UNIVERSIDADE FEDERAL DO PARÁ
CENTRO TECNOLÓGICO
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

NELSON CRUZ SAMPAIO NETO

Desenvolvimento de Aplicativos Usando Reconhecimento e Síntese de Voz

BELÉM - PARÁ

2006

UNIVERSIDADE FEDERAL DO PARÁ
CENTRO TECNOLÓGICO
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

NELSON CRUZ SAMPAIO NETO

Desenvolvimento de Aplicativos Usando Reconhecimento e Síntese de Voz

DM 22/2006

Dissertação de mestrado apresentada como exigência parcial para obtenção do Grau de Mestre em Engenharia Elétrica com ênfase em Computação Aplicada, elaborada sob orientação do Prof. Dr. Aldebaro Klautau Jr.

BELÉM - PARÁ

2006

UNIVERSIDADE FEDERAL DO PARÁ
CENTRO TECNOLÓGICO
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

NELSON CRUZ SAMPAIO NETO

Desenvolvimento de Aplicativos Usando Reconhecimento e Síntese de Voz

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal da Pará como requisito parcial para obtenção do Grau de Mestre em Engenharia Elétrica.

BANCA EXAMINADORA

Prof. Aldebaro Barreto da Rocha Klautau Júnior
ORIENTADOR - UFPA

Prof. Evaldo Gonçalves Pelaes
MEMBRO DA BANCA EXAMINADORA - UFPA

Prof. Antonio Marcos de Lima Araújo
MEMBRO DA BANCA EXAMINADORA - IESAM

Profa. Valquíria Gusmão Macedo
MEMBRO DA BANCA EXAMINADORA - UFPA

VISTO:

Prof. Evaldo Gonçalves Pelaes
COORDENADOR DO PPGEE/CT/UFPA

“No fim tudo acaba bem. Se ainda não está bem, é porque ainda não chegou o fim.”

Fernando Sabino.

Agradecimentos

Agradeço a DEUS pelas conquistas alcançadas.

Um agradecimento especial ao meu orientador Prof. Dr. Aldebaro Barreto da Rocha Klautau Jr. pela confiança depositada em mim e por ter me oferecido todo o suporte necessário para a realização deste estudo.

Agradeço também à Universidade Federal do Pará pela oportunidade de realizar o curso de Mestrado como aluno em tempo parcial e à Amazônia Celular S/A pela adaptação de meu horário de trabalho aos horários de aulas na Universidade.

Aos meus pais Fernando e Vera pelo amor, pela paciência, pelo incentivo constante e por terem me proporcionado uma excelente educação, que se tornou a base de todo o meu crescimento pessoal, acadêmico e profissional.

À minha irmã Zenaide e à Luciana por estarem sempre presentes em minha vida, torcendo por mim.

Por fim, agradeço a todos aqueles que direta ou indiretamente apoiaram e ajudaram para a conclusão deste trabalho.

Sumário

Lista de Figuras

Lista de Tabelas

Resumo

A fala é um mecanismo natural para a interação homem-máquina. A tecnologia de processamento de fala (ou voz) encontra-se bastante avançada e, em escala mundial, existe vasta disponibilidade de *software*, tanto comercial quanto acadêmico. A maioria assume a disponibilidade de um reconhecedor e/ou sintetizador, que pode ser programado via API. Ao contrário do que ocorre, por exemplo, na língua inglesa, inexiste atualmente uma gama variada de recursos para o português brasileiro. O presente trabalho discute alguns esforços realizados nesse sentido, avaliando a utilização da SAPI e JSAPI, que são as APIs da Microsoft e Sun, respectivamente. Serão apresentados, outrossim, exemplos de aplicativos: uma aplicação CALL (baseada em SAPI) usando síntese em inglês e português, reconhecimento em inglês e agentes visuais; e uma proposta para agregar reconhecimento e síntese de voz ao chat IRC através de APIs Java.

Palavras Chave: voz, síntese, reconhecimento, SAPI, JSAPI.

Abstract

Speech is a natural mechanism for human-machine interaction. Speech (or voice) technology is a well-developed field when one considers the international community. There is a wide variety of academic and industrial software. The majority of them assumes a recognizer or synthesizer is available, and can be programmed through an API. In contrast, there are no such resources in public domain for Brazilian Portuguese. This work discusses some of these issues and compares SAPI and JSAPI, which are APIs promoted by Microsoft and Sun, respectively. We also present two examples: a CALL application using SAPI-based speech synthesis in English and Portuguese, recognition in English, and visual agents; and a JSAPI-based software that incorporates speech synthesis and recognition to IRC through Java APIs.

Key Words: speech, synthesis, recognition, SAPI, JSAPI.

Capítulo 1

Introdução

A popularização das técnicas de síntese e reconhecimento de voz está calcada no fato de que as interfaces gráficas com o usuário apresentam deficiências quando tentam produzir sistemas de fácil aprendizado e utilização eficiente, tendo em vista que as aplicações têm se tornado cada vez mais complexas, aglutinando um grande número de funcionalidades, as quais são disponibilizadas em hierarquias largas e profundas de menus, além de um número incontável de diálogos com o usuário.

Este novo meio de interação computador-usuário através da voz tornou-se recentemente viável e tende a ser incorporado em sistemas computacionais doravante, graças ao intenso aumento da capacidade de processamento e ao aperfeiçoamento de técnicas de Inteligência Artificial, alcançados nos últimos anos. Ressalta-se, todavia, que o referido meio apresenta deficiências, e é recomendável que se tenha conhecimento delas.

O uso da voz tende a facilitar o aprendizado e uso de um software, além de dar mais liberdade ao usuário, permitindo que seus olhos ou mãos estejam livres para a realização de outras tarefas [3]. Esse ganho é ainda maior se o diálogo ocorrer no idioma nativo do usuário.

A comunicação baseada na voz é bastante diversa [4]. Diferentes grupos de usuários, ambientes e domínios de aplicação demandam métodos de interação flexíveis. Enquanto a chave principal, do ponto de vista tecnológico, é a integração

de componentes, a adaptabilidade e flexibilidade são características essenciais para se implementar as várias técnicas de interação responsáveis pela interface do aplicativo com o usuário. Se as camadas mais baixas dentro do conceito da arquitetura suportarem componentes adaptativos e técnicas de interação, as aplicações de voz serão igualmente eficientes em diferentes condições de uso.

O presente trabalho expõe os principais tipos de aplicações de voz e os pontos que devem ser levados em consideração no momento da elaboração da interface com o usuário e da escolha do *software*. Ao invés de se deter em aspectos da *ciência* da fala, tais como, o uso de cadeias de Markov escondidas (HMM) para reconhecimento de voz, a dissertação focaliza a *tecnologia* da camada de aplicação, a qual faz uso dessas tecnologias a partir das camadas mais baixas. Outros trabalhos desenvolvidos no Laboratório de Processamento de Sinais da UFPA abordam temas como HMM e similares. Duas tecnologias de software se destacam nessa área: SAPI e JSAPI.

A ferramenta *Microsoft Speech Application Development* vem amadurecendo ao longo dos anos e já se tornou um dos ambientes de desenvolvimento de aplicações de voz mais utilizados no mundo. O coração do sistema é a interface unificada de programação (SAPI) com seus conjuntos de classes, que provêem todos os componentes necessários para incorporar a tecnologia de voz na construção de aplicativos.

Outra tentativa de se criar uma API unificada é a *Java Speech API* da Sun. A JSAPI disponibiliza interfaces e objetos de programação similares aos encontrados na SAPI, porém apresenta limitações técnicas e carência de sintetizadores e reconhecedores que suportem JSAPI, apesar de não ser uma especificação tão recente (disponível desde 1998).

Durante o trabalho, utilizou-se, também, o *toolkit* do CSLU (*Center for Spoken Language*), o qual incorpora suporte ao português brasileiro através da "voz" AGA, e o componente *Microsoft Agent*, que permite criar uma alta interatividade com o usuário. Através do MSagent é possível fazer com que frases ditas pela aplicação em diversas línguas, inclusive a portuguesa, sejam atribuídas ao seu personagem.

Nesta pesquisa foi implementado um protótipo de aplicativo CALL (*Computer-*

assisted Language Learning) para o ensino da língua inglesa baseado em SAPI, no qual é usado síntese e reconhecimento em inglês, síntese em português e agentes visuais. Uma implementação em JSAPI com o sintetizador de voz de código livre FreeTTS também foi desenvolvida. Nela buscou-se inserir emoção à uma voz sintetizada. E, finalmente, é apresentada uma aplicação puramente Java que agrega reconhecimento e síntese de voz às “salas de bate-papo” na Internet.

O texto encontra-se organizado da seguinte forma: o capítulo seguinte apresenta uma breve descrição da ciência e tecnologia da fala. O Capítulo 3 conceitua e compara as relevantes APIs de voz SAPI e JSAPI, promovidas pela Microsoft e Sun, respectivamente. O estudo das APIs é importante porque são elas que proporcionam ao programador acesso e controle total sobre os softwares que oferecem tecnologia de voz, *expertise* essencial para o desenvolvimento de aplicações com alto grau de complexidade e interatividade. Os aplicativos desenvolvidos são descritos no Capítulo 4, seguindo-se o capítulo reservado às conclusões.

Capítulo 2

Tecnologias para Aplicativos Baseados em Voz

Dentre os ramos da tecnologia de voz, o reconhecimento e a síntese são os abordados no presente trabalho. Os mesmos são brevemente descritos nos itens subseqüentes. Em seguida, faz-se uma sintética apresentação dos tipos de interfaces baseadas em voz.

2.1 Reconhecimento e Síntese

2.1.1 Síntese de Voz

Dado um texto (em ASCII, por exemplo), a *síntese de voz* é o processo de geração de amostras de um sinal digital que deveria soar como se um humano tivesse dito o texto. A síntese de voz (ou TTS, de *text-to-speech*) não deve ser confundida com a simples digitalização, armazenagem e posterior *playback* de voz.

Um conceito importante dentro da síntese de voz é o da prosódia [5]. A prosódia refere-se ao perfil melódico da pronúncia, ou seja, é usada tanto para pausar a fala como para enfatizar certas palavras ou mesmo sílabas com o objetivo de transmitir um certo significado à frase, sendo um dos aspectos da voz mais afetado pela emoção. O processo para adicionar emoção à pronúncia de uma voz sintetizada compreende desde

a modificação de parâmetros básicos da voz como intensidade, quantidade de palavras por minuto, até a manipulação da entonação e ritmo dos fonemas. Por exemplo, as frases “Ele foi!” e “Ele foi?” utilizam-se das mesmas palavras, mas a diferença entre elas está na forma com que elas são pronunciadas.

2.1.2 Reconhecimento de Voz

O *reconhecimento de voz* (ou ASR, de *automatic speech recognition*) pode ser visto como o processo inverso, onde o sistema converte voz digitalizada em texto. Dependendo da aplicação, o resultado obtido pelo ASR pode ser repassado para módulos como o de gerenciamento de diálogos, tutores inteligentes, processamento de linguagem natural (PLN), entre outros.

Nos itens posteriores, serão apresentadas as principais técnicas utilizadas para transformar a voz humana em algo inteligível para o computador e vice-versa. Discute-se, também, como funcionam os principais tipos de aplicações de voz, o que se deve levar em conta no momento de projetar a interface do programa com o usuário e os métodos de avaliação.

2.2 Tipos de Aplicações de Voz

A tecnologia de voz possui grande potencial para a criação de aplicativos que possibilitem uma eficaz interação humano-computador, especialmente no atual contexto de acelerada miniaturização dos sistemas embarcados. Uma maneira de apresentar as aplicações de voz é organizando-as em duas categorias baseadas na modalidade de uso [4]. Na primeira categoria estão as aplicações convencionais, que utilizam a voz como principal fonte de interação, enquanto a segunda categoria engloba as aplicações multimodais, mais conhecidas como audiovisuais.

2.2.1 Aplicações Convencionais

As aplicações de voz já estão sendo utilizadas por muitas empresas, por exemplo, para atender as ligações de clientes que buscam informações, serviços ou registrar reclamações. Tais sistemas são conhecidos comercialmente como IVR (*Interactive Voice Response*). Os mesmos dispensam a operação humana, já que fazem uso da síntese de voz e das teclas do telefone (DTMF) para interação, e mais recentemente, também, do reconhecimento de voz, o que torna a interface mais natural e eficiente. A Microsoft e a Sun disponibilizam API's destinadas exclusivamente para o desenvolvimento de aplicações telefônicas, chamadas TAPI (*Telephony Application Program Interface*) e *Java Telephony API*, respectivamente.

As aplicações para *desktop* são uma outra área na qual se emprega a voz como forma de interação. Há algum tempo, já são de conhecimento popular as aplicações de síntese que lêem o conteúdo de um documento e as aplicações para ditado, onde o usuário pode falar ao invés de digitar o texto, como o ViaVoice da IBM. As atuais limitações técnicas do reconhecimento de voz interferem diretamente no desempenho dos aplicativos para ditado.

Outro tipo clássico de aplicação para *desktop* é conhecido como *command-and-control*, no qual tarefas operacionais usualmente feitas pelo teclado e pelo *mouse*, como salvar um documento ou iniciar um aplicativo, podem ser executadas via comando de voz, facilidade esta, também, muito útil para dispositivos com tamanho reduzido (aparelhos celulares, *palm tops*, *iPods* e outros). Essas aplicações, em função da sua gramática livre de contexto e vocabulário reduzido, tendem a apresentar maior eficiência de reconhecimento quando comparadas com as aplicações para ditado. Por outro lado, os sistemas para ditado disponíveis hoje no mercado, em sua maioria, incorporam técnicas básicas de *command-and-control* [6].

Nem sempre a voz é a única, a mais eficiente, ou a preferida modalidade de interface dentro da aplicação. A voz também pode ter um papel secundário, servindo de suporte e/ou alternativa para outras interfaces. Assim, a voz pode ser empregada conjuntamente e de várias maneiras com outras modalidades, surgindo as aplicações

conhecidas como multimodais, objeto do item subseqüente.

2.2.2 Aplicações Multimodais

Para [7], a síntese multimodal ou audiovisual define-se como a geração automática de um sinal de voz conjuntamente com a sua correspondente informação visual. A mesma tenta emular a comunicação entre pessoas, a qual é basicamente visual e falada. A síntese multimodal agrega valor para qualquer aplicação que a utilize, já que permite melhorar tanto a acessibilidade quanto o grau de compreensão da mensagem emitida. Cabe destacar que a correta sincronização entre a informação sonora e visual é crucial para conseguir o melhor aproveitamento desse tipo de síntese.

O conceito de aplicação multimodal é muito bem exemplificado pelos sistemas de animação facial que envolvem a sincronização da fala de um personagem com a animação da sua face, conhecidos como sistemas *talking heads* [8]. Adicionalmente aos *talking heads*, sistemas com interativos agentes animados estão sendo desenvolvidos [9], trazendo mais robustez à aplicação. Tais agentes auxiliam o usuário a corrigir seus erros e adiciona alternativas de comunicação entre usuário e aplicativo para diferentes situações e ambientes.

Outro estilo de aplicações audiovisuais são os *softwares* para aprendizado de língua estrangeira com o auxílio de computador, chamados comercialmente de CALL [10]. Esses ambientes de aprendizagem apresentam exercícios de pronúncia e atividades de ditado para o aluno, geralmente exemplificando o contexto com figuras. O uso de agentes animados é muito comum, tornando mais alegre a interface, de modo que se possa não apenas mostrar o texto de retorno na tela do computador, mas também permitir que o aluno, ao errar, ouça a pronúncia sintetizada correta de uma palavra ou frase, enriquecendo seu aprendizado.

Entre os desafios dos futuros sistemas de síntese de voz, uma das tendências que parece mais interessante é a síntese de textos multilingual [11]. Traduções de textos e leitura de documentos em empresas são exemplos de aplicações de voz multilingual. Acrescentar ao sistema novos idiomas para a síntese de voz, principalmente o por-

tuguês, é uma idéia cada vez mais madura e compartilhada por alguns projetos em desenvolvimento no Brasil (e.g., [9]). Considerando que em aplicativos CALL o usuário é muitas vezes uma criança, é importante se poder usar TTS em língua portuguesa para instruções e *feedback*.

2.3 Reconhecedores/Sintetizadores

Obviamente, cada produto possui sua especificação e tem sua melhor performance em determinados tipos de aplicação, razão pela qual é relevante o estudo, durante a fase de projeto, do que reconhecedores e/ou sintetizadores, conhecidos genericamente na literatura como *engines*, são capazes de fazer, suas limitações e as ofertas disponíveis no mercado. Enfim, é imprescindível a análise das propriedades desses *softwares* quando se projetam e implementam aplicações baseadas em voz.

2.3.1 Tecnologias para Reconhecimento de Voz

Primeiramente, o *engine* precisa identificar ou separar cada palavra dita em uma torrente. Três métodos são utilizados:

- Voz discreta: uma ligeira pausa entre cada palavra ou regra é necessária, para que seu início e fim sejam claramente identificados. É o método mais primitivo que existe, não necessitando de grande poder computacional, e é extremamente desconfortável em longos períodos de interação.

- Palavra-chave (*keyword spotting*): esse método é baseado na procura de palavras-chaves dentro de uma torrente, ou seja, palavras que não fazem parte de uma lista pré-definida são ignoradas. Tem a vantagem de parecer mais natural ao usuário que o método anterior, já que as pausas não são mais necessárias. Porém, precisam ser apoiadas por um vocabulário bem detalhado, para evitar conflitos na execução de determinadas tarefas. É ideal para aplicações que chamam menus e funções a partir deles.

- Voz contínua: as palavras são processadas sem a necessidade de pausa entre elas. A tarefa de identificar o início e o fim da palavra é realizada pelo próprio algoritmo, o que faz desse método o de interface mais amigável entre todos, já que o usuário pode falar como se estivesse conversando com outra pessoa. A desvantagem é que o poder computacional requerido é bastante elevado. É utilizado em aplicações para ditado.

O desempenho dos métodos acima discriminados pode ser influenciado pelo locutor. Características da fala como velocidade, entonação, pronúncia, variam bastante entre as regiões de um mesmo país. Esse fator, conhecido como dependência do locutor, é fundamental na implementação do *engine* de reconhecimento de voz. O *software* ideal seria aquele que assimilasse a palavra dita em qualquer sotaque da língua.

A dependência do locutor pode ser trabalhada de três formas, em conformidade com o propósito a que se destina a aplicação. Existem os *engines* desenvolvidos com dependência de locutor, que necessitam de um longo e cansativo treinamento adicional. Além do sistema se adaptar às características acústicas do usuário, o locutor também acaba ajustando seu estilo de voz para conversar com o computador. Esses *softwares* não demandam muita capacidade da máquina, proporcionam bom desempenho e são mais empregados em aplicações de uso individual, ou para pessoas com deficiência na fala.

Nos *engines* com independência de locutor, os treinamentos extras são dispensáveis, pois os mesmos são implementados para serem utilizados por qualquer pessoa, não interessando a sua origem, sendo ideal para ambientes públicos, como aeroportos. Por último, existem os *engines* adaptativos, que tampouco necessitam de treinamento, se aprimorando, contudo, com o uso. Com isso, podem ser falhos quando utilizados por um grupo muito variado. Tanto os *engines* independentes, como os adaptativos, precisam dispor de uma elevada capacidade computacional e apresentam resposta razoável.

Depois da palavra ser identificada, ela precisa ser encontrada no vocabulário, para que a aplicação tenha conhecimento de qual ação ela deva executar. Existem duas

formas de realizar essa busca: a primeira vasculha o banco de dados pela palavra em sua forma original. Para isso, todas as palavras necessárias no processo de reconhecimento devem estar armazenadas. A outra forma é separando a palavra em fonemas, que são as menores unidades sônicas diferenciadoras e indivisíveis que formam uma palavra. Assim, os fonemas são armazenados e não as palavras completas. A última técnica de busca descrita requer menos capacidade física de armazenamento de dados e maior utilização computacional da máquina que a primeira.

O último, mas talvez o elemento mais importante no reconhecimento de voz, é o vocabulário, ou seja, quais palavras poderão ser reconhecidas pelo *engine*. Porém, alguns autores concebem o vocabulário de modo diverso. Por exemplo, segundo [12], o vocabulário não representa o número total de palavras que o *engine* pode compreender, mas sim o número total de palavras não identificadas que o *software* pode solucionar a qualquer momento. Essa afirmativa demonstra que o tamanho do vocabulário é dimensionado pela quantidade de palavras de que ele dispõe para complementar uma determinada seqüência ou frase. Fato é que a precisão do vocabulário é inversamente proporcional ao seu tamanho, dificuldade esta que pode ser superada com um bom algoritmo de separação de palavras.

Após a palavra ser reconhecida pelo *engine*, ela é enviada à aplicação, que deverá entender o que foi dito. Ou seja, uma estrutura com regras, palavras e listas, conhecida como gramática, precisa ser definida na aplicação, para que a mesma saiba que ação executar quando uma determinada palavra lhe for enviada. Existem dois tipos de gramáticas: com contexto livre e para ditado. Para [13], na gramática com contexto livre, as palavras permitidas estão limitadas às regras que informam que palavras podem ser ditas, ou seja, possuem um domínio específico e limitado, diferente da gramática utilizada para ditado. O código abaixo exemplifica o que foi exposto:

```
regra_nome = ("Maria", "Paulo", "Chico")
regra_enviar = ("EnviarEmail para", <regra_nome>)
```

Duas regras foram definidas - a regra_nome cria uma lista com possíveis nomes de entrada e a segunda executa uma ação a partir da primeira. A gramática com con-

texto livre não trabalha com a idéia de que todas as palavras selecionadas precisam ser identificadas. O importante é encontrar uma relação mais abrangente possível entre as listas, as regras e o que pode ser pronunciado. Apresenta como grande vantagem a flexibilidade para expandir seu vocabulário e é ideal para atividades que não necessitam de grande vocabulário e lidam com frases curtas.

O sucesso de uma gramática para ditado depende, essencialmente, de um vocabulário eficiente. É importante que o vocabulário encontre um meio termo entre a quantidade de palavras e a unicidade entre elas, para evitar tanto a falta quanto a troca de palavras. Para ajudar nessa tarefa, os *engines*, normalmente, necessitam de treinamento prévio e possuem conjuntos de palavras mais utilizadas separadas por tópicos, como medicina e engenharia. Outras características importantes: qualidade do áudio de entrada (microfones e placas de som), nível de ruído presente no ambiente, além de uma boa performance computacional. Todavia, as aplicações para ditado continuam distantes do desejável diálogo espontâneo.

Com exceção da gramática, todos os processos necessários para o reconhecimento da voz são automáticos por parte do *engine* e não são controlados pelo desenvolvedor da aplicação. Portanto, para uma escolha adequada, as características do *engine* devem ser confrontadas com o tipo de aplicação que se almeja implementar, levando em consideração detalhes como o grau de precisão desejado, o local de utilização, o público alvo e o poder computacional disponível (processamento, memória, espaço para armazenamento e dispositivos de entrada).

2.3.2 Tecnologias para Síntese de Voz

O segundo tipo de serviço de voz oferece a possibilidade de converter um texto escrito em áudio. O procedimento para fazer com que a máquina fale o que se deseja não é tão complicado. O difícil é fazer com que essa fala seja menos robotizada e mais humana. Alguns fatores nos distanciam dessa almejada realidade, tais como, a dificuldade que os algoritmos encontram em tentar reproduzir o ritmo e as emoções que empregamos na construção de frases e o tratamento dado às palavras que possuem

escrita idêntica, porém são pronunciadas de forma diferente dependendo do contexto.

Cada palavra do texto é convertida em fonemas e marcadores prosódicos. Em seguida, através do processo conhecido como síntese de voz, o *engine* gera áudio a partir da análise dos fragmentos da língua separados no passo anterior. Os métodos utilizados para realizar a síntese de voz são:

- Gravação/Reprodução: é o método mais utilizado comercialmente. Nesse método, o áudio é obtido através de arquivos de som previamente gravados com voz humana, contendo as palavras ou frases que serão utilizadas pela aplicação. Ideal para soluções com respostas padrões, como máquinas de *call center*.

- Síntese por formantes: tenta reproduzir sinteticamente a voz humana, por intermédio de algoritmos e filtros especiais baseados nas freqüências "formantes". Atualmente, esse método não atinge a qualidade de voz obtida com o da síntese concatenativa, e daí não é muito popular.

- Concatenativa: segmentos de voz (fones, difones ou polifones) são pré-gravados em arquivos de som e concatenados para produzir qualquer palavra. Permitem atingir uma boa qualidade e naturalidade. Contudo, o desenvolvimento do dicionário com os segmentos é bastante trabalhoso.

Sob o ponto de vista do contexto a ser sintetizado, os *engines* podem ser divididos em duas categorias: sintetizadores genéricos e limitados ao domínio. Sintetizadores limitados ao domínio podem ser a solução apropriada a tarefas onde a melhor qualidade de voz disponível é necessária, e o domínio não requer textos livres a serem sintetizados. Um exemplo de *engine* limitado ao domínio é o FreeTTS [14], que possui uma voz destinada exclusivamente a informar a hora. Os sintetizadores genéricos, por serem "capazes" de sintetizar qualquer texto, são os mais utilizados em aplicações comerciais.

2.4 A Interface com o Usuário

Para se construir uma interface que utilize a voz como fonte de entrada e saída de informação, é preciso conhecer o estado da arte e da realidade das tecnologias de voz, assim como que tipo de elemento de interface é a voz. É importante que o usuário tenha consciência que a síntese e, principalmente, o reconhecimento de voz têm limitações [15].

A interação homem-máquina é bastante distinta de uma conversa informal entre duas pessoas. Os obstáculos encontrados hoje pelas tecnologias de voz não serão superados em um futuro próximo, mas com técnicas de interação apropriadas, muitas dificuldades podem ser vencidas e aplicações bem sucedidas podem ser desenvolvidas.

2.4.1 Questões sobre ASR

Por mais informado que esteja, o que o usuário realmente espera de uma aplicação de reconhecimento de voz é que ela o entenda naturalmente. É o conhecido grau de satisfação que, dependendo da expectativa que o usuário tenha pela implementação, pode ser baixo ou extremamente elevado. Logo, quanto maior for o grau de satisfação esperado pelo usuário, maior deverá ser a precisão de reconhecimento da aplicação.

Uma boa precisão é obtida, principalmente, através da escolha de um *engine* que atenda a maior parte das necessidades requeridas pela implementação, como o nível de ruído do ambiente e as características lingüísticas dos usuários; da elaboração de uma simples e mais eficaz possível estrutura gramatical, com o uso de uma lista de comandos válidos relativamente curta e que evite ao máximo a presença de dois ou mais comandos com pronúncias parecidas; e da qualidade do microfone e da placa de som utilizados, conhecidos como dispositivos de entrada, que podem facilmente fazer com que a aplicação confunda sons similares.

É interessante que a aplicação disponibilize em sua interface a opção para ativar e desativar o reconhecimento de voz, sempre informando o atual estado do *engine*.

Assim, o usuário poderá realizar atividades paralelas, como conversar com outra pessoa presente no ambiente ou falar ao telefone, sem necessariamente ter que fechar o aplicativo.

Dependendo da finalidade da aplicação, é sempre bem vindo um dispositivo para controlar o nível de confiabilidade, ou seja, o quão precisa a palavra necessita ser pronunciada para ser reconhecida. Aplicações em CALL, que geralmente trabalham com iniciantes na língua, precisam dispor desse controle para facilitar o aprendizado.

Caso um comando de voz não seja reconhecido como válido, ou o aplicativo não tenha entendido o que foi falado, uma boa idéia seria informar o usuário sobre o ocorrido e solicitar-lhe que repita o que antes fora dito, oferecendo-lhe, ainda, uma caixa de diálogo com uma lista dos possíveis comandos e uma breve descrição daquela etapa da aplicação. Em [6], o desenvolvedor faz uso dessa prática através de regras universais do tipo "*What Can I Say*" e "*Where Can I Go*".

Essa espécie de *help online*, bastante utilizada com ajuda de agentes animados, acarreta muitas horas de trabalho. Contudo, é uma ferramenta que contribui em demasia para a satisfação do operador, na medida em que lhe permite "caminhar com as próprias pernas". Nessa linha, outros pontos muito valorizados são o uso de comandos que fazem parte da linguagem computacional e a confirmação daqueles considerados perigosos dentro de um sistema, como "*Delete*", por exemplo.

2.4.2 Aspectos da implementação de TTS ("Text-to-Speech")

Em termos de TTS, o principal desafio é fazer com que a voz sintetizada soe menos robotizada e mais humana. Dentre os fatores que dificultam atingir esse objetivo, encontra-se a modelagem da prosódia, uma das principais responsáveis para que a voz sintetizada carregue o ritmo e emoção que um ser humano empregaria. O melhor exemplo da evolução da tecnologia TTS é o sistema da AT&T [16], o qual atinge uma voz bastante próxima da natural.

Uma das alternativas para formatar o texto a ser sintetizado é via linguagem XML (*Extensible Markup Language*). O texto XML permite que a aplicação controle

importantes características do texto de entrada e da voz padrão. Pausas, ênfases, *pitch*, volume, velocidade, vozes masculinas ou femininas, são alguns dos parâmetros que podem ser especificados para uma determinada palavra ou frase, sempre com a finalidade de minimizar a falta de naturalidade da voz sintetizada.

A opção da *Markup Language* possui limitações quando utilizada em sistemas com síntese concatenativa, onde segmentos de voz são previamente gravados com voz humana, o que torna bastante trabalhoso prover flexibilidade à voz, por meio desse método. Entretanto, os arquivos WAV pré-gravados devem ser utilizados em aplicações que demandam longos períodos de síntese, como leitura de documentos, e naquelas que se utilizam apenas de frases fixas, como aplicações para atendimento via telefone.

Assim como o reconhecimento, a síntese de voz deve ter um caráter opcional dentro da aplicação, pois nem todas as estações oferecerem os recursos necessários para sua empregabilidade e o próprio operador pode querer desativá-la para melhorar a performance da sua máquina. Assim é valorizado um retorno visual, além do falado, de todas as principais operações do sistema. Por fim, não é aconselhável o emprego de vozes humanas e das geradas eletronicamente na mesma seção, pois isso realça a diferença de qualidade entre elas, dificultando a familiarização do usuário com a interface [12].

2.5 Avaliação da Aplicação

São dois os principais tipos de avaliação de uma aplicação de voz, técnica e relacional [4]. A avaliação técnica é aquela que analisa, em tarefas específicas, o desempenho dos componentes técnicos do sistema, como por exemplo o sintetizador de voz. É a avaliação de interface (relacional) que procura através de entrevistas e simulações tornar o relacionamento homem-aplicação o mais amigável possível. A fase de avaliação é de extrema importância no processo de implementação. Os dados aqui coletados e a análise dos mesmos adicionam importantes informações técnicas e relacionais até então despercebidas aos olhos do desenvolvedor.

Independente do método de avaliação utilizado, é crucial que o estudo dos dados coletados atente para as deficiências da tecnologia de voz, que, como já dito, ainda está longe da perfeição. Além da limitação do estado da arte, outros fatores influenciam no resultado final da avaliação, como o ambiente e o equipamento para a realização das tarefas e o público alvo. Ambos devem se aproximar ao máximo do real, pois de outra forma são pontos capazes de degradar o nível de reconhecimento de voz por parte da aplicação. Por exemplo, uma aplicação de CALL deve levar em consideração que os alunos a estarão operando, provavelmente, em ruidosos laboratórios de informática, e que há diferença de sotaque entre as regiões de um mesmo país.

Em aplicações comerciais, a relação custo-benefício deve ser sempre analisada com muito cuidado. Em [17], é possível encontrar métodos e ferramentas para medir o custo para implantação e a eficiência de uma interface de reconhecimento e síntese de voz via telefone para *Call Centers*. Os principais benefícios trazidos por um IVR são a redução das despesas com operadores humanos e a economia de tempo nas ligações, mas sempre garantindo a informação ao cliente. A maneira utilizada para calcular automaticamente o sucesso da interação é a gravação das chamadas. Esse método não se preocupa com a satisfação do cliente ao ser atendido por uma máquina, que deve ser medida através da coleta de opiniões.

2.5.1 Avaliação Técnica

O método mais comum para medição de performance de reconhecimento de voz é o WER (*word error rate*). É calculado como a porcentagem dos erros de reconhecimento (somatória das palavras retiradas, substituídas e adicionadas) pelo total de palavras ditas na mensagem original.

Existe o método de avaliação léxico e sintático que é mais utilizado na fase de desenvolvimento do próprio sintetizador e/ou reconhecedor de voz. Pode ser empregado ainda na avaliação de componentes destinados a linguagem natural e gerenciamento de diálogos. Já para aplicações de alto nível não é de grande valia.

Outro método bastante difundido em aplicação de *command-and-control* é o que

avalia os erros de reconhecimento de sentença ao invés de palavras individuais. Na prática, isso significa que se uma sentença for mapeada com a mesma representação de uma frase referência, a mesma é entendida de forma correta. Nessa medição os erros de reconhecimento de palavras, cujo significado não é relevante ao entendimento da sentença, não são levados em consideração.

A perplexidade é uma medição padrão para caracterizar a dificuldade encontrada nas tarefas de reconhecimento no que tange ao modelo de linguagem utilizado, ou seja, como diferentes palavras são combinadas pelo *engine* de voz. Quanto maior a perplexidade mais livremente as palavras podem ser combinadas. Por outro lado, maior será a dificuldade do reconhecedor em identificar a correta seqüência das palavras. A perplexidade não atua nas propriedades acústicas das palavras, ou seja, erros ocasionados por palavras com sons semelhantes, porém com significados diferentes, não serão contabilizados por essa medição, mas sim pelo WER.

A qualidade da voz sintetizada pode ser medida através de dois fatores - entendimento e naturalidade. O nível de entendimento pode ser dividido em dois segmentos: primeiramente a precisão com que as palavras pronunciadas são recebidas pelo usuário, e em seguida de que forma elas foram compreendidas dentro do contexto proposto. A precisão pode ser calculada de maneira semelhante ao WER, comparando o que foi recebido a mensagens de referência. Já a compreensão pode ser avaliada através de questões e tarefas requisitadas ao receptor, que deve mostrar que conseguiu absorver o significado da mensagem.

A naturalidade só pode ser medida através da coleta de opiniões de um considerável número de usuários, o que torna tal aferição bastante subjetiva. Através de questionários, vários aspectos da voz sintetizada podem ser avaliados, como a fluência e a emoção do locutor. Ficou claro que o entendimento e a naturalidade estão relacionados, mas não diretamente atrelados. Sistemas com alto grau de sofisticação podem não ser dos mais agradáveis aos ouvidos dos usuários.

Quando aplicações baseadas em voz estão sendo construídas, a sua avaliação técnica deve ser interpretada de acordo com a sua empregabilidade. Em muitas

aplicações, 20% de WER é um valor perfeitamente aceitável, já para outras possuir um WER acima de 5% comprometeria seriamente o desempenho do sistema. Erros de reconhecimentos, que para algumas aplicações podem ser perfeitamente mascarados, para outras podem ser de fundamental importância.

2.5.2 Avaliação Relacional

Como já foi exposto anteriormente, a interface com o usuário é um ponto vital para o sucesso da aplicação. É isso que busca a avaliação relacional - melhorar ao máximo essa interação, através do treinamento e adaptação dos componentes técnicos, coletando opiniões dos usuários, definido gramáticas e vocabulário, etc.

Um método pouco utilizado para incrementar a relação homem-aplicação é a observação e gravação da cotidiana comunicação humana, os cumprimentos mais freqüentes, expressões características, entre outros, para futuras análises e modificações na interface da aplicação. O que desestimula nesse método é que as características da comunicação homem-homem diferem em várias maneiras da comunicação homem-aplicação. E, hoje, é utopia pensar de forma diferente. Outro método são os experimentos, como o WOZ (*Wizard of Oz*) [4], que simulam a interação de toda a aplicação ou apenas parte dela com usuários de teste antes de sua efetiva implementação. Também possui sérias limitações, já que simulações nem sempre reproduzem de maneira adequada o mundo real.

O terceiro e principal método de avaliação é o estudo real da interação homem-aplicação, onde a interface entre o sistema e os verdadeiros usuários pode ser examinada e testada, e dados podem ser coletados no real ambiente de uso da aplicação para futuros ajustes. Para [18], os resultados finais são estimulantes na medida em que demonstram que sistemas arquitetados e avaliados com base em dados de usuários reais podem ser tornar aplicações práticas e úteis.

Para se coletar as informações necessárias nos métodos acima apresentados, o questionário é um mecanismo muito eficaz [19]. Ele pode ser apresentado ao usuário de maneira escrita (papel) ou eletrônica (página Web), e pode contemplar respostas

limitadas (objetivas) ou respostas livres (subjetivas). Os questionários com respostas induzidas são os mais utilizados, já que são mais simples e permitem comparações com outros estudos já realizados em um determinado campo de pesquisa [20], além de evitar a ambiguidade dos testes de livre escolha [2].

Capítulo 3

Ferramentas para Desenvolvimento de Aplicativos com Suporte a Voz

Existe um grande número de empresas no mercado que apresentam soluções para que um desenvolvedor possa incorporar a tecnologia de voz em seus aplicativos. Empresas como a IBM e a Microsoft possuem soluções que adotam padrões como VoiceXML e SALT. A maioria dessas empresas adota uma API para o desenvolvimento de aplicativos. Na presente seção, discute-se duas delas, as quais bem representam as opções disponíveis. A primeira é a JSAPI (*Java Speech API*) da Sun [21] e a segunda é a SAPI (*Speech API*) da Microsoft [22].

O objetivo deste capítulo é servir como guia prático para desenvolvedores de software. Assim, alguns trechos de código (*snippets*) foram incorporados no corpo do texto, de maneira a melhor documentar os detalhes, os quais são deveras importantes em algumas situações.

3.1 JSAPI

A *Java Speech API* é uma especificação que permite aos desenvolvedores incorporarem tecnologia de voz aos seus aplicativos escritos no poderoso, compacto e orientado a objeto ambiente de programação Java. A JSAPI especifica uma inter-

face que independe da plataforma de desenvolvimento e suporta sistemas para ditado, *command-and-control* e síntese de voz.

A JAPI encontra-se entre a aplicação Java e o sintetizador e/ou reconhecedor de voz, disponibilizando um conjunto de classes, também escritas em Java, que representam a visão do programador sobre o *engine*, conforme mostra a Figura 3.1. A JSAPI é mantida como uma especificação de interface abstrata por boas razões. Para que as aplicações que utilizam a tecnologia de voz sejam portáteis, as mesmas não devem ter nenhuma conexão concreta com o *hardware* utilizado. Essa é a intenção proposta pela *Java Speech API*, onde a mesma interface de programação pode suportar uma variedade de produtos de uma maneira muito similar.

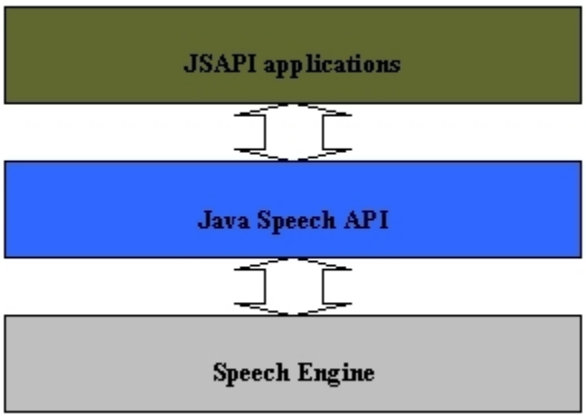

Figura 3.1: A arquitetura de alto nível de uma aplicação utilizando JSAPI

Apesar da Sun ter lançado a Versão 1.0 da *Java Speech API* em 1998, não comercializou qualquer tipo de implementação em JSAPI. Em vez disso, ela conta com um grupo de companhias que trabalham com ASR e TTS, entre elas a IBM, para desenvolverem aplicações e serviços baseados na especificação JSAPI. O fato de não existir referência de implementação por parte da Sun, não obriga necessariamente o desenvolvedor a trabalhar com *softwares* comerciais, como o Via Voice da IBM e o Cloud Garden [23], para adicionar interface de voz à sua aplicação. É possível fazer uso de

produtos com código livre, disponibilizados sem custo na Internet. O FreeTTS [14], para síntese de voz, é uma das opções existentes.

Os reconhecedores e sintetizadores de voz podem ser escritos na linguagem de programação Java. Essas implementações se beneficiam com a portabilidade da plataforma Java e com a crescente melhora de desempenho e velocidade das máquinas virtuais Java. O próprio FreeTTS e o Sphinx-4[1] [24] são exemplos de *softwares* livres desenvolvidos em Java. Todavia, a maioria dos produtos disponíveis são implementados em C ou C++ e acessados via plataformas específicas (API's) como *Apple Speech Managers* e *Microsoft Speech API*, ou através de API's proprietárias. Um aspecto importante é que usando as facilidades *Java Native Interface* e *Java Software Wrappers*, fornecedores podem implementar a JSAPI em uma camada de aplicação acima dos seus *softwares* de voz já existentes.

Para usar a *Java Speech API*, um usuário deve ter um mínimo de *software* e *hardware* disponível. Como já foi exposto, as exigências individuais de sintetizadores e reconhecedores de voz podem variar extremamente e os usuários devem ficar atentos às exigências computacionais do produto. Acima de tudo, é fundamental um *engine* de voz que suporte JSAPI. A especificação JSAPI 1.0, que normalmente acompanha o pacote de distribuição do *engine*, e o *kit* de desenvolvimento JavaTM 2 SDK Standard Edition v.1.4 [25], ou mais recente, precisam estar implementados na máquina.

3.1.1 Síntese de Voz usando JSAPI

Os sintetizadores são definidos e selecionados por características da voz tais como língua, sexo e idade. As especificações do núcleo *javax.speech.synthesis* são responsáveis pela síntese de voz na *Java Speech API*. Também definem o controle de saída da voz, das notificações de retorno, da gerência do vocabulário e do formato do texto a ser falado.

[1] Sistema de reconhecimento de voz do estado-da-arte. O Sphinx-4 atualmente suporta a JSGF (*Java Speech Grammar Format*), especificação que define o formato do texto da gramática de reconhecimento, mas não provê implementação à *Java Speech API*.

A classe Central da JSAPI é usada para encontrar e criar o sintetizador. Essa classe espera encontrar o arquivo *speech.properties*, provido pelo *engine*, nos diretórios user.home ou java.home/lib do sistema operacional. O método *SynthesizerModeDesc* é o responsável por receber as características que o sintetizador deve possuir.

```
SynthesizerModeDesc required = new SynthesizerModeDesc(
                    null,        // engine name
                    "general",   // mode name
                    Locale.US,   // locale
                    null,        // running
                    null);       // voice
```

O método de criação do sintetizador possui uma política implícita de seleção. Para o *createSynthesizer*, os parâmetros nulos indicam que a aplicação não se importa com as propriedades do sintetizador. Por exemplo, se a característica *mode* não for definida, todos os *engines* com vozes gerais e limitadas ao domínio serão selecionados.

Caso a aplicação não especifique a língua do sintetizador, a língua padrão do sistema, retornada por *java.util.Locale.getDefault()*, será utilizada. Se mais de um *engine* suportar a língua padrão, a classe Central dará a preferência a um *engine* que esteja *running* (em espera), e então a um que suporte o país definido no *locale*.

Com isso, é importante que a propriedade *locale* seja definida (Locale.US ou Locale.ENGLISH). Caso contrário, a plataforma utilizada pode ter como língua padrão, por exemplo, o português brasileiro e, nesse caso, a aplicação que esteja fazendo uso do FreeTTS apresentará falha, já que esse sintetizador trabalha apenas com vozes no idioma inglês.

Como já foi exposto, mais de um *engine* pode ser selecionado, dependendo das propriedades sugeridas. O objeto *EngineList*, descrito a seguir, surge como uma possibilidade de visualização dos sintetizadores disponibilizados, facilitando o estudo e a escolha do *software* que melhor se adequa à aplicação.

```
EngineList engineList =
    Central.availableSynthesizers(required);
for(int i = 0; i < engineList.size(); i++) {
   SynthesizerModeDesc desc =
        (SynthesizerModeDesc) engineList.get(i);
   System.out.println(desc.getEngineName()
                    + desc.getModeName()
                    + desc.getLocale());
   Voice[] voices = desc.getVoices();
   for (int j = 0; j < voices.length; j++) {
       System.out.println(voices[j].getName());
       System.out.println(voices[j].getGender());
       System.out.println(voices[j].getAge());
       System.out.println(voices[j].getStyle());
   }
}
```

Escolhido o sintetizador, o próximo passo é criá-lo, através do método *createSynthesizer* da classe Central. Os métodos *allocate* e *resume* alocam todos os recursos necessários e preparam o sintetizador para produzir a voz, respectivamente. Grande parte dos sintetizadores disponibilizam mais de uma voz. Quatro características estão presentes na voz: o nome, o sexo, a idade e o estilo. O método *getEngineModeDesc()* também nos permite discriminar essas vozes.

```
Synthesizer synth = Central.createSynthesizer(desc);
synth.allocate();
synth.resume();
SynthesizerModeDesc desc =
  (SynthesizerModeDesc) synth.getEngineModeDesc();
Voice[] voices = desc.getVoices();
Voice voice = voices[1];
```

As aplicações fornecem ao sintetizador de voz o texto a ser falado como um objeto *Java String*. A saída desse texto é conseguida através do método *speakPlainText*. Antes de sintetizar o texto, além da voz, propriedades como *pitch*, *pitch range*, velocidade (*speaking rate*) e volume de saída podem ser controladas pela interface *SynthesizerProperties* do sintetizador, que disponibiliza para cada propriedade um método *get* e um *set*, e ambos seguem os padrões *JavaBeans*.

O método *get* retorna o valor atualmente configurado. Já o método *set* define um novo valor para a propriedade e gera uma exceção em caso de falha, geralmente provocada porque o *engine* rejeitou ou limitou o valor solicitado pela aplicação. No caso do volume, o intervalo permitido é de 0.0 a 1.0. Logo, valores fora desse intervalo gerarão uma exceção. Por fim, o método *waitEngineState* garante que o texto seja sintetizado até o fim e os recursos do sintetizador sejam liberados pelo método *deallocate()*.

```
SynthesizerProperties props =
    synth.getSynthesizerProperties();
float Volume = (float) props.getVolume();
float SpeakingRate = (float) props.getSpeakingRate();
float Pitch = (float) props.getPitch();
float PitchRange = (float) props.getPitchRange();
try {
    props.setVoice(voice);
    props.setSpeakingRate(newSpeakingRate);
    props.setPitch(newPitch);
    props.setPitchRange(newPitchRange);
    props.setVolume(newVolume);
} catch (java.beans.PropertyVetoException e) {
    System.err.println("Erro encontrado");
}
synth.speakPlainText("My creator is a good man", null);
synth.waitEngineState(Synthesizer.QUEUE_EMPTY);
```

```
synth.deallocate();
```

A outra maneira de sintetizar o texto é através da JSML (*Java Speech Markup Language*) [21] e do método *speak*. A JSML, que segue o padrão XML, especifica a formatação do texto enviado aos sintetizadores de voz, permitindo que as aplicações controlem características importantes da voz produzida por um sintetizador. As pronúncias podem ser especificadas para palavras, frases, acrônimos e abreviaturas para assegurar uma melhor compreensão do texto. O controle explícito das pausas, do ritmo e da ênfase é oferecido para melhorar a naturalidade da fala.

Enquanto o texto é falado, o sintetizador pode ser solicitado a emitir notificações dos principais eventos à aplicação através da classe *SpeakableEvent*. Os eventos incluem o começo e o fim da saída de áudio correspondente ao texto enviado, o começo das palavras no texto, a saída dos marcadores embutidos no texto de entrada e possivelmente a saída de fonemas individuais. Esses eventos podem ser usados por aplicações para inúmeras finalidades, incluindo a coordenação entre múltiplos sintetizadores, a sincronização entre a fala e as animações faciais e o destaque das palavras no texto na medida em que são pronunciadas.

As aplicações podem fornecer informações de vocabulário aos *engines* de voz. O vocabulário é usado tipicamente para definir a pronúncia e a informação gramatical de palavras que podem estar sendo pronunciadas de maneira incorreta pelo sintetizador. A plataforma Java usa a configuração Unicode para todas as *strings*. A Unicode fornece uma excelente sustentação multilingual e inclui todo o IPA (*International Phonetic Alphabet*). O IPA é um conjunto padronizado de símbolos que documentam os fonemas. De posse dessa documentação e da interface *VocabManager* da JSAPI, o desenvolvedor tem a liberdade de redefinir a pronúncia de uma palavra e adicionar outras tantas.

Finalmente, uma inconveniência que pode ser encontrada na implementação da interface JSAPI é que as exceções *PropertyVeto* nem sempre são disponibilizadas corretamente quando os pedidos de mudança de propriedade são rejeitados. Em outras palavras, a aplicação de voz pode não estar apta a responder apropriadamente à situações onde as mudanças de propriedade são negadas, devido à circunstâncias como

limitações de recurso, por exemplo.

3.1.2 Reconhecimento de Voz usando JSAPI

Segundo [26], 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).

Assim como acontece com o sintetizador, a classe Central da JSAPI é usada para encontrar e criar o reconhecedor. O método *RecognizerModeDesc* é o responsável por receber as características que o reconhecedor deve possuir. Além das cinco propriedades já apresentadas, a classe Central é capaz de indicar se o modo de operação do *software* suporta a gramática para ditado. O código abaixo primeiro seleciona um *engine* de reconhecimento para ditado na língua inglesa e depois o cria, alocando todos os recursos necessários.

```
static Recognizer rec;
RecognizerModeDesc required = new RecognizerModeDesc();
required.setLocale(new Locale("en", ""));
required.setDictationGrammarSupported (Boolean.TRUE);
rec = Central.createRecognizer(required);
rec.allocate();
```

Uma gramática de regras suporta o diálogo discreto entre uma 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 [21]. Outra alternativa se dá através de objetos e métodos da própria JSAPI. As regras de uma gramática podem ser carregadas em tempo de execução através de um arquivo texto, uma URL ou uma regra por vez.

O código abaixo mostra uma gramática sendo carregada a partir de um arquivo contendo regras JSGF:

```
FileReader reader = new FileReader("grammar.txt");
RuleGrammar gram = rec.loadJSGF(reader);
```

A seguir, descreve-se como adicionar uma regra com a palavra "*hello*" à uma gramática, semelhante ao que foi feito acima, porém através de interfaces JSAPI:

```
RuleGrammar gram = rec.newRuleGrammar("sun.speech.test");
RuleToken word = new RuleToken("hello");
gram.setRule("ruleName", word, true);
```

A linha de código a seguir cria uma instância de gramática para ditado. Tal estrutura suporta a entrada de texto sem formato pré-determinado. 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 simplesmente ignoradas.

```
DictationGrammar gram = rec.getDictationGrammar(null);
```

Após o carregamento da(s) gramática(s), deve-se ligar um ou mais ouvidores (*listeners*) ao reconhecedor, eles serão responsáveis por interpretar o que foi dito, determinando qual processamento se refere ao que foi reconhecido.

```
rec.addResultListener(new teste());
```

Por fim, todas as gramáticas precisam ser disponibilizadas, salvas e ativadas, como descrito a seguir. Somente após estas três etapas é que o reconhecimento da gramática pode ocorrer. Os métodos da interface *Recognizer* que permitem iniciar e interromper o reconhecimento são, respectivamente, *resume()* e *pause()*.

```
gram.setEnabled(true);
rec.commitChanges();
rec.requestFocus();
rec.resume();
```

Neste momento, o reconhecedor espera que o usuário diga alguma coisa. Quando uma correspondência com a gramática ativa for detectada, o reconhecedor gerará um *resultEvent* que será recebido pelo método *resultAccepted*. Uma vez recebidas as informações sobre o que se "ouviu", pode-se desencadear um processamento pré-estabelecido pelo programador.

```
public void resultAccepted(ResultEvent e) {
   Result r = (Result) e.getSource();
   ResultToken tokens[] = r.getBestTokens();
   for (int i = 0; i < tokens.length; i++)
     System.out.println(tokens[i].getSpokenText() + " ");
   rec.deallocate();
   System.exit(0);
}
```

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. Por exemplo, em um nível de confiabilidade mínimo, todos os resultados são tratados como *resultAccepted*.

O nível de sensibilidade é o limite entre o som que deve ser processado e o que deve ser ignorado. É classificado com um valor que vai de 0.0 até 1.0, respectivamente referente aos sons mais baixos e mais altos. Seu valor *default* é 0.5, e quanto mais baixo for esse parâmetro, mais alto o usuário terá que falar.

```
RecognizerProperties props = rec.getRecognizerProperties();
```

```
props.setConfidenceLevel(new Float(0.0).floatValue());
props.setSensitivity(new Float(1.0).floatValue());
rec.commitChanges();
rec.resume();
```

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. São três os eventos gerados: um 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, basta que se associe um ouvidor de áudio (*RecognizerAudioListener*) ao reconhecedor.

3.2 SAPI da Microsoft

A SAPI é uma interface fornecida pela Microsoft de reconhecimento e síntese de voz para o desenvolvimento de aplicações baseadas nos sistemas operacionais Windows. Essa tecnologia faculta aos programadores acesso ao serviço de voz fornecido por um *engine* de síntese e reconhecimento de voz, conforme ilustra a Figura 3.2.

A comunicação entre a SAPI e o *engine* é feita através de uma interface Windows chamada DLL (*Dynamic Link Library*). O papel da DLL é agir como um tradutor entre as solicitações feitas pela API e as facilidades disponíveis no *engine*. Esse modelo, chamado WOSA (*Windows Open Services Architecture*), tem como vantagem oferecer à aplicação a possibilidade de acessar serviços providos por *softwares* de mais de um fabricante, desde que todos eles suportem SAPI.

A SAPI é uma interface do estilo COM (*Component Object Model*). Esses objetos estão incluídos nas interfaces DLL's, ou livrarias, de forma que em cada livraria podemos encontrar um ou vários objetos que nos vão permitir tratar um problema específico, e cada um deles, com o seu correspondente conjunto de propriedades, métodos e eventos.

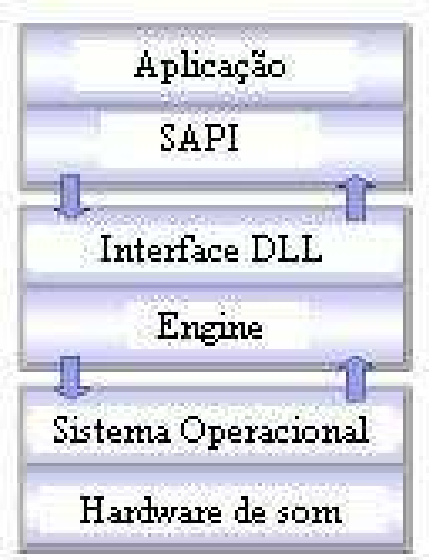

Figura 3.2: Arquitetura de uma aplicação utilizando a SAPI

Atualmente na versão 5.1, a *Microsoft Speech API* fornece uma interface de alto nível entre a aplicação e o *engine*. Contudo, tal interface também é capaz de controlar e gerenciar objetos e métodos da interface de baixo nível. Cada nível possui objetos próprios, que permitem aos programadores em C acessarem diretamente os serviços de voz:

- Interface de Alto Nível: nível que disponibiliza serviços básicos de voz. É excelente para construir menus de opções, executar funções através do comando de voz e leitura de textos simples.

- Interface de Baixo Nível: nível que fornece ao desenvolvedor maior flexibilidade na construção das aplicações, pois permite acesso aos atributos de configuração do *engine*.

A Microsoft disponibiliza gratuitamente o *kit* de desenvolvimento *Speech SDK* [22], com DLLs que suportam objetos OLE (*Object Linking and Embedding*), os quais permitem que uma aplicação seja controlada por outra, característica essencial

para que a camada mais alta da arquitetura SAPI também possa ser acessada por linguagens de programação conhecidas como de alto nível (Visual Basic, C Sharp, etc.). O *kit* contém ainda *engines* de reconhecimento e síntese de voz na língua inglesa.

A SAPI 5 possui interfaces para três aplicações fundamentais que são:

- Síntese de voz: transforma um texto escrito de entrada em voz sintetizada.
- Reconhecimento de comandos: o usuário dá como entrada uma série de comandos pré-definidos por meio da fala que serão transformados em textos escritos e que a aplicação entederá como ordens, executando-as.
- Ditado: transforma a fala contínua do usuário em texto escrito a ser utilizado pela aplicação.

3.2.1 Síntese de Voz usando SAPI

As implementações SAPI controlam o processo de síntese de voz através da interface COM *ISpVoice*. Depois de criar esse objeto, a aplicação precisa apenas chamar o método *ISpVoice::Speak* para gerar áudio de saída a partir de um texto ou arquivo de entrada. Adicionalmente, as propriedades do processo de síntese, como volume e velocidade da voz de saída, podem ser modificadas através das interfaces e métodos *ISpVoice::SetVolume* e *ISpVoice::SetRate*, tal como mostrado a seguir.

```
SpVoice Voice = new SpVoice();
Voice.Volume = 50;
Voice.Rate = 5;
Voice.Speak("Hello", SpeechVoiceSpeakFlags.SVSFlagsAsync);
Voice.Speak("C:/Teste.txt",
            SpeechVoiceSpeakFlags.SVSFIsFilename);
```

Também é possível redirecionar as amostras de áudio para algum destino especial que não seja a saída de áudio padrão adotada pelo sistema operacional da máquina, tais como arquivos de som (*ISpStream*), sistemas de telefonia ou qualquer *hardware*

específico (*ISpAudio*). No código abaixo, a voz sintetizada será salva em um arquivo de som.

```
SpeechStreamFileMode SpFileMode =
    SpeechStreamFileMode.SSFMCreateForWrite;
SpFileStream SpFileStream = new SpFileStream();
SpFileStream.Open("C:/Temp/Hello.wav", SpFileMode, false);
Voice.AudioOutputStream = SpFileStream;
Voice.Speak("Hello", SpeechVoiceSpeakFlags.SVSFDefault);
SpFileStream.Close();
Voice.AudioOutputStream = null;
```

Segundo [13], é desejável que a API permita ao desenvolvedor configurar os parâmetros de síntese de voz como velocidade, afinação, timbre, língua, etc., tanto através de invocação de funções, procedimentos e/ou métodos, como por meio da inserção de marcações no texto a ser sintetizado. Dessa forma, uma das alternativas para formatar o texto a ser sintetizado é via XML. A linguagem XML permite que a aplicação controle importantes características do texto de entrada e da voz padrão. Pausas, ênfases, características da pronúncia, volume e velocidade são alguns dos parâmetros que podem ser especificados para uma determinada palavra ou frase, sempre com a finalidade de minimizar a falta de naturalidade da voz sintetizada. Abaixo, segue exemplo de um código XML, que pode ser salvo em um arquivo texto e executado a partir do método *ISpVoice::Speak*.[2]

```
<volume level="50">
 This text should be spoken at volume level fifty
  <volume level="100">
```

[2]A estrutura do código e a relação dos elementos XML podem ser encontradas na documentação integrante do Microsoft Speech API SDK 5.1. Como se trata de um código XML, além do parâmetro que indica que a síntese será gerada a partir de um arquivo de entrada, é necessária a informação da referida linguagem. Assim o método ISpVoice::Speak segue a seguinte estrutura: Voice.Speak(“C:/Teste.txt”, SpeechVoiceSpeakFlags.SVSFIsFilename|SpeechVoiceSpeakFlags.SVSFIsXML);

```
   This text should be spoken at volume level one hundred
  </volume>
</volume>
```

O método *ISpVoice::SetVoice* permite que a voz padrão do sistema operacional, que é selecionada por características como língua, sexo e idade, seja alterada de acordo com a necessidade e finalidade da aplicação. Para isso, é necessário introduzir o conceito de *object token*. Conceitualmente, *token* é um objeto que representa um recurso disponível no registro da máquina e contém as seguintes informações:

- Um identificador que identifica unicamente o *object token*.
- Um nome universal que deve ser mostrado toda vez que o nome do *object token* for solicitado.
- O CLSID é usado para associar o objeto ao *token*.
- Um conjunto de atributos, que são as características particulares de um *object token*, como o fabricante de um reconhecedor e o sexo da voz.

Vários métodos SAPI retornam o referido *object token* de um específico tipo de recurso, tais como *ISpRecognizer::GetRecognizer* com o *object token* associado ao ASR *engine*; *ISpVoice::GetVoice* com o *object token* correspondente ao TTS *engine*. O código abaixo ilustra a escolha da "voz" a ser utilizada pela aplicação em função do valor presente em uma *comboBox*. A interface *ISpeechObjectTokens* recebe e enumera todos os recursos de voz disponíveis no computador.

```
SpObjectToken Token = new SpObjectToken();
SpObjectTokenCategory Cat = new SpObjectTokenCategory();
ISpeechObjectTokens TokenEnum;
TokenEnum = Voice.GetVoices(null,null);
if (comboBox1.Text == "Mary")
    {Voice.Voice = TokenEnum.Item(0);}
```

```
if (comboBox1.Text == "Mike")
    {Voice.Voice = TokenEnum.Item(1);}
if (comboBox1.Text == "Sam")
    {Voice.Voice = TokenEnum.Item(2);}
Debug.WriteLine(TokenEnum.Item(0).Id.ToString());
Debug.WriteLine(TokenEnum.Item(1).Id.ToString());
Debug.WriteLine(TokenEnum.Item(2).Id.ToString());
```

A SAPI comunica-se com a aplicação através do envio de notificações padronizadas para cada evento ocorrido. Para TTS, os eventos são usados, na sua maioria, para sincronizar ações em tempo real com a saída da voz sintetizada. Exemplos de eventos são início e fim de palavras, fonemas e até mesmo "visemes" (animações da boca). O mais importante evento ASR é o que indica que o *engine* reconheceu alguma seqüência de palavras. Na arquitetura SAPI, a interface *ISpEventSource*, que por sua vez é uma *ISpNotifySource*, é responsável por receber e tratar as notificações dos eventos gerados.

3.2.2 Reconhecimento de voz usando SAPI

Para executar ASR, uma gramática precisa ser definida para que o aplicativo saiba que ação executar quando uma determinada palavra lhe for enviada. Assim como o *ISpVoice* é o principal mecanismo para a síntese de voz, *ISpRecoContext* é a principal interface COM para o reconhecimento de voz. Através dela, a aplicação tem controle do conteúdo do que foi reconhecido e das gramáticas de reconhecimento, além do poder de parar e reiniciar o reconhecimento. O código abaixo mostra a criação de um objeto *ISpRecoContext* e a inicialização do reconhecimento.

```
SpSharedRecoContext RecoContext=new SpSharedRecoContext();
RecoContext.State = SpeechRecoContextState.SRCS_Enabled;
```

Como já mencionado, a SAPI dá suporte a ambas gramáticas - a CFG (*command-and-control*) e a probabilística, também chamada de gramática para ditado (*dictation*).

No código abaixo, uma gramática CFG (Grammar) contendo uma regra (Ruleone) foi criada, carregada e ativada. Daí, a regra será acionada toda vez que a palavra "*boy*" for reconhecida.

```
ISpeechRecoGrammar Grammar;
ISpeechGrammarRule Ruleone;
objectPropValue = "";
Grammar = RecoContext.CreateGrammar(0);
Ruleone = Grammar.Rules.Add("regraum",
  SpeechRuleAttributes.SRATopLevel |
  SpeechRuleAttributes.SRADynamic,1);
Ruleone.InitialState.AddWordTransition(null,"boy","",
  SpeechGrammarWordType.SGLexical,
  "boy",0,ref PropValue,0f);
Grammar.Rules.Commit();
Grammar.CmdSetRuleState("regraum",
  SpeechRuleState.SGDSActive);
```

O sucesso de uma aplicação que use uma gramática para ditado depende, dentre outros fatores, da qualidade do modelo de linguagem e de sua adequação à interação do usuário com o sistema. Para isso, os *engines* possuem modelos de linguagem específicos para dados tópicos, tais como medicina e engenharia. Abaixo, segue um exemplo de como uma gramática de ditado é criada, carregada e ativada.

```
ISpeechRecoGrammar Grammar;
Grammar = RecoContext.CreateGrammar(10);
Grammar.DictationLoad(null,SpeechLoadOption.SLOStatic);
Grammar.DictationSetState(SpeechRuleState.SGDSActive);
```

Depois da gramática ser definida, a aplicação precisa saber o que foi reconhecido. Para isso, utiliza-se o evento *RecoContext.Recognition*, conforme o código a seguir

descrito, onde uma caixa de texto (TextBox1) recebe o conteúdo do reconhecimento. No caso de se estar utilizando a gramática CFG, pode-se identificar qual foi a palavra ou frase, reconhecida através da interface *SpeechPhraseProperty*, que recebe todas as propriedades do reconhecimento, tais como a própria palavra e o seu parâmetro de identificação (provenientes do método *AddWordTransition*), o nível de confiança do reconhecimento (*SECLowConfidence*, *SECNormalConfidence* ou *SECHighConfidence*), entre outras.

```
RecoContext.Recognition += new
    _ISpeechRecoContextEvents_RecognitionEventHandler
    (RecoContext_Recognition);

private void RecoContext_Recognition(int StreamNumber,
  object StreamPosition,
  SpeechRecognitionType RecognitionType,
  ISpeechRecoResult Result)
{
   TextBox1.Text = Result.PhraseInfo.GetText(0,-1,true);
   SpeechPhraseProperty Phrase;
   Phrase = Result.PhraseInfo.Properties.Item(0);
   Debug.WriteLine(Phrase.Name.ToString());
   Debug.WriteLine(Phrase.Id.ToString());
   Debug.WriteLine(Phrase.Confidence.ToString());
}
```

3.3 Microsoft Agent

Apesar de não ser parte da SAPI, o MSagent (*Microsoft Agent*) [27] é uma tecnologia diretamente relacionada, já que permite criar poderosos personagens (*Agents*) e empregá-los em aplicações para a plataforma Microsoft Windows, além de associá-los

aos mecanismos de síntese e reconhecimento de voz, proporcionando uma interface gráfica muito rica (similar à descrita em [9]). Além de tornar a interface do *software* mais amigável, uma das aplicações mais populares desses agentes é fornecer uma espécie de *help on line*, onde seria possível consultar a lista de comandos de voz reconhecíveis em uma tela em separado, ou a pronúncia correta de uma determinada palavra.

Figura 3.3: Personagem do MSagent

A princípio o pacote de desenvolvimento MSagent disponibiliza quatro personagens padrões: Genie, Merlin (vide Figura 3.3), Robbie e Peedy. Cada personagem possui uma série de ações que ele pode executar. Por exemplo, o personagem pode saudar o usuário, voar pela tela, entre outras. Tais ações são definidas pelo criador do personagem. Em [28], é possível encontrar a documentação *on-line* do *Microsoft Agent*, incluindo a lista de animações para os seus personagens padrões. O *core* do componente, seus *Agents*, além de um editor que permite a criação de novos personagens, podem ser adquiridos gratuitamente na página da Microsoft na *Web* [27].

Dependendo da aplicação o usuário precisará ter esses arquivos instalados na máquina, todavia a Microsoft mantém um servidor de dados na *Web* com os arquivos

dos personagens do MSagent [29]. Portanto, é possível fazer com que o personagem seja obtido de um *site*. O arquivo .ACS, para uso local, possui todos os dados do personagem e suas animações no diretório *C:\Windows\msagent*. Já na otimização do acesso via *Web*, os arquivos estão divididos em .ACA e .ACF, separando os dados do personagem de suas animações.

Sua arquitetura usa Microsoft SAPI 4.0 [22] para suportar *engines* de reconhecimento e síntese de voz, sendo compatível com a grande maioria das linguagens de alto nível, como Microsoft Visual Basic, VBScript, JScript, C/C++, C# e J++; e ambientes de desenvolvimento que suportem COM e/ou interface ActiveX, incluindo Visual Basic for Application, Visual Studio .NET e Borland Delphi.

A qualidade da voz sintetizada e a precisão no reconhecimento da voz variam em função de vários fatores, como o método de síntese empregado, o *engine* utilizado, a linguagem desejada, entre outros. Os *engines* SAPI 4.0 Lernout & Hauspie TTS [30], disponíveis para *download* na língua inglesa e em várias outras, inclusive o português, foram licenciados pela Microsoft especificamente para uso com o *Microsoft Agent*. Adicionamente, outras companhias também oferecem *engines* de voz que suportam SAPI 4.0 e *Microsoft Agent*, como o Digalo Text-To-Speech [31], o Elan Speech Engine [32] e o Eloquent Technology ETI-Eloquence [33].

O código abaixo é uma ilustração do uso de agentes em uma aplicação SAPI com C#. O agente Merlin é carregado junto com o formulário e depois mostrado na tela. As duas últimas linhas proporcionam ações por parte do agente: na primeira ele realiza uma breve saudação e, em seguida, utiliza-se da tecnologia de voz para sintetizar o texto “*Exercise*”.

```
AxAgent.Characters.Load("Merlin",
   (object)"C:/Windows/Msagent/chars/merlin.acs");
Character = AxAgent.Characters.Character("Merlin")
Character.Show(null);
Character.Play("announce");
Character.Speak("Exercise",null);
```

Atualmente, a Microsoft disponibiliza apenas um reconhecedor de voz SAPI 4.0 para uso com o MSagent na língua inglesa, chamado *Microsoft Speech Recognition Engine* [27].

Similar ao que já foi exposto sobre reconhecimento, o componente *Microsoft Agent* nos permite adicionar comandos de entrada de voz responsáveis pela execução de alguma ação dentro da aplicação. Por exemplo, no código abaixo se o usuário falar "*Who is your Master?*" o personagem irá interagir respondendo a pergunta. Assim como este, vários outros *command-and-control* podem ser criados.

```
Character.Commands.Add("Who is your Master?",
   (object)"Who is your Master?",
   (object)"(Your(Master|Administrator))",
   (object)true,(object)true);
IAgentCtlUserInput ui;
ui = (IAgentCtlUserInput)e.p_userInput;
if(ui.Name == "Who is your Master?") {
  Character.Play("Pleased");
  Character.Speak("My Master name is John", null);
}
```

3.4 Comparação entre SAPI e JSAPI

As pesquisas realizadas demostraram que tanto a SAPI, como a JSAPI permitem o emprego das tecnologias de TTS e ASR. A especificação JSAPI é bem mais limitada, compacta e acessível que a SAPI. Como consequência dessa simplicidade, a JSAPI deixa a desejar em pontos de extrema importância. Ao contrário da SAPI, "a atual especificação de áudio da *Java Speech API* é incompleta" [21]. A JSAPI não permite o redirecionamento das amostras de áudio para arquivos e/ou dispositivos, além da saída de áudio padrão. Outro item que não é satisfatório é a sua interface léxica, que não possui uma definição clara.

Sobre a deficiência no controle léxico, as interfaces *VocabManager* e *DictationGrammar*, que abastecem o *engine* com informações sobre palavras que até então lhe são desconhecidas, chegam a ser discutidas na especificação JSAPI. Contudo, o seu objeto principal *Word* em nenhum momento é definido. Outra interface, a *Synthesizer.phoneme*, que retorna, de acordo com o IPA, a *string* fonética utilizada na pronúncia de uma determinada palavra, precisa ser empregada com restrições, pois apresenta uma falha grave, já que não consegue indicar uma pronúncia preferencial, caso um texto possua múltiplas formas de pronúncia.

A interface SAPI *ISpLexicon* faculta acesso ao vocabulário. O banco de dados léxico é um repositório de palavras e informações a elas relacionadas, como pronúncia e morfologia. A interface léxica da SAPI disponibiliza às aplicações e aos desenvolvedores de *engines* de voz um método padrão de como criar, acessar, modificar e sincrinizar seu conteúdo léxico.

A SAPI define dois tipos de vocabulário: aplicação e usuário. O primeiro engloba as palavras que todas as aplicações podem utilizar e, como está agregado ao *engine* de voz, é "*read-only*". Já o segundo tipo armazena as palavras específicas de um determinado usuário.

A Microsoft, por sua vez, não dá ao usuário a capacidade de consultar o conteúdo léxico dos seus *engines*, pois ela o considera "*by design*", ou seja, proprietário. Com isso, os desenvolvedores de *softwares* ASR e TTS, que façam uso dos reconhecedores e sintetizadores dessa empresa americana, só poderão manuserar as palavras do seu vocabulário (usuário), ou seja, as que eles próprios inserirem.

O código abaixo exemplifica a criação de um objeto da interface SAPI *ISpLexicon*. Em seguida, o substantivo "papai" é adicionado ao vocabulário do usuário com a pronúncia da palavra inglesa "*red*"[3] e, por fim, é realizada a síntese do registro inserido.

```
SpLexicon Vocab = new SpLexicon();
SpVoice Voz = new SpVoice();
```

[3]A relação de fonemas pode ser encontrada no *Appendix A - SAPI 5 Phonemes* da documentação integrante do Microsoft Speech API SDK 5.1.

```
Vocab.AddPronunciation
  ("papai", 1033, SpeechPartOfSpeech.SPSNoun, "r eh d");
Voz.Speak("papai", SpeechVoiceSpeakFlags.SVSFlagsAsync);
```

Através do método *GetWords*, é possível resgatar todas as palavras que fazem parte da base léxica de um determinado usuário, ou até mesmo da aplicação, caso essa esteja disponível. O código abaixo mostra como é feita essa consulta, com a quantidade de palavras obtidas sendo mostrada em uma caixa de texto (textBox1).

```
SpLexicon Vocab = new SpLexicon();
ISpeechLexiconWords word;
int y = 0;
word = Vocab.GetWords(SpeechLexiconType.SLTUser, out y);
textBox1.Text = word.Count.ToString();
```

Com relação ao combate à voz robotizada, a *Microsoft Speech API* possibilita a modificação de algumas características prosódicas básicas do texto a ser sintetizado através de comandos XML, incluindo a *tag vce*, que exerce um certo controle sobre a qualidade de voz, sendo raramente implementada pelos *engines* que suportam SAPI. Já com JSAPI, também é possível trabalhar com esses parâmetros, tanto através de invocação de procedimentos e/ou métodos, como pela JSML. A JSML é um subconjunto da linguagem XML. Isso significa que sua sintaxe não é amarrada a JSAPI, uma vez que a XML define um padrão aberto, independente de plataforma e amplamente extensível.

Ambas suportam o uso de gramáticas para ditado (probabilísticas) e CFG (contexto livre). As interfaces *ISpRecoResult* da Microsoft e *ResultListeners* da Sun são manuseadas pela aplicação para recuperar as informações sobre reconhecimento e falso reconhecimento do *engine*. A SAPI disponibiliza ainda dados de reconhecimentos hipotéticos, ou seja, textos apenas propostos pelo reconhecedor. Interessantes funcionalidades de relacionamento com o usuário podem ser encontradas nas duas API's, como opções para "ligar" e "desligar" o *engine* de voz, ativar e desativar a(s)

gramática(s) de reconhecimento e o controle do nível de confiabilidade do ASR. Já a opção para ajustar o nível de sensibilidade do reconhecedor foi encontrada apenas na JSAPI.

A escolha de qual interface utilizar depende de vários aspectos, que vão desde critérios técnicos, como o sistema operacional, a critérios bastante subjetivos, como facilidade de uso. A SAPI é uma interface do estilo COM com automação OLE, e por isso pode ser acessada a partir de linguagens de alto nível. Neste trabalho é empregada a linguagem de programação C Sharp, também conhecida como C#. "A JSAPI é uma extensão da plataforma Java" [34], logo a linguagem empregada no desenvolvimento do sistema deve ser a mesma utilizada para escrever a JSAPI, no caso Java. A JSAPI sugere que o aplicativo se comunique com o *engine* a partir de *Java Events*. Assim, a JSAPI pode ser utilizada em qualquer sistema operacional para o qual exista uma máquina virtual Java. Em contraste, a SAPI é restrita a sistemas operacionais Windows.

No que diz respeito a *engines*, existe uma gama de *softwares* que suportam SAPI e/ou JSAPI. Esse é um ponto importante, pois permite ao desenvolvedor avaliar a qualidade de *engines* de fabricantes diferentes e escolher o que melhor se aplica às pecularidades do seu sistema. A SAPI é adotada pela maioria das empresas do setor, tais como Dragon, IBM e Lucent. Já a JSAPI é bem menos difundida.

Capítulo 4

Aplicativos VUI Desenvolvidos

4.1 “CALL” (“Computer Aided Language Learning”)

CAI (*Computer Assisted Instruction*) é o processo de aprendizagem com o auxílio de computadores. Nos últimos 40 anos, houve um crescimento exponencial no uso de computadores como provedores de instruções. Durante essa rápida revolução tecnológica, a CAI se tornou mais refinada e os computadores se transformaram em um sofisticado, e por que não dizer essencial, instrumento de ajuda, na medida em que se inventou uma nova maneira de ensinar e aprender, envolvendo mais facilmente o estudante no universo intelectual. Com o passar dos anos, mais e mais sistemas CAI estão incorporando interfaces multimodal, como som, imagem, vídeo e programas interativos que suportem uma vasta gama de aplicações. Mais recentemente, iniciou-se a incorporação de interfaces de voz nos ambientes denominados CALL (*Computer-assisted Language Learning*).

Apesar de criar novas possibilidades para o desenvolvimento de ferramentas de aprendizagem, o impacto prático dos aplicativos CALL no campo da educação de línguas estrangeiras tem sido bem modesto, conseqüência da falta de padronização das suas interfaces e avaliações, e a carência de dados concretos que comprovem os benefícios pedagógicos desses sistemas [35]. Entretanto, especialistas, como [36], sustentam que, com a drástica e acelerada invasão dos computadores em escolas e

residências, os professores de línguas precisam, o quanto antes, rever seus conceitos e trabalhar na busca de mecanismos que ajudem a superar os obstáculos impostos pelas limitações presentes nas tecnologias de voz.

Nos últimos anos, a performance dos "computadores pessoais" tem evoluído com a produção de processadores cada vez mais velozes, fato que encoraja a implementação de algoritmos de processamento de voz em tempo real economicamente viáveis. A integração da alta qualidade de áudio, alta resolução de imagens e vídeos com a saída de textos e gráficos dos computadores convencionais tem gerado recentes sistemas multimídia que fornecem poderosas ferramentas de "treinamento" exploradas no campo CALL. O resultado desse esforço são protótipos com interfaces de voz para treinamento da pronúncia nos mais variados idiomas, leitura de textos e ensino de línguas estrangeiras em contextos pré-definidos [18].

Um importante campo de estudo dentro do reconhecimento e processamento de voz é o treinamento da pronúncia. Tutores inteligentes interagem com os alunos e os guiam na repetição de palavras e frases, ou na leitura de sentenças, com o propósito de praticar tanto a fluência (qualidade da pronúncia fonética), quanto a entonação (manipulação do parâmetro *pitch*) em uma determinada língua. Essas características prosódicas são particularmente importantes no aprendizado de uma língua, já que uma prosódia incorreta pode obstruir a comunicação entre os oradores. Erros na entonação podem dar ao ouvinte uma impressão completamente equivocada da postura tomada pela pessoa que está falando, assim como uma sentença sem fluência torna praticamente impossível para o ouvinte entender o contexto do diálogo.

Para muitos lingüistas, como [37], os aplicativos para treinamento da pronúncia devem ser empregados para acentuar a interação entre professor e aluno. Mas há quem pense o contrário. Para [38], a grande vantagem desses sistemas é que eles podem substituir os "tutores humanos", e com isso a presença de um instrutor torna-se desnecessária. Outra vantagem é que os alunos podem seguir seu próprio ritmo dependendo da sua velocidade de aprendizagem. Todavia, lingüistas podem assistir os técnicos, sugerindo quais aspectos da voz do praticante necessitam ser focados e es-

tipulando limites para os desvios identificados na pronúncia. Em [37], o autor sugere que os esforços sejam concentrados nos aspectos da voz que possam afetar sua inteligibilidade, e não em todo e qualquer sinal que a diferencie de um modelo nativo proposto.

O sucesso desse tipo de implementação depende muito do correto *feedback* que não conta com a própria percepção do estudante. Geralmente, esse retorno ao praticante é dado através de métodos automáticos de avaliação. Alguns sistemas dão nota à pronúncia do usuário, enquanto outros fazem uso de técnicas audiovisuais, que geram sinal de voz conjuntamente com a sua correspondente informação visual. Apesar de disporem de *softwares* de ASR extremamente poderosos e de boa precisão, esses ambientes de treinamento possuem duas limitações extremamente complexas [39]. Primeiramente, sem a inclusão de modelos acústicos para vozes não-nativas, a integração usuário-aplicativo fica comprometida e limitada a um pobre vocabulário, já a outra dificuldade é apontar explicitamente qual parte da sentença não foi corretamente pronunciada, e como ela pode ser corrigida [40].

Um estilo de *software* CALL muito difundido, principalmente por escolas de idiomas, faz uso de uma interface de alto nível e gramática definida dentro de um contexto, procurando sempre retratar situações cotidianas da vida real para a prática da língua estrangeira [41]. Basicamente, o aluno interage com o aplicativo através de exercícios pedagógicos, onde ele precisa escolher uma resposta dentro de um número limitado de alternativas que lhe são apresentadas na tela, ou pode ser desafiado a responder questões subjetivas. Em ambos os métodos, chamados *closed response* e *open response*, respectivamente, o usuário é introduzido a responder oralmente os questionamentos através de estímulos escritos, falados e visuais.

Nesta seção discute-se a implementação de um *software* CALL através das interfaces providas pelo *kit* de desenvolvimento *Microsoft Speech SDK 5.1*. A aplicação mescla o uso dos métodos *closed response* e *open response* para aprendizado de língua estrangeira (inglesa) com o auxílio do computador. Além dos exercícios objetivos e subjetivos propostos, existe a possibilidade de um treinamento prévio, onde

o aluno pode enriquecer seu vocabulário escutando via TTS palavras individualizadas, ou frases, sempre com o respectivo retorno visual, ilustrando o significado da palavra, ou a ação associada à frase. A Figura 4.1 mostra uma tela do protótipo de CALL que está sendo desenvolvido para o ensino da língua inglesa, em função da maior disponibilidade de *engines*[1].

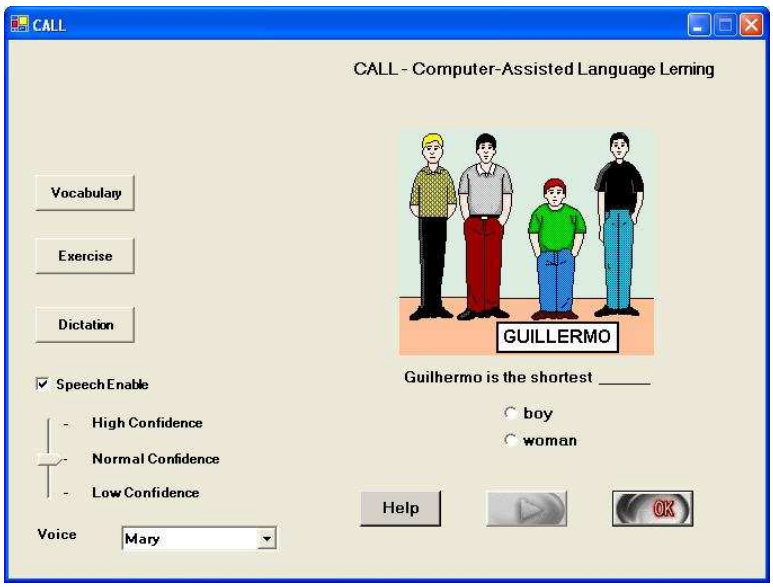

Figura 4.1: Uma das telas do aplicativo para ensino de língua inglesa

O sistema disponibiliza a facilidade de um agente personalizado, "Merlin", criado a partir do componente *Microsoft Agent* para prover *feedback* e assistência quando necessário juntamente com o sintetizador de voz Lernout & Hauspie TruVoice[2]. O agente retorna textualmente e oralmente o resultado do exercício, "*excelent*" ou "*wrong*", dependendo, claro, da resposta enviada pelo usuário via ASR, teclado ou *mouse*. O Merlin também participa do *help* criado para auxiliar o aluno durante os exercícios. Basta o aprendiz pronunciar a palavra "*help*", ou pressionar o botão cor-

[1]Foram utilizados os *engines* presentes no Sistema Operacional Windows XP da Microsoft, são eles: o *Microsoft English Recognizer v5.1* para ASR e os *Microsoft Mary*, *Sam* e *Mike* para TTS.

[2]É necessário a instalação do suporte SAPI 4.0 runtime para a utilização Microsoft Agent com qualquer software de voz, caso o sistema operacional seja o Windows XP.

respondente, que dicas lhe serão apresentadas como forma de ajuda na tentativa de resolução da questão. O código abaixo exemplifica todo o fluxo de ativação da regra *help*, desde a crição da gramática que a comporta, até a ação a ser executada com alto nível de reconhecimento (TTS por parte do agente animado). Para a inicialização do reconhecimento, basta que o método IniciaVoz ( ) seja enfocado em qualquer outra parte do código fonte.

```
private void IniciaVoz ()
{
SpSharedRecoContext RecoContext=new SpSharedRecoContext();
RecoContext.State = SpeechRecoContextState.SRCS_Enabled;
ISpeechRecoGrammar Grammar;
ISpeechGrammarRule Ruleone;
objectPropValue = "";
Grammar = RecoContext.CreateGrammar(0);
Ruleone = Grammar.Rules.Add("regraum",
  SpeechRuleAttributes.SRATopLevel |
  SpeechRuleAttributes.SRADynamic,1);
Ruleone.InitialState.AddWordTransition(null,"help","",
  SpeechGrammarWordType.SGLexical,
  "help",3,ref PropValue,0f);
Grammar.Rules.Commit();
Grammar.CmdSetRuleState("regraum",
  SpeechRuleState.SGDSActive);
RecoContext.Recognition += new
    _ISpeechRecoContextEvents_RecognitionEventHandler
    (RecoContext_Recognition);
}
```

```
private void RecoContext_Recognition(int StreamNumber,
 object StreamPosition,
 SpeechRecognitionType RecognitionType,
 ISpeechRecoResult Result)
{
    SpeechPhraseProperty Phrase;
    Phrase = Result.PhraseInfo.Properties.Item(0);
    if (Phrase.Id == 3 &&
      Phrase.Confidence.ToString() == "SECHighConfidence")
    {
        AxAgent.Characters.Character("Merlin").Speak
          ("Peter Pan is a boy",null);
    }
}
```

A familiarização do usuário com a interface se mostrou bastante dependente do TTS em língua nativa, no nosso caso a língua portuguesa, para instruções e *feedback*. Essa situação é compartilhada com outros projetos em desenvolvimento no Brasil (e.g., [9]). Uma alternativa aqui estudada foi a criação de uma estrutura multilingual com a incorporação do *toolkit* do CSLU à aplicação.

Apesar de relativamente precário, o suporte à TTS em PT_BR pela “voz” AGA do CSLU [42] é bastante útil. Em resumo, o processo de geração da voz AGA foi baseado na síntese por difones, um dos métodos mais populares para criação de voz sintética [43]. O difone é a região entre os “centros” de dois *fones* adjacentes. O centro de uma realização fonética é a região mais estável da mesma, enquanto que a transição de um fone para outro contém mais variação (efeitos da co-articulação), o que torna difícil a modelagem. Após a seleção do conjunto de *fones*, os difones são coletados utilizando palavras que possuam o respectivo difone no meio [44]. A coleta foi feita em uma câmara anecóica, com equipamento de gravação de alta-qualidade, incluindo um laringógrafo. O laringógrafo é utilizado para garantir a qualidade da gravação e

extrair informação sobre o *pitch* e pulso glotal. Após a coleta, as palavras foram alinhadas temporalmente com os difones e segmentadas posteriormente. Finalmente, os modelos prosódico e léxico são gerados para produzir uma voz o mais natural possível e lexicamente correta.

O *toolkit* do CSLU é uma plataforma para o desenvolvimento de aplicativos de diálogos. O *software* disponibiliza quatro níveis de interface para o desenvolvimento de aplicativos. No nível mais baixo, os desenvolvedores podem usar código C para acessar todos os componentes, tais como o *engine* ASR. Não há, contudo, suporte para uma API genérica como a SAPI ou JSAPI. Apesar de não ser a solução ideal, discute-se a seguir a estratégia utilizada para sintetizar em PT_BR usando CSLU. Uma das diretrizes adotadas, foi buscar uma solução simples, a qual não envolvesse modificações no código do CSLU. Assim, optou-se pela utilização de arquivos *batch* que invocavam o TTS do CSLU, sob o comando do aplicativo. Os passos adotados foram os seguintes ($cslu é a pasta onde o *software* CSLU foi instalado):

Dentro do diretório $cslu/Festival/1.4.2/festival/lib, o aplicativo cria um arquivo com o texto que deseja sintetizar. Nesse exemplo, assume-se que o nome desse arquivo é “comandos.txt”. O comandos.txt possui o seguinte conteúdo:

```
(voice_aga_diphone) (SayText "ENTRE O TEXTO AQUI")
```

Então, no diretório $cslu/Toolkit/2.0/bin, criou-se uma cópia do arquivo Festival.bat (chamada aqui de “call.bat”), e modificou-se sua última linha para:

```
$CSLU\Festival\1.4.2\bin\festival.exe --batch "comandos.txt"
```

Após esses passos, basta ao aplicativo criar um arquivo comandos.txt e invocar o arquivo call.bat. Por exemplo, a partir de um programa C, bastaria a chamada:

```
system("$cslu\\Toolkit\\2.0\\bin\\call.bat");
```

Nota-se que a solução apresentada acima é ineficiente no sentido de que o processo de escrita e leitura em disco é lento. Contudo, o mesmo é simples e pode ser usado em situações onde a velocidade e qualidade do TTS não são críticas.

Diante dos problemas encontrados com o CSLU, optou-se, novamente, pela empregabilidade do *Microsoft Agent*. Dado que os personagens Genie, Merlin, Peedy and Robby são compilados para utilizarem o *engine* para o inglês L&H TruVoice como sintetizador de voz *default*, fez-se necessária a instalação do L&H TTS3000 para língua portuguesa, igualmente licenciado pela Microsoft. Além do *engine*, mostrou-se fundamental a instalação do componente de linguagem. Esses componentes são livrarias (arquivos DLL) que adicionam suporte a síntese em uma determinada língua, com exceção da inglesa, ao componente *Microsoft Agent*, ou seja, com eles é possível alterar a propriedade TTSModeID do personagem. A linha de código abaixo mostra como selecionar a língua inglesa para um agente previamente carregado.

```
Character.LanguageID = 0x409;
```

No geral, tanto as vozes sintetizadas pelos *engines* da Microsoft, quanto as produzidas pelos licenciados, não foram bem aceitas nas simulações realizadas no laboratório de pesquisa, ambiente que muito lembra uma sala de aula informatizada, com usuários que possuem nível básico a intermediário na língua inglesa. A robotização da voz foi bastante criticada. Para melhorar sua naturalidade e clareza, comandos XML podem ser inseridos para alterar parâmetros da voz, como volume, *rate* e *pitch*, e com isso elevar os índices de compreensão e satisfação.

A tecnologia ASR por trás desse tipo de implementação CALL é bastante simples, mesmo nos mais modernos *softwares* encontrados no mercado. Como o reconhecimento é sempre focado em contextos (opções de resposta limitadas), o número de regras necessárias é reduzido, fato que simplifica as gramáticas empregadas. Tudo isso faz com que esses sistemas sejam bem robustos, e esse não se comportou de maneira diferente, apesar que nenhum algoritmo para acompanhamento dos erros de reconhecimento de voz foi aqui empregado. Constatou-se que realizar o treinamento sugerido pelo *engine* resulta em um considerável ganho de precisão no momento do reconhecimento contínuo. O uso de microfones *head-mounted* também se mostrou de grande valia, já que eles diminuem e mantêm constante a distância entre a boca e a entrada de áudio.

A avaliação da interface foi considerada boa nos experimentos realizados. Os usuários relataram, nas entrevistas de opinião realizadas, que a navegação entre os menus é de fácil entendimento e que os comandos de voz requeridos em cada etapa da aplicação são bem claros. Citaram como importantes as opções para ativar e desativar o reconhecimento de voz, o *help online* disponibilizado através do agente animado e o controle do nível de confiabilidade do ASR. Ainda com relação ao parâmetro *confidence*, caso o usuário fale alguma seqüência que combine com a gramática carregada e ativa, porém com nível de confiabilidade abaixo do exigido, a aplicação alerta o aluno do ocorrido e sintetiza a palavra ou frase, com o intuito de melhorar a pronúncia do praticante. Contudo, muitos questionaram o fato do programa não solicitar repetição, ou mesmo informar, quando o comando de voz não é reconhecido.

4.2 Software de Chat ("bate-papo") com Voz

IRC (*Internet Relay Chat*) é um sistema que permite que grupos de pessoas possam conversar a partir de qualquer lugar do mundo via Internet. Uma rede IRC consiste de um conjunto de servidores. Os clientes conectam-se ao IRC através desses servidores. Essa arquitetura composta por diversos servidores em cada rede tem a finalidade de ajudar a incrementar a performance e a capacidade de recuperação do sistema. Adicionalmente à facilidade de troca de informações com privacidade, os usuários de IRC podem fazer parte de canais, mais conhecidos como "salas de bate-papo", que possuem nomes individuais, geralmente, de acordo com o assunto lá discutido.

Segundo [45], desde que foi introduzido por Jarkko Oikarinen em 1988, o IRC rapidamente cresceu em popularidade e atualmente chega a ter mais de um milhão de pessoas ao redor do mundo conectadas ao mesmo tempo. O protocolo utilizado pelo IRC foi claramente definido cinco anos depois na documentação RFC 1459, tornando o sistema mais acessível. Como resultado dessa exposição, muitos programas *clients*, que permitem que usuários conectem-se a uma rede IRC, foram desenvolvidos.

Dentro desse ambiente, surgiu o *PircBot Java IRC API* [46]. O *PircBot* é

um *framework* Java que possibilita o desenvolvimento de programas *clients*, também chamados de *bots*, capazes de interagir automaticamente com uma rede IRC. O pacote *PircBot* é gratuito e escrito inteiramente em Java, com isso ele rodará em qualquer sistema operacional que tenha uma Máquina Virtual Java a partir da versão 1.1.

Existe um razoável número de implementações que se utilizam do *PircBot* para as mais variadas finalidades [46]. Podemos citar as que empregam as facilidades desse pacote na gerência de canais (monitorando o que é dito, quem entra e quem sai da sala); na avaliação de desempenho dos servidores através do estudo dos *logs* gerados; e na integração das redes IRC com canais de notícias internacionais e nacionais.

Paul Mutton, em sua aplicação "IRC Text to Speech with Java" [47], buscou desmistificar a idéia de que o uso do IRC diminui a produtividade nos ambientes de trabalho. Ele disse que, quando usado apropriadamente, o IRC é capaz de diminuir distâncias entre as sucursais de uma grande companhia, por exemplo, agilizando a organização e a realização de reuniões, e o tratamento das tarefas mais simples, onde não se requer muita formalidade. Mas, para isso, é preciso que o ususário não seja obrigado a direcionar toda a sua atenção ao diálogo que trafega pelo canal, isto é, que ele seja capaz de realizar atividades paralelas a uma reunião via IRC.

Seguindo esse entendimento, o presente trabalho buscou a complementação da pesquisa feita por Paul Mutton [47] com a integração do *PircBot* com um *engine* de reconhecimento de voz através da especificação JSAPI. O objetivo foi, através de uma aplicação puramente Java, dar ao usuário a possibilidade de enviar suas mensagens ao canal de IRC via comando de voz, falando naturalmente o que deseja transmitir. Assim, ele estará apto a continuar seu trabalho e somente desviar sua atenção ao IRC quando for absolutamente necessário. Além da *Java IRC API*, obviamente, a pesquisa aqui apresentada baseou-se no pacote de desenvolvimento *Cloud Garden TalkingJava SDK* e nos *engines* de voz *Microsoft English Recognizer v5.1* e *Microsoft Mike*.

O *Cloud Garden TalkingJava SDK* [23] é uma implementação completa da especificação JSAPI para plataforma Windows, compatível com qualquer *engine* SAPI 4 e SAPI 5 de reconhecimento e síntese de voz. Essa característica disponibiliza um

maior número de *softwares* de voz para o desenvolvedor Java. Também inclui pacotes extras, por exemplo, para redirecionamento de áudio, permitindo que o *stream* de áudio seja gravado/lido em arquivo ou transmitido através de uma rede. Para tal, utiliza-se de outra implementação da Sun, a *Java Media Framework*.

Para fazer uso da aplicação basta seguir os passos abaixo:

- Antes de tudo é necessário que o *kit* de desenvolvimento *JavaTM 2 SDK Standard Edition v.1.4* [25] esteja implementado na máquina.

- Deve-se fazer o *download* do arquivo com o pacote de distribuição do PircBot 1.4.4 em [46]. Em seguida, descompactar o arquivo recebido *pircbot-1.4.4.zip* em qualquer diretório da máquina. Por exemplo, caso o local escolhido para a extração seja a pasta *C:\PircBot*, o diretório *C:\PircBot\pircbot-1.4.4* será criado, juntamente com os seus sub-diretórios.

- O pacote *pircbot-1.4.4* contém, entre outras, uma classe abstrata denominada *PircBot*. Em função dela ser abstrata, não se pode criar instâncias a partir da mesma, mas é possível estendê-la e, com isso, empregar suas funcionalidades. Para tal basta copiar e colar o algoritmo abaixo em um arquivo chamado *irc.java* e salvá-lo na pasta *JAVA_HOME\bin*, caso não se queira perder tempo configurando as diretivas PATH e CLASSPATH com as variáveis ambientes Java:

```
import org.jibble.pircbot.*;
import java.util.Locale;
import javax.speech.*;
import javax.speech.synthesis.*;
public class irc extends PircBot {
 static Synthesizer synth;
 public irc(String name) {
   try {
     setName(name);
```

```
        SynthesizerModeDesc require =
          new SynthesizerModeDesc(
                null,
                null,
                Locale.US,
                null,
                null);
        synth = Central.createSynthesizer(require);
        synth.allocate();
        synth.resume();
      } catch (Exception w) { w.printStackTrace(); }
    }
    public void onMessage(String channel, String sender,
      String login, String hostname, String message) {
        String input = sender + "says:" + message;
        synth.speakPlainText(input, null);
    }
  }
```

- Pode-se fazer com que a classe “irc” responda à eventos pré-estabelecidos. Por exemplo, no código acima, toda mensagem enviada ao canal será sintetizada pelo *engine* alocado. Para isso, utilizamos o método *onMessage*. Esse método será chamado toda vez que alguém enviar uma mensagem ao canal. Se necessário, é possível trabalhar com múltiplos canais ao mesmo tempo, já que o nome do canal é passado como um dos parâmetros do método.

- Fazer o download do *Cloud Garden TalkingJava SDK* [23]. O arquivo recebido *TalkingJavaSDK-163.zip* deve ser descompactado na pasta *C:\CloudGarden*. O arquivo *C:\CloudGarden\packet.jar*, obtido com a ação anterior, também deve ser extraído no diretório *C:\CloudGarden*.

- As bibliotecas responsáveis pelo *wrapper* que a implementação *Cloud Garden* faz em cima de um *engine* SAPI, de forma que o programador possa usar a JSAPI: *C:\CloudGarden\cgjsapi.jar* e *C:\CloudGarden\cgjsapi163.dll*; devem ser copiadas nas pastas *JAVA_HOME\jre\bin* e *JAVA_HOME\jre\lib\ext*.

- O código abaixo deve ser copiado e colado no arquivo *JAVA_HOME\bin\irconn.java*. Ele executa a conexão automática com o codinome "testeirc" no servidor de IRC "eu.brasnet.org" e entra no canal de conversação "#belem". Com o módulo *Verbose* ativado, é possível visualizar e, conseqüentemente, salvar todos os eventos enviados e recebidos pelo canal na janela de *Prompt*. Uma vez conectado ao canal, utiliza-se da gramática probabilística JSAPI e do método *sendMessage* da classe abstrata *PircBot* para enviar as mensagens solicitadas pelo usuário via voz.

```
import javax.speech.*;
import javax.speech.recognition.*;
import java.io.FileReader;
import java.util.Locale;
import org.jibble.pircbot.*;
public class irconn extends ResultAdapter {
  static Recognizer rec;
  static irc bot = new irc("testeirc");
  public void resultAccepted(ResultEvent e) {
    Result r = (Result) e.getSource();
    ResultToken tokens[] = r.getBestTokens();
    String texto = "";
    for (int i = 0; i < tokens.length; i++)
       texto = texto+" "+tokens[i].getSpokenText();
    bot.sendMessage("#belem",texto);
  }
```

```
    public static void main(String args[]) {
      try {
         RecognizerModeDesc required =
            new RecognizerModeDesc();
         required.setLocale(new Locale("en", ""));
         required.setDictationGrammarSupported
            (Boolean.TRUE);
         rec = Central.createRecognizer(required);
         rec.allocate();
         DictationGrammar gram =
            rec.getDictationGrammar(null);
         gram.setEnabled(true);
         rec.addResultListener(new irconn());
         rec.commitChanges();
         rec.requestFocus();
         rec.resume();
         bot.setVerbose(true);
         bot.connect("eu.brasnet.org");
         bot.joinChannel("#belem");
      } catch (Exception e) {
         System.out.println("Erro");
         e.printStackTrace();
      }
    }
  }
```

- Para compilar e executar os códigos acima, é preciso abrir uma janela de *Prompt* e digitar os seguintes comandos no diretório *JAVA_HOME\bin*:

  Para compilar os códigos:

```
javac -classpath .\;C:\CloudGarden\cgjsapi.jar;
C:\PircBot\pircbot-1.4.4\pircbot.jar irc.java

javac -classpath .\;C:\CloudGarden\cgjsapi.jar;
C:\PircBot\pircbot-1.4.4\pircbot.jar irconn.java
```

Para executar a aplicação:

```
java -classpath .\;C:\CloudGarden\cgjsapi.jar;
C:\PircBot\pircbot-1.4.4\pircbot.jar irconn
```

Constatou-se, na prática, que a gramática para ditado é mais complexa que a gramática composta por regras definidas via código. Porém, analisando pelo lado do programador, ela possui uma usabilidade bem mais simples que a gramática com contexto livre. Isto porque a gramática probabilística é construída dentro do reconhecedor, fazendo com que o *engine* responda por toda a complexidade léxica e a abstraia da aplicação, o que torna esse tipo de gramática tipicamente mais exigente computacionalmente e menos precisa que uma simples gramática com regras.

O entendimento da *Java Speech API* foi fácil e rápido, assim como a sua combinação com outra API, no caso a *Java IRC API*. Essa última característica é de grande valia para atender expectativas básicas do mundo moderno, no momento que permite o enriquecimento da capacidade de comunicação *person-to-person* a partir de aplicações feitas na sua totalidade em Java.

Outro sintetizador de voz usado nessa pesquisa é o FreeTTS 1.2 [14]. O FreeTTS foi o primeiro sistema *text-to-speech* a ser escrito inteiramente em Java. É um projeto com código livre desenvolvido dentro dos laboratórios da Sun [48]. Seu algoritmo é baseado no Flite [49], da Universidade de Carnegie Mellon, e sua arquitetura no consagrado sintetizador de voz Festival TTS [50], desenvolvido pelo Centro de Pesquisa em Tecnologia de Voz da Universidade de Edimburgo. O Flite é mais compacto, mais rápido e menos flexível do que o Festival, e foi projetado, principalmente, para o uso em dispositivos e usuários portáteis. O FreeTTS busca o melhor dos dois sintetizadores

escritos em C, ou seja, performance e flexibilidade. Embora escrito em Java, é capaz de entregar um bom desempenho. Isso se deve muito ao fato da necessidade do JDK 1.4, que contem um novo pacote de I/O e diversos outros otimizadores, que dão maior rapidez à maioria dos programas desenvolvidos em Java.

Recentemente, o suporte para o MBROLA [51] foi adicionado por Marc Schröder [14]. Essa é uma característica valiosa para o FreeTTS, dado o número limitado de banco de dados de voz disponível. A qualidade das vozes disponíveis no FreeTTS não é um dos seus pontos fortes, pelo contrário, deixa muito a desejar quando comparadas a vozes de outros sistemas. O MBROLA fornece três vozes americanas e uma inglesa e são reconhecidas como sendo de boa qualidade [2]. Além disso, a "us1" é uma voz feminina. Antes desta, não havia nenhuma voz feminina disponível para o FreeTTS. Desde a sua disponibilização, nenhuma nova voz foi desenvolvida para o FreeTTS. As vozes disponíveis para o FreeTTS são:

- Kevin - voz de 8kHz e não limitada ao domínio.
- Kevin16 - voz de 16kHz e não limitada ao domínio.
- Alan - voz de 16kHz e limitada ao domínio.

A voz Alan é a única de alta qualidade e limitada ao domínio, isto significa que está disponível somente a um pré-determinado domínio da aplicação. Alan é utilizado somente para informar a hora do dia. Apesar de algumas restrições de portabilidade, o MBROLA já foi utilizado em um grande número plataformas [51]. Contudo, o FreeTTS não suporta o MBROLA no ambiente Windows [14]. Em razão disto, essa combinação não foi empregada na presente pesquisa.

Apesar da JSAPI disponibilizar ASR e TTS, o FreeTTS, sendo um sintetizador de voz, não implementa a interface de reconhecimento, mas sim uma parte da especificação de síntese de voz da JSAPI 1.0, e como conseqüência dessa parcialidade, a *Java Speech API* para o FreeTTS possui várias restrições. A JSAPI não é um programa, mas sim uma especificação, que define um conjunto de interfaces obrigatórias

e opcionais que o engine de voz precisa implementar para se tornar compatível com essa API.

Uma das principais fraquezas da JSAPI é exposta quando o *engine* suporta apenas as suas interfaces mandatárias, que fornecem funcionalidades bastante limitadas dentro do que já foi desenvolvido na área de tecnologia de voz. Por exemplo, a especificação *Java Speech Markup Language* é uma das funcionalidades opcionais da JSAPI que não é suportada pelo FreeTTS. Os objetos da JSML dão ao *engine* informações detalhadas sobre a pronúncia de palavras e frases e a capacidade de controlar a prosódia. O FreeTTS é capaz de extrair o texto a ser dito dentro da JSML. Contudo, nenhum dos métodos dessa linguagem é processado.

4.3 Implementação

Com o passar dos anos, o foco das técnicas de sintetização por uma voz clara, mas sem qualquer trabalho de entonação, foi perdendo espaço para a corrente de pesquisadores que defendiam a inclusão do efeito emoção na voz sintetizada. A intenção é tornar a síntese a mais natural possível para o ouvinte, permitindo que este consiga perceber sentimentos de alegria, raiva, tristeza ou medo apenas escutando o que foi sintetizado. As pesquisas, que estudam a relação entre voz e emoção, divergem em muitos pontos com relação a combinação dos fatores que influenciam na prosódia, dado que essa percepção é algo bastante particular e subjetivo.

Contudo, o consenso sobre as características que as propriedades da voz devem ter para simular maior emoção foi obtido por Murray&Arnott [1], de acordo com a Tabela 4.1. Esse estudo apresenta uma clara descriminação do papel de cada propriedade no resultado final do processo e foi empregado em pelo menos dois sistemas com reconhecida qualidade, são eles: o HAMLET, criado pelos próprios Murray&Arnott [19], e o sistema baseado na interação Festival-MBROLA desenvolvido por John Stallo [2].

Tabela 4.1: Guia proposto por Murray&Arnott [1]

	Raiva	Felicidade	Tristeza	Medo
Velocidade	+	++	- -	+++
Pitch	++++	+++	- -	++++
Pitch Range	+++	+++	- -	+++
Volume	+	+	-	+
Alteração no Picth	descendente	ascendente	descendente	descendente
Qualidade da Voz	ofegante	ofegante, berrando	resonante	irregular
Articulação	presa	mais confusa	confusa	precisa

O objetivo deste capítulo é acrescentar ao sistema atual do FreeTTS a habilidade de controlar a prosódia e os parâmetros emocionais da voz por ele oferecida, como a intensidade aplicada a certas palavras ou frases, inclusão de pausas, alteração do *pitch* e da velocidade da voz. O manuseio de tais características é possível através das interfaces *high level* da especificação JSAPI implementadas pelo FreeTTS. Desta forma, todas as modificações necessárias são feitas sem qualquer interferência no código original desse sistema.

Esta pesquisa é significativa porque a maioria das aplicações existentes de TTS não incluem emoção na voz. Em [2], o autor cita como uma falha dos sintetizadores de voz atuais, o fato deles não considerarem o efeito emoção, produzindo vozes monótonas, ou levemente distinguíveis da voz mecanizada. Dentre os sintetizadores comerciais, o único que oferece um grande número de comandos, incluindo a facilidade de manipular a pronúncia no nível fonético, é o DECtalk da Compaq [52], que foi o sintetizador escolhido por Murray&Arnott [19] para suas implementações. Já Stallo [2] optou pelo Festival em conjunto com o MBROLA, por oferecerem os mesmos recursos do DECtalk, mais algumas vantagens, como a disponibilidade gratuita, maior flexibilidade, entre outras.

Conseqüentemente, há uma necessidade real para adicionar o efeito emoção na voz sintetizada e seria um ganho valioso ao FreeTTS poder transmitir todo o contexto de uma mensagem. Além disto, é uma oportunidade maravilhosa de contribuir para a melhoria deste imaturo sintetizador, não somente por ter seu código livre, mas porque o FreeTTS tem a vantagem da portabilidade sobre outros sistemas de TTS, como o Festival e o MBROLA, ou seja, é capaz de funcionar em qualquer plataforma que possua uma máquina virtual Java [53].

Tornar a voz sintetizada mais realista e capaz de expressar emoções é a meta da presente pesquisa, apesar da limitação do FreeTTS com a JSML. A *synthesizer.properties* foi a interface utilizada para manusear as propriedades do sintetizador durante a sua operação e, com isso, alterar o efeito da voz de saída. Outras duas interfaces opcionais JSAPI, que poderiam contribuir com o manuseio dos fonemas, a *Vocabulary Management* e a *Synthesizer.phoneme*, foram pesquisadas, porém nenhuma é suportada pelo FreeTTS. No total, são cinco as propriedades controláveis pela *synthesizer.properties*:

- Voz - é o principal ponto de controle da saída de um sistema TTS. Cada voz é definida por subclasses que proporcionam variações em suas características, como o nome, sexo, idade e estilo de voz do locutor.
- Volume - responsável pela quantidade de palavras por minuto.
- Velocidade - modifica a intensidade de saída da voz.
- *Pitch* - modifica a F0 médio da pronúncia. A sigla F0 é a freqüência do primeiro harmônico e é, também, conhecida como a freqüência fundamental.
- *Pitch range* - modifica a escala em torno de que os valores F0 são permitidos para oscilar.

Como foi visto, a interface *synthesizer.properties* disponibiliza apenas um controle sobre os parâmetros básicos da prosódia. Com isso, as modificações em fonemas, conhecidas como *low level*, não serão implementadas. De acordo com [53], esta

etapa requer a habilidade de ter um controle mais fino sobre o processo gramatical da pronúncia, já que é necessário modificar a duração, o *pitch* e a informação mais importante no nível fonético. Os pesquisadores Murray&Arnott [19] e Stallo [2] desenvolveram regras prosódicas que servem como uma excelente indicação de quais configurações precisam ser alteradas para diferentes tipos de emoções.

Mesmo ciente da necessidade dessas alterações no processo de adição do efeito emoção à voz, a decisão de não fazer uso dos parâmetros *low level*, descritos nas três últimas linhas da Tabela 4.1, foi tomada, pois se constatou que um estudo aprofundado no código do FreeTTS teria que ser realizado para a configuração desses parâmetros. O FreeTTS possui sua própria interface lexical, dando liberdade ao programador para adicionar palavras com sua respectiva pronúncia e alterar as que já estão configuradas. Todavia, esta atividade foge do foco principal do trabalho, que é a empregabilidade da especificação JSAPI.

As emoções implementadas na presente pesquisa seguem parcialmente o guia proposto na Tabela 4.1 e levam em consideração as modificações sugeridas por Stallo [2]. A Tabela 4.2 mostra os parâmetros *high level* adotados por Stallo [2] para sintetizar a voz realçada com a emoção especificada. Nota-se que ele produziu pessoais ajustes a algumas regras de alto nível propostas na Tabela 4.1, especificamente na ênfase (*pitch*) aplicada à raiva. Os valores são expressos com base na voz neutra e indicam em quanto os valores padrões relativos a velocidade, *pitch*, *pitch range* e volume, precisam ser modificados. Apenas três emoções foram utilizadas, pois se mostrou ser um número suficiente para constatar, ou não, a eficiência da implementação e servir como experiência para a inclusão de novas emoções.

A pesquisa feita por Stallo [2] foi seguida na medida em que fornece uma base contínua para a comparação, já que o autor disserta com riquezas de detalhes sobre a teoria e os resultados obtidos, além de adicionar informações pessoais dos participantes. Assim, é possível concluir se o presente trabalho atingiu, ou não, o objetivo de entregar uma voz sintetizada capaz de expressar emoção.

Tabela 4.2: Valores sugeridos por Stallo [2]

Emoção	Velocidade	Pitch	Pitch Range	Volume
Felicidade	+10%	+20%	+175%	1.0
Raiva	+18%	-15%	-15%	1.7
Tristeza	-15%	-5%	-25%	0.6

Em [2], o autor afirmou que o efeito produzido na voz dependia do sintetizador utilizado. Isto também foi observado com o FreeTTS através de testes preliminares. Conseqüentemente, leves alterações aos parâmetros da Tabela 4.2 se mostraram necessárias, porém sempre fiéis aos princípios adotados. Os valores finais utilizados neste trabalho estão descritos na Tabela 4.3.[3]

Tabela 4.3: Modificações implementadas para adicionar emoção à voz neutra

Emoção	Velocidade	Pitch	Pitch Range	Volume
Felicidade	+11.8% (190)	+60% (160)	+80% (90)	0.9
Raiva	+23.5% (210)	-30% (70)	-20% (40)	1.0
Tristeza	-17.6% (140)	-25% (75)	-36% (32)	0.85
Neutra	170	100	50	0.9

4.4 Avaliação

Uma vez implementados os valores da Tabela 4.3 na voz sintetizada, foi preciso avaliar se a emoção aplicada à pronúncia pode ser reconhecida. O método de avaliação aplicado neste trabalho foi o questionário com respostas forçadas, seme-

[3] A exceção PropertyVeto (java.beans.PropertyVetoException) não se mostrou presente quando o pedido de mudança da propriedade volume da voz sintetizada ultrapassou o limite permitido (1.0) e foi negado.

lhante ao empregado por Stallo [2] em sua pesquisa, onde o participante precisa escolher uma palavra, dentre as que lhe forem apresentadas, que melhor representa a emoção por ele percebida na voz tocada. A vantagem de um questionário com estrutura muito parecida e com as mesmas sentenças utilizadas por Stallo [2] é a facilidade de comparação entre os resultados obtidos e a análise de discrepâncias que porventura possam aparecer. A avaliação se consubstanciou em uma série de 20 frases, divididas da seguinte maneira:

1. Cinco sentenças sem emoção no contexto e reproduzidas com voz neutra.
2. Cinco sentenças com emoção no contexto e reproduzidas com voz neutra.
3. Cinco sentenças sem emoção no contexto e reproduzidas com voz emotiva.
4. Cinco sentenças com emoção no contexto e reproduzidas com voz emotiva.

A primeira série foi usada para determinar a neutralidade das sentenças sem contexto emocial. A segunda, por sua vez, demostra a influência do conteúdo da sentença na classificação e o grau de entendimento da voz produzida pelo sintetizador. E, por fim, as duas últimas séries avaliam o sucesso da implementação. Além das quatro emoções aqui estudadas (raiva, felicidade, tristeza e neutralidade), foram inseridas, entre as possíveis respostas, as emoções alternativas de surpresa e nojo, além de uma comentada (outros), com a única finalidade de confundir o participante.

Todas as frases foram disponibilizadas em arquivos de som em um *website* [4], juntamente com o questionário, onde os participantes acessaram e deixaram suas respostas. Todos os trinta e sete participantes são brasileiros com nível de conhecimento no mínimo intermediário da língua inglesa. Grande parte já teve alguma vez na vida contato com vozes sintetizadas (89,29%) e responderam o questionário por vontade própria, após convite. Logo, não é garantido o uso de computadores modernos, fones de ouvido, caixas de som de boa qualidade e locais livres de ruído. O objetivo foi testar o uso do sintetizador em vários tipos de equipamentos e ambientes. Antes de tudo, os participantes foram orientados a escutar o texto a seguir, sintetizado com voz neutra, para que se familiarizassem com a voz sintetizada:

[4] http://www.sherlock-holmes.oi.com.br/nelson

"Hello. Welcome to the emotional text to speech evaluation on the web. Let's see if you can recognise different emotions in my voice. You will be asked to determine the emotion present in a series of 20 short sentences and this should take no more than 10 minutes of your time."

A JSAPI não dispõe de mecanismos que permitam a gravação do texto sintetizado em arquivos de som .wav. Para isso, foi necessário alterar a propriedade de direcionamento de áudio do próprio sintetizador FreeTTS. O código está disponibilizado abaixo e necessita se fazer presente antes que o método *speakPlainText* seja invocado.

```
if (voice instanceof
  com.sun.speech.freetts.jsapi.FreeTTSVoice) {
    com.sun.speech.freetts.Voice freettsVoice =
    ((com.sun.speech.freetts.jsapi.FreeTTSVoice)
    voice).getVoice();
  freettsVoice.setAudioPlayer(new MultiFileAudioPlayer());
}
```

4.5 Análise dos Resultados

Nos testes preliminares realizados por Stallo [2], as frases raivosas mostraram-se com uma sonoridade muito semelhante a apresentada pelas sentenças com sentimento de felicidade, muito provavelmente em função do sintetizador de voz não ser capaz de implementar todas as correlações entre diálogo e emoção contidas na Tabela 4.1, tais como articulação, parâmetros de qualidade de voz, entre outros. Diante dos resultados iniciais, Stallo [2] optou por reduzir o *pitch* e o *pitch range* da raiva com relação a voz neutra, dando um tom mais ameaçador e melancólico à voz, ao invés de incrementá-los como recomenda a literatura. Stallo [2] também fez uso, com leves modificações, de algumas regras prosódicas (*low level*) desenvolvidas por Murray&Arnott [19].

Após análise dos dados obtidos na fase de desenvolvimento, onde os valores sugeridos na Tabela 4.2 foram testados, decidiu-se não aumentar tanto o *pitch range* das

frases com o sentimento de felicidade, compensando essa perda com um acréscimo mais generoso no valor do parâmetro *pitch*. O objetivo foi reduzir o grau de excitação do locutor e melhorar o percentual de reconhecimento desse tipo de emoção, muito confundida com surpresa na pesquisa feita por Stallo [2]. Outra alteração significativa foi a redução mais brusca no valor do parâmetro *pitch* das emoções de raiva e tristeza, já que sobretudo as frases tristes apresentaram entonações pouco distinguíveis da neutralidade, e vice-versa. A confusão estabelecida entre felicidade/surpresa e tristeza/neutralidade se faz presente em vários trabalhos dedicados à essa área de pesquisa [54].

Tabela 4.4: Cinco sentenças sem emoção no contexto e reproduzidas com voz neutra

	Feliz	Triste	Raiva	Neutro	Surpreso	Nojo	Outros
Neutro	9.26%	12.54%	5.42%	60.27%	5.45%	1.64%	5.42%

A observação que pode ser feita diante dos resultados apresentados na Tabela 4.4 foi que houve um grande reconhecimento de que o locutor estava se comunicando sem expressar emoção (60,27%). O percentual de votos na opção triste não foi elevado (12,54%), diferentemente do que foi observado por Stallo [2], onde 36,9% dos participantes optaram por classificar a voz neutra como triste, contra 44,4% dos que escolheram a neutralidade. Esse melhor resultado pode ter sido alcançado em função dos ajustes realizados nos parâmetros da voz padrão do FreeTTS, que apresentou uma acentuada monotonia, apesar do sintetizador ter sido modulado para ser neutro.

Vale ressaltar a dependência entre o conteúdo da frase e a percepção dos ouvintes para descriminar emoções. Por exemplo, na sentença teoricamente neutra *"I saw her walking along the beach this morning"* 13,51% dos participantes perceberam que o locutor estava feliz e 62,16% optaram corretamente pela neutralidade. Por outro lado, a sentença *"The book is lying on the table"* se mostrou a mais neutra entre as cinco, com 83,78% dos participantes a classificando como sem emoção. É árdua a tarefa de se encontrar frases com conteúdo que não influencie na percepção do ouvinte [19].

Tabela 4.5: Cinco sentenças com emoção no contexto e reproduzidas com voz neutra

	Feliz	Triste	Raiva	Neutro	Surpreso	Nojo	Outros
Feliz	40.54%	2.70%	2.70%	32.43%	5.41%	2.70%	13.51%
Triste	-	43.24%	8.11%	35.14%	8.11%	-	5.41%
Raiva	2.70%	8.10%	44.59%	31.08%	7.95%	2.70%	2.70%
Neutro	5.41%	2.70%	5.41%	40.54%	27.03%	10.81%	8.11%

A Tabela 4.5 mostra que mais de 40% dos participantes atentou para o conteúdo da frase no momento de fazer a distinção entre as emoções que lhes foram apresentadas. Essa divisão, quase que idêntica, entre os votos emotivos e neutros, demonstra o bom desempenho do sintetizador no fator entendimento, e que a falta de estímulo emocional na voz causa dúvida e prejudica o completo convencimento do ouvinte. Os resultados foram semelhantes aos alcançados por Stallo [2], com exceção da frase com sentido de felicidade, que em sua pesquisa não teve boa percepção, apenas 13,3% das respostas, contra 68,9% de neutralidade. O presente trabalho não encontrou tal dificuldade, já que obteve 40,54% de percepção para a sentença com conteúdo feliz.

Um número relevante de entrevistados (27,03%) entendeu que o locutor estava surpreso ao pronunciar a frase *"Smoke comes out of a chimney"*. A frase foi definida na teoria como neutra, porém voltamos ao mesmo dilema exposto a pouco, encontrar frases com conteúdo sem tendências emotivas. Um detalhe contido em alguns comentários enviados - o de que era difícil a percepção de algumas das emoções disponibilizadas como opção (tais como nojo ou surpresa) - também pode interferir nesse e em outros resultados. Tal situação produz a hipótese de alguns dos entrevistados haverem optado por essas emoções anteriormente citadas de forma tendenciosa, em razão da crença de que elas fatalmente deveriam surgir como resultado, isto é, ser a opção correta em pelo menos um dos vinte itens (afinal, "por que ali estariam?" completando o pensamento sugestionado), ainda que não tenham, de fato, os entrevistados percebido a referida emoção em nenhum dos itens.

Tabela 4.6: Cinco sentenças sem emoção no contexto e reproduzidas com voz emotiva

	Feliz	Triste	Raiva	Neutro	Surpreso	Nojo	Outros
Feliz	54.05%	10.81%	8.11%	5.41%	16.22%	2.70%	2.70%
Triste	5.40%	43.25%	5.40%	31.08%	4.05%	5.40%	5.40%
Raiva	8.11%	-	35.14%	51.35%	2.70%	2.70%	-
Neutro	13.51%	10.81%	2.70%	45.95%	18.92%	-	8.11%

As sentenças aqui avaliadas são iguais as empregadas na seção de texto e voz neutros, restando clara, diante dos resultados apresentados na Tabela 4.6, a forte influência das alterações aplicadas à voz sintetizada na percepção das emoções por parte dos ouvintes. Na prática, as mudanças sugeridas e empregadas nos parâmetros do estímulo de felicidade surtiram o efeito desejado, com 54,05% de percepção e apenas 16,22% de surpresos, bastante superior ao conseguido por Stallo [2] (24,4% feliz e 40% surpresa).

Já os estímulos de raiva e tristeza não obtiveram a avaliação esperada, mesmo com as alterações impostas. Elas foram demasiadamente confundidas com a voz neutra. Isso sugere a necessidade das regras prosódicas nesses dois estímulos, principalmente nas sentenças sem conteúdo emocional explícito. Apesar que Stallo [2], mesmo fazendo uso das regras de baixo nível, encontrou a mesma dificuldade em dissociar vozes tristes de neutras em ouvintes que não têm o inglês como primeira língua, cerca de 42% dos entrevistados.

Os resultados encontrados na Tabela 4.7 reforçam o que foi constatado por outros autores que pesquisaram a emoção em voz sintetizada, ou seja, mostram uma elevada média de reconhecimento para todas as emoções estudadas. Adicionalmente, uma das perguntas da enquete procurou avaliar subjetivamente a qualidade da voz sintetizada, e 48,65% dos participantes encontraram facilidade em entender o que foi dito, apesar da reconhecida falta de naturalidade presente no TTS.

Tabela 4.7: Cinco sentenças com emoção no contexto e reproduzidas com voz emotiva

	Feliz	Triste	Raiva	Neutro	Surpreso	Nojo	Outros
Feliz	81.08%	-	2.70%	5.41%	5.41%	-	5.41%
Triste	2.70%	59.46%	18.92%	8.11%	-	2.70%	8.11%
Raiva	2.70%	2.70%	59.46%	17.56%	8.11%	8.11%	1.35%
Neutro	5.41%	10.81%	-	48.65%	10.81%	8.11%	16.22%

A percepção da emoção nem sempre reproduz o esperado, em função da clara subjetividade que envolve o campo emocional. No entanto, numa avaliação geral, os resultados aqui obtidos mostraram-se consistentes e comprovaram o sucesso da implementação de estímulos emocionais no FreeTTS.

Capítulo 5

Conclusão

A tecnologia de voz incrementa a habilidade da empresa em controlar como, quando e onde as transações de seus clientes serão realizadas. Muitas companhias já abraçaram a idéia de tecnologias como o IVR (*Interactive Voice Response*) para atendimento automático via telefone, *web sites* e máquinas de auto-atendimento. Esses sistemas de síntese e reconhecimento de voz, comprovadamente, alcançaram bons conceitos no que diz respeiro à eficiência e confiabilidade, e, como conseqüência, economizam à empresa e aos seus usuários tempo e dinheiro. Contudo, é importante ter em mente que o estado-da-arte está longe da perfeição, e que existem limitações práticas impostas por um conjunto de fatores, como o próprio usuário final, o ambiente e a finalidade da aplicação.

Há muitas aplicações que se utilizam das técnicas de processamento de voz para se tornarem mais atrativas, podendo ser citados tradutores multilinguais, processadores de texto com reconhecimentos para ditado e *command-and-control*, dispositivos de segurança com identificador de voz e as que se propõem ao estudo de idiomas. O ensino de línguas é um processo difícil e que requer um trabalho cuidadoso. Segundo [55], os educadores estão se mostrando cada vez mais suscetíveis aos *softwares* CALL na tentativa de manter a atenção dos seus alunos pelo maior tempo possível.

Este trabalho apresentou um aplicativo multimodal CALL com agentes animados baseado nas interfaces providas pelo *kit* de desenvolvimento *Microsoft Speech*

SDK 5.1 e nos personagens e facilidades do componente *Microsoft Agent*. A interface de programação de alto nível SAPI 5 mostrou-se uma ferramenta poderosa e flexível para o desenvolvimento de aplicações de voz nas plataformas Microsoft Windows. Têm como vantagens a vasta gama de *engines* e implementações de voz disponíveis no mercado que suportam e empregam SAPI; uma arquitetura coerente que permite acesso a características de configuração do *engine*; uma rede ampla e padronizada de notificações; e uma completa documentação integrante do pacote SDK.

Segundo a Microsoft, o novo sistema SAPI.NET, ainda em fase Beta de desenvolvimento, expandirá a capacidade da atual versão SAPI, especialmente na área de tecnologia de voz para *web pages*.

Através da "voz" AGA, integrante do *toolkit* do CSLU, e do componente *Microsoft Agent* a comunicação da máquina com o usuário na língua portuguesa tornou-se possível. As soluções foram citadas com entusiasmo nas avaliações do *software* CALL, principalmente a que empregou o MSagent, sobretudo em função da interatividade proporcionada por seus personagens.

A outra API aqui estudada foi a *Java Speech API*. A idéia de uma especificação de interface abstrata, independente de plataformas e de *engines* com código livre escritos em Java apresentou-se em 1998 como uma alternativa aos *softwares* comerciais das grandes empresas. Todavia, poucas foram as implementações desenvolvidas em JSAPI até o presente momento, talvez pelas deficiências e desinformações encontradas em muitos pontos. São exemplos do desinteresse criado em torno dessa especificação: a Sun nunca comercializou qualquer tipo de implementação; a IBM descontinuou o pacote para alocação de recursos *IBM Speech for Java*[1]; e nenhuma movimentação no intuito de tornar o *software* livre Sphinx-4 compatível com essa interface é observada.

Para avaliar a empregabilidade da *Java Speech API 1.0*, utilizou-se o *open source* FreeTTS 1.2 e a implementação *Cloud Garden TalkingJava SDK*. O *Cloud Garden* exemplifica o importante uso de facilidades Java na implementação da JSAPI em uma

[1]Esta implementação é baseada no produto comercial da IBM para reconhecimento e síntese de voz, o ViaVoice. A tecnologia do ViaVoice suporta o reconhecimento de discurso contínuo e permite o treinamento do reconhecedor, de maneira a ajustá-lo à voz do usuário e aumentar sua precisão.

camada de aplicação acima de *softwares* de voz já existentes (neste caso os que suportam SAPI). O FreeTTS, mesmo com suas limitações de suporte, apresentou resultados satisfatórios na implementação de estímulos emocionais na modificação da voz neutra. As pesquisas comprovaram a fácil integração entre as APIs Java e a atual inferioridade técnica da JSAPI quando comparada com a API da Microsoft.

A Sun está trabalhando na *Java Speech API* versão 2.0. Essa nova API dará atenção especial a compatibilidade com a JSAPI 1.0 e com a emergente *W3C Speech Interface Framework* [56]. Embora sejam raras as publicações disponíveis, a *Java Specification Request* da JSAPI 2.0 [57] listou suas principais novidades e tentativas de incrementar as funcionalidades presentes na versão anterior. São elas:

- Um SPI (*Service Provider Interface*) irá agir entre a camada JSAPI e o *engine* de voz de qualquer fabricante. Esse provedor de serviços, provavelmente, será baseado em padrões já existentes, como o *SAPI 5.0 Service Provider API*.

- As especificações JSGF e JSML seguirão o caminho dos grupos de *grammar format* e *synthesis markup languages* da *W3C Voice Browser*, respectivamente. Isso trará padronização e característica multilingual à essa API.

- Suporte ao redirecionamento de áudio. Gravação e transmissão de áudio poderão se tornar privadas em algumas aplicações.

- Uma arquitetura modular que suporte o mínimo de configuração e futuras implementações de recursos em vários tipos de plataformas (*javax.speech.desktops, javax.speech.server*).

- Maiores esclarecimentos sobre detalhes de funcionalidades já existentes, como sincronização via eventos, entre outras.

- Considerações sobre a compatibilidade com as API's telefônicas (JTAPI ou JAIN).

- Terá como alvo não apenas as plataformas *Java 2 Standard Edition* e *Java 2 Enterprise Edition*, mas também a plataforma *Java 2 Micro Edition*. Com isso, é grande a expectativa do uso dessa API em implementações para *handsets*.

Diante do exposto, espera-se que a *Java Speech API 2.0* apareça em breve com mais destaque no meio acadêmico e comercial que a sua antecessora. Para isso, porém, é crucial o alinhamento da JSGF e JSML com as especificações dos grupos W3C, afim de garantir que não ocorrerá fragmentação por padrões imcompatíveis e concorrentes.

Em suma, o presente trabalho buscou contribuir com a elaboração de um guia prático e comparativo das APIs SAPI e JSAPI para desenvolvedores de *software*. Trechos de código e implementações também foram apresentados, exemplificando os conceitos envolvidos e provando a viabilidade técnica da construção de robustas aplicações de voz. Além das interfaces de voz da Microsoft e da Sun, outros pacotes de desenvolvimento foram descritos e empregados: *CSLU toolkit*, *MSAgent*, *PircBot Java IRC API* e *Cloud Garden TalkingJava SDK*.

Futuramente, a academia poderá desenvolver o aprimoramento das atuais e a incorporação de novas funcionalidades ao *software* CALL, tais como permitir que um professor adicione “lições” de inglês sem que seja necessário editar o código fonte do programa; solicitar repetição, ou mesmo informar, quando o comando de voz não é reconhecido; e outras bem mais complexas, como o treinamento da pronúncia e o reconhecimento em português. Com relação ao FreeTTS, seria pertinente a realização de um estudo mais aprofundado no seu código fonte no intuito de agregar as regras prosódicas *low level* à voz sintetizada.

Referências Bibliográficas

[1] I. R. Murray and J. L. Arnott, "Toward the simulation of emotion in synthetic speech: A review of the literature of human vocal emotion," *Journal of the Acoustic Society of America, Vol. 2, 1097-1108*, 1993.

[2] J. Stallo, *Simulating emotional speech for a talking head*, Ph.D. thesis, Honours dissertation, Curtin University of Technology, Perth, Australia, 2000.

[3] Ben Shneiderman, "The limits of speech recognition," *COMMUNICATIONS OF THE ACM Vol. 43, No. 9, September*, 2000.

[4] Markku Turunen, "Speech application design and development," Tech. Rep., University of Tampere. Department of Computer Sciences, 2003.

[5] Thierry Dutoit, *An Introduction to Text-To-Speech Synthesis*, Kluwer, 2001.

[6] "http://www.chant.net/speechkit/default.asp," Visited in June, 2005.

[7] Francesc Alías and Ignasi Iriondo, "La evolución de la síntesis del habla en ingeniería la salle," Tech. Rep., Universitat Ramon Llull. Departamento de Comunicaciones y Teoria de la Señal Ingeniería y Arquitectura La Salle, 2002.

[8] P. S. Lucena, "Expressive talking heads: Um estudo de fala e expressão facial em personagens virtuais," M.S. thesis, PUC-RIO, 2002.

[9] Paula Lucena Rodrigues, Bruno Feijó, and Luiz Velho, "Expressive talking heads: uma ferramenta de animação com fala e expressão facial sincronizadas

para o desenvolvimento de aplicações interativas," in *Proceedings of Webmídia. SBC*, 2004.

[10] "http://edvista.com/claire/call.html," Visited in June, 2005.

[11] "http://www.belllabs.com/project/tts/," Visited in June, 2005.

[12] M. C. Amundsen, *MAPI, SAPI, and TAPI Developer's Guide*, Sams Publishing, 1996.

[13] B. M. Márquez Avendaño, *Implementación de un reconocedor de voz gratuito a el sistema de ayuda a invidentes Dos-Vox en español*, Monografia de Conclusão de Curso, Universidade das Américas-Puebla, 2004.

[14] "http://freetts.sourceforge.net," Visited in May, 2005.

[15] Nicole Yankelovich, Gina-Anne Levow, and Matt Marx, "Designing speechacts: Issues in speech user interfaces," *CHI 95 Conference on Human Factors in Computing Systems, Denver, CO, May 7-11*, 1995.

[16] "http://www.research.att.com/projects/tts/demo.html," Visited in March, 2005.

[17] Inc. Nuance Communications, *Enabling and Building Effective Conversational Applications*, White Paper, Nuance Solutions, 2005.

[18] Farzad Ehsani and Eva Knodt, "Speech technology in computer-aided language learning: Strengths and limitations of a new call paradigm," *Language Learning and Technology Vol. 2, No. 1, July, pp. 45-60*, 1998.

[19] I. R. Murray and J. L. Arnott, "Implementation and testing of a system for producing emotion-by-rule in synthetic speech," *Speech Communication, 16, 369-390*, 1995.

[20] M. Schröder, "Emotional speech synthesis: A review," *In The Proceedings of Eurospeech - Scandinavia, volume 1, pages 561 to 564, Denmark*, 2001.

[21] “http://java.sun.com/products/java-media/speech/,” Visited in March, 2005.

[22] “http://www.microsoft.com/speech/,” Visited in March, 2005.

[23] “http://www.cloudgarden.com/jsapi/,” Visited in March, 2006.

[24] “http://cmusphinx.sourceforge.net/sphinx4/,” Visited in March, 2005.

[25] “http://java.sun.com/j2se/1.4/,” Visited in March, 2005.

[26] J. F. Rodrigues Júnior and D. A. Moreira, “Estudo e desenvolvimento de aplicações java com reconhecimento e síntese de voz,” Tech. Rep., Universidade de São Carlos. Instituto de Ciências Matemáticas e de Computação, 2001.

[27] “http://www.microsoft.com/msagent,” Visited in March, 2005.

[28] “http://msdn.microsoft.com/library/default.asp,” Visited in March, 2005.

[29] “http://agent.microsoft.com/agent2/chars,” Visited in March, 2005.

[30] “http://www.nuance.com,” Visited in March, 2005.

[31] “http://www.digalo.com/agent,” Visited in March, 2005.

[32] “http://www.elantts.com/agent,” Visited in March, 2005.

[33] “http://www.eloq.com/etimsagentall2.html,” Visited in March, 2005.

[34] E. F. Damasceno, “Síntese e reconhecimento de voz em ambientes desenvolvidos em java,” *Semana Acadêmica do FESURV - Rio Verde - GO, Outubro*, 2004.

[35] C. Chapelle, “Call in the year 2000: Still in search of research paradigms?,” *Language Learning and Technology, 1(1), 19-43*, 1997.

[36] M. Warschauer, “Computer-assisted language learning: An introduction,” *In S. Fotos Ed., Multimedia language teaching, pp. 3-20. Tokyo: Logos International*, 1996.

[37] H. Fraser, "Phonetics, phonology, and the teaching of pronunciation - a new cd-rom for all esl learners and its rationale," *Eighth Australian International Conference on Speech Sciance and Technology, pp.180-185*, 2000.

[38] K. S. Ananthakrishnan, "Computer aided pronunciation system (caps)," M.S. thesis, University of South Australia, 2003.

[39] Helmer Strik, "Speech is like a box of chocolates...," *15th ICPhS Barcelona*, 2003.

[40] Silke Maren Witt, *Use of Speech Recognition in Computer-assisted Language Learning*, Ph.D. thesis, University of Cambridge, 1999.

[41] CCAA, *CALL Computer-assisted Language Learning*, CCLS PUBLISHING HOUSE C1626AK11 ISBN 85-340-0602-4, 1998.

[42] "http://cslu.cse.ogi.edu/toolkit/," Visited in March, 2005.

[43] K. A. Lenzo and A. W. Black, "Diphone collection and synthesis," in *ICSLP*, 2000.

[44] S. Isard and D. Miller, "Diphone synthesis techniques," in *Proceedings of the IEE International Conference on Speech Input/Output*, 1986, pp. 77–82.

[45] "http://java.sys-con.com/read/38107_p.htm," Visited in November, 2005.

[46] "http://www.jibble.org/pircbot.php," Visited in November, 2005.

[47] "http://www.onjava.com/lpt/a/5143," Visited in November, 2005.

[48] "http://research.sun.com/research/speech," Visited in May, 2005.

[49] A. W. Black and K. A. Lenzo, "Flite: a small fast run-time synthesis engine," *In Proceedings of the 4th ISCA Workshop on Speech Synthesis, page 204, Scotland*, 2001.

[50] “http://www.cstr.ed.ac.uk/projects/festival/manual/festival_toc.html,” Visited in May, 2005.

[51] “http://tcts.fpms.ac.be/synthesis,” Visited in May, 2005.

[52] “http://www.disc2.dk/tools/sgsurvey.html,” Visited in May, 2005.

[53] H. H. Dam and S. de Souza, “Applying talking head technology to a web based weather service,” *In Proceedings of the HF2002 Workshop on Virtual Conversational Characters: Applications, Methods and Research Challenges, Melbourne, Australia*, 2002.

[54] C. Serge de Souza, *First Implementation of VHML on the Java Text-to-Speech Synthesiser*, Monografia de Conclusão de Curso, Curtin University of Technology, 2002.

[55] F. Kilickaya, “Text-to-speech technology: What does it offer to foreign language learners?,” *CALL-EJ Online, ISSN 1442-438X, Vol. 7, No. 2*, 2006.

[56] “http://www.w3c.org,” Visited in March, 2006.

[57] “http://www.jcp.org/en/jsr/detail?id=113,” Visited in March, 2006.

Apêndice A - FreeTTS

A Sun Microsystems, Inc. não disponibiliza qualquer implementação da *Java Speech API*. Por outro lado, existem parcerias entre essa empresa americana e outras companhias da área de síntese e reconhecimento de voz para a produção de implementações em JSAPI. Como cada implementação utiliza a mesma API, não importa qual delas será empregada, já que a interface será sempre a mesma.

Neste trabalho foi utilizado o FreeTTS 1.2, um sintetizador de voz de código livre, escrito inteiramente em Java e que fornece suporte parcial à *Java Speech API v1.0*. A implementação FreeTTS suporta JSAPI com algumas restrições:

- Da feita que o FreeTSS é um sintetizador de voz, nenhuma interface de reconhecimento da JSAPI 1.0 é por ele implementada. Ou seja, apenas a especificação *javax.speech.synthesis* é parcialmente suportada.
- A *Java Speech Markup Language* é ignorada. O FreeTTS chega a processar a JSML, contudo nenhuma das altareções impostas no código é repassada para a voz sintetizada.
- O FreeTTS não gera os eventos WORD_STARTED e MARKER_REACHED.
- O *Vocabulary Management* não é suportado.
- O método *Synthesizer.phoneme()* não é implementado.
- As exceções *PropertyVeto* nem sempre são disponibilizadas corretamente quando os pedidos de mudança de propriedade são rejeitados.

Instalação

1. Antes de tudo é necessário que o *kit* de desenvolvimento *JavaTM 2 SDK Standard Edition v.1.4*, disponível em http://java.sun.com/j2se/1.4/, esteja implementado na máquina.

2. Fazer o download do arquivo compactado com o pacote de distribuição do FreeTTS 1.2 (http://freetts.sourceforge.net/). É importante salientar que o arquivo a ser encontrado é o *freetts-1.2.1-bin*, pois somente esse pacote contém os arquivos .jar, responsáveis pela comunicação entre a aplicação e o *engine* de voz.

3. Descompactar o arquivo em qualquer diretório da máquina. Por exemplo, caso o local escolhido seja a própria raiz do Sistema Operacional, os diretórios *C:\freetts-1.2.1* e *C:\META-INF* serão criados, juntamente com os seus sub-diretórios.

4. Ir na pasta *C:\freetts-1.2.1\lib* e executar o arquivo *jsapi.exe*. Uma vez aceito o contrato de licença, uma cópia do arquivo *jsapi.jar* será extraída na mesma pasta.

5. O arquivo de configuração *C:\freetts-1.2.1\speech.properties* deve ser copiado na pasta *JAVA_HOME\jre\lib*. Digamos que o *kit* Java 2 SDK também esteja instalado no diretório raiz, logo o destino do arquivo em questão seria *C:\j2sdk1.4.1_05\jre\lib*.

Execução

Para testar se todos os passos da instalação foram realizados com sucesso o pacote de distribuição oferece um programa que nos permite avaliar muitas das funcionalidades presentes no FreeTTS. Esse programa pode ser executado a partir da seguinte linha de comando no *Prompt* do diretório *C:\j2sdk1.4.1_05\bin*:

```
java -jar C:\freetts-1.2.1\lib\freetts.jar
Enter text: Hello.
<o texto "Hello" é sintetizado>
```

O programa *freetts.jar* quando invocado sem argumentos, assim como foi feito acima, lerá o texto escrito na linha de comando e convertê-lo em voz. Existem variadas opções que podem ser utilizadas na operação do *freetts.jar*, como a síntese do

texto a partir de um arquivo de entrada, seleção da voz, entre outras. Todas as funcionalidades existentes podem ser encontradas na documentação desse sintetizador (http://freetts.sourceforge.net/docs/index.php).

Outra maneira de fazer seu computador falar através do FreeTTS é construindo a sua própria implementação baseada na especificação JSAPI. O código abaixo é um exemplo de uma aplicação que sintetiza a palavra “Hello”.

```
import java.util.Locale;
import javax.speech.*;
import javax.speech.synthesis.*;
public class Teste {
 private static Synthesizer sintetizador;
 public static void main(String[] args) {
  try {
    SynthesizerModeDesc desejado =
      new SynthesizerModeDesc
      (null,‘‘general’’,Locale.US,null,null);
    sintetizador = Central.createSynthesizer(desejado);
    sintetizador.allocate();
    sintetizador.resume();
    sintetizador.speakPlainText("Hello.", null);
    sintetizador.waitEngineState(Synthesizer.QUEUE_EMPTY);
    sintetizador.deallocate();
  } catch (Exception e) {
      e.printStackTrace();
  }
 }
}
```

Para tal basta copiar e colar o algoritmo acima em um arquivo chamado *Teste.java* e salvá-lo na pasta *C:\j2sdk1.4.1_05\bin*, caso não se queira perder tempo configurando as diretivas PATH e CLASSPATH com as variáveis ambientes Java. Em seguida, abrir uma janela de *Prompt* e digitar os seguintes comandos no diretório *C:\j2sdk1.4.1_05\bin*:

Para compilar o código:

```
javac -classpath .\;C:\freetts-1.2.1\lib\jsapi.jar
Teste.java
```

Para executar a aplicação:

```
java -classpath .\;C:\freetts-1.2.1\lib\freetts.jar
;C:\freetts-1.2.1\lib\jsapi.jar Teste
```

Erros Frequentes

- **Exception in thread "main" java.lang.UnsupportedClassVersionError**: Provavelmente a versão do *java runtime* instalada na máquina é inferior a *jdk 1.4*. Pode-se verificar a versão atual digitando o comando:

  ```
  java -version
  ```

  Algo como *java version "1.4.0"* ou superior deve ser observado. Caso negativo, proceder conforme descrito no Passo 1 da instrução de instalação.

- **Exception in thread "main" java.lang.NoClassDefFoundError**: Esse erro ocorre quando a especificação JSAPI 1.0 encontra-se indisponível. Para solucioná-lo, basta executar o Passo 4 da instrução de instalação.

- **Exception in thread "main" java.lang.NullPointerException**: Erro relacionado com o arquivo *speech.properties*. A presença desse arquivo na pasta correta é fundamental para que a classe Central consiga localizar e criar o sintetizador. A classe *javax.speech.Central* controla a interação entre a API e a entrada e saída de áudio pré-processada pelo *engine*. Ela 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. Para eliminar a referida falha, é necessário revisar o Passo 5 da instrução de instalação.

Apêndice B - Desenvolvendo em SAPI

Este documento descreve as plataformas, linguagens de programação e *softwares* empregados e conhecimentos adquiridos durante o desenvolvimento do aplicativos CALL e CHAT.

- Plataforma: Microsoft Windows XP Professional Versão 2002.
- Linguagem de programação: C Sharp ou C#.
- Visual Studio .NET 2003: ambiente de desenvolvimento.
- Microsoft Speech SDK 5.1: Este *kit* contém todas as ferramentas, informações e exemplos necessários para incorporar tecnologia de voz às aplicações na plataforma Windows. As versões SAPI 5.0 e SAPI 5.1 podem coexistir na mesma máquina, contudo a desinstalação de uma delas pode danificar o funcionamento da outra. Por isso, é recomendado pela Microsoft, que a versão SAPI 5.0 seja retirada da máquina para a instalação da SAPI 5.1. O arquivo executável *Speech SDK 5.1* é instalado pelo Windows Installer e está gratuitamente disponível em http://www.microsoft.com/speech/download/sdk51.
- *Softwares* de voz SAPI 5: Foram utilizados os *engines* presentes no Sistema Operacional Windows XP, são eles: o *Microsoft English Recognizer v5.1* para ASR e os *Microsoft Mary*, *Sam* e *Mike* para TTS.
- Microsoft Agent Core: Para os usuários do Windows XP, não é necessária a sua instalação, pois este componente e o personagem Merlin podem ser encontrados, normalmente, no diretório *C:\Windows\msagent*. O Microsoft Agent Core é disponibilizado como uma ActiveX DLL. Logo, para ser utilizado como uma aplicação .NET, é necessário executar o arquivo *AxImp.exe*. Este executável é oferecido no *kit* .NET Framework (instalado juntamente com o Visual Studio .NET 2003):

```
C:\WINDOWS\msagent> AxImp agentctl.dll
```

Após a execução da linha de comando acima no ambiente DOS, duas novas DLLs serão geradas, são elas: *AxAgentObjects.dll* e *AgentObjects.dll*.

A arquitetura do Microsoft Agent Core usa SAPI 4.0 para suportar reconhecimento e síntese de voz. É necessária a instalação do *SAPI 4.0 runtime support*, caso o sistema operacional utilizado seja o Windows XP. Foram instalados também os *engines* SAPI 4 L&H TruVoice para síntese na língua inglesa e o L&H TTS3000 para a língua portuguesa, juntamente com o seu respectivo componente de linguagem. Todas estas ferramentas podem ser livremente adquiridas na página http://www.microsoft.com/msagent/downloads/user.asp.

As versões SAPI 4 e SAPI 5 podem funcionar no mesmo computador com os seus correspondentes *engines* de voz e aplicações simultaneamente, sem problemas de compatibilidade.

- A Figura 5.1 mostra o *Solution Explorer* do presente projeto .NET, contendo todas as DDLs necessárias para o desenvolvimento do aplicativo proposto.

Figura 5.1: Solution Explorer

- Para um melhor desempenho da aplicação, é necessário que um perfil de reconhecimento de voz seja criado e treinado com as informações sobre como reco-

nhecer a voz de cada uma das pessoas que usarão o aplicativo. Em propriedades de fala do painel de controle do Windows (Figura 5.2) podemos encontrar as instruções de como realizar esse treinamento e ajustar as configurações de reconhecimento e síntese de voz do *engine* disponibilizado pelo sistema operacional.

Figura 5.2: Tela das propriedades de fala