UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

ESCOLA DE ENGENHARIA

DEPARTAMENTO DE ELETRÔNICA E DE COMPUTAÇÃO

"UM SISTEMA DE CONVERSÃO TEXTO-FALA PARA WINDOWS"

Autor _____

Vagner Luis Latsch

Orientador _____

Prof. Sérgio Lima Netto

Examinador _____

Prof. Marcio Nogueira de Souza

Examinador _____

Prof. Fernado Gil Vianna Resende Junior

DEL

AGOSTO DE 2002

Agradecimentos

À minha família, pelo apoio e incentivo para o término deste projeto e à Adriana, pela tolerância e compreensão nos períodos de ausência quando dediquei-me ao desenvolvimento deste projeto.

Aos professores Sérgio Lima Netto, pela orientação e paciência em relação aos contratempos do término deste projeto, Fernando Gil Vianna Resende Junior, pela orientação nos primeiros passos no processamento da fala e a ambos pelas oportunidades oferecidas.

Do outro lado do oceano, em terras além mar, meus agradecimentos, em memória, ao prof. Carlos Espain de Oliveira, uma grande perda para a ciência e para os que conviveram com ele. Ao prof. Diamantino Freitas, professor, orientador e amigo, que agradeço pelo incentivo, entusiasmo e apoio ao longo do período de estada em Portugal. Agradeço ainda aos colegas João Paulo Teixeira, pela contribuição em idéias e ao refinamento do banco de dados; Paulo Gouveia, realizador dos algoritmos de conversão grafema-fonema; Daniela Braga, pelo apoio lingüístico e pelos momentos literários; Helder Ferreira pelos momentos de colaboração em software; Paulo Meireles e Antônio Moura, pelos momentos de descontração e Maria João Barros, pela companhia, incentivo e idéias.

Vagner Luis Latsch

Resumo

Um conversor Texto-Fala é um sistema que a partir de um texto, produz fala sintetizada correspondente à leitura. A fala sintetizada já vêm sendo utilizada como interface em várias aplicações, como no auxilio de deficientes visuais em acessar sistemas de informação ou em centrais de atendimento por telefone. No entanto, para utilização e aceitação desses sistemas, é necessário inteligibilidade e naturalidade da voz, além de fatores como velocidade de resposta e controle.

A Microsoft oferece um conjunto de componentes para aplicações que desejam utilizar sistemas de voz, inclusive conversão texto-fala, no sistema operacional Windows. Este projeto tem o objetivo de implementar um sistema de conversão conforme esta arquitetura, para que possa ser utilizado por componentes e aplicações compatíveis; possa servir como modelo de aprendizado e experimentação e ofereça uma plataforma base para desenvolvimentos futuros. Para isso, será necessário o estudo da programação por componentes e o estudo detalhado da arquitetura, que serão descritos ao longo deste projeto.

O sistema de conversão a ser inserido no componente, possui uma estrutura que busca soluções para atender os requisitos de desempenho como por exemplo velocidade de resposta. Para permitir marcas de controle no texto e, futuramente, a utilização de gramáticas, o texto é convertido para uma estrutura de dados que permanece até a etapa de síntese. O modelo de síntese utilizado é um sintetizador por formantes com concatenação de difones, no qual resulta em inteligibilidade e naturalidade precárias, no entanto o componente implementado serve de base para a inserção de novos modelos de síntese.

palavras chaves: síntese de voz; conversor Texto-Fala; TTS; COM; Microsoft Speech API 4.0.

Índice

Capítulo 1 - Introdução

Um conversor texto-fala ou TTS (Text-To-Speech) é um sistema que a partir de um texto irrestrito, produz fala sintetizada correspondente à leitura. Um texto irrestrito é composto pelos caracteres de uma língua escrita que são agrupados e organizados conforme regras ortográficas, morfológicas e sintáticas. A fala é produzida pelo homem variando as características de seu aparelho fonador e, quando produzida conforme as regras fonéticas e fonológicas de uma língua falada, possui associação com a língua escrita. A leitura de um texto é portando a associação da língua falada com a língua escrita. A síntese da fala consiste em reproduzir os sons da fala humana, seja por concatenação e manipulação de segmentos pré-gravados ou por modelos e regras. Portanto, um sistema TTS possui duas características principais: relacionar a língua escrita e a língua falada e sintetizar os sons da língua falada.

Um sistema TTS para ter boa aceitabilidade[1] precisa ser capaz de pronunciar corretamente um texto e produzir fala inteligível e natural. No entanto, outros fatores como velocidade de resposta, controlabilidade, limite em memória e capacidade de ser configurado e modificado, também alteram a aceitabilidade do sistema. Tais fatores são determinados diretamente pela estrutura de dados e estrutura de software, no qual requerem significativa atenção.

Os sistemas operacionais têm procurado incluir sistemas de voz, síntese e reconhecimento, como meio de interface com o usuário. A exemplo disto, a Microsoft vem desenvolvendo um conjunto de APIs (*Aplication Program Interfaces*), chamadas Speech APIs, que definem a arquitetura destes sistemas, no formato de componentes, para serem utilizados no sistema operacional Windows. Estas APIs permitem que o usuário tenha em seu computador sistemas de diferentes fornecedores e diferentes línguas, e se um sistema TTS é implementado conforme esta arquitetura, poderá ser utilizado por qualquer aplicação compatível. A aplicação faz a interface entre o usuário e o sistema TTS e limita-se a conhecer os métodos expostos pelos componentes, conforme definidos pela Speech API, e programá-los da maneira que deseja interagir com o usuário, podendo estar contida numa página HTML, ser um leitor de emails ou uma aplicação simples com entrada de texto. No entanto este projeto não tem o

[1] aceitabilidade representa o nível de aceitação de um sistema pelo usuário, ou seja, o quando um sistema é satisfatório para o usuário [1].

objetivo de implementar uma aplicação, mas o sistema TTS que localiza-se num nível mais baixo como um componente do sistema operacional.

Basicamente, a implementação do sistema TTS em uma arquitetura compatível, requer o estudo da arquitetura, a implementação de uma arquitetura base e a inserção do sistema de conversão. A definição de responsabilidades da arquitetura para tornar o sistema um componente compatível, influirá tanto na montagem da estrutura de software como na estrutura de dados do sistema. A vantagem de um sistema compatível e baseado em componentes é que a aplicação e o sistema podem ser desenvolvidos separadamente.

1.1. Motivação

A comunicação máquina-homem ou homem-homem através da voz já vem sendo utilizada em várias áreas, mas não amplamente. A aplicação do TTS na comunicação entre a máquina e o homem, pode auxiliar deficientes visuais no acesso a sistemas públicos de informação, como a verificação de horário de transportes coletivos, serviços de localização, e ainda à tecnologia em geral, como a utilização do computador pessoal e acesso a Internet. Além destas aplicações, o sistema TTS é utilizável em sistemas de acesso por telefone, por exemplo centrais de vendas por telefone e comércio eletrônico, tendo a função de descrever menus, mensagens ao usuário ou até ler mensagens de correspondência eletrônica. Na comunicação homem-homem, deficientes orais e/ou auditivos conseguiriam se comunicar através do som através de sistemas TTS portáteis, e ainda, utilizando tradutores automáticos, seria possível se comunicar em outras línguas.

O desenvolvimento de um sistema TTS implica na participação de várias áreas de conhecimento, como a lingüística, inteligência artificial, processamento de sinais, análise de sistemas, etc. o que classifica-o como um assunto multidisciplinar.

Apesar de existir várias aplicações possíveis e já utilizáveis para o sistema TTS, ainda é uma tecnologia em desenvolvimento, principalmente para o português, tanto europeu quanto brasileiro, que ainda requer avanços para ser aceita e absorvida pelo mercado.

1.2. Objetivos do projeto

O projeto tem por objetivo implementar um sistema TTS conforme a arquitetura de sistemas TTS comerciais atuais, dentre os quais foi escolhido a Microsoft Speech API 4.0 como arquitetura base. Esta escolha se deve inicialmente por permitir que o sistema TTS a ser desenvolvido possa ser utilizado por componentes e aplicações já desenvolvidas compatíveis com esta API. Por outro lado, a escolha de por uma arquitetura pré-definida visa utilizá-la como modelo para aprendizado e experimentação, considerando ser uma arquitetura "madura", já experimentada e utilizada pela Microsoft. Pretende-se portanto criar uma plataforma base para desenvolvimentos e por isso os algoritmos de conversão inseridos serão de caráter experimental.

Em suma, o projeto consiste em implementar um componente, designado de motor Texto-Fala, que implementa as funções do sistema TTS e que poderá ser utilizado por qualquer aplicação compatível com a versão 4.0 da Speech API. Para demonstração, será utilizado o MS Agent 5.0 da Microsoft.

O sistema de conversão do texto em fala, a ser inserido na arquitetura, só considerará válido os caracteres pertencentes ao alfabeto português, pontuação e tags, conforme a Speech API. Portanto não haverá expansão de abreviaturas ou numerais. A conversão para fonemas será feita a partir dos algoritmos fornecido pela Faculdade de Engenharia da Universidade do Porto, para o português europeu, no âmbito deste projeto. Em relação a prosódia, o contorno para a frequência fundamental será gerado pelo modelo de Fujisaki;

Após definida a arquitetura de software, será implementada uma estrutura de dados que: permita a permanência das informações do texto até o estágio final de processamento, suporte marcas inseridas no texto e permita a inclusão posterior de gramáticas.

O sistema de síntese, incluído no sistema de conversão do texto em fala será um sintetizador por formantes que utiliza a concatenação de difones parametrizados. A escolha se deve a utilização de um banco de dados proveniente de um sistema de desenvolvimento e síntese chamado MULTIVOX, fornecido pela Faculdade de Engenharia da Universidade do Porto no âmbito deste projeto de fim de curso.

Requisitos do sistema:

- Computador PC com processador acima de 100Mhz;
- mínimo de 32Mb de memória;
- mínimo de 2Mb de espaço em disco;
- Sistema operacional Windows 98, NT, 2000 ou XT;
- Microsoft Speech API 4.0;

1.3. Organização dos capítulos

No capítulo 2, a seguir, serão discutidas as etapas de um sistema conversor texto-fala, os fatores que influenciam na estrutura do software e por fim uma breve descrição dos sistemas TTS atuais; No capítulo 3 será feita a compilação dos conceitos da programação por componentes COM, no qual se baseia a Microsoft Speech API 4.0, seguida da descrição da API específica para Text-To-Speech. No capítulo 4 será descrita a implementação da arquitetura do sistema TTS, conforme necessário para a compatibilidade. Em seguida, no capítulo 5 será descrita a implementação do sistema TTS inserido na arquitetura, dividida em dois segmentos: a estrutura do sistema e a estrutura de dados. A síntese da fala será descrita no capítulo 6, onde será descrito o modelo do sintetizador por formantes, incluindo: o formato do banco de dados; o modelo de excitação; o modelo de Fujisaki para determinação do F0; a implementação do filtro de trato vocal e o método utilizado para variação de pitch e duração.

Capítulo 2 – Sistemas de conversão texto-fala

2.1. Introdução

Um conversor texto-fala é um sistema que a partir de um texto irrestrito, produz fala sintetizada correspondente à leitura. As etapas que compõem o sistema podem ser divididas em duas: o processamento e conversão do texto em unidades fonéticas e a síntese das unidades fonéticas. A primeira etapa se subdivide na normalização do texto, *parser* sintático, determinação da pronúncia e da prosódia. A etapa de síntese consiste em produzir a fala sintetizada, no qual existem diferentes técnicas a serem utilizadas, como a concatenação de unidades temporais ou modelos paramétricos.

Um sistema TTS para ter boa aceitabilidade precisa ser capaz de pronunciar corretamente um texto e produzir fala inteligível e natural. No entanto, outros fatores como: velocidade de resposta, controlabilidade, limite em memória e capacidade de ser configurado e modificado, também contribuem para a aceitabilidade de um sistema. Tais fatores são determinados diretamente pela estrutura de dados e estrutura de software, no qual requer significativa atenção.

Neste capítulo serão discutidas as etapas de um sistema conversor texto-fala, apresentando resumidamente alguns problemas e sugerindo algumas soluções. Os métodos de síntese da fala, que vêm sendo estudados a 6 décadas, serão brevemente apresentados. Em seguida, os fatores que influenciam diretamente na estrutura do software, também serão discutidos, e por fim será feita uma breve descrição dos sistemas TTS atuais.

2.2. Etapas do sistema

Os passos básicos na conversão do texto em fala, conforme mostra a figura 2.1, incluem: a normalização do texto, onde os caracteres não pertencentes ao alfabeto são traduzidos para a forma pronunciável (R$1,20 -> um real e vinte centavos), ou abreviaturas (Sr. -> Senhor),

etc.; o parser sintático, onde é obtida a estrutura sintática de sentenças e a classificação de palavras (nome, verbo, adjetivo, etc.); pronúncia, onde caracteres ortográficos (grafemas) são mapeados para fonemas apropriados e marcados com a posição da tônica, conjugando regras grafema-fonema e dicionários; determinação prosódica, que associa características de ritmo e

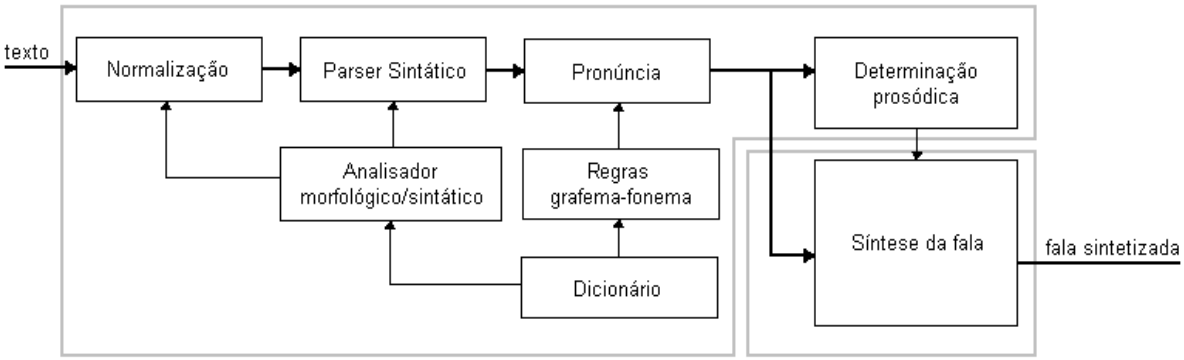


Figura 2.1.: Diagrama em blocos de um TTS.

pitch ao texto para maior inteligibilidade e naturalidade; e por fim a síntese, etapa onde a fala é produzida [1]. Os dicionários, além de dados de pronúncia, podem conjugar informações morfológicas, sintáticas e ainda formas não normalizadas, como abreviaturas e números. Desta forma, a consulta a um dicionário pode ser feita em várias etapas.

2.2.1. Normalização do texto

No texto existem inseridos: abreviaturas, acrônimos e caracteres não pertencentes ao alfabeto, como pontuação, dígitos, barras, etc.. Pontos podem indicar uma abreviação, um número decimal ou fim de sentença e precisa ser diferenciado. Os dígitos podem ser convertidos de diferentes formas, dependendo do contexto, por exemplo 1/4 pode ser convertido em um quarto ou primeiro de abril. Tais situações são difíceis para o sistema resolver não tendo acesso a informação semântica. Existem vários caminhos para a solução deste problema, entre eles: utilizar modos de configuração onde o usuário determina regras sobre como ele gostaria que determinadas formas fossem convertidas, no entanto se o texto não for escrito pelo usuário, podem surgir vários erros de interpretação; outro método é utilizar textos marcados, onde os numerais, abreviaturas, etc., possuem etiquetas que indicam o modo como deverão ser convertidos. Esta solução permite a personalização do texto, no entanto requer grande esforço do usuário.

2.2.2. Parser sintático

A análise sintática é necessária para a normalização do texto, para a pronúncia correta da palavra e é um dos vários componentes na determinação da prosódia. Na normalização do texto a interpretação e expansão corretas de um número pode depender do contexto sintático, por exemplo: “O Flamengo desceu 1 (um) ponto no campeonato” ou “O Flamengo perdeu mais 1(uma) vez”. Na pronúncia, palavras como “gosto”, podem ser verbos ou substantivos: “Eu gosto de frutas com gosto ácido”, e portanto a análise sintática é necessária para a formação da pronúncia apropriada. Na determinação prosódica, palavras conteúdo como nomes, adjetivos e verbos geralmente recebem acento de intonação, enquanto palavras função não. Ainda na determinação prosódica, o parser sintático também se aplica na classificação de grupos que possuem características prosódicas semelhantes.

Existe a necessidade de processar o texto em unidades menores, onde a utilização da pontuação como delimitador pode não ser suficiente. Desta forma, estruturas sintáticas ou grupos prosódicos, podem ser utilizados como unidade quando considerados com processamento quase independente.

2.2.3. Pronúncia

A pronúncia correta de uma palavra (a seleção apropriada de fonemas e a marcas de tônica) é um fator crítico para inteligibilidade e naturalidade de um sistema TTS.

A conversão de caracteres do alfabeto (grafemas) para caracteres fonéticos (fonemas) por regras grafema-fonema, é insuficiente para todas as palavras encontradas em determinada língua. Portanto, muitas palavras são exceções às regras e sua forma correta de pronúncia precisa estar incluída em um dicionário de exceções.

Conversão grafema-fonema:

As regras de conversão grafema-fonema determinam a pronúncia de uma palavra estritamente da ortografia. Um conjunto de regras de conversão são determinadas assumindo que exista

correspondência entre um ou mais caracteres em uma palavra escrita e um ou mais fonemas na pronúncia. Esta correspondência pode variar dependendo dos caracteres vizinhos na palavra. Estas regras são parecidas com aquelas que freqüentemente as crianças aprendem para ler, no entanto regras mais complexas contém centenas de pequenas regras ordenadas. É possível a implementação por redes neurais, mas tem mostrado ser menos eficiente do que as regras tradicionais. Um problema para as regras de conversão são as palavras onde a pronúncia das vogais dependem da posição da tônica (leva, levou). Portanto, é preciso haver regras de determinação da sílaba tônica para a conversão correta destes casos. Outro problema para as regras de conversão são os nomes próprios derivados de outra línguas, que pode ser solucionado encontrando inicialmente qual a língua mais provável daquela palavra e em seguida aplicando regras de conversão apropriada àquela língua.

Dicionários:

As palavras que fogem as regras, podem estar contidas em um dicionário. O tamanho do dicionário depende da acurácia das regras, e existem várias maneiras de fazer um dicionário de exceções mais eficiente, com um número mínimo de dados. Uma maneira é a analogia de pronúncia, que foi proposta pela observação de que o homem usa palavras conhecidas para pronunciar palavras desconhecidas. A pronúncia da palavra original é escolhida entre uma ou mais palavras similares, das quais a pronúncia é conhecida. A maior dificuldade é arquitetar uma medida de similaridade. Uma solução simples é basear o grau de similaridade no número de letras em comum entre a palavra conhecida e a palavra original. Decompor a palavra em trechos maiores do que caracteres, tal como rema, tema, etc., pode auxiliar em definir similaridade para a proposta de pronúncia por analogia e reduzir mais ainda o tamanho do dicionário.

2.2.4. Determinação prosódica

A prosódia se refere à maneira como algo é dito, ou seja, como variações de intonação, duração e intensidade são utilizadas para modificar a pronúncia de uma palavra ou sentença. Uma prosódia apropriada é importante tanto para a naturalidade quanto para a inteligibilidade de uma leitura.

A prosódia carrega tanto informação lingüística, quanto extra-lingüística ou para-lingüística, que diz respeito à atitude, intenção ou estado emocional e físico. De um ponto de vista linguístico a prosódia envolve tanto a estrutura segmental quanto unidades estruturais mais largas do que segmentos fonéticos, chamados supra-segmentais.

Intonação:

Em sistemas TTS, a meta em determinar a intonação é encontrar um modelo apropriado para a variação da freqüência fundamental, ou contorno de F0, para cada segmento de texto a ser falado. Fisiologicamente o F0 corresponde a freqüência na qual as cordas vocais vibram.

Geralmente, um modelo de contorno de F0 está associado a uma sentença ou grupo. Tipicamente o F0 declina gradualmente ao longo de uma sentença, existindo picos e vales sobre o declínio. Um aumento no fim do contorno do F0 é gerado para questões sim/não (Você quer ir ao cinema?). Um decaimento final é gerado para afirmações e para questões sem resposta sim/não (Eu gosto de ir ao cinema.). Entre sentenças é gerado um decréscimo e acréscimo contínuo.

A partir do texto são extraídos parâmetros para o modelo que irá gerar uma forma para o contorno do F0. Geralmente um dos parâmetros é a função sintática das palavras usadas na sentença, acrescentando acentos ou vales ao contorno. Por exemplo, palavras função como artigos ou preposições tipicamente não recebem acentos, ao contrário de palavras conteúdo. Os tipos de modelos diferem tanto na flexibilidade e naturalidade, quanto na facilidade de implementação e na teoria fonológica implícita.

Duração segmental:

Cada unidade sonora, segmento ou fone, possui uma duração inerente, chamada de duração segmental. Esta duração pode ser modificada de acordo com o contexto sintático, fonético e/ou fonológico. Por exemplo, segmentos precedidos de vizinhança de frase são alongados, vogais tônicas são alongadas, etc. Sistemas TTS tipicamente utilizam regras para determinar a duração.

Intensidade:

Este é o fator de menor importância na componente da prosódia, e não é modelado por muitos sistemas.

2.2.5. Síntese da fala

A síntese de fala é feita, por alguns sistemas, através da concatenação de unidades como frases ou palavras pré-gravadas, armazenadas em banco de dados. No entanto estes sistemas são restritos ao vocabulário ou requerem enormes banco de dados em memória.

Para sistemas TTS, onde a entrada é um texto irrestrito, a voz é gerada por manipulação dos sons básicos da língua, fones. A maioria das línguas têm somente 30 a 50 unidades ou fones. No entanto, a síntese da fala baseada nesta unidade encontram dois fatores de complicação: a região de transição entre um fone e outro e a modificação que o fone sofre em relação ao contexto onde está inserido. Geralmente são preferíveis unidades que são recortadas sob o ponto de vista da análise do sinal, por exemplo o difone. Difones são obtidos pela divisão do sinal de fala em segmentos, recortados entre a região de estabilidade de um fone e o fone seguinte, preservando as características de coarticulação entre eles. O número de difones é igual a combinação dos fones da língua dois a dois, ou seja, 40 fones equivalem a 1600 difones. Desta forma, quando os difones são concatenados em seqüência correta, seja por unidades temporais (amostras de sinal) ou unidades parametrizadas, resultam em um sinal "quase" espectralmente contínuo [1].

Os requisitos de um sintetizador são: a capacidade de produzir sinais com as características espectrais do sinal de fala e ainda ser capaz de alterar a estrutura temporal e a freqüência fundamental sem produzir distorções apreciáveis. Ao longo de 6 décadas estes sistemas vêm sendo aprimorados, e alguns marcos importantes para este desenvolvimento são [2]:

> 1939, Dudley: Sistema chamado *Voder*, exibido na Exposição Mundial de 1939 em Nova Iorque e desenvolvido pela *Bell Telephone Laboratories*. O sinal de fala era gerado através de um teclado que comandava dez filtros passa-banda e um pedal para manipular a frequência fundamental.

1960, Fant: Formalizou a teoria acústica da fala. Um filtro linear simula as ressonâncias do tubo acústico formado pela faringe, cavidade bocal e lábios enquanto as fontes de excitação são a de vozeamento produzido pelas cordas vocais e da turbulência causada pela diferença de pressão devida a uma constricção do fluxo de ar.

1964, Rabiner: Na sua tese de doutoramento apresentou um sistema que foi o precursor do método de síntese por regras. O sistema usa como entrada uma seqüência de símbolos fonéticos com marcas de acento, fronteira de palavras e pausas.

1973, Holmes: Usando um sintetizador de formantes paralelo, sintetizou pela primeira vez uma frase que o ouvinte médio não consegue destinguir do original.

1979, Allen et al.: O sistema MITalk de síntese de fala a partir de texto desenvolvido no MIT desde a década de 60.

1980, Klatt: A publicação do programa fonte do sintetizador de formantes cascata/paralelo promoveu a sua utilização em testes perceptuais por diversos laboratórios.

1981, Klatt: *Klatt* desenvolveu no MIT um novo sistema de síntese de fala chamado *Klatttalk*. O sistema continha um dicionário de 6000 palavras e dispondo de um analizador sintático rudimentar e de regras de síntese segmental do próprio Klatt.

1990, Carpentier e Moulines: Apresentam a técnica PSOLA (*Pitch Syncronous Overlap-Add*) para a variação da duração e frequência fundamental na síntese por concatenação. Mostraram ser possível reconstruir o sinal de fala variando estes dois aspectos da prosódia, sem degradação considerável de qualidade. Os bons resultados obtidos justificam a popularidade atual dos sistemas de síntese por concatenação.

O método PSOLA, um eficiente representante dos modelos por concatenação de unidades temporais, reproduz a fala com mais fidelidade, reproduzindo características que ainda não são modeláveis; as caraterísticas espectrais do sinal original são mantidas, no entanto requer um grande esforço na montagem do banco de dados e recorde das unidades para que não

aconteçam descontinuidades na freqüência ou na fase. O maior problema dos sistemas concatenativos é o fato de serem pouco flexíveis quanto a variação de duração e freqüência fundamental.

Os sistemas mais adequados à regras, como o sintetizador por formantes, produzem uma voz "sintética" ou pouco natural; requerem maior esforço no refinamento do modelo e definição das regras, no entanto são sistema muito flexíveis, capazes de modificar características da voz, o que sistemas concatenativos não permitem. Sintetizadores de canto, geralmente utilizam síntese por regras ou híbridos, buscando flexibilidade.

2.3. Fatores estruturais

Quando um texto ou frase são enviadas ao sistema por ação do usuário, a velocidade de resposta que é percebida como "imediata" é de cerda de 200 ms. Tempos maiores do que este dão o entender de que a ação não foi registrada e assim o usuário executa novamente a ação sobrecarregando o sistema. Se excedem de 1 segundo o usuário tem a impressão de um sistema muito lento ou que não responderá nunca. Deste modo, minimizar o tempo de resposta, ou o tempo de síntese após o primeiro envio de texto, é uma responsabilidade para o sistema TTS [1].

A controlabilidade do sistema significa que o usuário pode interagir no processo, enfatizando determinada palavra, inserindo um pausa, modificando a velocidade de pronúncia ou designando a forma como uma data ou um número devem ser expandidos. No entanto, estas alterações não precisam necessariamente ser permitidas em tempo real. Alguns sintetizadores definem um conjunto de marcas padronizadas, que acrescentadas ao texto permitem o controle do sistema. Por outro lado, se o usuário deseja modificar parâmetros padrões, como velocidade de pronúncia ou frequência fundamental base, sem marcar o texto, ou ainda, se deseja que o sistema "sempre" se comporte de determinada maneira em relação a uma abreviatura, ou acrônimo, serão necessários métodos para configuração do sistema.

Os sintetizadores apresentam o conflito entre máxima qualidade e mínimo de *hardware* requerido (memória, velocidade, espaço em disco,...). O limite de espaço em memória praticamente determina o método de síntese a ser utilizado. No caso da utilização do TTS em

plataformas *multitasking*, onde o sistema TTS permanece em *background* de outra aplicação é conveniente que a utilização da memória seja limitada, assim como a porcentagem de utilização do processador.

Um requisito para qualquer sistema, é a capacidade de ser facilmente modificado e atualizado. Além disso, o sistema TTS ainda não é um sistema em seu estado final, é um sistema que ainda requer ser frequentemente atualizado, acrescentando uma nova regra de conversão, modificando o dialeto de sua língua, incluindo novo modelo para prosódia ou um novo método de síntese. Isto implica que o sistema TTS, assim como qualquer sistema em desenvolvimento e de complexidade considerável, seja flexível a futuras modificações e acréscimos.

2.4. Desenvolvimentos atuais

A maioria dos sistema texto-fala comerciais da atualidade têm obtido bons resultados com a concatenação de unidades temporais, utilizando banco de dados com unidades diferenciadas em tamanho como trifones e difones e diferenciadas em frequência fundamental. Os maiores esforços tem se dado na determinação da intonação prosódica através de características sintáticas do texto. Existem sintetizadores que utilizam dicionários léxicos, gramáticas e conjuntos de regras linguísticas para determinação de duração, frequência fundamental e intensidade. Esta abordagem tem utilizado produtos da Linguistica Computacional e da Inteligência Artificial.

Em consequência do desenvolvimento da tecnologia no processamento da fala, os sistemas operacionais têm procurado incluir sistemas de voz, síntese e reconhecimento, como interface. Desenvolvedores para plataformas Unix vêm aprimorando os sistemas Festival e MBrola, que oferecem suporte para a geração de novos banco de dados para diferentes vozes ou línguas. A Microsoft vem desenvolvendo um conjunto de APIs (*Aplication Program interfaces*), chamadas Speech APIs, que definem a arquitetura destes sistemas no formato de componentes para serem utilizados no sistema operacional Windows.

Tendo em vista a utilização e controle destes sistemas através da internet ou pelo telefone, a linguagem XML (*eXtended Markup Language*) vem sendo expandida e utilizadas por

sistemas TTS. Através de marcações específicas, é possível controlar o sistema para modificar a intonação prosódica, inserir pausas, inserir marcas de sincronismo, definir pronúncia de palavras, etc. permitindo acrescentar ao texto características de intencionalidade ou personalidade.

Os sistemas TTS para a língua Portuguesa vêm sendo desenvolvidos por iniciativas de universidade brasileiras e portuguesas ou, no caso de sistemas comerciais, por iniciativa privada, por exemplo: Aculab (Inglaterra) e Elan (França), as quais já fornecem seus produtos.

2.5. Conclusão

Neste capítulo, foram discutidas as etapas de um sistema conversor texto-fala: normalização do texto, parser sintático, pronúncia e determinação prosódica, ressaltando-se a presença de um analisador sintático na maioria das etapas de processamento do texto. Foram também descritos alguns fatores a serem considerados, relativos à aceitabilidade do sistema, como velocidade de resposta e controle. A necessidade de um arquitetura de fácil manutenção e desenvolvimento também foi justificada.

Em seguida foi feito uma breve apresentação do estado dos sistemas atuais, onde destaca-se a utilização de produtos da lingüística computacional no processamento do texto, a utilização da linguagem XML como marcação de texto e a iniciativa da Microsoft de inserir interface de voz ao Windows através da Speech API.

Capítulo 3 – Microsoft Speech API 4.0

3.1. Introdução

A Speech API 4.0 é um conjunto de componentes COM que permitem a uma aplicação utilizar a voz como interface no sistema Windows, tanto por reconhecimento quanto por síntese da voz. Os componentes são utilizáveis por diferentes linguagens, por exemplo C++, Java ou Visual Basic, e em diferentes níveis de simplicidade ou complexidade. Os componentes fundamentais, onde estão inseridos os sistemas de síntese e reconhecimento, são chamados de *engines* ou motores. Em relação à síntese, a arquitetura do sistema TTS é definida pela DirectTextToSpeech API, que trata-se da definição de um conjunto de componentes, interfaces e responsabilidades para o sistema de conversão. Na figura 3.1., tem-se um esquema simplificado dos componentes de um sistema TTS compatível.

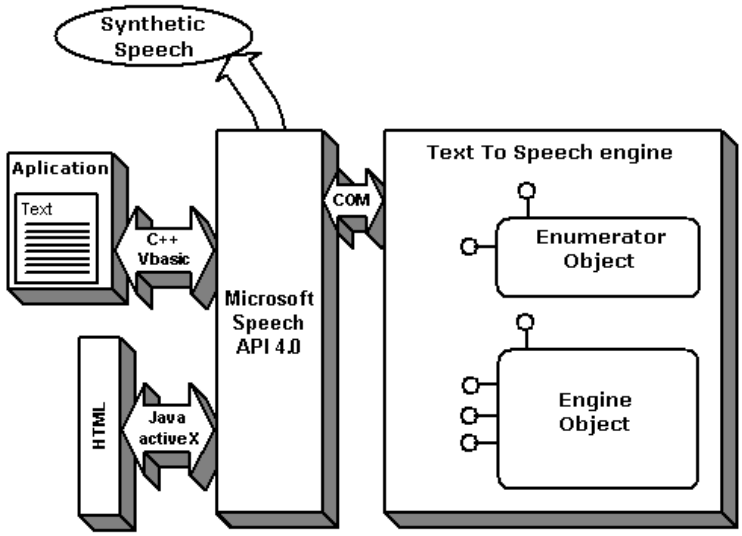


Figura 3.1.: Esquema simplificado de uma aplicação TTS compatível com a Speech API 4.0

Neste capítulo, será feito a compilação e resumo dos conceitos da programação por componentes COM, no qual se baseia a Speech API. Em seguida, serão descritos os principais componentes da arquitetura da DirectTextToSpeech API: um objeto de áudio, um objeto enumerator e um objeto engine e suas respectivas interfaces.

3.2. COM (Component Object Model)

COM é uma arquitetura de software por componentes, que abrange serviços de manipulação de documentos, controle, transferência de dados, e outras interações de software. Estes serviços oferecem, distintamente, diferentes funcionalidades para o usuário, no entanto os componentes compartilham requisitos fundamentais para se conectarem e se comunicarem um com outro, de maneira bem definida. COM estabelece estes conceitos definindo um padrão binário para interação entre componentes ou entre componente e aplicação; sendo independente da linguagem de programação; sendo multiplataforma (Windows, Windows NT, Macintosh, UNIX) e extensível. Desta forma, é possível que componentes sejam fornecidos por diferentes fornecedores /1/ . Além disso, COM permite mecanismos para

- comunicação de componentes entre processos ou através de rede,
- compartilha memória entre componentes,
- relata o estado do componentes e erros
- e permite leitura dinâmica de componentes.

"É importante notar que COM é uma arquitetura geral para componentes. Enquanto a Microsoft esta aplicando COM em áreas específicas tal como controle, composição de documentos, automação, etc., qualquer desenvolvedor pode tirar vantagem da estrutura e fundamentação que COM permite" /1/.

3.2.1. Conceitos fundamentais

COM define vários conceitos fundamentais, principalmente:

- um padrão binário para chamada de função entre componentes;
- agrupamento de funções em interfaces;
- uma interface básica que permite: um caminho para os componentes, dinamicamente, descobrirem as interfaces implementadas por outros componentes e um contador para permitir verificar o próprio tempo de vida e se auto liberar quando apropriado.
- um mecanismo para identificar unicamente cada componente e cada interface;
- um leitor de componentes para definir e ajudar o gerenciamento de interação entre componentes.

Padrão binário:

Para qualquer plataforma, COM define um caminho padrão para modelar tabelas de funções virtuais em memória e um caminho padrão para chamar funções por estas tabelas. Assim, qualquer linguagem de programação que permite utilizar ponteiros para funções virtuais (C, C++, Small Talk, Ada) pode ser usada para criar componentes que operam com outros componentes. O duplo acesso indireto (um ponteiro para um ponteiro para a tabela) permite partilhar a tabela de funções entre múltiplas instâncias da mesma classe do objeto, o que pode reduzir consideravelmente o espaço em memória necessário.

Componentes e interfaces:

Em COM, um objeto é uma parte de código compilado que oferece algum serviço para o resto do sistema. Para evitar confusão, é preferível referir um objeto COM como um objeto componente ou simplesmente um componente, evitando confusão com programação orientada a objetos tal como em C++.

Um componente usualmente possui dados (ou atributos) associados, mas diferente de objetos C++, um componente nunca terá acesso direto a outro componente. Ao invés disso, componentes sempre acessam outro componente através de ponteiros para interfaces. Esta é uma característica primária da arquitetura de COM, que permite preservar encapsulamento completo de dados e processos.

As aplicações interagem entre si e com o sistema, através de coleção de grupos de funções que compõem uma interface. A interface é a definição de comportamentos e responsabilidades. Um ponteiro para um componente é na realidade um ponteiro para uma das interfaces que o componente implementa. Na figura 3.1., tem-se a representação usual de um componente. As interfaces A e B são grupos de funções e a interface *IUnknown* é a interface padrão para requisitar a conexão para A ou B. É comum utilizar a letra "I" para indicar uma interface.

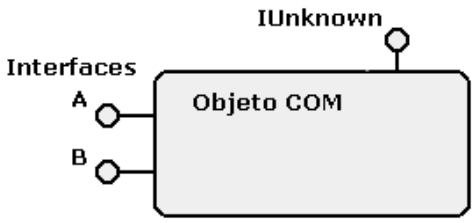


Figura 3.1.: representação de um componente COM.

Interface básica - IUnknown:

COM define uma interface especial, *IUnknown*, para implementar algumas funcionalidades essenciais. Todo componente requer esta interface implementada e convenientemente todo objeto COM ou interface deriva de *IUnknown*. Esta interface tem três métodos: *AddRef*, *Release* e *QueryInterface*.

AddRef e *Release* são simplesmente métodos de contagem de referência. O método *AddRef* de um componente é chamado quando outro componente deseja usar uma interface. O método *Release* é chamado quando o outro componente não precisa mais daquela interface. Enquanto a referência do componente é diferente de zero, ele permanece em memória; quando chega a zero, o componente pode seguramente se liberar da memória, pois nenhuma interface está sendo usada.

QueryInterface é o mecanismo que permite descobrir, em tempo de execução, se uma interface específica é implementada por um componente. Quando uma aplicação quer usar alguma função de um componente, ela chama o método *QueryInterface* do componente, requisitando um ponteiro para a interface que implementa a função.

Obs.: O método *QueryInterface* internamente incrementa o contador de referência por *AddRef* antes de retornar o ponteiro.

Identificador único – GUID:

COM usa GUID (*Globally Unique Identifiers*), um inteiro de 128 bits, que garante identificar toda interface e todo componente como únicos. Este identificador é chamado *CLSID* quando se refere a classe de um componente e *IDD* quando se refere a uma interface. Desenvolvedores criam seus próprios *GUIDs* para os componentes ou interfaces próprias.

Component Object Library:

É uma biblioteca do sistema operacional que permite o mecanismo de COM. Um componente é implementado em um módulo, ou seja, no formato de uma DLL ou um EXE, portanto são necessários métodos específicos para criar e acessar o componente. Uma aplicação,

inicialmente, precisa iniciar a biblioteca COM e em seguida requisitar a criação do componente, indicando o *CLSID* da classe do componente. A biblioteca procura o código no banco de dados de registros que informa onde o objeto se encontra implementado. Se o código está implementado em uma *DLL*, é chamada a função *DLLGetClassFactory*, exportada pela *DLL*, na qual a biblioteca instancia a *ClassFactory* do componente. Todo componente COM precisa implementar esta classe e a interface padrão *IClassFactory*, da qual a biblioteca utilizará os métodos para instanciar o componente e gerenciar memória e definir a comunicação entre processos ou através de rede.

Conexão entre componentes:

COM suporta dois mecanismos através do qual um componente pode utilizar outro: *containment* e *aggregation*. Na figura 3.2. tem-se a ilustração dos dois métodos. Por conveniência, o objeto sendo utilizado é chamado de objeto *inner* e o objeto que faz uso é chamado objeto *outer*.

- ***Containment*:** o objeto *outer* comporta-se parecido como um cliente do objeto *inner*. O objeto *outer* "contém" o objeto *inner* e quando o objeto outer deseja usar um serviço do *inner*, simplesmente requisita interface do objeto *inner*. Em outras palavras, o *outer* usa os serviços do *inner* para implementar algumas de suas funcionalidades.

- ***Aggregation*:** o objeto *outer* expõe a interface do *inner* como se fosse implementada por ele mesmo. Isto é útil quando o *outer* sempre precisa delegar a mesma interface para o *inner*.

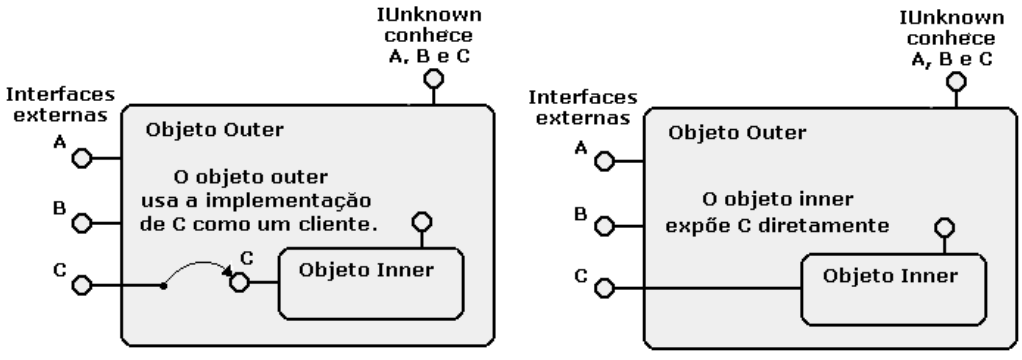


Figura 3.2.: representação de conexão por Containment ou por Aggregation.

3.3. DirectTextToSpeech API

Os componentes da DirectTextToSpeech API estão distribuídos em diferentes níveis de complexidade e flexibilidade e são utilizáveis por diferentes linguagens, por exemplo C++, Java ou Visual Basic. Em um nível mais baixo, como componentes ActiveX, tem-se menor controle, porém maior simplicidade. Analogamente, em um nível mais alto tem-se maior controle e consequentemente maior complexidade. Os componentes COM de nível mais alto são os componentes fundamentais, onde estão inseridos os sistemas de síntese e reconhecimento, chamados de *engines* ou motores. O usuário pode ter instalado em sua máquina motores de diferentes línguas ou diferentes fornecedores, pois existem componentes da Speech API que enumeram os motores instalados na máquina.

Os componentes principais envolvidos no sistema TTS são: um objeto (ou componente) de áudio, um objeto enumerator e um objeto engine, que são criados e agrupados nesta mesma ordem. Em segundo plano estão objetos de notificação, que são usados para trocar notificações do engine para a aplicação e do áudio para o engine. A interação geral entre os componentes é a seguinte:

- A aplicação cria o *objeto de áudio* para onde o TTS enviará o sinal de fala sintetizado. A Microsoft já implementa um *objeto AudioDest* para comunicar-se com dispositivo de áudio do sistema operacional.

- A aplicação, através do *objeto TTS enumerator*, procura um engine instalado que implementa um tipo de voz que se quer usar. Em seguida, é criada uma instância do engine que recebe a referência para o objeto de áudio.

- O *objeto engine* quando é criado dialoga com o objeto de áudio para encontrar um formato de áudio digital comum. Uma vez encontrado, o engine cria um *objeto de notificação* para o áudio, por onde o objeto de áudio enviará informações sobre seu estado.

- A aplicação pode registrar um *objeto de notificação* de onde receberá informações do engine a respeito do estado do processo de conversão ou mudanças de atributos.

- Estabelecido os acordos de comunicação e interface, a aplicação passa um ou mais textos para o engine que serão colocados em fila e convertidos. O texto pode possuir *tags* inseridas, que modificam características de intonação ou marcam o texto para sincronismo. Em conjunto com o texto é enviado uma *interface de notificação de buffer*, da qual a aplicação recebe notificações a respeito do processamento do *buffer* de texto enviado.

Algumas tarefas para a inicialização do sistema TTS, descritas acima, são de responsabilidade da aplicação, outras são tarefas do engine. Apesar do enfoque não estar na aplicação, é preciso conhecer a arquitetura e quais os passos básicos do sistema. Na figura 3.3. estão ilustrados todos os componentes e interfaces envolvidos no sistema, descritos nos tópicos a seguir.

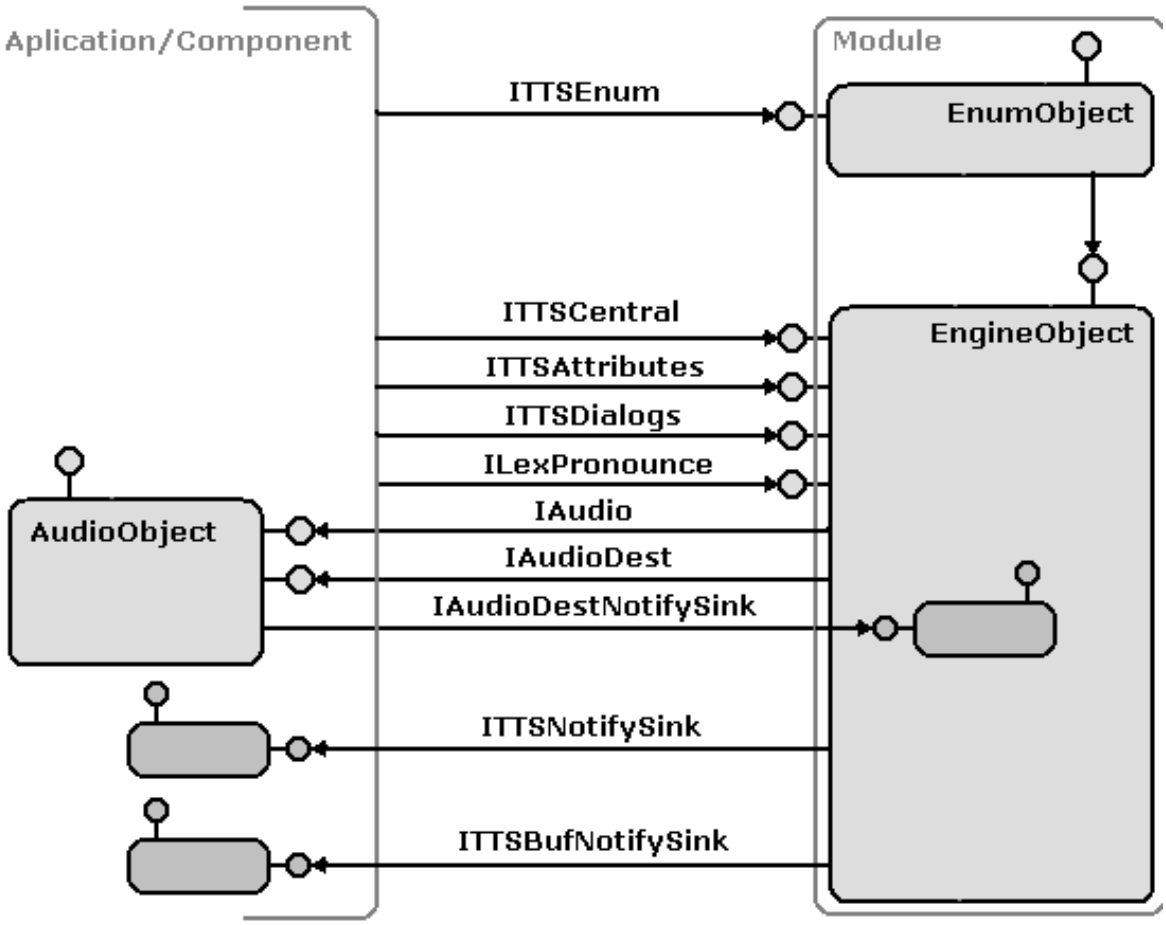


Figura 3.3.: representação da arquitetura DirectTextToSpeech API.

3.3.1. Objeto de áudio

A Microsoft fornece vários objetos de áudio para diferentes objetivos. Neste caso o objeto de áudio que interessa é o objeto *MultimediaAudioDestination* que permite à aplicação criar um objeto para envio de áudio baseado no driver do dispositivo de áudio instalado na máquina, que pode ser usado para *Text-To-Speech*. O objeto é criado pelo identificador *CLSID_MMAudioDest* retornando o endereço da interface *IUnknown*, da qual são obtidas as referências para as outras interfaces, que agrupam as seguintes funcionalidades:

Interface	**Descrição**
IAudio	Permite gerenciar o *buffer* de áudio interno e controlar atributos do dispositivo de áudio representado;
IAudioDest	Envia informações e dados para o objeto *AudioDest*;
IAudioDestNotifySink	Notifica o *engine* a respeito de mudanças do *buffer* de áudio;
IAudioMultiMediaDevice	Permite acessar características específicas do dispositivo de áudio.

Quando o objeto de áudio é criado pela primeira vez, é usado *multimedia WAVEMAPPER* como dispositivo. No entanto, para utilizar outro dispositivo, após criado o objeto de áudio a aplicação deve chamar a função membro IAudioMultiMediaDevice::DeviceNumSet, identificando o dispositivo de áudio a ser usado como saída.

Quando o engine é criado, ele irá indicar ao objeto de áudio o formato do wave desejado e em seguida registrar a interface de notificação *IAudioDestNotifySink*, por onde o engine receberá informações do objeto de áudio.

3.3.2. Objeto engine enumerator

O objeto engine enumerator é usado para enumerar os modos disponíveis que o sistema implementa. Modos de um sistema significa diferentes vozes, diferentes formatos de áudio ou diferentes línguas e dialetos. Suas interfaces são:

Interface	**Descrição**
ITTSEnum	Enumera e seleciona os modos de operação do engine.
ITTSFind (Opcional)	Procura e seleciona um modo que satisfaz as características procuradas.

A aplicação não interage diretamente com o objeto engine enumerator, mas utiliza o componente *Text-To-Speech enumerator*, implementado pela Speech API, para redirecionar as chamadas de funções para o objeto engine enumerator, implementado pelo engine. Nota-se que ambos componentes possuem responsabilidades e interfaces similares, no entanto a diferença fundamental entre eles é que o *Text-To-Speech enumerator* enumera todos os modos de todos os engines instalados na máquina. Por sua vez, o objeto engine enumerator lista somente os modos implementados por um engine particular.

Quando uma aplicação chama a função membro *ITTSFind::Find* de *Text-to-Speech enumerator*, ou alternativamente quando a aplicação chama as funções membro da interface *ITTSEnum* para enumerar os modos, cada objeto engine enumerator de cada engine registrado, é criado.

A interface *ITTSFind* é opcional para o objeto engine enumerator, por que se o objeto *Text-to-Speech enumerator* requisitar esta interface e ela não existir, automaticamente ele utilizará *ITTSEnum* para listar os modos e encontrar um modo que satisfaça as características procuradas. Se o modo apropriado é encontrado, é retornado a estrutura *TTSModeInfo* que contém informações a respeito do modo, inclusive o GUID. Então, o objeto engine enumerator irá criar o objeto engine no modo escolhido, identificado pelo GUID. Uma vez que o objeto engine existe, o objeto engine enumerator não é mais necessário sendo automaticamente liberado quando a aplicação libera o *TTS enumerator*.

3.3.3. Objeto engine

O objeto engine implementa o sistema de conversão do texto em fala. O objeto engine enumerator é que cria o objeto engine, passando o endereço da interface *IUnknown* do objeto de áudio. O objeto engine recebe notificação do objeto de áudio a respeito do estado do áudio que está sendo tocado, ou se há espaço disponível para mais dados. Desta forma é possível o engine gerenciar o envio de dados para o áudio. Quando o objeto engine inicia, ele utiliza as interfaces de áudio para determinar o formato de áudio especificado e enviar a interface por onde receberá as notificações. Se o engine é criado e inicializado com sucesso, é retornado o endereço da interface *ITTSCentral*, através da qual é possível requisitar as outras interfaces do engine que são:

Interface	Descrição
ILexPronounce	Permite a aplicação controlar a pronúncia de determinadas palavras.
ITTSAttributes	Controla os atributos do engine. As funções membro permitem ajustar pitch, velocidade e volume da voz;
ITTSCentral	Controla o engine. As funções membro permitem uma aplicação enviar texto para o engine; converter caracteres de texto para representação fonética; interromper, continuar, e reiniciar a saída de áudio; obter informação a respeito do modo do motor e registrar a interface de notificação;
ITTSDialogs	Cria janelas de diálogo que permitem o usuário configurar o engine, ou controlar a pronúncia de alguns símbolos;
ITTSBufNotifySink	Notifica a aplicação a respeito das mudanças no buffer que contém o texto sendo falado;
ITTSNotifySink	Notifica a aplicação a respeito de eventos no processamento do texto, tais como mudança de atributos, o tempo que o áudio começa ou termina; quando um fonema está sendo falado e a informação gráfica correspondente à posição dos lábios;

Para que a aplicação receba notificações do objeto engine através de *ITTSNotifySink,* ela precisa criar um objeto que suporte esta interface e registrá-la através da interface *ITTSCentral* do objeto engine.

3.4. Conclusão

Neste capítulo foi visto que COM estabelece um padrão binário para interação entre componentes, é independente da linguagem de programação, é multiplataforma e extensível. Foi visto também os conceitos fundamentais de COM, entre eles o agrupamento de funções em interfaces, o uso de uma interface padrão utilizada para criar instâncias do componente, um mecanismo de identificação única para cada componente, e uma biblioteca do sistema, responsável por gerenciar os componentes. Foi também citado o mecanismo de reutilização (agregamento) entre componentes.

Em seguida foram descritos os componentes da *DirectTextToSpeech API* e as respectivas interfaces. Os principais componentes e interfaces descritos foram: o objeto (ou componente) de áudio, responsável por reproduzir o som enviado pelo sistema de síntese, e suas interfaces *IAudio, IAudioDest, IAudioDestNotifySink, IAudioMultiMediaDevice*; o objeto engine enumerator, responsável por listar os modos de fala dos engines, e a interface *ITTSEnum*; e finalmente o objeto engine, responsável por implementar o sistema de síntese, e as interfaces *ILexPronounce, ITTSAttributes, ITTSCentral, ITTSDialogs, ITTSBufNotifySink, ITTSNotifySink.*

Capítulo 4 - Implementação da arquitetura TTS

4.1. Introdução

A partir da descrição dos componentes que compõem a DirectTextToSpeech API, deseja-se criar um sistema que seja compatível com estas definições, ou seja, criar um software que possua uma arquitetura compatível com esta API. A arquitetura pode ser entendida como o "invólucro" onde o sistema de conversão estará inserido.

Implementar a arquitetura TTS consiste em inicialmente criar o projeto de um módulo, ou seja, o lugar físico para os componentes, que neste caso será uma DLL; em seguida implementar as funções utilizadas pela biblioteca COM para localizar e instanciar um componente (implementações genéricas de um componente COM); depois são declarados os componentes e as interfaces que compõem a arquitetura, que são o objeto enumerator e o objeto engine; e finalmente são implementadas as funções membro das interfaces que os objetos expõem.

Os tópicos a seguir se referem a construção de um projeto no Microsoft Visual C++, plataforma de desenvolvimento julgada adequada para implementação do motor. Procurou-se sempre seguir a metodologia de programação baseada em objetos ao longo da implementação.

No último tópico, será descrita uma aplicação para demonstração, na qual é possível verificar se a arquitetura está compatível, ou seja, se os componentes do motor são encontrados pelo sistema operacional e se as interfaces são criadas e as funções membro são invocadas corretamente.

4.2. Implementação do módulo do motor

O módulo do motor é o lugar físico onde os componentes estão inseridos, que neste caso está compilado em uma DLL dinâmica. A DLL exporta quatro funções que o sistema utiliza para

criação, registro e liberação dos componentes. Para as funções serem exportadas, é preciso declará-las em um arquivo de extensão “.def”.

DllCanUnloadNow: esta função é utilizada pelo sistema operacional para saber se a DLL está inativa e se pode ser liberada. Internamente a função verifica se o contador dos objetos estão em zero e se estiver a DLL pode ser liberada.

DllGetClassObject: esta função é utilizada pelo sistema operacional para requisitar a criação do componente. Todo componente COM precisa implementar um objeto *ClassFactory*, que possui a interface padrão *IClassFactory*, da qual o sistema utilizará as funções membro para instanciar o componente e obter o apontador para a interface *IUnknown.*

DllRegisterServer: esta função é utilizada para registrar o componente no momento em que é instalado no sistema. A instalação é feita executando-se o arquivo “regsvr32.exe”, (fornecido junto do sistema operacional) passando como argumento o caminho onde a DLL se encontra.

Como citado anteriormente, é através da interface *IClassFactory* que o sistema instancia o componente. Quando a aplicação solicita a criação de um componente ao sistema, é passado a identificação da classe do componente (CLSID) para que o sistema procure no registro onde está o módulo (DLL) onde a *ClassFactory* e o componente estão implementados. Desta forma a busca pela *ClassFactory* é feita no local de registro sob a chave:

HKEY_CLASSES_ROOT\CLSID

{GUID}	"*TTS Engine*"	(GUID e nome)
InprocServer32	"..*ttsengine.dll*"	(caminho da dll)
ThreadingModel	**"Apartment"**	

O engine precisa ainda estar instalado num espaço específico do registro, onde o componente *TTS enumerator* irá buscar todos os motores de síntese instalados, que é:

HKEY_LOCAL_MACHINE\ Software\ Voice\ TextToSpeech

Engine

"*TTS Engine Portuguese*"	"*{GUID}*"	(Nome e GUID)

DllUnregisterServer: esta função é utilizada para desinstalar a DLL, retirando do registro as inscrições descritas anteriormente.

Para otimizar o código, as funções exportadas pela DLL foram implementadas em uma classe chamada *CModule* que encapsula o processo de contagem de objetos para gerenciamento de múltiplas instâncias; implementa as funções intermediárias para registro do motor; e através de um ponteiro para *ClassFactory* implementa as funções para instanciar o componente.

A instância de um componente é obtida da seguinte forma: após identificada a DLL (ou módulo) onde está o componente, o sistema chama a função *DllGetClassObject* passando o GUID do componente que deseja criar. Esta função chama o método *CModule::GetClassObject*. Este método verifica se o registro pertence ao seu componente, instancia o objeto *ClassFactory* e através de *IUnknown::QueryInterface* obtém o ponteiro para a interface *IClassFactory*. Em seguida o sistema operacional chama o método *IClassFactor::CreateInstance* para instanciar o componente desejado, que neste caso é o objeto engine enumerator.

4.3. Implementação do objeto enumerator

Existem diferentes formas de implementar em C++ um componente que possui várias interfaces e ainda utiliza outros componentes. Por exemplo, pode ser criada uma classe para cada interface, que estão agregadas à um objeto principal. Outro tipo de implementação é um único objeto que engloba todas as interfaces numa única tabela de funções virtuais, este método é usado em ATL /1//2/. Pode-se dizer que a técnica utilizada neste projeto possui um pouco das duas. De modo a esclarecer melhor o tipo de implementação usada, será utilizado como exemplo a implementação do objeto engine enumerator e a interface *ITTSEnum*.

Quando chamada a função *IClassFactory::CreateInstance* o objeto engine enumerator será criado e retornado um apontador para uma interface *IUnknown*. Em seguida, através de *QueryInterface*, será requisitada a interface *ITTSEnum*. No entanto, uma aplicação pode ser construída utilizando caracteres em UNICODE ou ANSI, para Windows NT ou 98 por exemplo. Como deseja-se que o motor de síntese funcione em qualquer plataforma Windows, o motor precisa implementar as interfaces nas versões W e A. Estas interfaces possuem as mesmas funções membro e o mesmo comportamento, no entanto o formato de dados de

entrada e saída são diferentes. Algumas funções possuem argumento de entrada em formatos idênticos, mas os caracteres contidos nas estruturas destes argumentos são de tipos diferentes, o que torna inviável para um objeto implementar duas funções com o mesmo tipo de argumento, porém tratamento diferente. Por isso, foi escolhido construir um objeto para a interface W, outro para A e um objeto principal, independente do número de interfaces que ele possua. Por exemplo, o objeto enumerator precisa implementar obrigatoriamente a interface *ITTSEnum*, portanto foram implementados três objetos: *CTTSEnumW* que implementa a interface *ITTSEnumW*; *CTTSEnumA* que implementa a interface *ITTSEnumA* e *CTTSEnumObject* que expõe a interface do componente conforme requisitada. Nota-se que se houvessem mais interface a serem implementadas, mesmo assim seriam construídos 3 objetos, como será visto na construção do objeto engine.

A implementação das funções, que possuem comportamento idêntico, nos leva ainda a otimizar o código agregando os objetos W e A de forma que somente os métodos de um objeto seja implementado e o outro somente repasse a chamada fazendo a conversão dos dados de entrada e saída. Assim, foi escolhida a interface UNICODE para ser implementada, tendo em vista que somente o código UNICODE possui os símbolos para IPA (Alfabeto Fonético Internacional), por conter 16 bits para um caracter ao invés de 8, sendo preferível em implementações posteriores.

O tipo de "agregamento" entre os objetos é mostrado na figura 4.1. Os objetos que implementam as interfaces W e A são declarados de modo que possam ter acesso aos métodos e atributos do objeto principal. Em contrapartida, o objeto principal possui acesso aos objetos que implementam as interfaces, sendo responsável por criá-los.

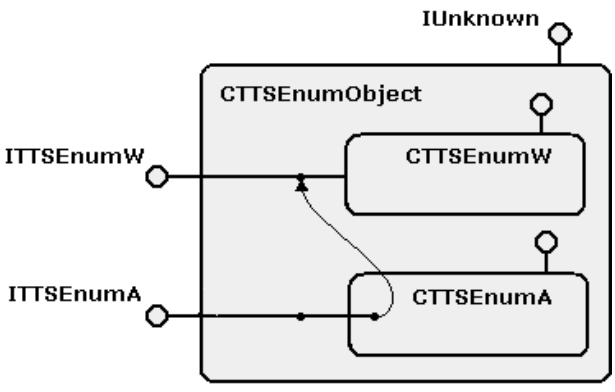


Figura 4.1.: agregamento entre os objetos.

O objeto engine enumerator ou *CTTSEnumObject,* herda a interface *IUnknown,* implementa as funções membro e implementa uma função de inicialização que cria os objetos das interfaces. Os objetos das interfaces, *CTTSEnumW* e *CTTSEnumA,* precisam herdar as interfaces que implementam[1], não precisam herdar a interface *IUnknown* pois já estão implícitas, mas implementam as funções membro da interface *IUnknown*, e das interfaces *ITTSEnum* W e A, respectivamente.

Quando *CTTSEnumObject* é criado, este cria um objeto para a interface W e outro para A, passando aos objetos sua interface padrão *IUnknown* para agregar-se como *Outer*. Juntamnte passa seu próprio apontador para que as interfaces tenham acesso aos seus métodos e atributos. O método *CTTSEnumObject::QueryInterface* retorna um ponteiro para *CTTSEnumW* se for requisitado uma interface W, para *CTTSEnumA* se for requisitado uma interface A, ou para a interface *IUnknown*. Quando as funções *CTTSEnumW::AddRef* e *CTTSEnumW::Release* são chamadas, o objeto *CTTSEnumW* incrementa seu contador de referência e repassa a chamada para o objeto *outer*. O objeto *CTTSEnumA* só existe se o *CTTSEnumW* existir, portanto ele não precisa de contadores de referência, repassando essa responsabilidade ao objeto *CTTSEnumW*. Por simplicidade, adiante será citada somente a interface sem considerar as diferenças entre UNICODE e ANSI.

4.3.1 Interface ITTSEnum

O objeto *Text-To-Speech enumerator* da Speech API requisita a criação do objeto enumerator para montar uma lista de modos disponíveis, através das funções membro da interface *ITTSEnum*, descritas abaixo:

Clone	Faz uma cópia do objeto enumerator no estado atual. Esta função é utilizada para guardar a posição de uma seqüência de modos do motor;
Next	Retorna o próximo item da seqüência de modos;
Reset	Reinicia a seqüência de enumeração ao inicio;
Skip	Adianta a seqüência;

[1] Todas as interfaces e seus respectivos identificadores (IID) estão declarados no arquivo “speech.h”.

Select	Cria o objeto engine em um modo específico, passando um ponteiro para o objeto de áudio, criado anteriormente na aplicação; inicia-o; e requisita a interface *ITTSCentral.*

O modo de um engine é caracterizado por determinado tipo de voz, por diferentes freqüências de amostragem ou por filtros de efeito que modificam a voz original caracterizando uma nova voz. As características de um modo são passadas para a aplicação em uma estrutura chamada TTSModeInfo, que possui os seguinte campos:

gEngineID	GUID que identifica o motor;
szMfgName	nome do desenvolvedor;
szProductName	nome do produto;
gModeID	GUID que identifica o modo do motor;
szModeName	nome do modo;
language	linguagem do modo;
szSpeaker	nome da voz;
szStyle	estilo da voz (sussurrada, rouca,...);
wGender	gênero da voz;
wAge	idade correspondente a voz;
dwFeatures	capacidades do motor;
dwInterfaces	interfaces implementadas pelo motor;
dwEngineFeatures	capacidades específicas.

Após implementada as funções, exceto *ITTSEnum::Select*, é possível testar a compatibilidade inicial do engine, registrando a DLL e verificando se a aplicação consegue listar os modos que o engine implementa. Neste caso, foi implementado dois modos para demostrar o funcionamento das funções, no entanto os dois modos possuem as mesmas características, exceto o nome.

4.4 Implementação do objeto engine

Neste tópico, a partir do projeto do módulo do motor onde foi implementado o objeto enumerator, será implementado o objeto engine e completada a função *ITTSEnum::Select* da interface do objeto enumerator para criar o objeto engine.

O primeiro passo para a criação do objeto engine é a declaração de sua classe. Como no caso do objeto enumerator, o objeto engine também precisa implementar as duas versões para cada interface, UNICODE e ANSI, e portanto tem-se novamente o mesmo tipo de agregamento entre os três objetos que são: *CTTSEngineObject* que é o objeto principal que expões as interfaces; o objeto *CTTSEngineW* que implementa as interfaces W e *CTTSEngineA* que implementa as interfaces A utilizando a interface W. Na figura 4.2. podemos ver um esquema da implementação dos objetos.

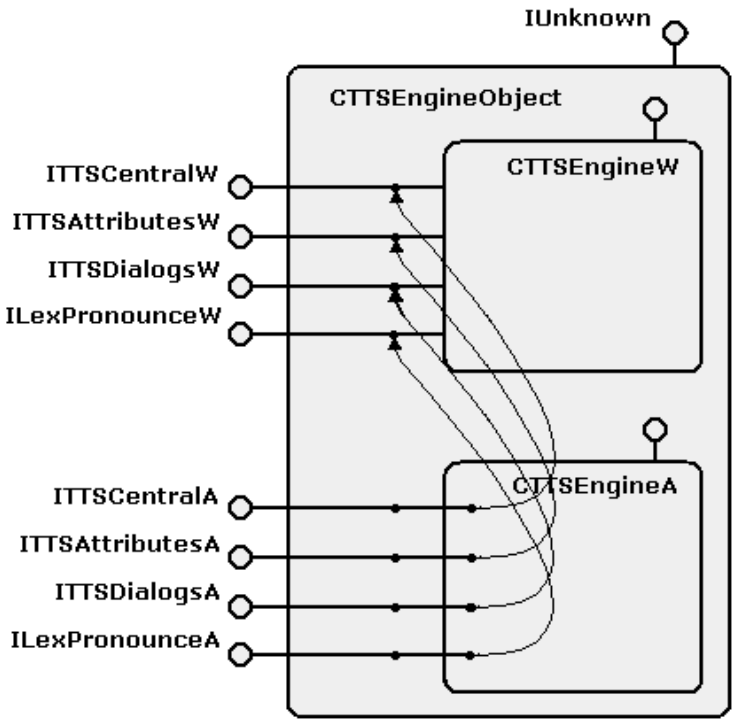


Figura 4.2.: agregamento entre os objetos.

Os objetos *CTTSEngineW* e *CTTSEngineA* herdam as interfaces e implementam as funções membros que serão descritas posteriormente. Tal como o objeto *CTTSEnumObject*, o objeto *CTTSEngineObject* herda somente a interface padrão, implementa as funções membro e implementa uma função de inicialização que cria os objetos W e A. A função

CTTSEngineObject::QueryInterface fornecerá o ponteiro para a interface requisitada, fazendo um "<*type cast*>" ao ponteiro dos objetos W e A. O "<*type cast*>" movimenta o ponteiro do objeto para a posição da interface requisitada dentro da tabela de funções virtuais que o objeto implementa.

Quando o objeto *CTTSEngineObject* é criado, o formato de áudio será verificado através da função *CTTSEngineObject::InitAudio.* Se o formato entre o engine e o objeto de áudio forem compatíveis, o engine fará a requisição das outras interfaces de áudio *IAudio* e *IAudioDest*, que serão utilizadas para o envio de áudio.

Implementada as funções de criação e inicialização do objeto *CTTSEngineObject*, a função *ITTSEnum::Select* pode ser completada, e portanto é possível verificar a compatibilidade do engine. As funções membro das interfaces podem ser implementadas somente com retorno sem efetuar nenhuma ação.

4.4.1 Interface ITTSAttribute

As funções desta interface tratam da mudança dos parâmetros base do engine, através das funções:

PitchGet	recebe o valor do pitch base corrente do engine;
PitchSet	modifica o valor do pitch base corrente. A aplicação pode também determinar que o engine vá para os valores extremos;
SpeedGet	recebe o valor da velocidade de fala (palavras/minuto) média corrente;
SpeedSet	modifica o valor da velocidade de fala base corrente também podendo ser alterada para os valores extremos;
VolumeGet	recebe o valor do volume base corrente;
VolumeSet	modifica o valor base do volume.

Obs.: Todos estes atributos também podem ser modificados através de tags contidas no texto.

A definição destas funções determina que é preciso estabelecer valores mínimos, máximos e médios para os três parâmetros (o volume é determinado de 0x0000 até 0xFFFF). Às vezes uma aplicação utiliza o mecanismo de forçar o engine a um valor mínimo e em seguida ler esse valor. Depois utiliza o mesmo método para os valores máximos e assim encontra o intervalo válido dos atributos.

4.4.2 Interface ITTSDialogs

As funções da interface ITTSDialog são utilizadas para configurações personalizadas do motor, que são:

AboutDlg	mostra a janela de identificação do engine;
LexiconDlg	mostra uma janela de diálogo para permitir ao usuário acesso ao objeto léxico, para modificar a pronúncia de palavras;
GeneralDlg	mostra a janela de configurações gerais do engine;
TranslateDlg	mostra a janela de configurações da tradução de números, abreviaturas ou símbolos.

Inicialmente, será implementada quatro janelas básicas de diálogo para serem futuramente definidas e implementadas. Para facilitar a montagem das janelas, é preciso modificar os parâmetros de compilação para suportar a biblioteca MFC [7]. Permitido o uso desta biblioteca, 4 janelas são criadas quando são chamada as respectivas funções acima.

4.4.3 Interface ITTSCentral

As funções desta interface representam as funções principais do sistema de síntese. Na tabela abaixo temos a descrição de cada função membro da interface.

Inject	injeta tags de controle no texto que está sendo falado;
ModeGet	retorna as informações do modo corrente do motor;

Phoneme	converte o texto para a representação fonética. O código dos caracteres utilizados como retorno desta função podem ser CHARSET_IPAPHONETIC ou caracteres específicos indicados por CHARSET_ENGINEPHONETIC;
PosnGet	retorna o byte do buffer de áudio que está sendo tocado;
TextData	inicia o processo de conversão texto-fala. O texto pode conter tags inseridas no texto, marcadas por barras;
ToFileTime	não implementada;
AudioPause	imediatamente pausa o processo de reprodução;
AudioResume	reinicia a reprodução após pausado;
AudioReset	apaga o buffer de áudio;
Register	registra o objeto de notificação entre o engine e a aplicação;
UnRegister	desfaz o registro do objeto de notificação.

A implementação das funções inicia com a função *ITTSCentral::ModeGet*, que retorna a estrutura *TTSModeInfo* preenchida com as características do modo que está sendo usado. O objeto engine possui como atributo uma estrutura que é preenchida quando o modo é selecionado pela função *ITTSEnum::Select* do objeto enumerator. Portanto basta retornar uma cópia desta estrutura.

As funções *ITTSCentral::Register* e *ITTSCentral::Unregister* serão discutidas quando forem implementadas as interfaces de notificação.

As funções *ITTSCentral::AudioReset, AudioPause* e *AudioResume* precisam verificar o estado do sistema de conversão e em seguida atuar sobre o decorrer do processo e sobre o objeto de áudio. Desta forma, o gerenciamento interno de áudio e o controle do estado do sistema foram implementados utilizando 3 variáveis booleanas: m_fClaimed, m_fPaused, e m_fDataPending, que informam o estado corrente do sistema. A variável m_fClaimed é verdadeira se o áudio já foi aberto pela função *IAudio::Claim* que abre o dispositivo de áudio; a variável m_fPaused é verdadeira enquanto o áudio permanecer interrompido e a variável m_fDataPending é verdadeira se existe dados para serem processados. Assim, as funções verificam as variáveis; executam a ação correspondente e modificam as variáveis.

A função *ITTSCentral::Inject* não terá efeito em tempo real.

A função *ITTSCentral::Phoneme* foi implementada a partir do algoritmo de conversão grafema-fonema que recebe uma string e retorna uma string codificada. No entanto o uso desta função exige que a alocação de memória seja feita de maneira específica[1].

A função *ITTSCentral::TextData* verifica o estado do áudio e inicia-o caso estiver desativado. Em seguida o texto é enviado para os objetos responsáveis pela conversão em fala. O texto será colocado em buffer e é iniciado o processo de conversão. No capítulo a seguir será descrito a implementação do sistema TTS.

4.4.4 Interface IAudioDestNotifySink (audio => engine)

O engine recebe notificações do objeto de áudio, através da interface *IAudioDestNotifySink*, para gerenciar o *buffer* de áudio. Sendo uma interface de comunicação interna, esta interface foi implementada num objeto separado, chamado *CAudioDestNotify*, criado no momento de criação do engine. Quando o engine é inicializado, ele passa o ponteiro do objeto *CAudioDestNotify* através da função *IAudio::PassNotify*. Quando o engine é finalizado, antes liberar as interfaces de áudio, ele precisa notificar novamente passando o parâmetro NULL para que o áudio libere o objeto de notificação. Em seguida o objeto de notificação pode ser liberado. As funções membro da interface, chamadas pelo áudio de acordo com os eventos, são:

AudioStart	é chamada toda vez que o áudio é aberto. É resultado da chamada da função *IAudio::Claim;*
AudioStop	notifica ao engine que nenhuma informação de áudio será reproduzida por um período indefinido;
BookMark	notifica que o *buffer* de áudio encontrou uma marca no *buffer*. Esta função é utilizada para sincronizar o texto com o som. Bookmarks são inseridos no buffer de áudio através da função *IAudioDest::BookMark*;

[1] A aplicação envia um ponteiro para um espaço de memória e a função precisa realocar este espaço utilizando de CoTaskMemAlloc.

FreeSpace	Notifica ao engine que os dados internos do *buffer* foram modificados. O engine pode responder a esta notificação enviando mais dados para o *buffer* de áudio.

4.4.5 Interface ITTSNotifySink (engine=>aplicação)

O engine deve enviar notificações à aplicação a respeito de mudanças de atributos ou de estado através da interface *ITTSNotifySink*. A aplicação envia a interface para o engine através da função *ITTSCentral::Register*. A Speech API define que o engine deve estar preparado para receber mais de um registro de notificação, no entanto esta é uma diretiva de visão futura, e observa-se que atualmente a maioria das aplicações utilizam somente uma interface de notificação. Portanto, o engine manterá somente o último registro como utilizável, substituindo os registros anteriores.

Para não sobrecarregar o objeto engine, foi criado um outro objeto responsável por gerenciar o envio de notificações chamado *CTTSNotify*. Este objeto implementa as funções *Register* e *Unregister* e ainda os métodos para enviar notificação para a aplicação. Assim como as outras interfaces, a interface de notificação pode ser ANSI ou UNICODE, portanto o objeto é responsável por esta identificação e o envio dos parâmetros nos tipos adequados. Esta interface possui 4 funções membro:

AttribChanged	esta função é repassada quando o as funções de mudança de atributo são chamadas;
AudioStart	avisa quando o áudio foi inicializado. É enviada quando é recebida a notificação *IAudioDestNotifySink::AudioStart*;
AudioStop	avisa quando o áudio foi terminado. É enviada quando é recebida a notificação de *IAudioDestNotifySink::AudioStop;*
Visual	fornece informações para controle gráfico. Esta função retorna o fonema que está sendo falado e uma estrutura que contém dados de controle para a posição dos lábios.

A função *CTTSNotify::Visual* foi implementada para verificar o funcionamento da interface de notificação para simular o movimento dos lábios. No entanto o tempo disponível para o projeto não é suficiente para recolher as informações de posição dos lábios para todos os fonemas.

4.4.6 Interface ITTSBufNotifySink (engine=>aplicação)

A cada envio de texto ao engine, é enviado juntamente uma interface de notificação *ITTSBufNotifySink*, por onde a aplicação recebe notificações a respeito de mudanças no buffer ao longo do processo de conversão do texto em fala. As funções membro desta interface são:

BookMark	notifica a aplicação que o engine encontrou um Bookmark no texto que está sendo falado;
TextDataDone	notifica a aplicação que o texto em buffer foi enviado para o objeto de áudio ou a aplicação fez reset ao engine;
TextDataStarted	notifica a aplicação que o *buffer* de texto foi colocado no inicio da fila de textos a serem falados;
WordPosition	notifica a aplicação qual palavra que está sendo falada. Usualmente é utilizado caracteres de espaço para definir o limite de palavras.

Para gerenciar o envio destas notificações foi implementado um objeto a parte. Para cada texto enviado, uma nova interface é também enviada, no entanto observa-se que geralmente uma única interface é reenviada, portanto inicialmente foi implementado o suporte para somente uma interface durante todo o processo.

4.5 Demonstração

Conforme dito anteriormente, se o sistema TTS é compatível com a Speech API então pode ser utilizado por qualquer aplicação também compatível. Para verificação e demonstração do funcionamento do sistema TTS implementado, será utilizado um aplicativo desenvolvido pela Microsoft, que permite verificar o funcionamento de todas as funcionalidades do sistema de síntese. Na figura 4.3. tem-se a janela inicial da aplicação.

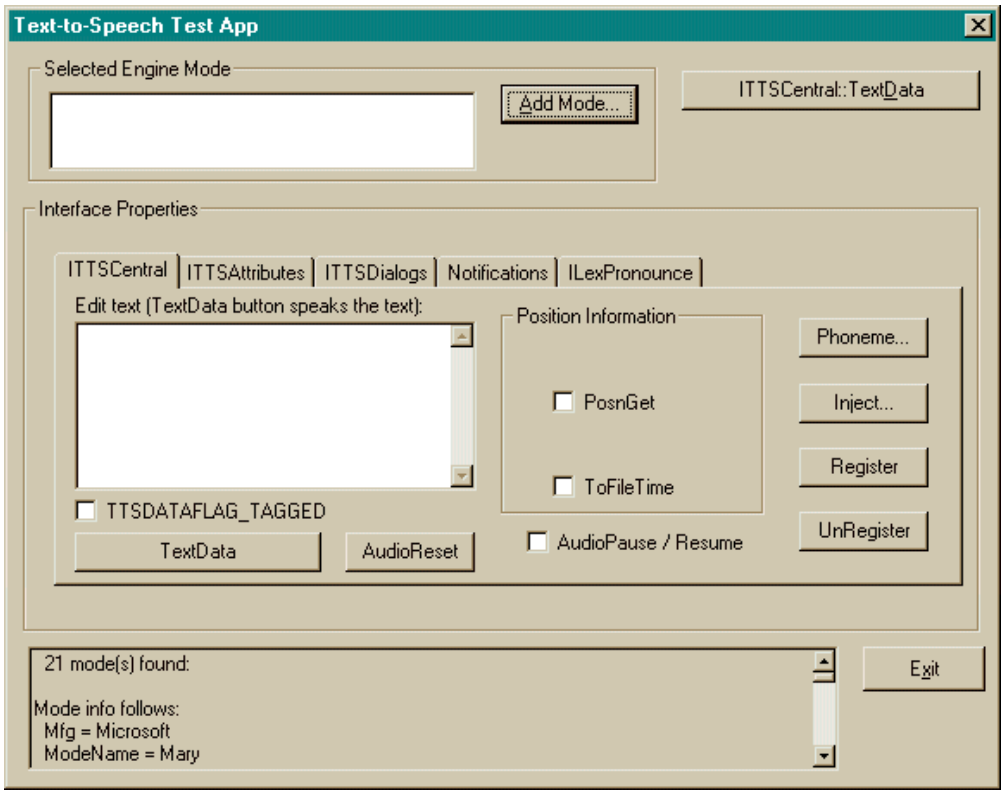


Figura 4.3.: Aplicação de demonstração.

No painel inferior da aplicação são demonstrados os resultados de todas as ações feitas pela aplicação. Ao iniciar, foram listados neste painel todos os sistemas TTS compatíveis, instalados e registrados. No painel superior, é mostrado o modo do motor selecionado. Através do botão *AddMode*, é possível verificar as características dos modos de todos os sistemas, conforme a figura 4.4., e escolher um a ser utilizado. Quando escolhido um modo, o componente do sistema TTS é colocado em memória e todas as suas interfaces são obtidas.

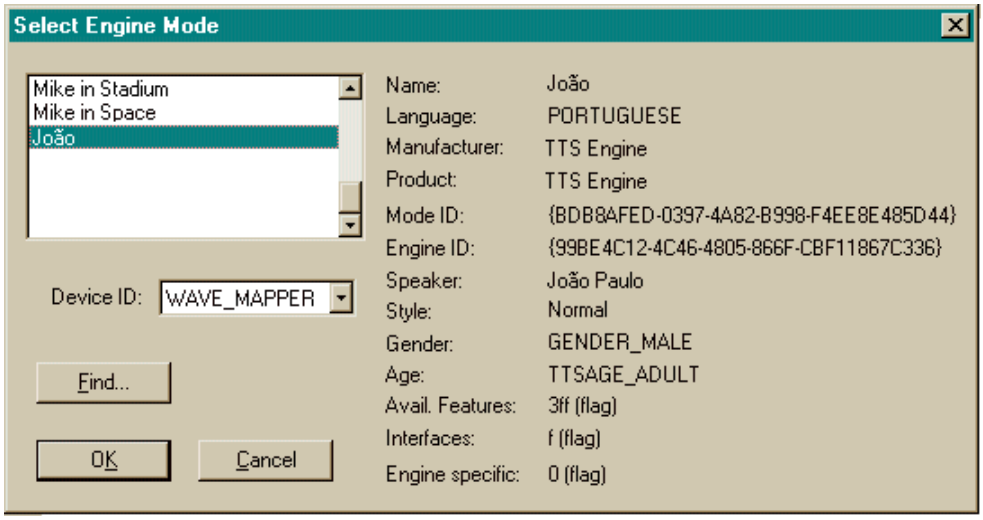


Figura 4.4.: janela de demonstração dos modos instalados

O painel central, é composto por 4 janelas sobrepostas, onde cada uma corresponde ao grupo de funções de uma interface. Os botões e outros controles possuem o mesmo nome das funções e exercem ações correspondentes. Por exemplo, na figura 4.3. a interface *ITTSCentral* é mostrada como a janela principal, onde no painel de edição é inserido o texto a ser sintetizado. O texto pode ou não conter tags e é enviado para o sintetizador quando selecionado o botão *TextData*. Outro exemplo é o painel *Notifications*, figura 4.5., onde são mostradas as notificações enviadas pelo engine.

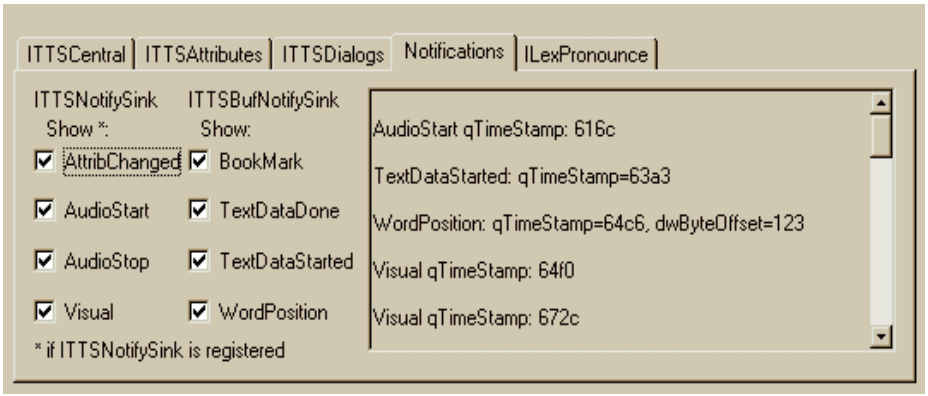


Figura 4.5.: painel de visualização de recebimento de notificações

Neste exemplo de aplicação, todas as funcionalidades são expostas, no entanto, em uma aplicação prática a ser utilizada por um usuário comum, o gerenciamento dessas funções não são de responsabilidade do usuário, mas da aplicação. A exemplo disto, temos um componente, também desenvolvido pela Microsoft, que encapsula síntese e reconhecimento de voz na interface de um personagem animado. O componente, desenvolvido em parceria com L&H (Learnout & Houspie) chama-se MSAgente. A figura 4.6. mostra um dos personagens, Mérlin.

Figura 4.6.: MSAgent Mérlin.

O componente pode ser utilizado diretamente em uma página html ou por uma aplicação executável, desenvolvida em Visual Basic, Java ou C++. O programador pode criar novos movimentos para o personagem ou até criar um novo personagem. Através do reconhecimento da voz, é possível determinar ações, como fechar uma janela ou fazer uma busca na internet, quando determinadas palavras são reconhecidas. Utilizando a síntese de voz, através de uma única função (Speak) é possível descrever menus, dar avisos, ou interagir com o usuário.

4.6 Conclusão

Neste capítulo foi descrito a implementação da DLL onde o motor está inserido incluindo as implementações genéricas de um componente COM. A seguir, foi discutido a reutilização de componentes como método de otimização de código permitindo que o motor fosse implementado em duas versões simultaneamente, UNICODE e ANSI.

Em seguida foram implementados o objeto enumerator e a interface *ITTSEnum* com as funções membro: *Clone, Next, Reset, Skip e Select.* O objeto engine também foi implementado, juntamente com as interfaces: *ITTSAttribute*, com as funções membro *PitchGet, SpeedGet, SpeedSet, VolumeGet, VolumeSet; ITTSDialogs*, com as funções membro *AboutDlg, LexiconDlg, GeneralDlg, TranslateDlg; ITTSCentral*, com as funções membro *Inject, ModeGet, Phoneme, PosnGet, TextData, AudioPause, AudioResume, AudioReset*, Register, UnRegister; IAudioDestNotifySink, com as funções membro *AudioStart, AudioStop, BookMark, FreeSpace; ITTSNotifySink* com as funções *membro AttribChanged, AudioStart, AudioStop*, Visual; e finalmente a interface *ITTSBufNotifySink* com suas funções membro *BookMark, TextDataDone, TextDataStarted, WordPosition.*

Por último foi apresentado uma aplicação para demonstração, na qual foi possível verificar que a arquitetura é compatível, ou seja, os componentes do motor são encontrados e identificados pelo sistema operacional, as interfaces são criadas e as funções membro são invocadas corretamente.

Capítulo 5 – Implementação do sistema TTS

5.1. Introdução

A arquitetura discutida e implementada até o momento serve de base para inserção do sistema que fará a conversão do texto em fala. O objeto engine é considerado o objeto principal, que mantém todos os atributos gerais do sistema; é responsável por estabelecer o contato com a aplicação e gerenciar os objetos subjacente que implementam o sistema TTS. Quando o engine recebe um texto para ser sintetizado, é desejável que o processo de síntese seja iniciado sem que o engine precise ficar "preso" ao processo de conversão, ficando disponível para outras operações secundárias ou para envio de mais texto. Também é desejável que o tempo de resposta, ou seja o tempo entre o envio do texto e a produção do som, seja reduzida; e ainda é desejável que a leitura do texto não possua interrupções dedicadas ao tempo de processamento do texto posterior. Essas metas afetam diretamente a estrutura de software do sistema.

Em paralelo à estrutura do software, temos a estrutura dos dados que transitam entre as etapas de conversão. Em sistema TTS atuais, um grande esforço tem sido dedicado a extração de fatores lingüísticos do texto, principalmente para determinação da prosódia. Esta abordagem lingüística requer uma organização de dados complexa e bem estruturada, que já vem sendo discutida por profissionais da área da lingüística computacional. O uso de tags, inseridas no texto, para acréscimo ou modificação das características para síntese, como pronúncia de palavras, ênfase, mudança de velocidade, etc., também requerem uma estrutura de dados mais complexa do que a seqüência de caracteres. Além disso, parte destes dados precisam estar presentes até a etapa de geração do sinal, e mesmo que este projeto não pretenda incluir processamento lingüístico, será implementada uma estrutura de dados que vá em direção a este desenvolvimento futuro [12].

Neste capítulo será descrito a implementação da estrutura de dados e estrutura do software, que formam o sistema TTS.

5.2. Estrutura de dados

Para o projeto da estrutura de dados, inicialmente o texto é interpretado como um grupo de unidades, ou objetos, inter-relacionados, organizados conforme uma determinada hierarquia, que possuem atributos próprios e são capazes de determinar ou modificar seus próprios atributos. A partir deste conceito, a implementação da estrutura procurou seguir uma implementação orientada a objetos, no entanto sem muito rigor de análise[1].

Em primeira análise, observa-se que o processo de conversão do texto, precisa ser feito em uma unidade menor que o texto, para minimizar o tempo de espera e as pausas para processamento. Deste modo, a frase limitada por pontuação (.:;?!) foi considerada a maior unidade de processamento e portanto o primeiro elemento que compõe o texto ou o objeto de maior hierarquia. A escolha da frase se deve aos seguintes motivos: se ocorrerem pausas no processamento a pausa acontecerá no fim de uma frase e não ao meio de uma palavra ou entre palavras, nos quais afetaria a inteligibilidade; e a consideração da quase[2] independência entre frases.

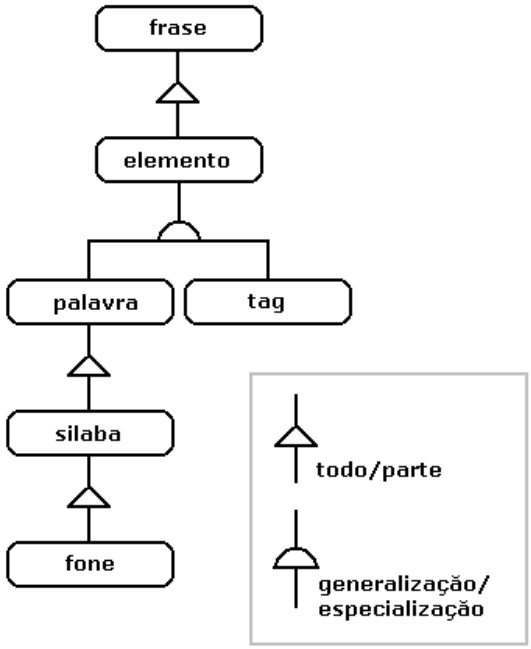


Figura 5.1.: Diagrama de classes da estrutura de dados.

Assumindo que a frase é composta por elementos, que podem ser tags ou palavras (posteriormente números, abreviaturas, símbolos, etc.), chega-se aos objetos hierarquicamente abaixo da frase, conforme pode ser visto na figura 5.1.. Em seguida, uma palavra possui uma realização fonética e desta originam as sílabas (tônicas ou átonas), que sugerem o próximo nível de objetos. As sílabas podem ser consideradas como conjuntos de fones, que representam o objeto de menor hierarquia. Desta forma, uma frase é indiretamente composta por um conjunto de fones, que são objetos com atributos acústicos.

[1] A analise é uma etapa fundamental para a implementação orientada a objetos, no entanto neste projeto pretende-se ter uma abordagem inicial para este tipo de concepção de estrutura de dados.

[2] a pausa entre frases, indica certa independência na coarticulação de fonemas vizinhos, no entanto não se pode afirmar que exista independência prosódica entre frases.

Um conjunto trata-se de uma lista encadeada de objetos de mesma classe, onde o elemento imediatamente superior na hierarquia possui o ponteiro para o inicio da lista. A escolha por implementar listas encadeadas é discutível, no entanto procurou-se sempre utilizar alocação dinâmica de memória, no qual a implementação por listas pareceu ser uma boa escolha. Outros modelos mais complexos que utilizam estruturas lingüísticas para a formação do texto, incluindo dicionários léxicos e gramáticas já vem sendo propostos e utilizados em corretores gramaticais, analisadores sintáticos e em sistemas TTS comerciais.

Quando o texto é enviado ao sistema, a primeira etapa é a montagem da estrutura de dados, no entanto o texto precisa estar normalizado para que os objetos sejam recortados corretamente. Neste projeto, não foi dado atenção ao tratamento de numerais, abreviaturas, etc. Os caracteres diferentes dos limitadores de frase, não pertencentes ao alfabeto português e que não formam uma tag, são substituídos por espaços em branco. O formato das tags respeitam o padrão definido pela Speech API, onde: todas as tags começam e terminam com uma barra invertida (\); o caracter para a barra não é permitido sem um a tag; são case-insensitive; e são dependentes de espaços em branco. Todas as tags são opcionais, a não ser a tag “mrk” que marca o texto. O conjunto de tags consideradas essenciais que foram implementadas são:

Sintaxe	Descrição	Observação
Mrk=number\ Ex: \Mrk=75000\	Indica um bookmark no texto;	Quando o engine encontra esta tag, ele notifica a aplicação por *ITTSBufNotifySink::Bookmark*.
Pit=*number*\ Ex: \Pit=90\	Determina o pitch base em Hz.	Equivalente a função *ITTSAttributes:: PitchSet*
Spd=*number*\ Ex: \Spd=120\	Determina a velocidade média de fala em palavras por minuto.	Equivalente a função *ITTSAttributes:: SpeedSet*
Vol=*number*\ Ex: \Vol=32768\	Determina o volume base.	Equivalente a função *ITTSAttributes:: VolumeSet.* O nível de volume é linear de 0 até 65535.
Rst\	Reinicia o engine para os parâmetros iniciais.	

Após o texto normalizado, inicia-se a montagem da estrutura a partir do recorte de frases.

5.2.1. Montagem da estrutura

Um objeto frase é criado e inicializado com o conjunto de caracteres limitados por pontuação. O método de inicialização irá procurar os elementos que podem ser: palavras delimitadas por espaços ou tags delimitadas por barras (\). Quando encontrado os caracteres que dão origem a uma palavra, um objeto elemento-palavra é criado e inicializado com a seqüência de caracteres que o compõe. No entanto, se os caracteres definem uma tag, um objeto elemento-tag é criado e inicializado, onde serão extraídos os atributos da tag. Voltando ao elemento-palavra, seu método de inicialização fará a conversão dos caracteres (grafemas) para fonemas, que apresentará como resultado uma sequência de caracteres no formato Sampa (anexo A), onde as sílabas serão separadas por espaços e com marcação de tônica. Assim, um objeto sílaba é criado e inicializado com os fonemas que compõem a sílaba, recortados por espaços. Este método irá criar um objeto fone e iniciá-lo, no qual irá recortar o primeiro fonema, procurar em uma tabela a classificação, o índice numérico daquele fonema e a classificação do fone associado àquele fonema, chegando à hierarquia mais baixa da estrutura.

Se a sílaba possui mais de um fone, outro objeto fone é criado, no qual o seu endereço de memória será passado ao fone anterior, e assim o processo se repete até que não haja mais fonemas na sílaba. Terminada a inicialização da silaba, retorna-se a palavra, que irá procurar pela sílaba seguinte e repetir o processo de criação de uma nova sílaba, passando o ponteiro da silaba criada para a silaba anterior, e assim sucessivamente até que não haja mais sílabas na palavra. Terminada a inicialização da palavra, retorna-se a frase que irá procurar pelo próximo elemento (palavra ou tag), criá-lo e passar o ponteiro para o elemento anterior e assim sucessivamente até que não haja mais elementos. Esta etapa é um *parser* do texto, no entanto sem abordagem sintática. Na figura 5.2. a seguir tem-se um exemplo da estrutura de uma frase após montada.

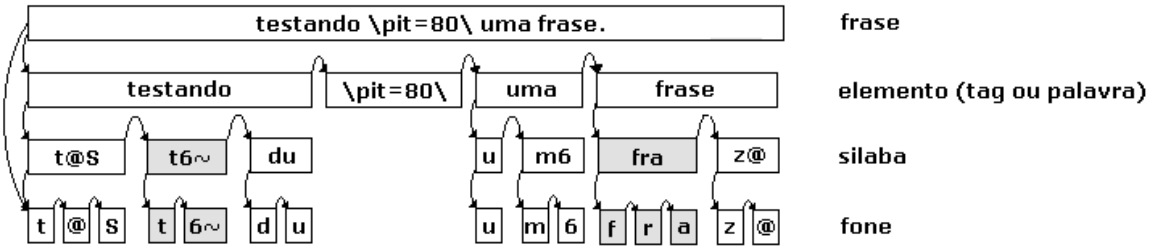


Figura 5.2.: exemplo de estrutura de frase.

Após montada a frase, toda a informação contida no texto foi armazenada na estrutura. Em seguida, outros parâmetros para síntese serão definidos através de aplicação de regras ou leitura de banco de dados. Estes métodos, foram divididos em dois e são: a leitura de difones; e a determinação das durações e dos acentos para o contorno de F0. Apesar destes métodos pertencerem ao objeto frase, eles serão aplicados ao longo da estrutura de software, que será descrita a seguir.

5.3. Estrutura do sistema

Quando o engine recebe um segmento de texto, através da função *TTSCentral::TextData*, o texto é colocado em buffer e é iniciada a conversão. Logo em seguida ao envio de texto, o engine precisa estar livre para receber um novo trecho de texto, independentemente do andamento do processo de conversão em fala. Portanto, a técnica utilizada para que o engine não fique preso ao sistema de conversão, é a implementação do sistema TTS em um *Thread Object*. Quando o engine é criado o objeto em *Thread* (CTTS) é criado em modo suspenso. Cada vez que o engine recebe texto, ele repassa o texto para o buffer de CTTS, "acorda" (resume) o CTTS e retorna, ficando livre para outra ação. Enquanto isso, o objeto CTTS, compartilhando o tempo de uso do processador, verificará se existe texto em buffer e inicia o recorte de frases, montagem da estrutura, etc. Se outro texto é enviado ao engine, é requisitado ao CTTS que coloque outro texto em buffer, que será processado logo que o processamento do texto anterior termine. O buffer de texto trata-se de uma lista encadeada de blocos de caracteres que podem ser equivalentes a parágrafos, frases ou até palavras, dependendo da escolha do usuário. Quando esvaziada a lista, o CTTS entra novamente em modo suspenso.

O processo de conversão em fala, conforme a figura 5.3. e descritas a seguir, inicia com a etapa de montagem da estrutura, ou parser do texto, seguido da leitura do banco de difones e a determinação dos parâmetros prosódicos. A etapa de síntese será descrita no capítulo a seguir.

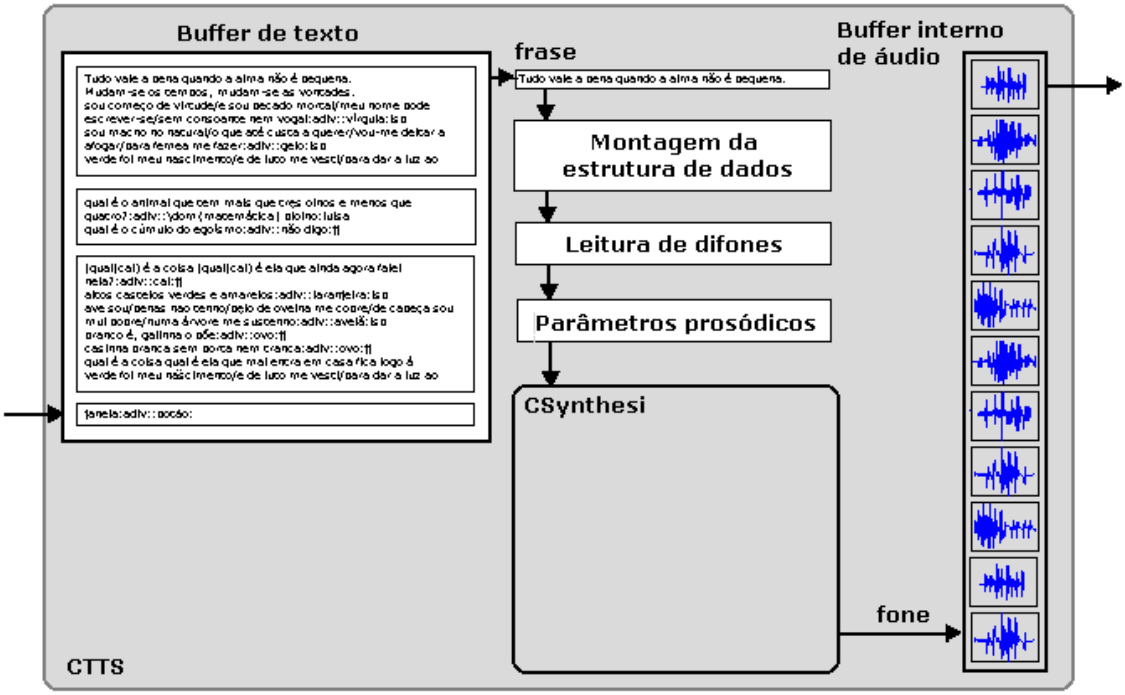


Figura 5.3.: estrutura do sistema TTS.

5.3.1. Leitura do banco de dados

O formato do banco dados depende do modelo de síntese utilizado. Posteriormente serão dados maiores detalhes a respeito do modelo de síntese, neste momento interessa quais os dados que serão lidos e como.

O banco de dados, proveniente de um sistema multi-língua, foi desenvolvido na Faculdade de Engenharia da Universidade do Porto, para o Português Europeu [3]. Trata-se de um conjunto de difones compostos por frames (no máximo 25), uma marca que indica a transição entre os fones que deram origem ao difone e um identificador. Os frames foram obtidos através da parametrização de segmentos de 10ms, do qual não se possui quase nenhuma documentação do método utilizado. Cada frame é uma estrutura contendo 5 formantes, 5 larguras de banda, amplitude e uma informação vozeado/não vozeado.

O identificador do difone é formado a partir do código do fone anterior e do fone em questão. Por exemplo, o difone “pa” formado pelos fones “p” (01) e “a” (24) terá o código (0124). Os difones estão gravados em seqüência, simulando uma matriz indexada em coluna pelo fone anterior e em linha pelo difone posterior, vide Anexo B, e o arquivo é lido somente uma vez e posto em memória. Foi visto que um difone é composto por parte do fone anterior e parte do fone em questão, que eqüivalem à parte da região estável do fone anterior, a região de transição e parte da região estável do fone seguinte. Concatenando-se dois difones, temos um fone completo antecedido e precedido por regiões de transição ou coarticulação.

A listas de fones resultantes da montagem da estrutura, estabelecem a seqüência de concatenação dos difones. Para a leitura de um difone são necessários dois fones, o fone anterior e o fone atual. Portanto, após lido um difone, são copiados para o fone atual os frames contidos após a marca que indica a fronteira entre fones. Em seguida, o difone atual passa a ser o fone anterior e então é lido o difone seguinte. Os frames do fone anterior serão concatenados com os frames do novo difone, até a marca de fronteira, e assim sucessivamente até que todos os fones possuam seus respectivos frames. Desta forma o fone continua sendo a menor unidade na hierarquia e torna-se possível a síntese fone a fone.

5.3.2. Determinação dos parâmetros prosódicos

Duração: A duração de cada fone é determinada inicialmente por um valor médio, encontrado na fala natural [2]. A seguir, caso o fone em questão seja equivalente a uma vogal em sílaba tônica, a duração é aumentada em 20%.

Intensidade: A intensidade de cada fone é inicialmente definida como unitária, no entanto caso o fone em questão também seja equivalente a uma vogal em sílaba tônica, a intensidade é aumentada em 10%.

Parâmetros para a geração do F0: O contorno do F0 será definido pelo modelo de Fujisaki, que será melhor explicado na etapa de síntese, no entanto nesta etapa é necessário definir os parâmetros que serão utilizados para o cálculo do modelo. Resumidamente o modelo requer dois tipos parâmetros: os acentos de comando e os acentos de frase. Foi considerado que uma

frase possui somente um acento de frase, localizado a 150ms antes do inicio da frase, e fixado como padrão. Os acentos de comando foram definido para os limites da vogal, inserida em uma da sílaba tônica. Os efeitos destes parâmetros serão melhor demonstrados na etapa de geração da curva de F0.

5.3.3. Síntese da frase

A partir do momento em que todos os parâmetros para síntese já foram determinados e organizados na estrutura de dados, a frase, ou melhor, os fones que compõem a estrutura da frase, podem ser sintetizados. A etapa de síntese, ou geração do sinal, foi implementada em um objeto chamado CSynthesi, conforme mostrado na figura 5.2, que encapsula todos os algoritmos de processamento de sinal que será descrito no capítulo a seguir. Portanto, ao receber uma frase, o objeto CSynthesi sintetiza-a fone a fone, preenchendo o buffer interno de áudio de CTTS. Nota-se que para ser possível, posteriormente, enviar notificações à aplicação, a respeito do fone que está sendo falado, é necessário que o sinal enviado ao áudio possua marcas de transição de fones. Deste modo, o buffer interno de áudio de CTTS é uma lista de estruturas que contém o trecho de áudio equivalente ao fone e marcas utilizadas para o envio de notificações.

O envio do sinal de áudio, contido no buffer interno de CTTS, para o objeto de áudio é feito por um método do engine, chamado *CTTSEngineObject::SendToAudio*. Este método verifica o espaço disponível no objeto de áudio e copia os dados do buffer interno para o objeto de áudio até que o espaço disponível esteja completo, ou o buffer interno não possua mais dados ou foi encontrado uma marca de fim de buffer. Se os dados do buffer interno contiverem marcas, estas serão enviadas como bookmarks ao objeto de áudio**.**

O método de envio de áudio é chamado pela primeira vez pelo objeto CSynthesi, logo após o buffer interno de áudio ter sido preenchido com no mínimo 1 segundo de áudio, pois o objeto de áudio, após ser inicializado, possui um buffer mínimo de 1 segundo. Portanto, o CSynthesi só poderá chamar a função SendToAudio após sintetizar um segundo de áudio da primeira frase e coloca-lo no buffer interno e em seguida continua sintetizando os fones seguintes e preenchendo o buffer interno, até que a frase termine. Enquanto isso, quando o objeto de áudio modifica seu buffer interno, a função de notificação *IAudioDestNotifySink::FreeSpace* é

chamada e portanto, a função *SendToAudio* é chamada para enviar mais dados para o áudio e assim sucessivamente até que seja encontrado em buffer um dado marcado com o fim de buffer e o objeto de áudio é fechado. Nota-se que a escrita e a leitura dos buffers de áudio interno e também de texto são feitas em concorrência. Portanto estes buffers foram implementados com exclusão mútua de memória.

Ao fim da síntese de cada frase e do envio dos fones para o buffer interno, a estrutura montada para a frase é destruída, outra frase é retirada do buffer de texto, sintetizada, e o processo se repete até que o buffer de texto esteja vazio e objeto *CTTS*, em *thread*, seja posto em suspenso.

O uso de *threads* acrescenta considerável complexidade ao software, mas se justifica não só pelo fato de deixar o objeto engine livre, mas principalmente por tornar possível que a síntese da frase seja assíncrona com o envio para o áudio, permitindo que o processo de síntese avance independente do fornecimento de mensagens de espaço disponível no áudio. No diagrama de tempos da figura 5.4., é demonstrado o envio de duas frases para a síntese, considerando que o tempo de síntese de uma frase seja igual a duração.

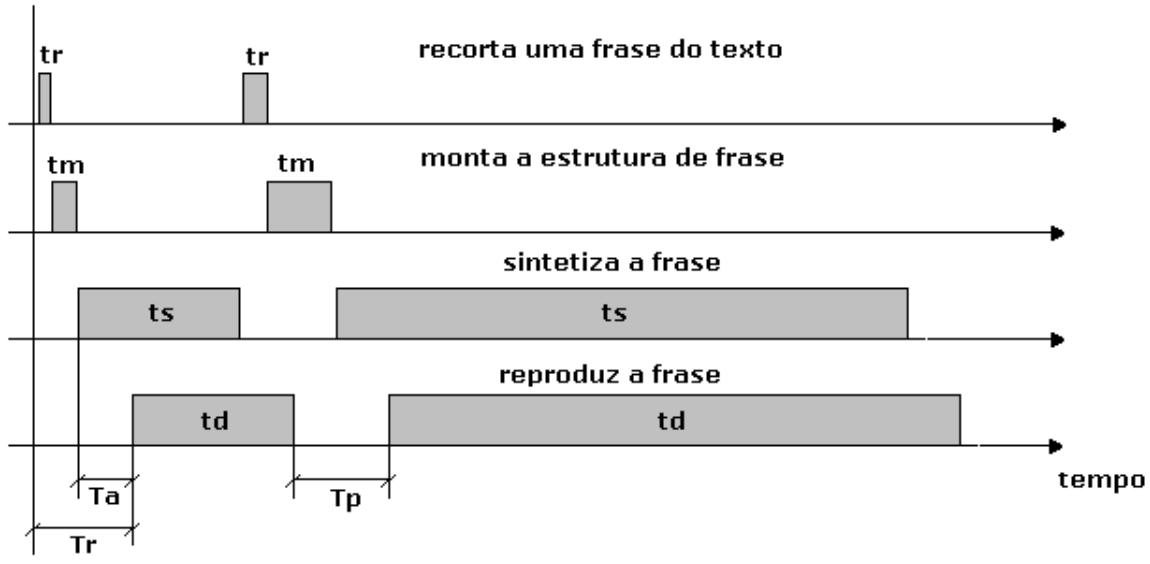


Figura 5.4.: diagrama de tempos supondo o tempo de síntese igual a duração da frase.

onde:	Tr:	tempo de resposta inicial;
	Ta:	tempo de síntese de 1 segundo de áudio;
	Tp:	tempo de pausa entre frases;
	td:	duração da frase;
	ts:	tempo de síntese da frase;
	tm:	tempo de montagem da estrutura e definição dos parâmetros;
	tr:	tempo de recorte da frase do texto;

A partir da análise do diagrama, chega-se a conclusão que a síntese da frase fone a fone, permite utilizar o tempo de fala de um fone para síntetizar o fone posterior, sendo: Tr = tr+tm+Ta, dado que Ta é quase constante e aproximadamente ts/td, **Tr = tr+tm+ts/td** e Tp = (tr+tm)–(td-ts). Daí podemos concluir que para não haver pausas entre frases, Tp ≤ 0, **tr+tm+ts ≤ td**, ou seja, basta que o tempo de síntese, recorte e montagem da estrutura seja menor ou igual a duração da frase, o que parece óbvio. Assim, o código foi otimizado até que esta condição fosse satisfeita. Consequentemente, o tempo de resposta será independente do tamanho da frase e dado pela expressão acima.

5.4. Conclusão

Neste capítulo foi implementada a estrutura de dados, interpretando o texto como um grupo objetos inter-relacionados e organizados conforme uma determinada hierarquia. A frase foi escolhida como unidade de processamento para minimizar o tempo de espera e para não haver pausas no meio de segmentos menores. Os objetos que estruturam a frase, palavras, tags, sílabas e fones, foram ordenados em listas encadeadas. Foi implementado a interpretação das tags inseridas no texto, que permite a modificação das características da síntese, como pronúncia de palavras, ênfase, mudança de velocidade, etc.. A utilização da estrutura de dados permite a inclusão posterior de processamento lingüístico, e ainda, que informações retiradas do texto possam ser utilizadas já na etapa de geração do sinal. O sistema de TTS foi implementado em uma Thread separada, com buffers internos de texto e de áudio e com permissão de acesso mútuo. Deste modo, foi possível: tornar o envio de texto da aplicação para o engine independente do processo de conversão, minimizar o tempo de resposta e extinguir as pausas de processamento entre frases.

Capítulo 6 – Implementação do sintetizador

6.1. Introdução

Após o envio da frase para *CSynthesi::SintetizaFrase*, este método irá percorrer a lista encadeada de fones e sintetizá-los separadamente através do método *CSynthesi::SintetizaFone,* onde estão contidos os algoritmos do modelo do síntetizador.

O modelo do sintetizador está descrito no diagrama da figura 6.1. e trata-se de um sintetizador de formantes. Para sons vozeados, o filtro de trato vocal é excitado por um sinal de forma de onda que modela o fluxo de ar através da glote a partir do período de pitch determinado pelo contorno do F0. Para sons não vozeados, é utilizado um gerador de ruído. Os fatores multiplicativos *Ar* e *Ag*, as formantes e as larguras de banda são parâmetros contidos nas frames, que serão concatenadas. O filtro de trato vocal simula as ressonâncias, indicadas pelas formantes e pelas larguras de banda e em seguida o sinal resultante passa por um filtro que simula a radiação produzida nos lábios.

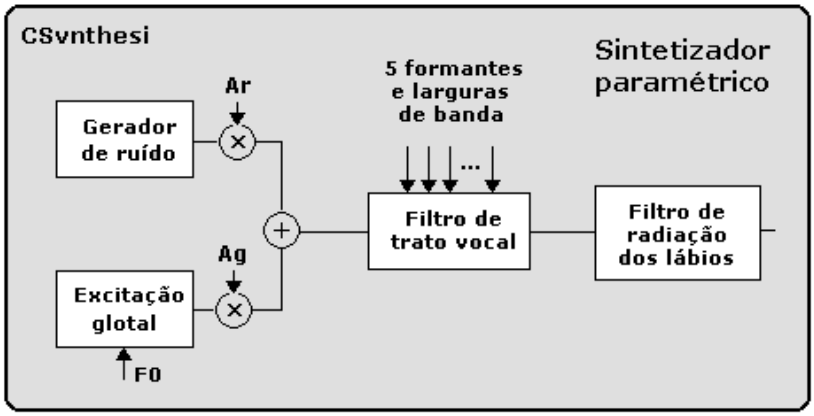


Figura 6.1.: diagrama em blocos do sintetizador.

Neste capítulo será descrita a implementação do modelo LF como excitação para sons vozeados, do modelo de Fujisaki para a geração do contorno do F0, do filtro de trato vocal, do filtro de radiação e do método utilizado para variação do pitch e duração de um fone.

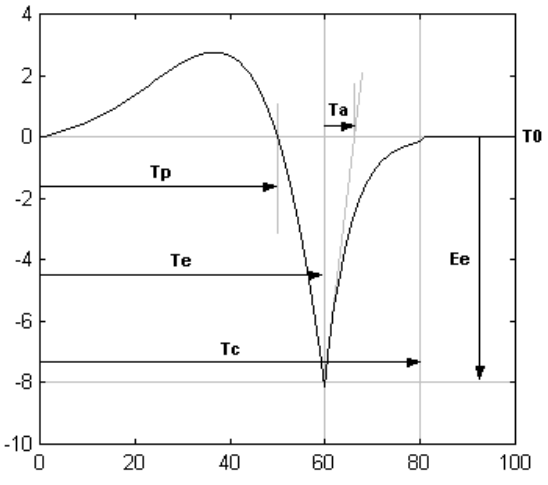
6.2. Excitação glotal

A forma de onda da excitação do filtro de trato vocal é determinada pelo modelamento do fluxo (ou velocidade de volume) do ar através da glote. O modelo poderia simular basicamente a ação da abertura e do fechamento da glote, sendo assim um trem de pulsos com periodicidade igual ao período de pitch. No entanto para uma boa qualidade da voz, é necessário um modelo mais completo. Foi escolhido o modelo LF (Liljencrants and Fant) por possibilitar a associação dos parâmetros do modelo com características físicas e portanto maior controle sobre a qualidade da voz [4].

6.2.1. Modelo LF

O modelo baseia-se na derivada da velocidade de volume glotal. Na figura 6.2., tem-se a forma do modelo em um ciclo glotal: o primeiro segmento equivale à abertura da glote, o ponto máximo de excitação do trato vocal e o fechamento; o segundo segmento equivale ao fluxo residual depois das cordas vocais se fecharem até o inicio de um novo ciclo. O modelo é descrito por:

$$u_g'(t) = \begin{cases} E_0 e^{\alpha t} \sin(\omega_g t) & se \quad 0 < t < T_e \\ -\frac{E_e}{\varepsilon T_a}\left[e^{-\varepsilon(t-T_e)} - e^{-\varepsilon(T_c-T_e)} \right] & se \quad T_e < t < T_c < T_0 \end{cases}$$



onde : $\omega_g = \frac{\pi}{T_p}$, $E_e = E_0 e^{\alpha T_e} \sin(\pi \frac{T_e}{T_p})$

e $\varepsilon T_a = 1 - e^{-\varepsilon(T_c - T_e)}$

De modo que o fluxo de ar na glote no final de um ciclo seja nulo, a área sob a curva deve ser nula, ou seja:

$$u_g(T_0) = \int_0^{T_0} u'_g(t)dt = 0$$

Figura 6.2.: modelo LF de um ciclo glotal.

Os parâmetros do modelo podem ser diretamente E_0, α, ω_g e ε, que não são óbvios, ou podem estar referenciados no tempo por T_e, T_p, T_a e T_c, conforme a figura, que são mais fáceis de manipulá-los e são:

T_a: constante de tempo da curva exponencial da segunda fase do ciclo glotal. Indica a velocidade de fechamento das cordas vocais e define o parâmetro ε.

T_p: instante de máximo fluxo glotal;

T_e: instante do máximo negativo do modelo;

T_c: instante em que se completa o ciclo;

Implementação: a implementação é baseada na rotina contida no toolbox, referenciado em [4], para Matlab. Inicialmente a rotina calcula os parâmetros das curvas, satisfazendo interativamente a condição imposta por T_e (área sob curva igual a zero). Em seguida, os parâmetros são aplicados à equação e obtém-se a forma de onda do modelo. Para teste e verificação, foi feito um aplicativo em Builder C++, mostrado na figura 6.3.

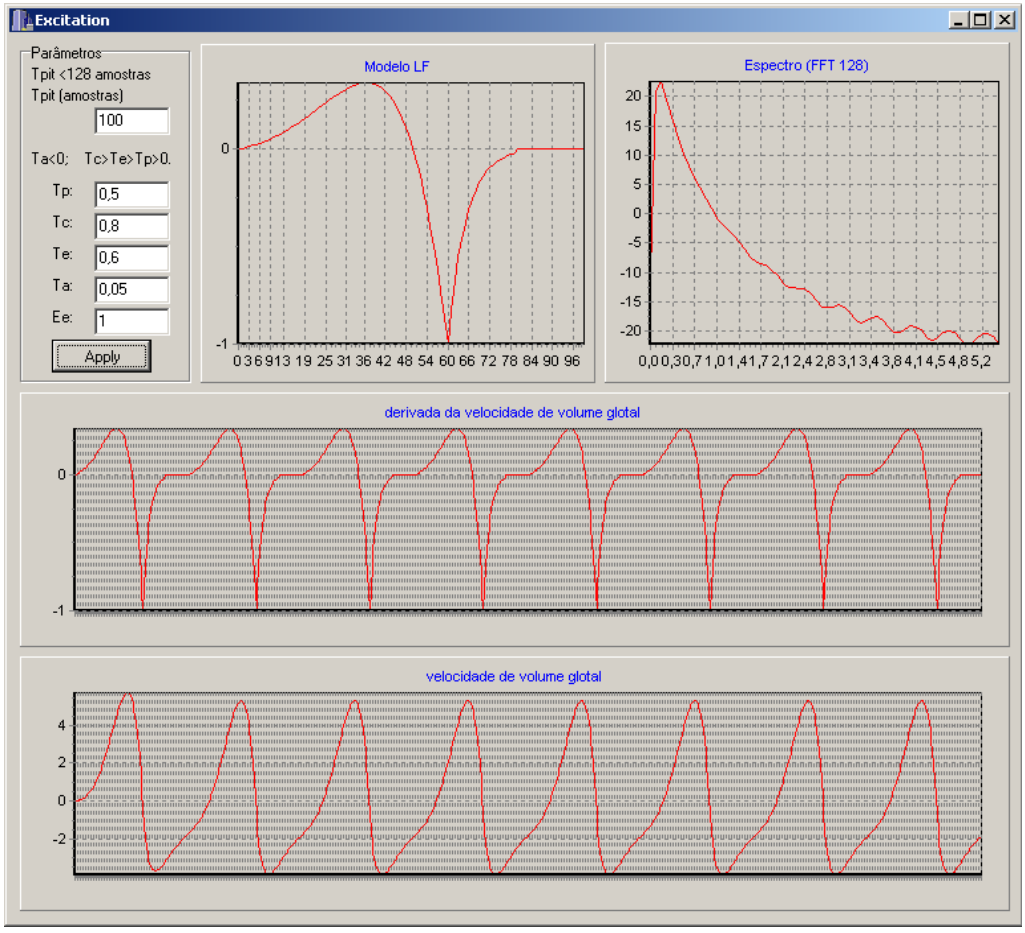


Figura 6.3.: aplicativo de demonstração da excitação vozeada.

No primeiro gráfico foi desenhado o modelo de um ciclo glotal, de acordo com os parâmetros temporais escolhidos. Ao lado, tem-se o módulo do espectro em frequência de um ciclo. O decaimento espectral médio da combinação dos dois segmentos deve ser de -12dB/oitava, mas pode variar de –6dB até –18dB de acordo com os parâmetros escolhidos [5]. A descontinuidade no instante T_c resulta nas ondulações de alta frequência no espectro. Nos 2 gráficos inferiores da figura, temos respectivamente a forma de onda de 8 ciclos concatenados, equivalente ao sinal contínuo da derivada da velocidade de volume glotal, e a integral deste sinal, que é a velocidade de volume, o sinal desejado para excitação do filtro.

Foi acrescentado ao pulso glotal um sinal de ruído de aspiração modelado conforme a figura 6.4. Os parâmetros do modelo são ajustados de modo a obter um tipo de voz mais “áspera” ou, com parâmetros mais suaves, mais natural. O modelo com amplitudes diferenciadas é explicado devido ao aumento do fluxo de ar quando a glote está totalmente aberta [4]. Portanto os parâmetros offset e dur podem ser ajustados de acordo com os parâmetros do modelo glotal.

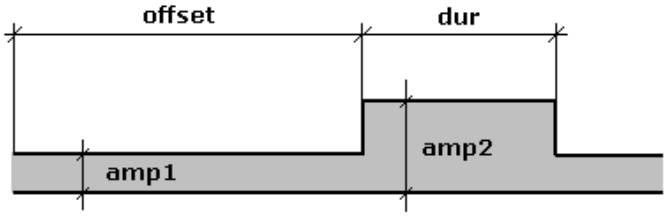


Figura 6.4.: modelo de ruído de aspiração.

6.2.2. Contorno de F0 - modelo Fujisaki

O modelo de Fujisaki para o contorno do F0 foi inicialmente proposto para o Japonês e vem sendo utilizado para outras línguas. Experimentalmente foi verificado que o modelo também pode ser aplicado ao português [11].

Resumidamente, o contorno de F0 de uma sentença, ou trecho contínuo de fala, pode ser estimado como a resposta do mecanismo de vibração das cordas vocais a um conjunto de comandos que carregam informações lingüisticas, não lingüisticas e/ou para-linguisticas [10]. O efeito da informação lingüística é a mais evidente e diz respeito ao acento léxico da palavra,

sintaxe e estrutura da sentença. Dois diferentes tipos de comandos têm sido necessários para a formação do contorno: um comando (comando de frase) parecido com impulso, para determinação de um padrão com subida e descida relativamente lenta ao longo do tempo, associado à estrutura sintática; e um outro tipo de comando (comando de acento), parecido com um degrau com largura limitada, para determinação do inicio e fim de um padrão com subida e descida relativamente rápida. Cada padrão de contorno acima pode ser aproximado pela resposta a um sistema linear de segunda ordem aos respectivos comandos [12]. Se o contorno do F0 for representado como o logaritmo da freqüência fundamental ao longo do tempo, então ele pode ser aproximado pela soma destes componentes, e expressado por:

$$\ln \quad F_0(t) = \ln \quad F_b + \sum_{i=1}^{I} Ap_i \, Gp(t - T_{0i}) + \sum_{j=1}^{J} Aa_j \{Ga(t - T_{1j}) - Ga(t - T_{2j})\}$$

onde:

$$Gp(t) = \begin{cases} \alpha^2 t \exp(-\alpha t) & se \quad t \geq 0 \\ 0 & se \quad t < 0 \end{cases} \quad \text{e} \quad Ga(t) = \begin{cases} \min\left[1 - (1 + \beta t)\exp(-\beta t), \gamma\right] & se \quad t \geq 0 \\ 0 & se \quad t < 0 \end{cases}$$

Os símbolos da equação indicam:

- F_b : valor base da freqüência fundamental na ausência de acentos;
- I : número de comandos de acento de frase;
- J : numero de comandos de acento;
- *Api* : amplitude do *ith* comando de frase;
- *Aaj* : amplitude do *ith* comando de acento;
- *T0i* : tempo do i*th* comando de frase;
- *T1j* : onset do j*th* comando de acento;
- *T2j* : fim do j*th* comando de acento;
- α : frequência angular natural do mecanismo de controle de frase para o i*th* comando de frase (nomeadamente 3/s);
- β : frequência angular natural do mecanismo de controle de acento para o i*th* comando de acento (nomeadamente 20/s);
- γ : parâmetro que indica o nível de arredondamento do componente de acento (geralmente igual a 0.9).

Na figura 6.5. tem-se o exemplo do contorno de F0, obtido com um comando de frase em T0 = 0 e três comandos de acentos posicionados no inicio, meio e fim da sentença.

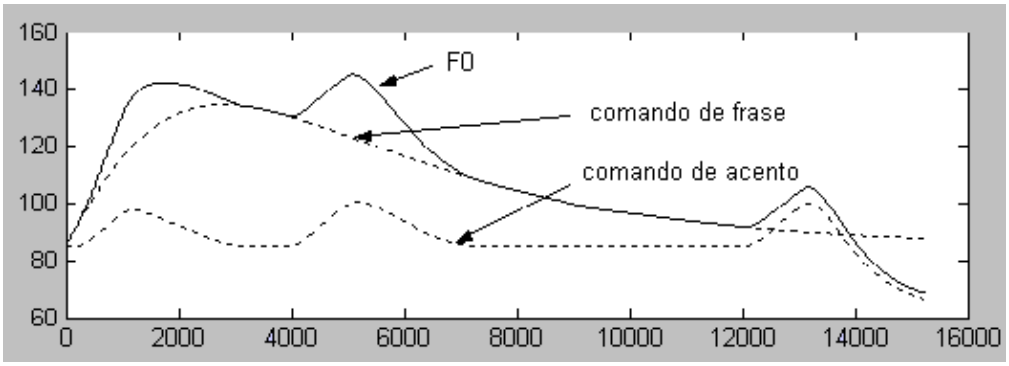


Figura 6.5.: exemplo de contorno de F0 gerado pelo modelo de Fujisack.

Os parâmetros para o modelo, são determinados na etapa de determinação dos parâmetros prosódicos da estrutura de frase, descrita anteriormente na estrutura do TTS. O parâmetro alfa foi modificado para ser proporcional à duração da frase e a posição do comando (T0) foi previamente fixado para uma frase declarativa. Para outro tipos de frases como interrogativas, exclamativas ou enumerativas, ou grupos prosódicos, é necessário o processo de análise por síntese, de modo a encontrar a posição e intensidade corretas dos comandos de frase. Em relação aos comandos de acento, a mesma análise é necessária, no entanto como neste projeto não se dispõe de um analisador sintático, os acentos de comando foram posicionados somente nas vogais tônicas da frase.

Na figura 6.6. está ilustrada, no primeiro gráfico, a sequência de fonemas da frase: “Uma frase declarativa”. No gráfico a seguir, são mostrados os comando de acento posicionados nas vogais tônicas. No gráfico abaixo, tem-se o contorno do F0 gerado pelo modelo conforme os comandos de acento e um comando de frase em T0 = 0; e finalmente, o sinal sintetizado, gerado conforme o contorno do F0.

Na produção das consoantes fricativas e outros sons não-vocalizados, o modelo considera que não há movimentação das cordas vocais e portanto o fluxo de ar é livre até o trato vocal. Este fluxo é modelado por um sinal de ruído que excitará o filtro de trato vocal.

A geração do sinal de ruído é feita a partir de números aleatórios com distribuição uniforme. O sinal de ruído passa por um filtro passa baixo de segunda ordem para manter o sinal de excitação com o mesmo decaimento espectral do impulso glotal [4] (-12dB/oitava).

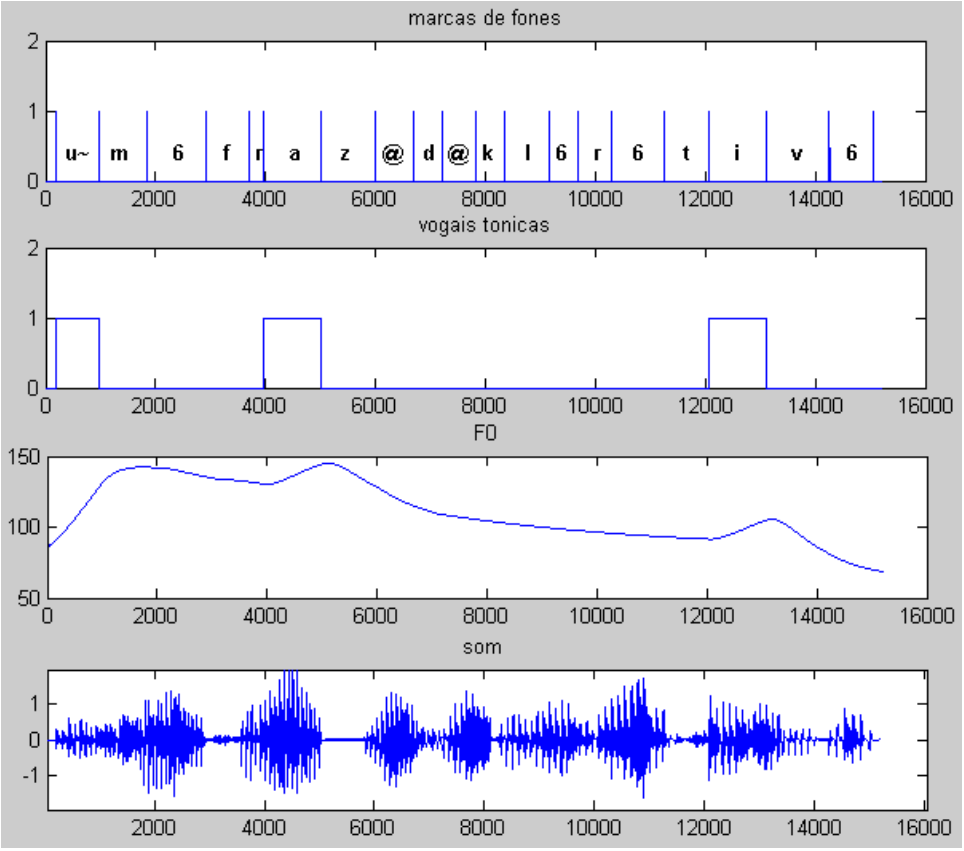


Figura 6.6.: marcas de fronteira entre fones; comandos de acento na vogal tônica; contorno de F0 e som sintetizado.

6.3. Filtro de trato vocal

Em comparação com o aparelho fonador humano, o filtro de trato vocal modela a cavidade bucal por um conjunto de tubos de seções diferenciadas. O filtro é excitado ora por ruído, ora por um sinal periódico, conforme descrito nos tópicos anteriores. O filtro é modelado por um IIR *all-pole*, cujos coeficientes podem ser coeficientes de predição linear (LPC), extraídos de segmentos de voz natural, no entanto não possuem significado físico direto. Outro método para a determinação dos coeficientes do filtro é através das formantes. Uma formante corresponde à uma ressonância do trato vocal, que pode ser produzida por um filtro com dois pólos complexos conjugados. Portando, cada som presente na fala possui suas formantes, bem

definidas no caso dos sons vocalizados, e não muito bem definidos para sons não vozeados. Por tratar-se de um filtro somente com pólos, a grande desvantagem deste modelo está na produção de sons não vocalizados ou nasais.

Conforme dito anteriormente, a síntese de um fone é feita por concatenação de frames de 10ms. Cada frame possui 5 formantes e mais uma formante é acrescentada em alta frequência para corrigir o decaimento espectral. Deste modo a implementação filtro pode ser interpretado como a associação de 6 filtros, cada um correspondente a um filtro de segunda ordem, cujos coeficientes são dados pela relação [6] :

$$H(z) = \frac{g}{a_0 + a_1 z^{-1} + a_2 z^{-2}}$$

onde: $a_0 = 1$;

$a_2 = \exp(-2\pi BT)$;

$a_1 = 2\sqrt{a_2}\cos(2\pi FT)$;

$g = a_2 - a_1$;

sendo B: largura de banda;

F: frequência central da formante;

T: 1/frequência de amostragem;

verificação: Na figura 6.7. tem-se um aplicativo feito em Builder C++ [9], que desenha a resposta em frequência do filtro gerado pela associação em cascata dos 6 filtros de segunda ordem, obtidos a partir das 6 formantes e larguras de banda indicadas, através da relação mostrada acima.

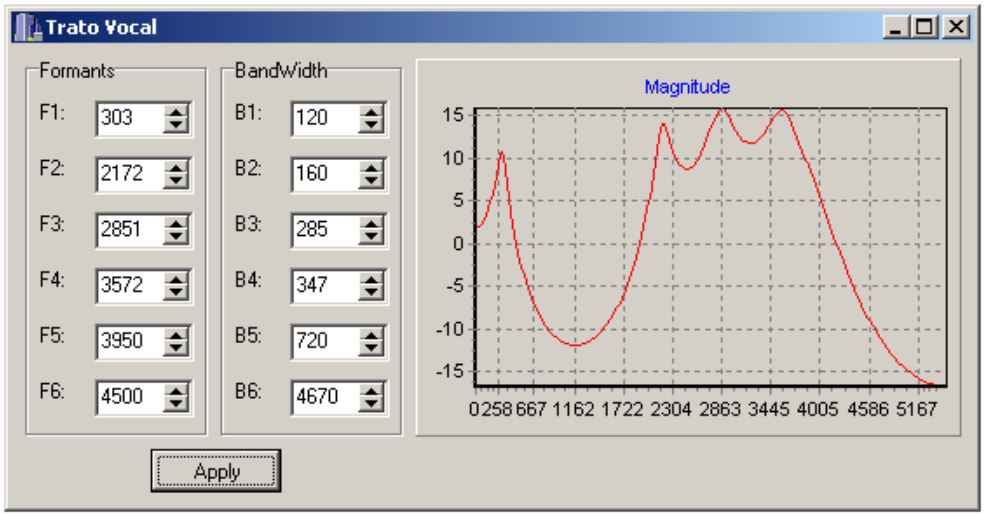


Figura 6.7.:aplicativo de demonstração do filtro de trato vocal.

6.4. Filtro de radiação dos lábios

O filtro de radiação dos lábios simula o efeito dos lábios. O modelo deste filtro, encontrado na literatura, é reduzido a um filtro de primeira ordem [3], que consiste em dar enfâse as altas frequências. Sua função de transferência é dada por:

$$H(z) = \frac{1}{1 - Rz^{-1}} \qquad \text{onde:} \quad R = 0.98;$$

Nomeadamente $R = 1$, no entanto o coeficiente de 0.98 mostrou-se mais adequado em respeito a qualidade final do som.

6.5. Síntese do fone

Um fone é composto por um conjunto de frames copiado dos difones. Nos frames estão contidos as formantes e larguras de banda para a determinação dos coeficientes do filtro de trato vocal, as amplitudes da excitação e um sinal vozeado/não vozeado. Inicialmente cada frame equivale a um segmento de sinal com a duração de 10ms, que será modificada de acordo com a duração e pitch desejada para o fone. Em um determinado instante de tempo, o pitch é dado pelas equações do modelo de contorno para o F0, que define a largura dos período glotais. Deste modo, o número de ciclos glotais contidos em um fone serão tantos quantos as durações permitirem e enquanto houver vozeamento.

A montagem da excitação suscita duas formas de implementação: vetores do tamanho do fone, ou vetores da dimensão do frame. Na primeira forma, a duração dos frames é ajustada uniformemente para atender a duração do fone; em seguida, é feito um vetor para as amplitudes do sinal glotal e do ruído, com a dimensão do fone, formados pela interpolação dos parâmetros de amplitude e vozeado/não vozeado do frame; depois são gerados: o vetor de ruído para excitação não vozeada e o vetor de impulsos glotais, por concatenação de janelas do modelo LF, de acordo com o período de pitch, atendendo o limite das durações; finalmente, os vetores de amplitude são respectivamente multiplicados pelos vetores de sinal e os resultados são somados, gerando um sinal único de excitação, com trechos vozeados e não vozeados. A geração dos ciclos glotais independentes do frame, obedecendo somente o

período de pitch e a duração do fone, resulta no aparecimento de transições do frames no meio de um período glotal. Por isso, os parâmetros do frame precisam ser finamente interpolados, que representa um peso computacional considerável. A vantagem deste método está na simplicidade do código.

No segundo método, é considerado que cada frame corresponde a um ciclo glotal e portanto, terá dimensão igual ao período de pitch correspondente àquele instante. No entanto para que acerto das durações seja atendido, alguns frames precisarão ser inseridos ou retirados. Os frames acrescentados ou retirados, são aqueles da região mais estável do fone. Desta forma, a síntese pode ocorrer frame a frame até o final do fone. Este método possui a vantagem de não haver transições no meio de um ciclo glotal, tanto em amplitude como em formantes ou larguras de banda, cuja interpolação de 50% da janela já ameniza os efeitos de transição entre os parâmetros do frame. Outra vantagem deste método está no reduzido espaço de memória utilizado, não precisando de alocações de vetores de parâmetros. O problema deste método é a manipulação de inserção ou retirada de frames.

Adotando o segundo método para implementação, primeiramente são calculados quantos frames precisam ser inseridos ou retirados para atender o pitch e a duração do fone. Em seguida, após a geração do sinal de excitação de um frame, este sinal passa pelo filtro de trato vocal, pelo filtro de radiação dos lábios e finalmente é normalizado para 16 bits para ser enviado ao buffer de áudio.

6.6. Conclusão

Neste capítulo foram descritos os blocos de um sintetizador por formantes, incluindo o gerador de excitação, o filtro de trato vocal e o filtro de radiação dos lábios. O modelo para a geração do sinal de excitação vozeada foi o modelo LF, onde o algoritmo foi demonstrado em uma interface gráfica. O algoritmo do filtro de trato vocal, que simula as ressonâncias das formantes, também foi demonstrado em uma interface gráfica. Foram descritas e implementadas as equações do modelo de Fujisaki para o contorno do F0. Em seguida foi discutido o método para implementação da síntese do fone, no qual a síntese frame a frame foi preferida pelo reduzido espaço de memória utilizado e pela ausência de transições bruscas entre frames.

Capítulo 7 – Conclusão e propostas

A arquitetura implementada apresenta compatibilidade com a Speech API 4.0, permitindo que o sistema TTS seja utilizado por qualquer componente ou aplicação compatível. A experiência adquirida na implementação de uma arquitetura por componentes representa o aprendizado de uma técnica de programação eficiente. O estudo da arquitetura definida pela Microsoft ressaltou a atenção para as responsabilidades que um sistema TTS precisa ter para ser aceito pelo usuário além de naturalidade e inteligibilidade, que é a capacidade de operar como um sistema que simule funcionamento em tempo real, com restrições a utilização do tempo do processador, espaço em memória e fundamentalmente tempo de síntese.

O sistema TTS a ser inserido na arquitetura foi implementado de modo a buscar solução, principalmente para a questão de desempenho, simulando a operação em tempo real através da utilização de *threads* e uso compartilhado de buffers. Foi também implementada uma estrutura de dados, buscando uma orientação por objetos, de modo a permitir organização e facilidade de acesso aos atributos do texto.

As etapas do sistema de conversão do texto em fala, foram implementadas parcialmente: a normalização do texto reconhece somente *tags*, pontuações e caracteres do alfabeto português; o algorítmo de conversão-grafema fonema, cedido pela Faculdade de Engenharia da Universidade do Porto é eficiente para a maioria dos casos, no entanto requer o acréscimo de exceções. O modelo de Fujisaki para o contorno do F0 acrescentou uma naturalidade fundamental à fala sintetizada, no entanto, requer maior diversidade de parâmetros para controle do modelo.

Em relação à etapa de síntese, o modelo de formantes é eficiente na síntese dos segmentos vozeados, no entanto não consegue modelar perfeitamente sons não vozeados ou nasais. A inclusão do modelo LF para excitação vozeada, aumenta a naturalidade dos sons vozeados, porém utiliza boa parte do tempo computacional disponível para a síntese. O banco de dados foi maximamente aproveitado e para o aperfeiçoamento da síntese, será preciso construir um novo banco de dados.

Finalmente, conclui-se que o objetivos de um sistema TTS em uma arquitetura compatível com sistemas comerciais atuais, foi alcançado. Ainda que a voz sintetizada seja pouco natural e inteligível, outros sistemas podem ser facilmente inseridos na arquitetura. Nota-se que nenhum teste de comparação do sistema TTS foi efetuado devido ao fato de que os objetivos deste projeto está em abordar os problemas e implementar uma plataforma moderna. Os algoritmos incluídos, assim como o sintetizador por formantes, serviram de protótipo para os próximos desenvolvimentos.

Em relação aos desenvolvimentos futuros, alguns deles já foram indicados, mas o principal deles é a busca de maior naturalidade da fala. Para isso propõe-se a inserção de outros modelos de síntese, ainda que paramétricos, devido a sua flexibilidade, e a implementação de softwares de desenvolvimento que permitam a montagem direta de banco de dados de outras vozes, como feminina e variantes características de região. Propõe-se ainda a diversificação de parâmetros para o modelo de intonação e a inserção de um modelo para as durações.

Para que seja possível a leitura de números, abreviaturas e outros símbolos, nos quais são ignorados neste projeto, o sistema desenvolvido precisa da inserção da etapa de normalização do texto e como apoio às etapas de processamento do texto, precisa ser inserido um analisador morfo-sintático. Neste campo de processamento do texto e determinação da prosódia, a sugestão secundária, porém não menos importante, é a participação direta de áreas afins da lingüistica e do processamento da linguagem natural.

Em relação às técnicas de programação, é sugerido que o sistema seja mais rigorosamente analisado, conforme uma técnica específica por exemplo UML *(Unified Model Language)*, permitindo melhor documentação, modularização, atualização, manutenção e outras vantagens que um sistema bem documentado e estruturado permite.

A Microsoft já apresentou a Speech API 5.0, incluída no Windows XP, que está construída em uma plataforma totalmente diferente, no entanto sugere-se que a plataforma 4.0 ainda não foi explorada suficientemente para que seja necessário a migração para outra versão. Ao invés, é preciso direcionar os esforços em aperfeiçoar o sistema TTS em sua plataforma atual [12].

Referências

[1] Syrdal, A. , R. Bennett e S. Greenspan, "Applied Speech Technology", CRC press, 1993.

[2] Oliveira, Luís Miguel V. V. C. de. "Síntese de fala a partir do texto", Universidade Técnica de Lisboa – Instituto Superior Técnico, Outubro de 1996.

[3] Teixeira, J.P.R., "Modelização paramétrica de sinais para aplicação em sistemas de conversão texto-fala," Tese de Mestrado, FEUP 1995.

[4] Childers, Donald G. "Speech processing an synthesi toolboxes," John Wilwy &Sons, inc. 1999.

[5] J. R. Deller, J. G. Proakis and J. H. Hansen, "Discrete-Time Processing of Speech Signal", Macmillan, 1993.

[6] Witten, I. H. "Principles of Computer Speech," Academic Press, inc. 1982.

[7] Petzold, Charles. "Programming Windows 95,"Microsoft Press, 1996.

[8] Richter, Jeffrey. "Advanced Windows," Microsoft Press, 1996.

[9] V. L. Latsch, F. G. Resende, Jr., and S. L. Netto, "An On-Line Laboratory Course on Speech Analysis", International Conference on Engineering and computer education, 1999.

[10] Sagisaka, Y. Et al 1997; "Computing Prosody", Spring New York, USA, ISBN 0-387-94804-X, ch 3, pp. 27-40.

[11] Freitas, D, D. Braga, M. J. Barros, V. Latsch, J. P. Teixeira, "Correlation between phonetic factors and linguistics events regarding a prosodic pattern of European Portuguese: A pratical proposal", ICSP2001, Korea. 2001.

[12] Latsch, V. and Barros Maria J., "Latest results in the development of the European Portuguese signal generation at FEUP/IPB: from formants to time frames and LPC", COST258 Maastricht meeting – The outlook for speech synthesis today. Novembro de 2001.

Internet:

/1/ Dr. GUI on Components, COM and ATL. http://msdn.Microsoft.com/library

/2/ Microsoft Speech API 4.0. www.microsoft.com/iit/onlinedocs.

Anexo A - Código Sampa para o Português Europeu

Consonants:

	Symbol	Word	Transcription		
plosives:	p	pai	paj		
	b	barco	"barku		
	t	tenho	"teJu		
	d	doce	"dos@		
	k	com	ko~		
	g	grande	"gr6nd@		
fricatives:	f	falo	"falu		
	v	verde	"verd@		
	s	céu	sEw		
	z	casa	"kaz6		
	S	chapéu	S6"pEw		
	Z	jóia	"ZOj6		
nasals:	m	mar	mar		
	n	nada	"nad6		
	J	vinho	"viJu		
liquids:	l	lanche	"l6nS@		
	L	trabalho	tr6"baLu		
	r	caro	"karu		
	R	rua	"Ru6		

Vowels and diphthongs

i	vinte	"vint@	/lápis	"lapiS
e	fazer	f6"zer		
E	belo	"bElu		
a	falo	"falu		
6	cama	"k6m6	/madeiram6"d6jr6	
O	ontem	"Ont6~j~		
o	lobo	"lobu		
u	jus	ZuS	/futuro	fu"turu
@	felizes	f@"liz@S		
i~	fim	fi~		
e~	emprego	e~"pregu (or em-)		
6~	irmã	ir"m6~		
o~	bom	bo~		
u~	um	u~		
aw	mau	maw	etc.: iw, ew, Ew, (ow)	
aj	mais	majS	etc.: ej, Ej, Oj, oj,	
6~j~	têm	t6~j~	etc.: e~j~, o~j~, u~j~	