalação destes softwares é direta sendo a versão Windows um executável responsável por todo o processo, a versão Linux é distribuída em formato rpm, que realiza toda a instalação através do comando:

rpm –ivh ViaVoice_runtime-3.0-1.1.i386.rpm

Como se trata de programação em Java é necessário ter a instalação prévia do Java Development Kit 1.1.7 ou superior, ou um similar de outro fabricante.

2.2 Configuração

Após a instalação alguns ajustes devem ser realizados para que as aplicações consigam utilizar as funcionalidades do pacote de reconhecimento e mesmo para que possam ser compiladas. Apesar dos arquivos de lote destinados à instalação realizarem a atualização destas variáveis, não o fazem permanentemente, é necessária a redefinição das variáveis de sistema nos arquivos de lote da inicialização do sistema operacional:

Sistema Operacional	Arquivo de inicialização	Ajuste para os arquivos binários	Ajuste para a biblioteca Java ibmjs.jar
Windows	autoexec.bat	deve-se acrescentar o caminho completo do diretório ibmjs/lib à variável de sistema PATH para que os arquivos binários lá presentes possam ser acessados.	deve-se acrescentar o caminho completo do arquivo ibmjs/lib/ibmjs.jar à variável de sistema CLASSPATH para a compilação e execução dos aplicativos que serão desenvolvidos.
Linux	.bashrc, dependendo da versão deve-se verificar a documentação	deve-se acrescentar o caminho completo do diretório ibmjs/lib à variável de sistema LD_LIBRARY_PATH para que os arquivos binários lá presentes	Idem

Tabela 2: ajustes adicionais para a compilação e execução das aplicações Java SDK

Para melhor funcionamento das aplicações, no que se refere ao reconhecimento de voz, junto com os pacotes do software ViaVoice descritos está disponível no diretório binário da instalação um aplicativo denominado **enroll** (Dme.exe no Windows) destinado a auxiliar o usuário no ajuste do microfone, a analisar o nível de ruído do ambiente de funcionamento e a realizar o treinamento do software com a voz do usuário. No entanto, esta etapa da configuração não é essencial, podendo ser realizada em outro momento.

2.3 Interface

O Java SDK oferece uma interface peculiar às API's Java, ou seja, totalmente orientada a objetos, com classes e métodos bem definidos no que se refere ao funcionamento e utilização e com documentação no estilo Javadoc: um hipertexto bem elaborado e interligado descrevendo todo os pacotes disponíveis.

O ViaVoice Runtime Dictation fornecido pela IBM é semelhante à versão comercial, oferecendo aplicativos de ajuste, treinamento e gerenciamento de usuários. A diferença é a ausência do assistente, presente na versão comercial, que oferece o controle de todo a área de trabalho utilizando a voz.

2.4 Performance

A documentação distribuída determina a configuração mínima como: Pentium 166Mhz MMX com 32 MB de memória, no entanto este sistema apresentou um rendimento muito baixo, com longos períodos de leitura de disco e porcentagem de acerto muito baixa no reconhecimento das palavras ditadas, da ordem de 7,5 acertos em 10 palavras.

Uma configuração apresentando uma melhor performance seria um Pentium, ou similar, com 233 Mhz e 48 MB de memória, apresentando uma taxa de acerto da ordem de 8,5. Processadores AMD K6 funcionam perfeitamente com o software.

Nos primeiros dois meses do projeto a pesquisa se deu com a versão 5.0 do ViaVoice, quando então foi disponibilizada a versão 7.0 (apenas para Windows) presente na atual distribuição comercial (Millenium). Esta segunda versão apresentou uma melhora significativa especialmente no reconhecimento de voz que é a funcionalidade mais crítica para a performance, com uma taxa de acerto da ordem de 8,5 em 10 palavras pronunciadas. Para Linux está disponível a versão 3.0 apenas em inglês, com uma performance da ordem de 7,8 acertos em dez palavras.

Quando da disponibilização da versão 7.0, a versão em português também foi colocada para download. Testes foram realizados com esta e apresentaram resultados da ordem de 9 acertos em 10 possíveis.

2.5 Suporte

O suporte oferecido pela IBM para o desenvolvimento das referidas atividades de programação é carente de maior detalhamento, ficando a determinação das funcionalidades específicas do software e os detalhes de instalação dependentes da realização de testes. O suporte direto aos usuários não comerciais é inexistente, há apenas um fórum de discussão onde diversos programadores se comunicam. Pode ser acessado em http://www.alphaworks.ibm.com/aw.nsf/discussion?ReadForm&/forum/speechforjava.nsf/discussion?createdocument.

3 Análise dos pacotes Java para reconhecimento de voz

Como descrito no projeto, a API é constituída de três pacotes que juntos oferecem a possibilidade do desenvolvimento de aplicações com funcionalidades completas para síntese e reconhecimento de voz, cada uma destas funcionalidades é tratada separadamente através de duas entidades: o sintetizador (javax.speech.synthesis.Synthesizer) e o reconhecedor (javax.speech.recognition.Recognizer). Estes são criados pela classe Central, uma abstração do software e do hardware que suportam a API.

A unidade principal utilizada é a palavra, o próprio constituinte das linguagens e da comunicação verbal, elas representam o que é dito e o que é reconhecido e são utilizadas na forma de cadeias de caracteres ou Strings.

Os sintetizadores essencialmente transformam cadeias de caracteres em voz digitalizada, no entanto podem fazê-lo de diversas maneiras para uma maior qualidade do que o computador "diz".

Os reconhecedores procuram determinar (e não entender) o que é dito e são entidades dependentes de gramáticas. As gramáticas são definições pré-estabelecidas ou customizadas que determinam o conjunto de possibilidades do que pode e espera-se que seja dito. O reconhecedor simplesmente tenta estabelecer uma correspondência entre a entrada de som e o que está determinado em sua(s) gramática(s) associada(s).

3.1 A Central de processamento e a aplicação Engine

A seguir será detalhado o funcionamento da interação entre as aplicações SDK Java e a tecnologia ViaVoice. Esta interação é realizada por duas entidades: a classe Central (javax.speech.Central) e a aplicação dependente de plataforma denominada Engine que é distribuída junto com a o pacote de software ViaVoice.

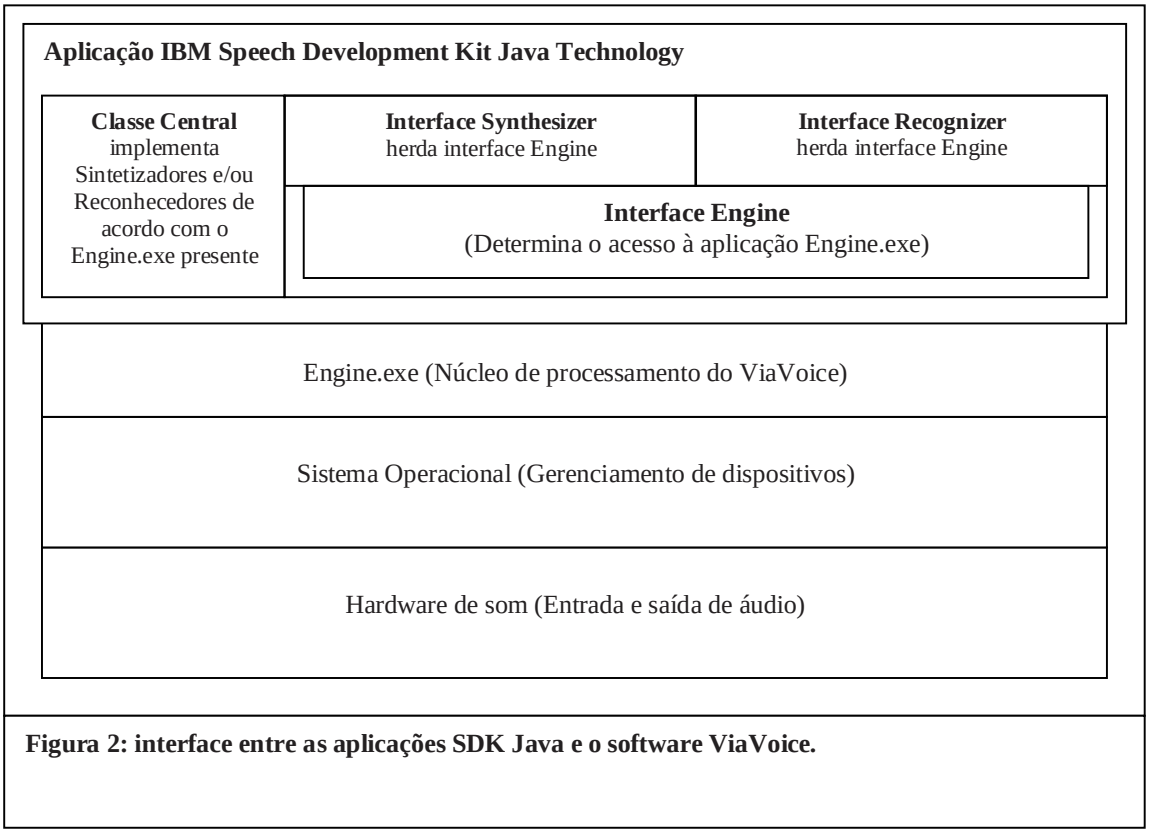

Figura 2: interface entre as aplicações SDK Java e o software ViaVoice.

A classe javax.speech.Central controla a interação entre a API e a entrada e saída de áudio pré-processada pelo ViaVoice, ela é responsável por criar instâncias de objetos sintetizadores e reconhecedores. Para tanto, a classe Central tem acesso ao software suporte do ViaVoice cujo núcleo é a aplicação Engine (Engine.exe no Windows).

O Engine é o motor do reconhecimento e da síntese de voz, e suas funcionalidades específicas para cada plataforma só podem ser acessadas após a classe Central implementar a interface Java cujo nome também é Engine (javax.speech.Engine). Esta interface é herdada tanto pelos reconhecedores como pelos sintetizadores, isto é, ela prove o acesso ao ViaVoice de forma genérica enquanto cada entidade possui sua própria especificação: ou síntese ou reconhecimento.

Portanto, a classe Central é responsável por garantir a independência de plataforma das aplicações. Ela acessa o software base e implementa as entidades essenciais segundo as características do Sistema Operacional em questão. É por esta razão que tanto os reconhecedores quanto os sintetizadores são interfaces que derivam a interface Engine e que são implementadas unicamente pela classe Central.

```
...
        //requisita a criação de um reconhecedor junto à classe Central
javax.speech.recognition.Recognizer recognizer = Central.createRecognizer(null);
        //aloca recursos para a instância criada
recognizer.allocate(  );
...
                                    ----------
...
        //requisita a criação de um sintetizador junto à classe Central
javax.speech.synthesis.Synthesizer synthesizer = Central.createSynthesizer(null);
        //aloca recursos para a instância criada
synthesizer.allocate( );
...
```

Exemplo 1: código para criar um reconhecdor e um sintetizador

3.2 Síntese

A síntese é realizada de forma direta, ou seja, após a criação de um objeto sintetizador o comando:

```
synthesizer.speak("olá");
```

É válido e faz com que o computador gere um som em sua saída de áudio equivalente a um ser humano dizendo: olá. Da mesma forma, pode-se determinar a pronúncia de um texto inteiro especificando-se um arquivo endereçado via URL.

3.2.1 O papel da JSML

Para aumentar a qualidade e a naturalidade da voz gerada é possível fornecer informações adicionais para estabelecer: estilos estruturais (parágrafos ou sentenças), enfatização de palavras, determinação de pausas, especificação da natureza das declarações (exclamações, interrogações), controle do nível sonoro de forma dinâmica, ritmo de pronúncia, etc. Para tanto o sintetizador é compatível com uma linguagem de marcação da mesma natureza do XML denominada "Java Speech Markup Language" ou simplesmente JSML que está detalhada no apêndice.

Através de uma marcação simples das sentenças é possível a síntese de voz com as características acima descritas. Desta forma:

```
synthesizer.speak("<SENT>Computadores <EMP>podem</EMP> falar.</SENT>");
```

Exemplo 2: com o comando acima temos:
"Computadores podem falar." Com ênfase na palavra "podem".

3.2.2 Propriedades do sintetizador

Podemos ainda determinar as propriedades de nosso sintetizador para melhorarmos a interface de voz. As propriedades que podem ser determinadas são:

Propriedade	Argumento	Observação
Voz utilizada (Voice)	Objeto Voice	Um objeto Voice possui quatro características: nome, sexo (masculino, feminino ou neutro), idade (criança, adulto novo, velho ou de média idade, adolescente ou neutro) e estilo (negócios, casual, robótica ou leitura)
Freqüência utilizada (Pitch)	Hertz	
Intervalo de freqüência (PitchRange)	Hertz	Calculado pela freqüência atual (fa) e o intervalo de freqüência (if) o intervalo fica será fa-(0.2*if) ≤ x ≤ fa+(0.8*if)
Velocidade de fala (SpeakingRate)	Palavras/minuto	
Volume	Absoluto $0.0 \leq x \leq 1.0$	

Tabela 3: propriedades de um sintetizador.

Abaixo temos um exemplo de como acessar e redefinir as propriedades de um sintetizador.

```
Voice voz=new Voice();
voz.setGender(voz.GENDER_MALE);    //determina que a voz será masculina
voz.setAge(voz.AGE_CHILD);         //e de criança
      //as propriedades do sintetizador só podem ser redefinidas através de uma
      //instancia de SynthesizerProperties
SynthesizerProperties sintetizadorProps = (SynthesizerProperties)synthesizer.getEngineProperties();
sintetizadorProps.setVoice(voz);          //redefine a voz
```

Exemplo 3: código que gera um objeto Voice com voz masculina de criança e a associa ao sintetizador.

3.2.3 Redefinir as propriedades ou utilizar JSML

Algumas das propriedades vistam podem ser alteradas através do uso da JSML. Como JSML é derivada do XML suas tags têm que ser abertas e fechadas em todos os casos. Portanto, através dela podemos determinar quando a alteração irá ocorrer desde que determinemos quando a propriedade deverá voltar para o estado anterior.

```
A <EMP>Fapesp, <PROS RANGE="-30%">fundação de amparo à pesquisa</PROS> tem-se destacado pelos<PROS RATE="-20%" VOL="+15%"> excelentes </PROS> resultados alcançados.</EMP>
```

Exemplo 4: utilização do XML para alterar, dinamicamente, as propriedades do sintetizador. Através da tag <PROS> podemos aumentar ou diminuir atributos como volume (VOL), intervalo de freqüência (RANGE) e velocidade de fala (RATE).

Desta maneira, o JSML proporciona uma maneira mais intuitiva e dinâmica para especificar como o som será gerado. Já a alteração direta das propriedades determina mudanças permanentes que devem ser especificadas em tempo de compilação, um custo bem maior para o processo de programação.

3.2.4 Interatividade

O sintetizador é capaz de receber um evento para cada palavra pronunciada ou em qualquer ponto específico da síntese, estes eventos podem ser tratados unindo-se ao sintetizador um ouvidor (SpeakableListener) que terá acesso ao que foi dito e que determinará qual atitude (processamento) será realizada em virtude disto. Esta possibilidade acrescenta uma enorme interatividade às aplicações, por exemplo: podemos ter um texto narrado ao passo que imagens são mostradas para elucidar a narração ou podemos navegar na narrativa determinando pontos em que o usuário deve ser questionado sobre qual caminho deverá ser tomado.

3.2.5 A interface Speakable

Uma funcionalidade adicional do pacote de síntese é a interface "speakable" (pronunciável). Ela pode ser implementada por qualquer objeto, de qualquer natureza. Um objeto speakable tem uma especificação JSML associada e é um argumento válido para o método "speak". Adiciona-se assim uma enorme robusteza ao processo de síntese: pode-se "pronunciar" qualquer tipo de objeto (implementando "speakable") a qualquer instante com uma única linha de comando. Por exemplo:

```
class speakableFrame extends java.awt.Frame implements javax.speech.synthesis.Speakable
{
        public String getJSMLText()
        {
                return("Isto é um Frame que pode ser pronunciado.");
        }
}
                        --------------------------------
...
speakableFrame SP_Frame = new speakableFrame();
speak(SP_Frame);
...
```

Exemplo 5: o código acima gera a frase:
"Isto é um Frame que pode ser pronunciado."

3.3 Reconhecimento

Considerando todo o conjunto de fonemas que podem ser pronunciados pelo ser humano ou mesmo todas as palavras de uma única língua, o reconhecimento de voz de forma simples e direta através de uma premissa do tipo:

recognizer.listen();

Não é válida simplesmente por não ser possível, pelo menos por enquanto. Ao invés disso, o reconhecimento de voz aqui presente é realizado limitando-se o conjunto do que pode ser compreendido. Limita-se primeiramente à língua em questão: ou português ou inglês, ou qualquer outra conhecida. Atualmente esta limitação é inerente ao software ViaVoice e não é possível, por exemplo, instalar mais do que uma versão do software em línguas diferentes em uma mesma máquina, correndo-se o risco de nenhuma delas funcionar.

Tendo-se um conjunto como o determinado por um idioma completo, devemos realizar uma segunda limitação utilizando-se subconjuntos deste idioma. Nestes subconjuntos são enumeradas uma a uma cada palavra que poderá ser dita.

É desta maneira que se dá o reconhecimento de voz através do software em questão: determina-se o que pode ser dito através de um conjunto bem definido e o reconhecedor procura estabelecer uma correspondência entre um dos elementos deste conjunto e o que é recebido na entrada de áudio. Estes conjuntos são denominados gramáticas.

As gramáticas funcionam da seguinte forma: primeiramente é criada uma gramática por um reconhecedor atribuindo-se a ela um conjunto de regras (as gramáticas de ditado têm regras pré-estabelecidas). Tendo-se as regras a gramática tem um conjunto de possibilidades de reconhecimento, ou seja, analisando a entrada de áudio a aplicação pode determinar se o que está sendo dito corresponde ao que se espera que seja dito.

Após esta etapa deve-se ligar um ou mais ouvidores (listeners) à gramática, eles serão responsáveis por interpretar o que foi dito, determinando qual processamento refere-se ao que foi reconhecido. Se uma correspondência for detectada, a gramática gerara uma entidade chamada resultado (result) contendo informações sobre o que se "ouviu".

Quando o resultado chega ao ouvidor, este pode determinar que som (comando) gerou o evento e desencadear o processamento pré-estabelecido pelo programador.

Existem dois tipos de gramáticas:

3.3.1 DictationGrammar

Gramática de ditado, este formato suporta a entrada de texto sem formato pré-determinado, continuamente como a leitura de um texto. Estas gramáticas possuem um conjunto pré-estabelecido de sons que definem o domínio da gramática. O usuário pronuncia as palavras uma após a outra em qualquer seqüência desde que pertençam ao domínio em uso. Palavras não pertencentes ao conjunto geram resultados incorretos ou são ignoradas simplesmente.

Este tipo de gramática é fornecido pela IBM de maneira a abranger domínios específicos para áreas de conhecimento como direito, medicina ou ciência da computação já que se podem prever as palavras técnicas e as mais comuns que serão utilizadas.

A presente API não oferece a possibilidade de se criar gramáticas de ditado, pode-se apenas adicionar ou remover palavras, representadas simplesmente por uma cadeia de caracteres. Junto com a distribuição do software base do ViaVoice é fornecida uma única gramática desta natureza denominada "dictation grammar text" de uso geral.

As gramáticas de ditado são sensíveis ao contexto, isto é, tentam limitar as possibilidades da próxima palavra a ser reconhecida baseando-se na palavra anterior, na seqüência de palavras anteriores ou mesmo no parágrafo anterior. Por exemplo: se um substantivo é dito, espera-se logo após um verbo.

```
        //cria uma instancia da gramática padrão associada ao reconhecedor
DictationGrammar dictationGrammar = recognizer.getDictationGrammar(null);
        //associa um ouvidor à gramática criada
dictationGrammar.addResultListener(dictationListener);
        //adiciona a palavra "rebutar" ao conjunto abrangido pela gramática
dictationGrammar.addWord("rebutar");
```

Exemplo 6: o código acima cria uma instância de gramática de ditado, associa a ela um ouvidor e adiciona uma nova palavra a seu conjunto de possibilidades.

3.3.2 RuleGrammar

Gramática de regras, suporta o diálogo discreto entre a aplicação e o usuário. O que pode ser dito é definido explicitamente através de um conjunto de regras descritas utilizando-se uma linguagem denominada "Java Speech Grammar Format" ou JSGF que será detalhada no apêndice.

```
        //criamos um objeto reader que tem acesso a um arquivo texto contendo as regras
Reader reader = new FileReader("songs.gram");
        //carregamos a gramatica através do reconecedor
songsGrammar = recognizer.loadJSGF(reader);
        //associamos um ouvidor à gramática
songsGrammar.addResultListener(ruleListener);
        //habilitamos a gramática
songsGrammar.setEnabled(true);
```

Exemplo 7: código para carregamento, criação, e habilitação de uma gramática de regras, também aqui pode-se observar como um ouvidor é associado à gramática.

Através do JSGF podemos determinar os comandos esperados: frases inteiras, palavras isoladas ou palavras/frases seguindo um determinado padrão. As regras de uma gramática podem ser carregadas em tempo de execução através de um arquivo texto, uma URL ou uma regra por vez.

As regras são identificadas por nomes e possuem uma cadeia de caracteres associada representando o que foi dito, elas podem ser alteradas, deletadas ou inseridas dinamicamente. Ex.:

```
grammar x;
...
<regra 10> = Fechar programa. {close};
...
```

Exemplo 8: A regra acima chama-se "regra 10", pertence à gramática "x", corresponde à pronuncia da frase "Fechar programa." e a String "close" representa esta regra dentro de uma aplicação.

3.3.3 Comparando as gramáticas

Comparando as gramáticas temos que a DictationGrammar é muito mais abrangente, sendo mais bem utilizada em discursos contínuos (textos) e não quando se espera que o usuário ministre comandos para a aplicação. É uma gramática que demanda grande capacidade de processamento e que proporciona aplicações suportando o ditado de sentenças ou mesmo textos completos.

A RuleGrammar é adequada quando se deseja o controle da aplicação via comandos discretos, deve-se utilizá-la aproveitando-se de suas funcionalidades dinâmicas, especificando a cada instante apenas o que o usuário pode realmente dizer através da alteração de seu conjunto de regras em tempo de execução.

Nos testes realizados, a RuleGrammar alcançou resultados de até 100% de acerto, taxa bem maior do que a Dictation Grammar que apresentou resultados da ordem de 70% a 85%. Isto se deve à diferença no tamanho entre os conjuntos de palavras suportados por cada gramática.

3.3.4 Os Resultados e os Ouvidores (Listeners) de Resultados

Quando uma correspondência é identificada entre o que é dito e o que está definido em uma gramática, um resultado é criado. Ao mesmo tempo um evento é gerado e o resultado despachado junto com ele. Este evento chega a todos os ouvidores (ResultListener) ligados à gramática em questão e estes são capazes de ter acesso a uma cadeia de caracteres representado o que foi dito. Desta maneira qualquer tipo de processamento pode ser desencadeado perante um resultado de reconhecimento.

```
ResultListener ruleListener =
        new ResultAdapter( ) {
          public void resultAccepted(ResultEvent e) {
           try
           {
              FinalRuleResult result = (FinalRuleResult) e.getSource();
               String tags[] = result.getTags();                //cada palavra representando a regra
                                                                    //corresponde a um tag (String)
                     //junta todas as tags em um unico buffer
               StringBuffer SB_comando = new StringBuffer();
              for(int i=0;i<tags.length;i++)                  //faz o append de cada token reconhecido
              {
                           SB_comando.append(tags[i]);             //junta ao fim do buffer cada token
               }
               if ((SB_comando.toString()).equals("comando")) {
                     System.out.println("Voce disse comando.");
                        }
               else if ((SB_comando.toString()).equals("close")) {
                     System.exit(0);       //fecha programa
                }
            }catch (Exception ex) {
                  ex.printStackTrace();
             }
          }
        }
...
gramaticaDeRegras.addResultListener(ruleListener);
...
```

Exemplo 9: código que implementa um ouvidor que responde a dois comandos: comando e close.

3.3.5 Eventos de áudio

Os reconhecedores geram, além dos resultados referentes às gramáticas, eventos de áudio (RecognizerAudioEvent) de forma que as aplicações possam receber informações sobre a entrada do som. Temos três eventos gerados: um periódico com informações sobre o volume da entrada do som, e outros dois quando se tem o início ou o fim de um diálogo. Estes eventos são gerados automaticamente e para serem utilizados deve-se apenas associar um ouvidor de áudio (RecognizerAudioListener) ao reconhecedor.

```
static RecognizerAudioListener audioListener =
        new RecognizerAudioAdapter(){
          public void audioLevel(RecognizerAudioEvent e) {
             System.out.println("volume " + e.getAudioLevel());
          }
        };
                                        ------------------------
...
                        //
AudioManager AM_temp =  recognizer.getAudioManager();  //todo reconhecedor tem um
                                                                  //audio manager associado
AM_temp.addAudioListener(audioListener);      //é a ele que se associa o audioListener
...
```

Exemplo 10: código que implementa um RecognizerAudioListener e o associa a um reconhecedor. Com este código teremos periodicamente impresso na saída padrão o volume da entrada de som.

3.3.6 Eventos do motor (engine)

Como visto no item sobre a Central de Processamento, a Engine é o coração tanto do sintetizador quanto do reconhecedor. Portanto, é útil que tenhamos algum meio de obter informações sobre seu funcionamento. Isto é possível associando-se a uma destas duas entidades um EngineListener, ele é responsável por tratar os eventos gerados pela atividade do engine que suporta o funcionamento dos reconhecedores e/ou dos sintetizadores.

Os eventos gerados são: motor alocado, alocação de recursos, motor desalocado, desalocação de recursos, erros do motor, motor pausado e motor despausado. O de maior interesse é o que informa sobre erros do motor, estes são gerados, por exemplo, quando o sintetizador tenta utilizar a saída de som e esta já está sendo usada.

```
static EngineListener engineListener =
        new EngineAdapter() {
          public void engineError(EngineErrorEvent e) {
             System.out.println("Erro na engine: " +e.getEngineError().getMessage());
          }
        };
                                        ------------------------
...
recognizer.addEngineListener(engineListener);
...
```

Exemplo 11: código que implementa um EngineListener e o associa a um reconhecedor para tratar erros do motor.

3.4 Visão geral

Analisando a API completa podemos concluir que o desenvolvimento de aplicações desta natureza pode ser realizado de uma maneira ampla, alternando as propriedades de síntese e reconhecimento em função de eventos desencadeados pelo usuário. Estes recursos podem ser acrescidos a qualquer aplicação, independente de sua natureza, expandindo a interface usuário/aplicação através de um novo paradigma: a voz.

A seguir temos um exemplo com o código completo de uma aplicação simples com os recursos mínimos oferecidos pelo software.

```
class SDK_teste{
    static Synthesizer synthesizer;                  //sintetizador
    static Recognizer recognizer;                    //reconhecedor
    static RuleGrammar ruleGrammar;                  //gramática de regras
    static ResultListener SDK_testeListener =                //implementação do ouvidor
    new ResultAdapter(){
        public void resultAccepted(ResultEvent e){
            try{
                FinalRuleResult result = (FinalRuleResult) e.getSource();
                String tags[] = result.getTags();
                StringBuffer SB_comando = new StringBuffer();
                for(int i=0;i<tags.length;i++) {
                    SB_comando.append(tags[i]); }
                if ((SB_comando.toString()).equals("comando")) {          //comando "command"
                    System.out.println("Voce disse comando.");  }
                 else if ((SB_comando.toString()).equals("close")) {      //comando "close"
                     System.exit(0);            }          }
             catch (Exception ex)     {
                ex.printStackTrace();  }          }        };
     public static void speak(String word){                               //método speak
        try{
            synthesizer.speak(word,null);        }
        catch(Exception e){
            System.out.println(e.getMessage());          }       }
     public static void main(String args[]){
        try{
            Locale.setDefault(new Locale("en","EN"));
            synthesizer = Central.createSynthesizer(null);          //cria sintetizador
            synthesizer.allocate();                                 //aloca recursos
            speak("I am the computer");
            recognizer = Central.createRecognizer(null);            //cria reconhecedor
            recognizer.allocate();                                  //aloca recursos
            recognizer.addResultListener(SDK_testeListener);    //associa ouvidor
                        //carrega arquivo texto com as regras da gramática
            ruleGrammar = recognizer.loadJSGF(new FileReader("SDK_teste.gram"));
            ruleGrammar.setEnabled(true);                           //habilita gramática
            recognizer.commitChanges();                  //valida as mudanças do reconhecedor
            recognizer.requestFocus();
            recognizer.resume();            }                //inicia reconhecimento
        catch(Exception e){
            System.out.println(e.getMessage());          }       }                }
```

Exemplo 12: código completo de uma aplicação SDK Java que responde a dois comandos de voz e sintetiza a seguinte mensagem: "I am the computer"

Na figura 3 é apresentada a arquitetura de funcionamento do software SDK Java, ressaltando que, da mesma forma que os modelos de interface gráfica, a API possui uma forte natureza orientada a eventos,.

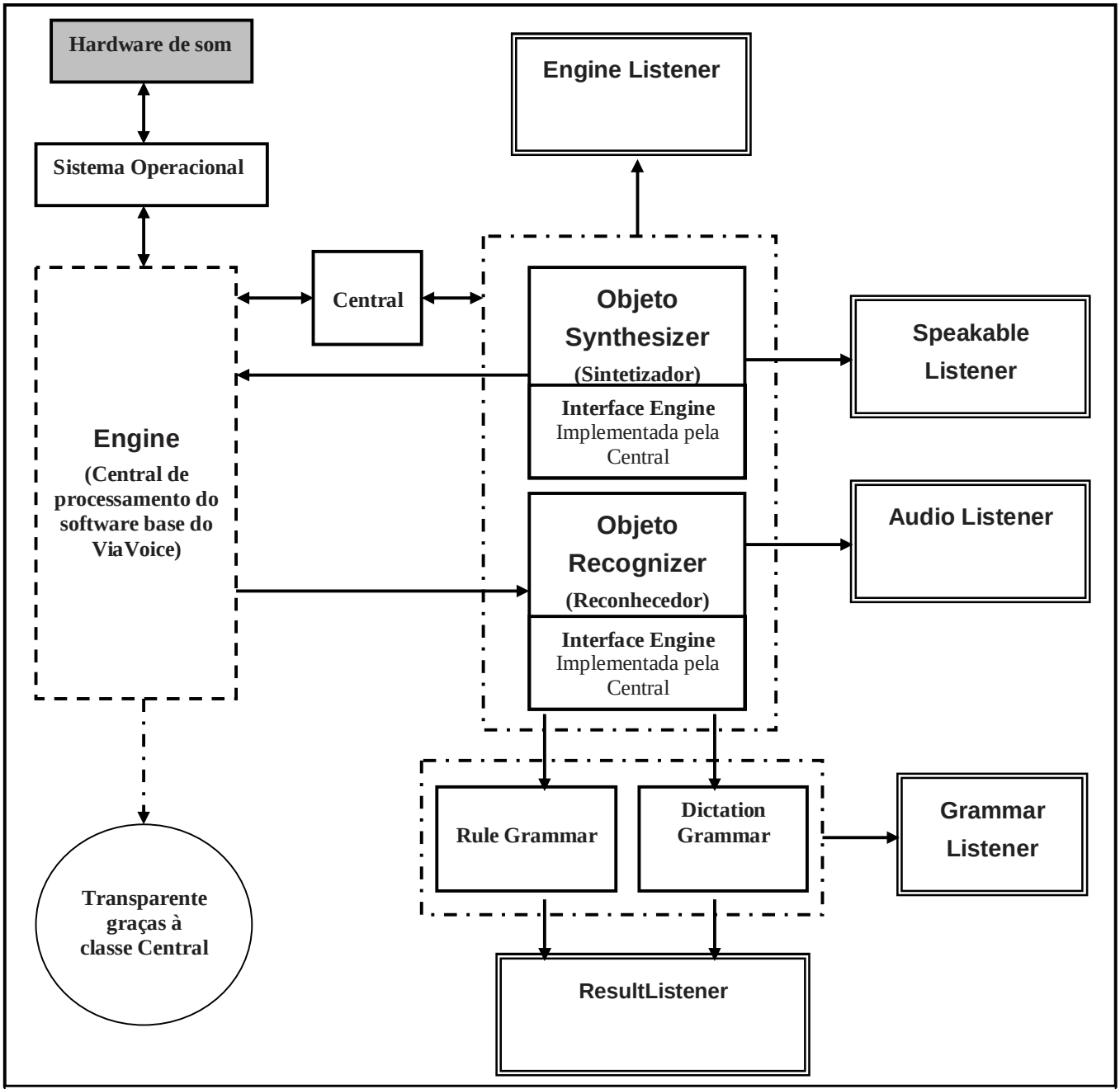

Figura 3: diagrama descritivo da estrutura de funcionamento da tecnologia SDK Java.

Os eventos, gerados constantemente durante o processamento, permitem monitorar toda a atividade do software, desencadeando a atividade pré-estabelecida pelo programador dentro dos ouvidores. Na ilustração temos os principais listeners que são capazes de responder a eventos gerados pelas aplicações.

4 Desenvolvimento de aplicativos com a tecnologia

Durante as duas primeiras etapas do projeto: a familiarização com o software e análise dos pacotes, várias aplicações foram desenvolvidas em caráter experimental para testar, compreender e fixar cada novo conceito desvendado a respeito do pacote de software. Quando um nível de domínio considerado bom sobre a matéria foi alcançado, decidiu-se projetar uma aplicação mais objetiva, capaz de unir a maioria das possibilidades oferecidas pela API.

4.1 O projeto Voice mp3 Player

Foi definido que a aplicação deveria utilizar, no mínimo, os recursos do sintetizador e os do reconhecedor em ambos os contextos: da gramática de ditado e da gramática de regras. A síntese deveria ser realizada de forma a verificar o correto gerenciamento dos recursos de hardware e o reconhecimento deveria proporcionar uma comparação adequada entre as diferentes gramáticas ao passo que utilizasse o paradigma de regras de forma dinâmica, ou seja, uma aplicação com diferentes contextos para o controle do software.

Agrupando todas estas características projetou-se uma aplicação denominada Voice mp3 Player, um software Java baseado nas API's: SDK Java (interface via voz) e Java Media FrameWork 2.0 (para a reprodução de arquivos de mídia). Trata-se de um reprodutor de arquivos Mpeg Layer 3 totalmente controlado através da voz. Com a utilização do microfone o usuário pode criar uma coleção de arquivos e desencadear a reprodução de qualquer um deles ao pronunciar seu nome. Durante a reprodução das músicas estão disponíveis comandos de pausa, parada, avanço, retrocesso, controle de volume, etc tudo via voz. Também através de comandos sonoros o usuário pode navegar em uma árvore de diretórios para escolher o arquivo desejado para a reprodução.

Além destas funções características, outras também estão disponíveis: é possível ditar sentenças que serão repronunciadas pelo sintetizador ou digitar trechos inteiros para que sejam lidos em "voz alta" pela máquina. Estes recursos adicionados para utilização e teste das possibilidades da API.

A natureza desta aplicação demanda um correto gerenciamento dos recursos de hardware, em especial dos recursos de som, já que a síntese de voz concorre com a reprodução dos arquivos. O reconhecimento demanda elaboração e gerenciamento minuciosos da gramática de regras já que não poderia ser determinado um conjunto fixo de arquivos mp3 para sua utilização, ao invés disso, novos arquivos podem se tornar disponíveis dinamicamente e conseqüentemente é necessário uma nova gramática a cada coleção de músicas.

A reprodução dos arquivos pode ser desencadeada através de comandos ministrados em ambos os contextos: da gramática de ditado e da gramática de regras. Promovendo uma comparação entre elas e fornecendo conclusões sobre a real utilidade de cada uma.

4.1.1 A interface gráfica

A interface gráfica possibilita o controle da aplicação sem a utilização de comandos de voz, no entanto o desenvolvimento se deu centrado na utilização da voz e a GUI pode apresentar falhas já que não foram executados testes exaustivos.

Figura 4: a interface gráfica do projeto Voice MP3 Player.

4.1.2 A interface via voz

O software desenvolvido, em sua utilização via voz, possui dois modos de funcionamento:

- modo arquivo: o usuário fornece a raiz de uma árvore de diretórios e o programa cria um conjunto contendo todos os arquivos mp3 encontrados em qualquer ponto da árvore. É possível, então, comandar a reprodução de qualquer um dos arquivos apenas dizendo seu nome, sem necessidade de fornecer sua localização. A cada conjunto de arquivos criado uma nova gramática de regras é definida e carregada para que os nomes das músicas sejam prontamente reconhecidos.

Neste modo de operação são utilizadas duas gramáticas: uma contendo os nomes das músicas do conjunto atual de arquivos (songsGrammar) e outra mais simples (basicGrammar) com comandos que devem ser utilizados quando uma música está sendo reproduzida. Elas estão definidas da seguinte forma:

- songsGrammar: utilizada quando nenhuma música está sendo reproduzida. Reconhece os seguintes comandos:

Inglês	Português	Ação desencadeada
song	música	Habilita a gramática de ditado para que se dite o nome da música desejada. Quando o usuário disser ok, a música será reproduzida.
play	tocar	Inicia a reprodução do arquivo cujo nome está na caixa de texto da GUI.
tracks	trilhas	Mostra a relação de músicas disponíveis na gramática de regras (habilitada automaticamente).
repeat	repretir	Habilita a gramática de ditado e tudo que o usuário disser será repetido pelo sintetizador após a palavra "ok".
ok	certo	Finaliza o comando repeat ou o comando songs.
say	diga	Abre uma caixa de diálogo onde o usuário pode escrever uma sentença para ser lida pelo sintetizador.
collection	coleção	Entra no modo coleção, exibindo o último diretório visitado.
resume	continuar	Faz com que a música volte a ser reproduzida caso tenha sido pausada.
up	mais alto	Aumenta o volume em 20 %.
down	mais baixo	Abaixa o volume em 20 %.
kill	fechar	Fecha o programa.
"nome de arquivo"		Qualquer nome de arquivo que seja mostrado pelo comando tracks faz com que a música seja tocada.

Tabela 4: conjunto de comandos da gramática songsGrammar.

- basicGrammar: gramática que define um conjunto de comandos básicos utilizados apenas quando uma música estiver sendo tocada. Estes comandos foram definidos pois quanto mais complexa a gramática maior processamento é necessário para tratar qualquer entrada de som e, com a música tocando isto se reflete em falhas na reprodução dos arquivos. Esta segunda gramática, portanto, limita ao máximo o número de comandos que podem ser ministrados diminuindo a carga de processamento durante a reprodução.

Inglês	**Português**	**Ação desencadeada**
Play	tocar	Reinicia a reprodução do arquivo cujo nome está na caixa de texto da GUI.
Stop	parar	Interrompe a reprodução da música que está sendo tocada e habilita a gramática songsGrammar para que outra música seja ordenada.
Pause	pause	Pausa a reprodução e habilita a gramática songsGrammar para que a reprodução continue (com o comando resume) ou para que outra música seja ordenada.
Up	mais alto	Aumenta o volume em 20 %.
Down	mais baixo	Abaixa o volume em 20 %.
Back	para trás	Volta a reprodução da música em 10 % de seu tempo de reprodução.
For	para frente	Adianta a reprodução da música em 10 % de seu tempo de reprodução.
Kill	fechar	Fecha o programa.

Tabela 5: conjunto de comandos da gramática basicGrammar.

• modo coleção: o usuário cria um subsistema de arquivos, organizando suas músicas em diretórios e subdiretórios e este subsistema pode ser navegado utilizando-se comandos de voz. Este modo de operação utiliza uma única gramática de regras denominada dirEntriesGrammar, contendo todos os comandos que podem ser ministrados junto com os nomes das entradas de um determinado diretório. Aqui a dinâmica da gramática de regras é utilizada habilitando-se uma a uma as regras que são úteis quando a música é iniciada e quando é interrompida. Além disso, a cada novo diretório examinado a gramática é redefinida e recarregada para que seus arquivos sejam comandos válidos para a gramática.

Inglês	**Português**	**Ação desencadeada**
Play	tocar	Inicia (ou reinicia) a reprodução do arquivo cujo nome está na caixa de texto da GUI.
Stop	parar	Interrompe a reprodução da música que está sendo tocada e redefine as regras.
Pause	pause	Pausa a reprodução e redefine as regras.
Back	para trás	Volta a reprodução da música em 10 % de seu tempo de reprodução.
For	para frente	Adianta a reprodução da música em 10 % de seu tempo de reprodução.
Up	mais alto	Aumenta o volume em 20 %.
Down	mais baixo	Abaixa o volume em 20 %.
Close	fim	Sai do modo coleção.
Previous	anteriror	Acessa o diretório pai do diretório corrente.
Resume	continuar	Faz com que a música volte a ser reproduzida caso tenha sido pausada.
Kill	fechar	Fecha o programa.
"nome de arquivo"		Qualquer nome de arquivo que esteja sendo mostrado na lista de arquivos desencadeia sua reprodução

Tabela 6: conjunto de comandos da gramática dirEntriesGramamar.

O programa funciona perfeitamente tanto em português como em inglês. Para cada língua é necessário instalar a correta versão do ViaVoice e substituir os arquivos que definem a gramática, existindo versões em português e em inglês.

5. Elaboração e desenvolvimento do tutorial

O tutorial, quarta etapa prevista pelo projeto de Iniciação científica, foi elaborado visando possibilitar o aprendizado da tecnologia estudada durante o período em questão. Rico em ilustrações e códigos exemplos totalmente funcionais, desenvolveu-se com o objetivo de tornar instantânea a possibilidade de desenvolvimento dos novos aplicativos acessados via voz, diminuindo a fronteira entre a teoria e a prática, abordadas concomitantemente no decorrer do trabalho e, consequentemente, também em seus resultados.

Em dois idiomas – figura 5, o português nativo para o bolsista, e o inglês que é, informalmente, a língua oficial da Internet, o curso oferecido foi desenvolvido na linguagem de marcação *html* com funcionalidades adicionais conseguidas através da linguagem de *script* denominada *Javascript*. Finalizado, teve sua publicação a partir do mês de Janeiro do ano de 2001 na grande rede de computadores através da url: http://www.grad.icmc.sc.usp.br/~jfrjunio/tutorial. O título escolhido "**Tutorial Speech Development Kit Java Technology"** é a síntese do documento que resume o trabalho realizado no decorrer da Iniciação Científica, apresentando de maneira sucinta informações essencialmente técnicas de um novo conceito de programação e de um novo paradigma de comunicação homem-máquina.

Figura 5 – Página inicial do tutorial.
http://www.grad.icmc.sc.usp.br/~jfrjunio/tutorial

O *design* teve especial atenção para a apresentação do conteúdo, procurando aliar uma aparência agradável com uma linguagem didática, foi projetado com sessões bem definidas e ilustrações claras e simples, item a item os tópicos tratados demonstram, e não apenas apresentam, como se dá a utilização do SDK Java. Preenchido com dezenas de exemplos práticos totalmente funcionais tanto no ambiente Windows como no ambiente Linux e em ambas as línguas Português e Inglês, é possível ao pesquisador fixar novos conceitos analisando código funcional ao término de cada capítulo.

5.1. Conteúdo

Os dados apresentados no curso foram extraídos em sua maioria da documentação da API de programação Java presente na distribuição do software, sendo o restante conseguido a partir da inferência realizada sobre os resultados dos muitos experimentos efetuados, os quais vários são apresentados no documento. Dados também puderam ser recolhidos no material oferecido pela empresa IBM detentora da tecnologia, notadamente o fórum de discussões *on line*, onde pesquisadores de todo mundo trocam informações sobre suas experiências.

Figura 6 – Menu dos tópicos que constituem o tutorial.

Para a estruturação do material foram definidos os tópicos principais considerados essenciais para a compreensão e fixação do curso. Em ordem seguem os nove temas escolhidos: Introdução, Arquitetura, Sintetizadores, JSML, Reconhecedores, JSGF, Ouvidores, Resultados e A API, como podem ser observados na figura 6:

1) **Introdução**: introduzindo a API SDK Java e apresentando os conceitos fundamentais de seu funcionamento, este tópico faz um resumo da técnica de programação possibilitada pelo software. O reconhecimento e a síntese são introduzidos ao leitor estabelecendo inicialmente a natureza do conteúdo que ainda está por vir e traçando os objetivos a serem alcançados com o guia.

2) **Arquitetura**: o conteúdo deste tópico foi elaborado pelas experiências realizadas com o software. As conclusões se tornaram evidentes ao se examinar o funcionamento dos aplicativos em dois ambientes distintos, no Windows e no Linux. Decorrendo assim a esquematização da arquitetura da tecnologia em dois níveis de detalhamento:

   - simplificada: observando-se o funcionamento do software desenvolvido esta primeira arquitetura pôde ser descrita definindo-se os papéis dos principais componentes de seu funcionamento: o hardware de som, o sistema operacional, a aplicação *Engine* e, no topo desta estrutura, uma aplicação SDK Java desencadeando o processamento programado pelo desenvolvedor, como se vê na figura 7.

Figura 7 – Arquitetura simplificada apresentada no tutorial.

- completa: estendendo a esquematização simplificada, é realizada uma aproximação nos detalhes da aplicação SDK Java mencionada na arquitetura da figura 7. Assim são apresentados os componentes Reconhecedores, Sintetizadores, as Gramáticas (de ditado e de regras), os Ouvidores (de reconhecimento, da síntese, do *engine*, das gramáticas e do áudio) e a Central de Processamento. É demonstrado, desta maneira, a disposição destes elementos observando-se a técnica de desenvolvimento das aplicações orientadas à voz, isto é mostrado na figura 8.

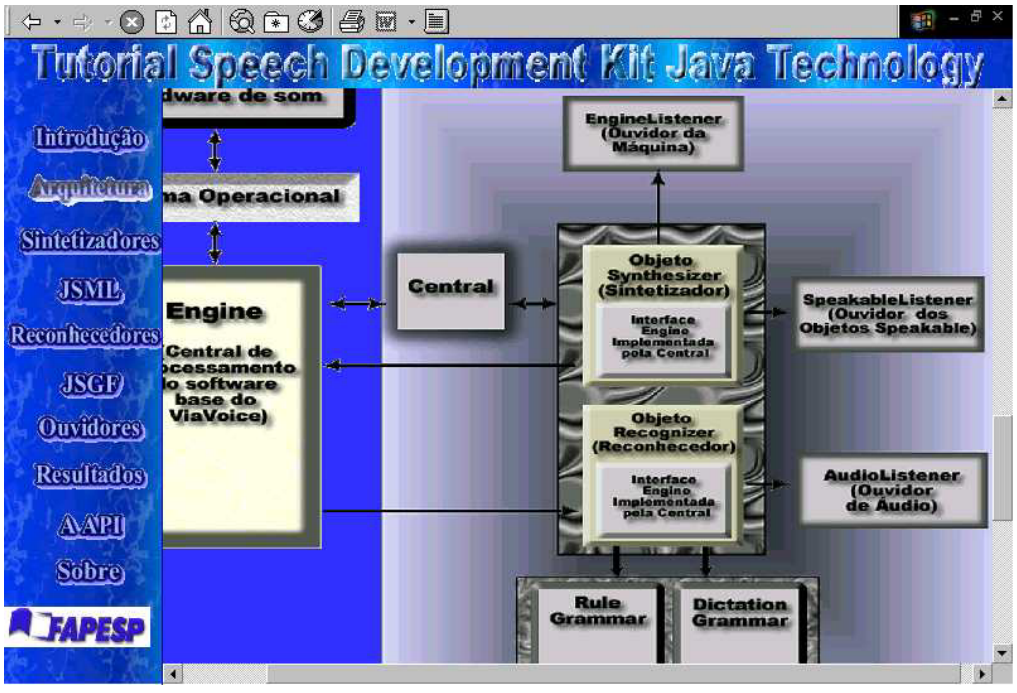

Figura 8 – Arquitetura completa apresentada no tutorial.

3) **Sintetizadores**: neste item é apresentada a entidade Sintetizadora (Objeto *Synthesizer*) definindo-se sua função e como se dá a criação através da Interface *Engine*, a responsável pelo acesso dos aplicativos Java aos executáveis ViaVoice. São vistas suas duas formas de criação e em seguida seus principais métodos que realizam a síntese de voz. Ainda são revisados os Ouvidores de Objetos *Speakable*, capazes de "interpretar" o que a máquina, e não o usuário, pronuncia e por fim é explanado o conceito da fila de itens que é o conjunto dinâmico de elementos a serem sintetizados.

4) **JSML**: nesta etapa é compreendida a linguagem de marcação que auxilia a obtenção de uma síntese mais natural realizada pela máquina. São mostrados sua função, utilização, posicionamento e possibilidades. Por fim, entrando na parte técnica do conceito são detalhadas as oito principais *tags* para marcação do texto da síntese.

5) **Reconhecedores**: finalmente é introduzida a técnica do reconhecimento, que torna possível a interação homem-máquina através da voz. Aqui são mostrados o esquema de operação dos Reconhecedores (Objetos *Recognizer*), a definição dos conceitos de Resultados (Objetos *Result*) e o nível de confiabilidade, um dos principais atributos dos reconhecedores. É ensinado como se realiza a criação de um objeto desta natureza através da Central de processamento e, em seguida, a importância e o papel das gramáticas de ditado e de regras.

6) **JSGF**: complementando o tópico anterior faz-se a exposição da gramática de formatação de regras, a JSGF, com a qual se determina o funcionamento das gramáticas de regras ao especificar, dinamicamente, os comandos que serão compreendidos pelo computador. São vistos: sua estrutura e seus principais usos.

7) **Ouvidores**: o maior dos capítulos do curso apresenta em detalhes todos os ouvidores da API de programação em estudo. Os ouvidores são uma característica marcante das técnicas de programação orientadas a eventos e, através destes é possível fazer com que a máquina responda ao usuário, neste caso, através da voz. Temos:

   - Ouvidor *EngineListener*: permite o tratamento dos eventos gerados durante o ciclo de funcionamento da aplicação *Engine*.
   - Ouvidor *SynthesizerListener*: trata eventos gerados durante o ciclo de funcionamento da entidade Sintetizadora, isto é, da função de síntese do *Engine*.
   - Ouvidor *SpeakableListener*: recebe eventos gerados pela síntese, isto é, pelos fonemas pronunciados pela máquina. É capaz de fazer com que a máquina interprete o que está sendo sintetizado.
   - Ouvidor *RecognizerListener*: trata eventos gerados durante o ciclo de funcionamento da entidade Reconhecedora, isto é, da função de reconhecimento do *Engine*.
   - Ouvidor *RecognizerAudioListener*: ouvidor específico para eventos operacionais de áudio, como o volume e a presença, ou não, de som.

- Ouvidor *GrammarListener*: ouvidor utilizado para controle dos eventos gerados pelas operações efetuadas em um objeto Grammar e, consequentemente, em seus objetos herdeiros: *RuleGrammar* e *DictationGrammar*.
- Ouvidor *ResultListener*: este é o principal dos ouvidores da tecnologia SDK Java, ele é responsável por interceptar e interpretar os eventos gerados pelos resultados (*Results*) criados pelos reconhecedores em parceria com as gramáticas, tanto de regras como de ditado, perante a entrada de áudio. É através da implementação desta interface que as aplicações são capazes de "ouvir" o que está sendo dito pelo usuário e determinar o processamento adequado em função do que foi entendido.

8) Resultados: os objetos deste tipo (Objetos *Result*) são fundamentais para a compreensão do funcionamento e utilização da API, eles são o resultado do reconhecimento de voz e este capítulo procura cobrir suas propriedades, variações e utilização. É apresentada uma ilustração do ciclo de reconhecimento abrangendo maiores detalhes sobre os resultados criados no diversos instantes do processamento.

Também neste item são mostrados, através de ilustrações – figura 9 – os ciclos de funcionamento do *Engine*, dos Sintetizadores, Reconhecedores e o ciclo de Reconhecimento, no qual o áudio é captado e processado. Com os ciclos apresentados, os ouvidores puderam ser explicados com maior clareza, sendo o conteúdo aprimorado através dos exemplos práticos presentes para cada um dos subitens.

9) **A API**: na finalização os conceitos vistos são posicionados em relação à API SDK Java, com uma breve descrição dos três pacotes que a compõem: o pacote javax.speech a partir do qual as funções de síntese (javax.speech.synthesis) e reconhecimento (javax.speech.recognition) são implementadas.

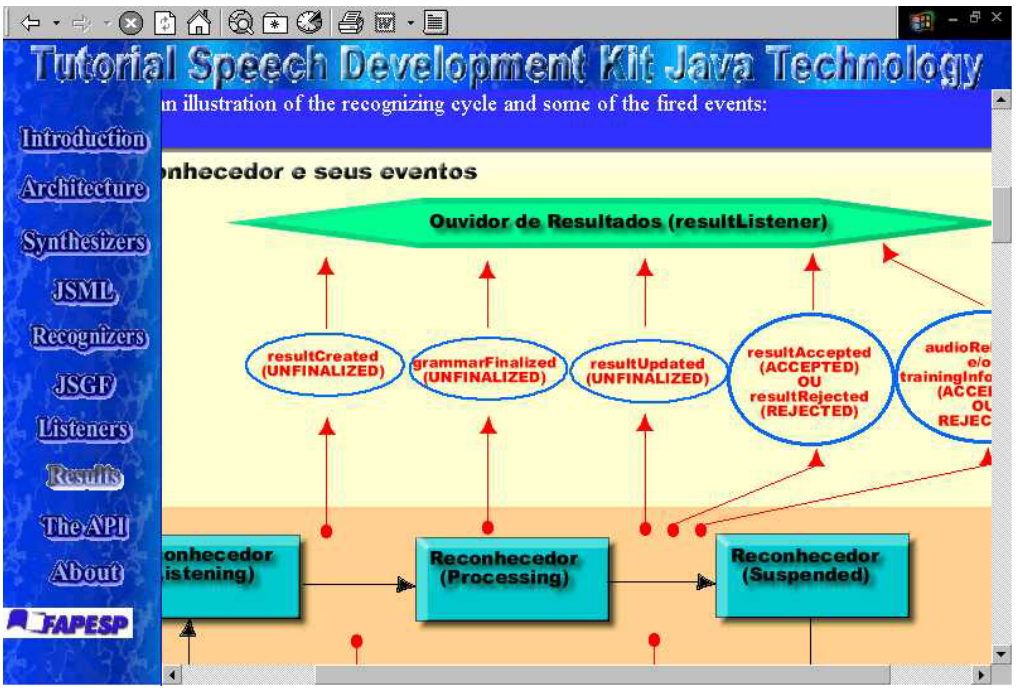

Figura 9 – O reconhecedor, seu ciclo e eventos apresentados no capítulo Resultados.

5.2. Design

Visando tornar o tutorial amigável e visivelmente agradável, o design teve especial atenção, desde a combinação de cores até a disposição dos links, a elaboração das imagens e a ordem de apresentação. Todos estes elementos foram projetados para que se tivesse uma apresentação simples, atraente e organizada. Na figura 10 podemos observar o esquema de cores, macissamente utilizando a cor azul.

Pode-se observar a disposição à esquerda dos links dos tópicos do curso, abaixo e à direita estão links extras que possibilitam avançar os capítulos através dos intuitivos *links* em forma de flechas, ou acessá-los aleatoriamente através dos *links* de cada parte do documento.

Figura 10 – Visão geral da interface das páginas do tutorial.

Os exemplos são assinalados através da imagem apresentada na figura 11, tornando clara a presença dos códigos demonstrativos no decorrer do conteúdo apresentado. Trabalhadas ilustrações não foram poupadas como se observa nas figuras 8, 9 e 10, enriquecendo os conceitos apresentados ao mostrar esquemas visuais que explicam a arquitetura e o ciclo de processamento dos elementos dos pacotes de software.

Constantes referências à documentação da API SDK Java são utilizadas, com inúmeros links para os tópicos fonte utilizados na pesquisa. Isto é observado na figura 11, onde também se nota que o leitor não é remetido para fora do documento quando acessa essa documentação adicional, os links abrem uma nova janela do browser em questão para que se possa continuar a leitura em meio a consultas na documentação.

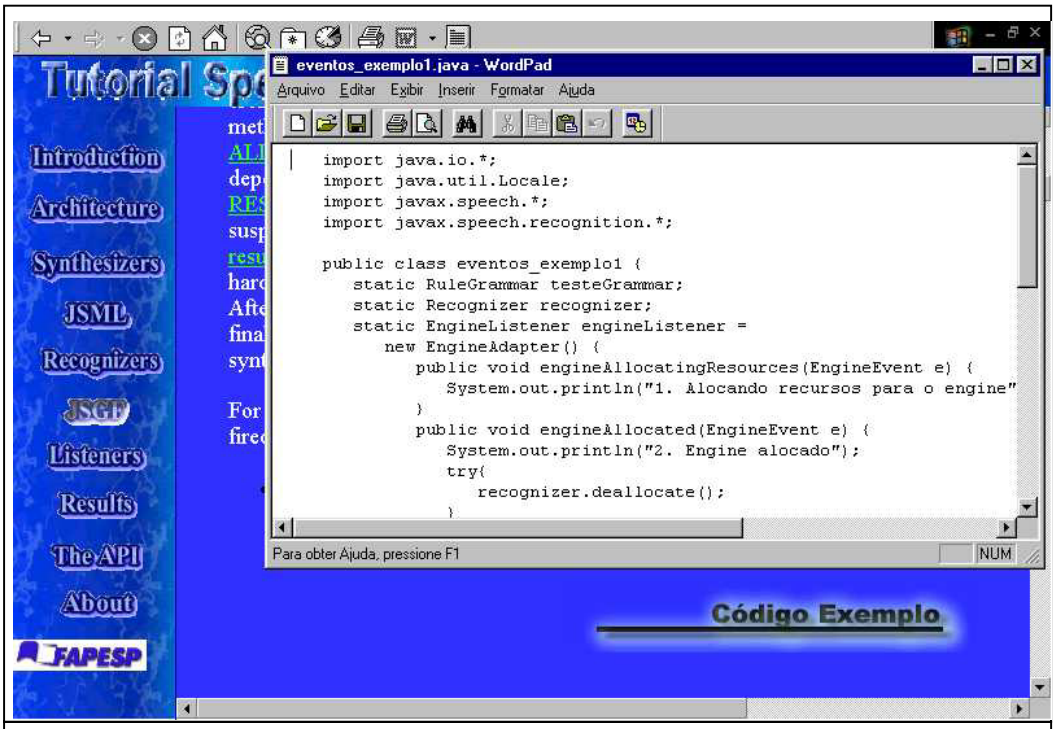

Figura 11 – Há uma presença constante de links para código exemplo.

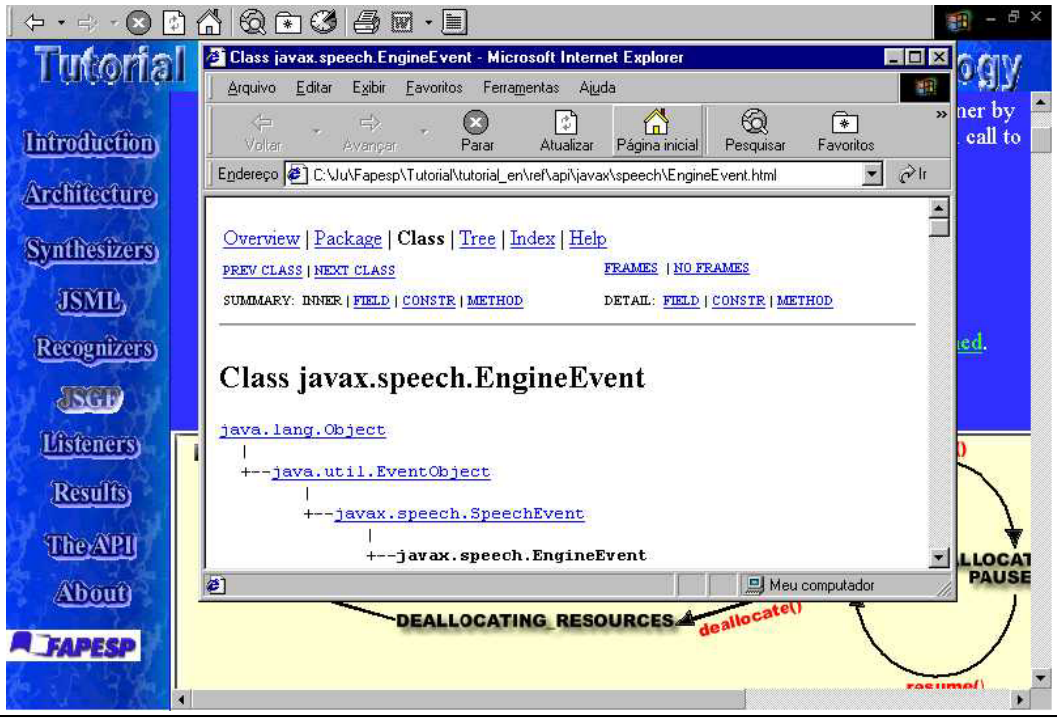

Figura 12 – Em todo o documento existem links para a documentação principal da API.

6. Revisão dos pacotes Java para reconhecimento de voz

No desenvolvimento final do projeto, o pacote de programação em estudo pôde ser mais bem estudado e utilizado em atividades práticas. Em conseqüência, novas descobertas a respeito da utilização da API foram realizadas tornando possível uma melhor elaboração do item 2 do projeto - Análise dos pacotes Java para reconhecimento de voz.

O reconhecimento, a função mais complexa da tecnologia apresentou possibilidades que foram notadas após um maior tempo de utilização da API e, algumas destas características são relatadas a seguir.

6.1. O nível de confiabilidade do reconhecedor

Quando o reconhecedor cria um novo resultado, ele associa a este objeto um valor entre 0.0 e 1.0 indicando o quanto o processamento do som foi bem sucedido, isto é, o quanto a entrada de som correspondeu exatamente a uma das possibilidades presentes na(s) gramática(s) associada(s) ao reconhecedor.

Ao contrário do que se pensava, e do que foi relatado no relatório prévio do projeto, o sucesso (*resultAccepted*) ou fracasso (*resultRejected*) do reconhecimento não é determinado por um patamar único. De fato, este índice pode ser estabelecido através do método ***setConfidenceLevel*** que aceita um valor variando de 0.0 até 1.0, sendo o valor padrão igual a 0.5.

Assim, o nível de confiabilidade é o limite entre o que é aceito e o que é rejeitado, lembrando que ambas, rejeições e aceitações são reportadas e disponibilizadas como eventos de reconhecimento cabendo ao desenvolvedor decidir que processamento desencadear perante um ou outro. Desta maneira um nível de confiabilidade máximo fará com que a maioria dos resultados sejam tratados como eventos de *resultRejected*, resultados indicando que o reconhecimento não foi realizado de maneira suficientemente correta. E um nível de confiabilidade mínimo tem efeito contrário: todos os resultados são tratados como aceitos, *resultAccepted*.

6.2. O nível de sensibilidade do reconhecedor

O nível de sensibilidade diz respeito ao volume da entrada de som. Esta é classificada com um valor que vai de 0.0 até 1.0, respectivamente referente aos sons mais baixos e mais altos. O nível de sensibilidade é o limite entre o som que deve ser processado e o que deve ser ignorado.

Assim, em ambientes ruidosos deve-se estipular uma sensibilidade baixa fazendo com que o ruído de fundo, mais baixo do que a voz entrando direto no microfone, seja ignorado pelo reconhecedor, mas ao mesmo tempo exigindo que o usuário fale mais alto. Uma sensibilidade mais alta deve ser utilizada apenas em ambientes de silêncio, onde não haja ruído competindo com a voz do usuário pela entrada de som.

A seguir temos o exemplo 13 com código demonstrativo de como definir as propriedades de confiabilidade e sensibilidade.

```
recognizer.suspend();                    //o reconhecimento é suspenso
recognizer.pause();                      //o engine é pausado
                    //são definidas as novas propriedades
recognizerProperties.setConfidenceLevel(new Float(0.0).floatValue());    //confiabilidade
recognizerProperties.set setSensitivity(new Float(1.0).floatValue());    //sensibilidade
recognizer.commitChanges();              //valida as alterações, despausa o engine
recognizer.resume();                     //retorna o reconhecimemto
```

Exemplo 13: código exemplo de como se definir os níveis de confiabilidade e sensibilidade do reconhecedor.

6.3. A captura de áudio do reconhecedor

Uma função importante dos reconhecedores é a possibilidade da captura automática do áudio que gerara um determinado resultado. Com esta propriedade definida, além das informações padrões relevantes a um resultado: a gramática responsável pelo reconhecimento, o token associado e a representação do fonema, é possível ter disponibilizada a gravação do som que o gerou, isto é, a gravação da voz do usuário para posterior reprodução. Este som é armazenado na forma de um objeto java.applet.AudioClip que pode ser recuperado durante o tratamento dos eventos de reconhecimento, temos um código ilustrativo no exemplo 14.

```
. . .
static ResultListener dictationListener =                    //implementa um ouvidor de resultados
  new ResultAdapter( ) {
    public void resultUpdated(ResultEvent e) {                //tratamento de resultados aceitos
      FinalResult result = (FinalResult) e.getSource( );
      if(result.isAudioAvailable( )) {
         AudioClip audioClipVoz=result.getAudio( );           //recupera o áudio
         AudioClipVoz.play( );                                //reproduz o áudio
      }
. . .
}
. . .
recognizerProperties = recognizer.getRecognizerProperties();
recognizerProperties.setResultAudioProvided(true);
. . .
```

Exemplo 14: código exemplo de como utilizar a captura e reprodução do áudio reconhecido.

6.4. Estatísticas de reconhecimento

Como requerido na avaliação do relatório parcial, de agosto de 2000, seguem dados estatísticos comparativos sobre o funcionamento das duas modalidades do reconhecimento: utilizando gramática de ditado e utilizando gramática de regras, em português e inglês norte americano ambas no ambiente Windows.

Os dados foram conseguidos submetendo o software ao reconhecimento de palavras isoladas e de frases inteiras. Para cada coleta foram utilizados 30 elementos distintos, apresentados nas tabelas 11 e 12 do Apêndice C, lidos sequêncialmente. Após o processamento de cada item, um resultado positivo era anotado caso o software compreendesse o item lido apresentando sua forma escrita na tela. Os resultados conseguidos estão listados nas tabelas 7 e8.

Em português:

Funcionamento	Palavras isoladas	Frases inteiras
Gramática de ditado	11/30 ~= 37 %	8/30 ~= 27 %
Gramátia de regra	28/30 ~= 93 %	27/30 = 90 %

Tabela 7 - Avaliação comparativa das modalidades de reconhecimento do software IBM ViaVoice em português, base da API SDK Java.

Em inglês norte americano:

Funcionamento	Palavras isoladas	Frases inteiras
Gramática de ditado	9/30 = 30 %	7/30 ~= 23 %
Gramátia de regra	22/30 ~= 73 %	20/30 = 67 %

Tabela 8 - Avaliação comparativa das modalidades de reconhecimento do software IBM ViaVoice em inglês, base da API SDK Java.

Percebe-se que a tecnologia ainda não está aperfeiçoada e que indubitavelmente a utilização da gramática de regras teve uma melhor performance neste teste simples. No entanto, deve-se notar que cada modalidade de funcionamento tem sua função específica, ministrar um ditado à aplicação (gram. de ditado) ou simplesmente submeter-lhe comandos (gram. de regras), não podendo compará-las diretamente.

7. A Aplicação "Verificador de Pronúncia"

Como última atividade prática realizada durante a iniciação científica procurou-se desenvolver uma aplicação que fosse capaz de utilizar e demonstrar todas as funcionalidades oferecidas pela tecnologia. Para tanto foi projetado, e realizado posteriormente, o "Verificador de Pronúncia".

7.1. O Funcionamento

O aplicativo trata-se de um software capaz de verificar e aperfeiçoar a pronúncia do usuário. A verificação foi testada nos idiomas Português e Inglês, no entanto, todos os idiomas suportados pelas diferentes versões do IBM ViaVoice: português brasileiro, francês, alemão, italiano, japonês, espanhol, inglês norte-americano e inglês britânico. A única necessidade é a alteração de um arquivo texto que contém as sentenças as quais o usuário irá ler para ter sua pronúncia verificada.

Ao usuário é apresentado um conjunto de frases, as quais devem ser lidas uma após a outra. O áudio então é gravado e submetido ao processamento da aplicação para que um parecer a respeito de sua pronúncia seja fornecido. Este parecer fornece uma avaliação permitindo que o usuário acesse dados que possibilitem melhorar sua fala em um determinado idioma.

Após a leitura de um item, o usuário é capaz de escutar sua própria voz, que foi submetida à avaliação do software. Para perceber seus erros e conhecer a maneira correta de se pronunciar as frases o software oferece a síntese instantânea das sentenças, através do sintetizador do ViaVoice. Desta forma, comparando as duas pronúncias: sua própria e a da máquina o usuário pode, em pouco tempo, aprender a falar qualquer idioma a qual tenha acesso. Repetindo um determinado exercício até que o parecer da aplicação seja positivo indicando a correta entrada de áudio.

7.2. A abrangência da aplicação

Procurou-se ao máximo utilizar os recursos da tecnologia SDK Java na concepção do "Verificador de Pronúncia", nos domínios da síntese e do reconhecimento. Assim:

• Síntese: a voz do computador pode ser alterada de todos as maneiras possíveis segundo a API, o volume, as características da idade e do sexo do sintetizador. Todas as configurações podendo ser realizadas graficamente, como se vê na figura 13.

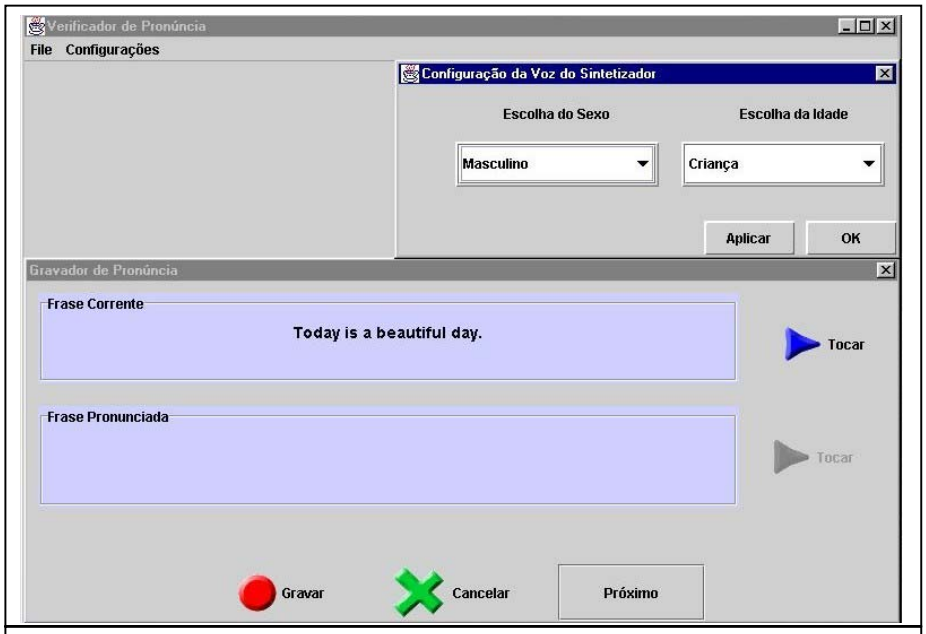

Figura 13 – A síntese pode ser alterada de inúmeras maneiras, incluindo a idade e o sexo da voz do sintetizador.

• Reconhecimento: para utilizar todas as possibilidades do reconhecimento há a opção de se realizar a verificação da pronúncia utilizando-se os dois tipos de gramática exclusivamente.

- Regras: neste modo de funcionamento cada frase a ser pronunciada é encarada como um **comando** composto por várias palavras sendo que a compreensão de uma sentença resulta na confirmação da pronúncia correta – figura 14 – e o contrário resulta em fracasso.

Este modo de funcionamento mostrou-se muito eficaz, sendo que uma pronúncia correta por parte do usuário resulta em confirmação de 90% dos casos, aproximadamente.

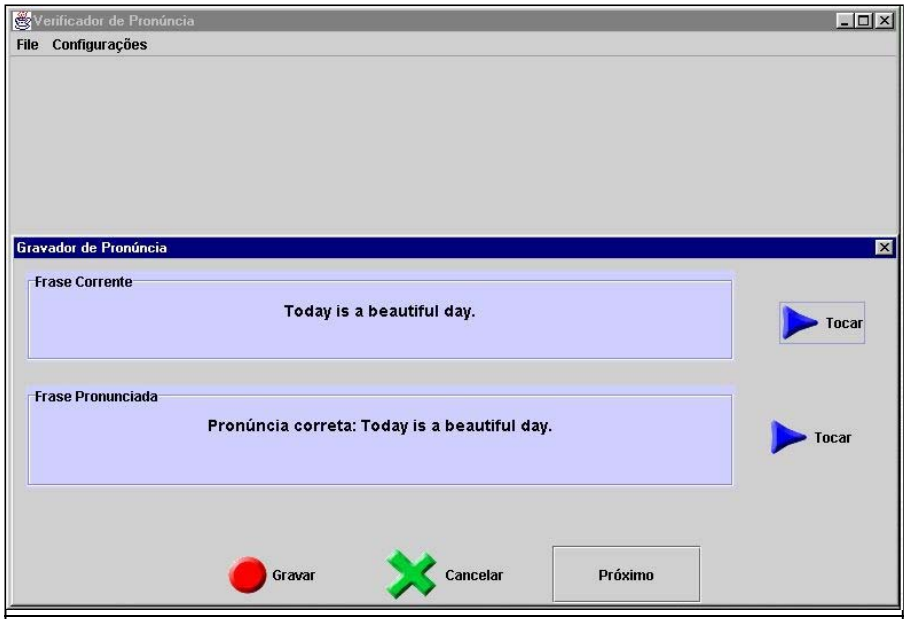

Figura 14 – Perante uma pronúncia correta, o software é capaz de identificar a corretitude da fala.

- Ditado: no modo em que um conjunto bem mais extenso de palavras pode ser compreendido pelo computador, cada frase é encarada como uma **seqüência de palavras** ditadas pelo usuário, devendo estas palavras ser as mesmas que a aplicação tem armazenado. A correta pronúncia é conseguida comparando-se o que foi compreendido pelo computador, durante o ditado da frase, com a frase em questão esperada pelo software, palavra por palavra. É necessária a correspondência de todos os elementos.

Esta variação de funcionamento mostrou-se inadequada para a aplicação, mesmo a pronúncia adequada poucas vezes, apenas 27% aproximadamente, resulta na confirmação da pronúncia.

Obs.: é considerada pronúncia adequada ou correta, a pronúncia de uma pessoa cuja língua nativa seja o idioma avaliado pela aplicação. Os dados de aproveitamento, acima mencionados, de cada modalidade de funcionamento foram conseguidos com o software operando com a língua portuguesa, nativa do aluno bolsista.

Além da possibilidade de utilização dos dois tipos de gramática ainda é possível alterar o nível de sensibilidade e a exigência por corretitude – figura 15 – do reconhecedor.

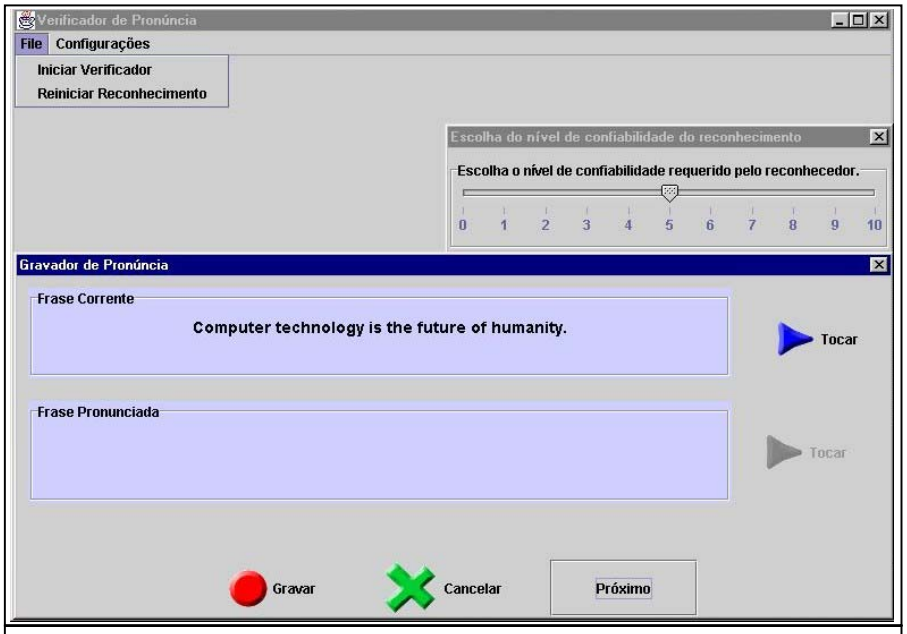

Figura 15 – É possível, entre outras configurações, alterar o nível de confiabilidade (*confidence level*) exigido pelo software.

8 Participação do VIII Simpósio Internacional de Iniciação Científica da USP

Como atividade relevante desenvolvida durante o período da bolsa temos a participação do Oitavo Simpósio Internacional de Iniciação Científica da USP (VIII SIICUSP), cujas apresentações da área de ciênicas exatas se deram na cidade de São Carlos no mês de Novembro de 2000.

Para a apresentação dos resultados do trabalho foi preparado um documento desenvolvido com auxílio da aplicação *Microsoft PowerPoint* para ser impresso e exposto durante o tempo de exposição do Simpósio. O cartaz foi impresso com as dimensões de 0,85 m de largura por 0,85 m de comprimento em 256 cores e foi apresentado durante 4 horas com apreciação de professores designados pela comissão organizadora. O trabalho foi aprovado na apreciação e um resumo do projeto está disponibilizado no CD distribuído aos participantes do Simpósio.

Uma cópia do documento preparado – figura 16- para a apresentação está contida no material enviado.

Figura 16 – Apresentação preparada para participação do Oitavo Simpósio Internacional de Iniciação Científica da USP.

9. Apresentação final do “Tutorial Speech Development Kit Java Technology ”

A conclusão da bolsa de Iniciação Científica e principal resultado esperado em decorrência dos trabalhos é a apresentação do tutorial desenvolvido com a finalidade de transmitir os conhecimentos adquiridos durante a pesquisa e permitindo que outros pesquisadores possam utilizar a nova tecnologia estudada. A apresentação é prevista para o mês de Fevereiro, no início do período letivo visando atingir um número maior de alunos com disponibilidade e interesse.

O mini-curso será ministrado nas dependências do ICMC-USP estando disponível para todo o campus de São Carlos, para alunos do mestrado, doutorado e graduação do cursos de engenharia, física, química e matemática. Para ministrar o tutorial foi preparada uma apresentação (desenvolvida com auxílio da aplicação *Microsoft PowerPoint*) – figura 17 – que está disponível no material enviado com o relatório.

Figura 17 – Apresentação preparada para ministrar o Tutorial Speech Development Kit Java Technology.

Apêndice A - Java Speech Markup Language

Trata-se de uma linguagem de marcação derivada da linguagem, também de marcação, Extendend Markup Language (XML). É utilizada para descrever entrada de texto nos sintetizadores da API de programação IBM Speech Development Kit Java Technology. Seus elementos são capazes de prover informações detalhadas sobre como pronunciar o texto a que o sintetizador está sendo submetido.

Estão incluídos elementos que descrevem a estrutura de um documento, a pronúncia das palavras e das frases, além de marcadores de texto, elementos de controle de ênfase, ritmo, freqüência, volume, etc. Uma marcação adequada aumenta a qualidade e a naturalidade do texto sendo sintetizado.

Ao todo são oito elementos:

Elemento	Descrição	Exemplo
PARA	Determina que o texto marcado trata-se de um parágrafo.	<PARA>Este é um curto parágrafo. </PARA> <PARA>O assunto mudou, então este é um novo parágrafo.</PARA>
SENT	Determina que o texto marcado trata-se de uma sentença.	<PARA><SENT>Até, logo.</SENT> <SENT> Te vejo logo mais.</SENT></PARA>
SAYAS	Especifica como o texto sendo marcado deve ser pronunciado.	<SAYAS SUB="0x0">zero a zero</SAYAS>
EMP	Determina com qual ênfase o texto será pronunciado. Os valores podem ser: "strong", "moderate", "none", or "reduced".	<EMP LEVEL="reduced">Cochichar em público é falta de educação.</EMP>
BREAK	Determina uma parada durante a síntese. Pode ser definido em milissegundos ou em tamanho: "none", "small", "medium", or "large".	<BREAK MSECS="300"/> <BREAK SIZE="small"/>
PROS	Determina atributos do discurso: volume (VOL número absoluto entre 0.0 e 1.0), ritmo (RATE em palavras por minuto), freqüência (PITCH em hertz), Intervalo de Freqüência (RANGE em hertz).	<PROS RATE= "50" VOL="+80%">Eu vou falar alto e claro, só mais uma vez.</PROS>
MARKER	Quando este elemento é alcançado, ao sintetizador é realizada uma requisição de notificação.	Resposta <MARKER MARK= "sim_nao" /> sim ou não.
ENGINE	Fornece informações específicas para o(s) sintetizador(es) identificado(s) pelo ENGID , que pronunciarão o que está definido em DATA. Do contrário pronunciam o que está marcado.	O sapo faz <ENGINE ENGID= "Sapo_Sint1.0" DATA= "<ribbit=1>" sintetizador inadequado </ENGINE>

Tabela 9: os elementos da JSML.

Apêndice B - Java Speech Grammar Format

Os sistemas de reconhecimento de voz, provêem o computador com a habilidade de escutar o que o usuário fala e determinar o que foi dito. As tecnologias correntes ainda não suportam reconhecimento de voz sem restrições: a habilidade de escutar qualquer diálogo em qualquer contexto e compreender corretamente. Para alcançar uma corretitude razoável e tempo de resposta factível, os reconhecedores atuais limitam o que pode ser ouvido através do uso de gramáticas.

A JSGF define uma maneira de descrever o que é denominado de gramática de regras. Utiliza uma representação textual que pode ser lida e editada tanto pelos desenvolvedores como pelos programadores.

• gramáticas: são compostas por um nome e por um conjunto de regras.Todas as gramáticas definidas pela JSGF devem ter um nome. Ex.:

grammar comandosInicias;

<regra1> = ...;

<regra2> = ...;

...

• regras: cada uma das regras deve ter um nome, um token representando o que o usuário pode dizer e um identificador que é passado para a aplicação identificando a regra correspondente ao que foi reconhecido na entrada de áudio.

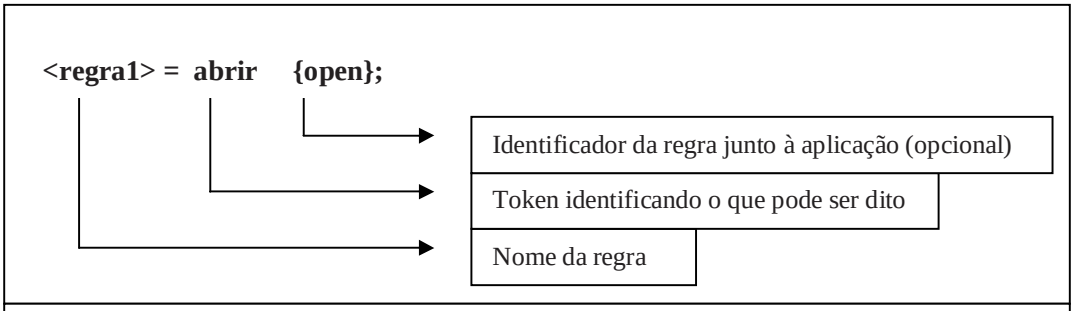

Figura 18: estrutura de uma regra.

Há uma grande versatilidade na definição do que o usuário pode dizer. As regras podem ser especificadas de várias maneiras:

Uso	Exemplo	O que se pode dizer
Apenas palavras	<regra1> = abrir {open};	“abrir”
Frases inteiras	<regra2> = abra o programa (open);	“abra o programa”
Alternativas	<regra3> = abrir | abra o programa {open};	“abrir” ou “abra o programa”
Opções	<regra5> = [por favor] abrir;	“abrir” ou “por favor abrir”
Uma regra pode referenciar outra regra	<regra6> = <regra5> {open};	O mesmo que a regra5
Operador de fechamento *	<regra7> = 13579* vezes;	“1”, “3”, “5”, “7”, “9”, “11”, “13”, “99”, “93”, ou qualquer algarismo (in-clusive nenhum) formado por 1, 3, 5, 7 e 9 em qualquer ordem seguido da palavra “vezes”.
Operador de fechamento +	<regra8> = 13579+ vezes;	Neste caso é obrigatório que um número seja dito antes da palavra “vezes”

Tabela 10 - Utilização da Java Speech Grammar Format - JSGF.

Através da combinação destas representações, podemos determinar conjuntos específicos de palavras ou frases que definirão a interface entre a aplicação e o usuário.

Apêndice C – Dados utilizados para a coleta de estatísticas do reconhecimento

Índice	Palavras	Frases
1	Gato	Errar é humano.
2	Bandeira	Acidentes acontecem.
3	Carro	Educação é tudo.
4	Lancheira	O sucesso é ser feliz.
5	Cidade	Brasil acima de tudo.
6	Brasileiro	O avião é o transporte mais seguro.
7	Computação	Antes só do que mal acompanhado.
8	Pesquisa	Vale mais um fato do mil palavras.
9	Fapesp	A Fundação de Amparo à Pesquisa do estado de São Paulo.
10	Estudar	Quem cala consente.
11	Domingo	Que Deus nos abençoe.
12	Escrever	O cachorro é o melhor amigo do homem.
13	Relógio	Ler é o melhor lazer.
14	Universidade	A terra é azul.
15	Televisão	Caem muitas chuvas em Janeiro.
16	Trabalho	Todo trabalho honesto é digno.
17	Professor	Quanto maior o homem maior é a queda.
18	Memória	O petróleo move o mundo.
19	Calculadora	A vida é incerta, primeiro a sobremesa.
20	Relatório	A TV brasileira não tem ética.
21	Cabeça	As pesquisas avançam rapidamente.
22	Direção	Todo homem tem direito à saúde.
23	Quarenta	Ao vencedor as batatas.
24	Natação	A Internet está interligando o mundo.
25	Caneta	O mundo agora é digital
26	Mochila	A paz é fundamental
27	Segurança	Da água depende a vida.
28	Criança	Pergunte antes, não se arrependa depois.
29	Acontecer	Experiência é sabedoria.
30	Pessoal	A saúde brasileira está crítica.

Tabela 11 - Os elementos utilizados para coleta de dados estatísticos em português.

Índice	Palavras	Frases
1	Cat	Humans make mistakes.
2	Sword	Accidents happen.
3	Car	The seas are overflowing.
4	Lunch	Success is to be happy.
5	Citizen	USA above everything.
6	American	The airplane is a safety transport way.
7	Computation	Better alone than a bad company.
8	Research	A fact is worth one thousand words.
9	Juice	After months finally we made it.
10	Studying	Men reached the moon in 1969.
11	Sunday	May God bless us.
12	Write	Dog is men's best friend.
13	Clock	To read is the best hobby.
14	University	The earth is blue.
15	Television	January is very rainy.
16	Work	Every honest work is worthy.
17	Teacher	Bigger the man, bigger the fall.
18	Memory	The petroleum moves the world.
19	Calculator	Life is uncertain, have desert first.
20	Table	The Brazilian TV has no ethics.
21	Head	Researchs are advancing quickly.
22	Money	Java is becoming the number one language.
23	Forty	To the winner, potatos.
24	Soccer	Internet is intercommunicating the world.
25	Pen	Now the world is digital.
26	Bag	Peace is fundamental.
27	Security	Life depends on water.
28	Chidren	Ask before, no to repent later.
29	Strawberry	Experience is knowledge.
30	People	The Brazilian health is troubly.

Tabela 12 - Os elementos utilizados para coleta de dados estatísticos em inglês.

Referências

1. “IBM alphaWorks” [http://www.alphaworks.ibm.com/aw.nsf/discussion?ReadForm&/forum/speechforjava.nsf/discussion?createdocument].

2. Java Speech API Specification

[http://java.sun.com/products/java-media/speech/forDevelopers/jsapi-doc/index.html]

3. “ViaVoice Dictation for Linux”

[http://www-4.ibm.com/software/speech/enterprise/collateral/dictationforlinux.pdf]

4. “ViaVoice SDK, Java Technology Edition”

[http://www-4.ibm.com/software/speech/dev/sdk_java.html]

5. “IBM ViaVoice Speech Recognition SDK for Linux”

[http://www-4.ibm.com/software/speech/enterprise/collateral/sdkforlinux.pdf]

6. Java Speech API Programmer’s Guide

[http://java.sun.com/products/java-media/speech/forDevelopers/jsapi-guide/index.html]

7. Java Speech Markup Language Specification

[http://java.sun.com/products/java-media/speech/forDevelopers/JSML/index.html]

8. Java Speech Grammar Format Specification

[http://java.sun.com/products/java-media/speech/forDevelopers/JSGF/index.html]