
UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE CIÊNCIAS EXATAS E NATURAIS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Josué Leal Moura Dantas

# Correio Eletrônico com Processamento de Fala

Belém
2012

UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE CIÊNCIAS EXATAS E NATURAIS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Josué Leal Moura Dantas

# Correio Eletrônico com Processamento de Fala

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal do Pará como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação.

Belém
2012

**Dados Internacionais de Catalogação na Publicação (CIP)**

**Biblioteca Central da UFPa**

Dantas, Josué Leal Moura

Correio eletrônico com processamento de fala / (Josué Leal Moura Dantas); orientador, Aldebaro Barreto da Rocha Klautau Júnior. - 2012

78 f. il. 28 cm Dissertação (Mestrado) - Universidade Federal do Pará, Instituto de Ciências Exatas e Naturais, Programa de Pós-Graduação em Ciência da Computação, Belém, 2012.

1. Processamento de Sinais. I. Klautau Júnior, Aldebaro Barreto da Rocha, orient. II. Universidade Federal do Pará, Instituto de Ciências Exatas e Naturais, Programa de Pós-Graduação em Ciência da Computação. III. Título.

CDD - 22. ed. 621.3822

Josué Leal Moura Dantas

# Correio Eletrônico com Processamento de Fala

**Dissertação de Mestrado** apresentada para obtenção do grau de Mestre em Ciência da Computação. Programa de Pós-Graduação em Ciência da Computação (área de concentração: Sistemas Inteligentes). Instituto de Ciências Exatas e Naturais. Universidade Federal do Pará.

Data da Aprovação: Belém-PA, 05-07-2012

**Banca Examinadora:**

Prof. Dr. Aldebaro Barreto da Rocha Klautau Júnior
Programa de Pós-Graduação em Engenharia Elétrica - UFPA - Orientador

Prof. Dr. Nelson Cruz Sampaio Neto
Instituto de Ciências Exatas e Naturais - UFPA - Co-orientador

Prof. Dr. Jefferson Magalhães de Morais
Instituto de Ciências Exatas e Naturais - UFPA - Membro

Prof. Dr. Antônio Marcos de Lima Araújo
Instituto Federal do Pará - IFPA
Instituto de Estudos Superiores da Amazônia - Membro

UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE CIÊNCIAS EXATAS E NATURAIS - ICEN
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

# Correio Eletrônico com Processamento de Fala

Autor: Josué Leal Moura Dantas

DISSERTAÇÃO SUBMETIDA À AVALIAÇÃO DA BANCA EXAMINADORA APROVADA PELO COLEGIADO DO PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO DA UNIVERSIDADE FEDERAL DO PARÁ E JULGADA ADEQUADA PARA OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIA DA COMPUTAÇÃO COM ÊNFASE EM SISTEMAS INTELIGENTES.

APROVADA EM: 05/07/2012

**BANCA EXAMINADORA:**

---

Prof. Dr. Aldebaro Barreto da Rocha Klautau Júnior
(ORIENTADOR - UFPA)

---

Prof. Dr. Nelson Cruz Sampaio Neto (CO-ORIENTADOR - UFPA)

---

Prof. Dr. Jefferson Magalhães de Morais (MEMBRO - UFPA)

---

Prof. Dr. Antônio Marcos de Lima Araújo
(MEMBRO - IFPA/IESAM)

**Visto**

---

Prof. Dr. Sandro Ronaldo Bezerra Oliveira
(COORDENADOR DO PPGCC/ICEN - UFPA)

UFPA-ICEN-PPGCC
CAMPOS UNIVERSITÁRIO DO GUAMÁ
BELÉM-PARÁ-BRASIL
2012

*"Porventura pode uma mulher esquecer-se tanto de seu filho que cria, que não se compadeça dele, do filho do seu ventre? Mas ainda que esta se esquecesse dele, contudo eu não me esquecerei de ti. Eis que nas palmas das minhas mãos eu te gravei; os teus muros estão continuamente diante de mim".*
*(Isaías 49:15-16)*

# Agradecimentos

Agradeço acima de tudo ao meu Deus, que nunca me desamparou e certamente tem me concedido muito mais do que eu mereço, sem teu auxílio Pai não teria conseguido nada do que alcançei. A ELE seja toda a glória pelos séculos dos séculos!

Ao melhor orientador de mestrado que já tive, Aldebaro Klautau. Agradeço pela oportunidade que me deste, pela paciência e dedicação, enquanto eu viver serei grato pelo que fizeste. Pelos conselhos, aulas práticas de economia, enfim por seres meu pai em Belém.

Ao meus famíliares. Minha mãe, que sempre orou e também me pressionou a terminar meu mestrado, sei do seu amor por mim. Ao meu pai que hoje se encontra inválido por trabalhar no sustento da família, obrigado pai. Ao meu irmão que me ajudou em vários aspectos, mas sobretudo no financeiro me socorrendo quando sem dinheiro. Ao meu avô e avó, que possuem carinho todo especial por mim.

A minha namorada pela paciência e confiança nesse tempo em que precisei ficar em Belém.

A família lapsiana que muito me ajudou. Ao Nelson por me ajudar desde o inicio, quando mesmo sem me conhecer fez um projeto para que eu conseguisse uma bolsa. A Kelly pelo apoio em todos os aspectos. Rafael, por muito me auxliar em muitas de minhas tarefas, tu és meu brother. Hugo, por me ajudar com a JLaPSAPI. Bem como os demais do grupo FalaBrasil e irmãos do LaPS.

Aos meus amigos de Parauapebas, sobretudo da igreja Adventista do Bairro Maranhão. Ao Pablo Pietro, que gentilmente me acolheu em sua casa. Bem como outros colegas com quem dividi aluguel, como o Algean, Natan, Décio, Jean, e vizinhos como Nice, Tércio, Thiago, Renata, Arielle e Thaís.

Aos amigos da igreja Adventista do Marco, sobretudo o Sergio, Savio e Sunne, por me receberem em sua casa. A Dona Euda, Vivian, Paulo e Márcio Glack pelo grande ajuda em Belém.

Os professores Sandro Bezerra e Benedito Ferreira, coordenadores do PPGCC, que muito me auxiliaram nessa jornada. Ao professor Benedito por me conceder uma das primeiras bolsas, obrigado professor.

À CAPES, pelo apoio financeiro.

# Resumo

A interação homem-máquina tem mudado ao longo dos anos. Atualmente, a fala tem sido utilizada nesse processo, sobretudo, provendo acessibilidade a deficientes físicos. Nesse aspecto, já existem ferramentas que permitem o desenvolvimento de software livre com o módulo de reconhecimento e síntese de voz, entre outras línguas, para o português brasileiro. Para o reconhecimento foi usado neste trabalho o sistema Coruja, que suporta a especificação JSAPI. Já para síntese foi utilizado o FreeTTS com uma voz para o PB do projeto MBROLA. O decodificador utilizado foi o Julius, que é um software livre. Baseado nas ferramentas citadas, este trabalho consistiu no desenvolvimento de uma extensão com reconhecimento e síntese de voz que possibilita o controle do Thunderbird, um gerenciador de emails de código livre. Avaliou-se de forma qualitativa o desempenho da ferramenta nos dois modos de reconhecimento, gramática e ditado, dentro do contexto da aplicação alvo. Um aspecto importante, em relação à configuração de um arquivo de entrada do Julius, é o ajuste fino dos seus parâmetros, pois esses melhor ajustados permitem uma melhor taxa de reconhecimento e menor tempo de resposta do sistema. Com isso, o trabalho busca contribuir também com resultados acerca do ajuste do decodificador Julius para que a extensão desenvolvida tenha tempo de resposta e acurácia adequadas para a aplicação de correio eletrônico.

**Palavras-chave**: Reconhecimento automático de voz, Síntese de Voz, Acessibilidade, Ajuste fino, Thunderbird.

# Abstract

The interation between user and computer have change in last years. Nowadays, the speech have been used in this process, mainly, providing acessibility for handicappeds. In this aspect, already have tools that allow the developing of free software with mode of speech recognition and synthesis, between other languages, for brazilian portuguese. For the recognition was used the Coruja system, that support these JSAPI specification. Already for synthesis was used the FreeTTS with a voice for PB of MBROLA project. The decoder utilized was the Julius, this is a free software. Based in the cited tools, this work consisted in a developing of an extension with speech recognition and synthesis that allow a control of Thunderbird, the mails manager of free code. Evaluate qualitative form the performance of two ways of recognition, grammar and dictation, in context of target application. A important aspect, in relation the configuration of input file of Julius, is the pruning of parameters, because these better adjusted allow better recognition taxe and small system response time. Therefore, the work find contribute too with results about the adjustment of Julius decoder for that the builded extension have the response time and accuracy appropriate for email application.

**Keywords**: Automatic Speech Recognition, JLaPSAPI, Thunderbird, Julius.

# Sumário

# Lista de Figuras

# Lista de Tabelas

# Lista de Publicações

No decorrer do curso de mestrado foi elaborada a publicação em conferência abaixo.

**1-** Dantas, J.; Oliveira, R.; Santos H.; Neto, N.; Klautau, A. *Um Sistema para Melhorar a Usabilidade de um Gerenciador de Correio Eletrônico Baseado em Reconhecimento de Fala.* The 8th Brazilian Symposium on Information and Human Language Technology - STIL 2011, Mato Grosso - Brasil, 2011.

# Lista de Abreviaturas

**API** *Application programming interface*
**ASR** *Automatic speech recognition*
**BSD** *Berkeley Source Distribution*
**COM** *Component Object Model*
**HMM** *Hidden markov models*
**IBGE** Instituto Brasileiro de Geografia e Estatística
**IDE** *Integrated Development Environment*
**JNI** *Java Native Interface*
**JS** *Javascript*
**JSAPI** *Java Speech API*
**JSGF** *Java Speech Grammar Format*
**LaPS** Laboratório de Processamento de Sinais
**MA** Modelo Acústico
**MFCC** *Mel-frequency cepstral coefficients*
**ML** Modelo de Linguagem
**PB** Português Brasileiro
**PPGCC** Programa de Pós-Graduação em Ciência da Computação
**RAV** Reconhecimento Automático de Voz
**SAPI** *Speech API*
**SDK** *Speech Development Kit*
**TTS** *Text-to-Speech*
**UFPA** Universidade Federal do Pará
**WER** *Word Error Rate*
**WIP** *Word Insertion Penalty*
**xRT** *Real Time Factor*
**XUL** *XML User-Interface Language*

# Capítulo 1

# Introdução

Um importante passo para a melhoria na interação entre os usuários e os aplicativos do seu computador será a possibilidade de comunicação entre ambos. Para isso, as tecnologias relacionadas a voz deverão ser utilizadas. Nesse aspecto, dois importantes processos são necessários, primeiramente o processo pelo qual o computador interpreta o que o usuário fala, reconhecimento. O segundo, e igualmente importante, é a produção de fala pela máquina, síntese.

Com base neste contexto, este projeto propõe o desenvolvimento de um *software*, na forma de extensão, para utilização de um correio eletrônico (email) utilizando síntese e reconhecimento automático de voz (RAV). O desenvolvimento de tal aplicação se baseia na existência de ferramentas que possibilitam reconhecimento e síntese de voz para a língua alvo. O foco do trabalho será o português brasileiro (PB), mas poderá ser disponibilizado em outros idiomas, para isto, basta o uso de uma *engine* de voz para a língua alvo.

Para realizar o reconhecimento será utilizado o sistema Coruja. Segundo [Neto et al. 2010] , este sistema, já permite o reconhecimento de fala para o PB. Sendo este resultado da pesquisa que vem sendo desenvolvida no grupo FalaBrasil do LaPS (Laboratório de Processamento de Sinais) da Universidade Federal do Pará. Já para síntese será utilizado o FreeTTS criado pelo *Speech Integration Group* da *Sun Microsystems*, juntamente com o MBROLA que é um projeto da Faculdade Politécnica de Mons (Bélgica). Essas tecnologias serão encapsuladas dentro de uma extensão que será adicionada ao Thunderbird, da fundação Mozilla.

## 1.1 Contextualização e Terminologias do Trabalho

O envio e recebimento de mensagens por meio da internet é uma atividade muito utilizada na atualidade. Estima-se que em abril de 2010 haviam cerca de 2,9 bilhões de contas de e-mail [1]. Muitos usuários possuem mais de uma conta, por isso, fazem uso de gerenciadores. O Thunderbird destaca-se entre os existentes, por contar com cerca de 15 milhões de usuários [2], além de se enquadrar na categoria de software livre.

Levando em conta a informação acima, muitas empresas e indivíduos fazem uso dessa ferramenta para propagar seus serviços, idéias, entre outras finalidades. Isto implica no recebimento diário, por parte dos usuários, de uma quantidade relevante de mensagens. Estudos realizados por [Barley et al. 2010], concluíram que a grande quantidade de e-mails recebidos atualmente, sobretudo, por pessoas de negócio, trazem estresse para esses usuários, por demandar parte significante de seu tempo.

Outra problemática é a utilização do serviço de email por parte dos deficientes físicos. As últimas informações do IBGE (Instituto Brasileiro de Geografia e Estatística) sobre o número de deficientes físicos no Brasil, estima que cerca de 14,5% da população brasileira possui alguma deficiência.

As tecnologias assistivas, que auxiliam usuários com determinadas deficiências a realizarem tarefas usando o computador, atualmente, permitem o uso de um headphone (microfone e fone de ouvido) para comunicação do indivíduo usando voz, como o Skype, por exemplo. Porém, tal aplicação é usada de forma síncrona, ou seja, se dois usuários desejarem usar esse recurso devem se conectarem ao mesmo tempo. No entanto, o e-mail permite uma forma assíncrona de comunicação, permitindo assim a troca de informação, embora os usuários acessem seus e-mails em ocasiões diferentes ou não.

O reconhecimento e a síntese de voz são importantes tecnologias assistivas. Essas tecnologias, podem melhorar substancialmente a qualidade de vida de pessoas com necessidades especiais, tais como os deficientes auditivos, visuais, além de usuários que possuem deficiência em algum dos membros superiores, e por esse motivo são impossibilitados de utilizar o mouse ou o teclado.

O uso de tecnologias assistivas, tal como a proposta deste projeto, permitirá o acesso de deficientes físicos a informação, quer seja em lugares públicos, ou em seus computadores pessoais, através da manipulação de seus e-mails pelo Thunderbird.

Além de permitir a manipulação do TB por usuários cujas deficiências não sejam visuais e auditivas, o software também beneficiará os outros usuários, pois permitirá que

[1] http://www.email-marketing-reports.com/metrics/email-statistics.htm
[2] http://www.spreadthunderbird.com/content/thunderbird-3-faq

estes trabalhem em outras aplicações, mesmo ouvindo seus e-mail ou escrevendo-os, semelhante ao que muitos usuários costumam fazer ao ouvir música e manipular outros aplicativos. Os usuários poderão ainda realizar outras atividades que não fazem uso do computador. Além disso, o reconhecimento da voz, possivelmente permitirá a escrita de um e-mail mais rápido do que a digitação deste, tomando como premissa que a maioria dos usuários fala mais rápido, do que consegue digitar algo.

Assim, esta pesquisa visa analisar uma solução para as duas problemáticas supracitadas, pois além de permitir os usuários a realização de outras tarefas enquanto gerenciam seus e-mails, possibilitará ainda o acesso a essa importante ferramenta de troca de informações, por pessoas com diversas deficiências físicas.

## 1.2 Justificativa

O Thunderbird ainda não possui nenhuma extensão que permita a manipulação deste através de comando de voz. Já o Outlook da Microsoft pode ser manipulado através da fala. Dentre as funcionalidades suportadas, permite o acesso os menus, composição de emails, entre outras. Porém, entre as linguagens suportadas (no momento, chinês simplificado, chinês tradicional, inglês americano e japonês) não se encontra o PB. Nota-se que algumas das funcionalidades são permitidas somente para o inglês [Outlook 2011].

Desta forma, é necessário que a tecnologia de voz alcance também os softwares livres, tal como o Thunderbird, tendo em vista que o Outlook é uma solução comercial. Por se basear na gratuidade de distribuição é de se supor que este alcançará uma parcela significativa dos usuários, sobretudo, os que possuem baixo poder aquisitivo. Além disso, é importante a disponibilização dessas funcionalidades para o PB, que ainda não é suportado pelo Outlook.

## 1.3 Objetivos

### 1.3.1 Objetivo Geral

Usar as ferramentas que possibilitam o desenvolvimento de um sistema de computação para o Thunderbird, na forma de uma extensão (plugin), com uma interface via fala, que permita a interação do usuário com o referido gerenciador de e-mails.

### 1.3.2 Objetivos Específicos

- Estudar a API do Thunderbird e os requisitos do projeto Mozilla;
- Entender o Coruja e utilizá-lo no projeto.
- Adicionar um sintetizador a extensão a ser desenvolvida.
- Desenvolvimento do módulo para integração entre as tecnologias Mozilla e as utilizadas na interface via fala;
- Analisar a acurácia do reconhecedor e buscar parâmetros do Julius que otimizem o processo de reconhecimento;

### 1.3.3 Contribuições

O trabalho apresenta as seguintes contribuições:

- Construção de um conversor para que o Coruja, usando a JLaPSAPI, possa receber como entrada um arquivo no formato *Java Speech Format Grammar* (JSFG) e gere os arquivos necessários ao decodificador Julius.
- Utilização de uma base de áudio, LaPSMail, para o teste do decodificador, usando as sentenças que são mais utilizadas no processo de manipulação de um gerenciador de emails.
- Desenvolvimento da extensão que utiliza o Coruja para reconhecimento de fala, e o FreeTTS para a síntese.
- Ajuste dos parâmetros do decodificador, Julius, para melhorar o desempenho da aplicação.

## 1.4 Metodologia de Pesquisa

Primeiramente, foram escolhidas as interfaces (APIs) para a realização da síntese e do reconhecimento. Essas interfaces, são compatíveis com a *Java Speech API* (JSAPI). O Coruja, em sua versão atual, será usado no processo de reconhecimento e o FreeTTS para a síntese, sendo que o projeto MBROLA será utilizado juntamente com o FreeTTS, já que ele permite o uso de vozes para o PB.

Depois disso, foi necessário o conhecimento da codificação utilizada pelo Mozilla Thunderbird (TB). Além disso, foi necessário o conhecimento das especificações para a geração de uma extensão para o TB.

Um aspecto importante, realizado depois dos dois passos anteriores, é a integração entre o TB e o código Java que faz uso do reconhecedor e do sintetizador. Isso é possível

graças a utilização do Liveconnect, que em suma, permite a comunicação entre o código em Java e o Javascript, utilizado pelo TB.

Em última instância, os parâmetros do decodificador foram analisados, usando tentativa e erro. Assim, buscamos os valores que otimizassem o processo de reconhecimento para a extensão desenvolvida.

## 1.5 Organização do trabalho

Os próximos capítulos estão dispostos da seguinte forma:

**Capítulo 2**. Tecnologias de Voz. Neste capítulo, são apresentados os conceitos referentes aos processos de síntese e reconhecimento. Além de apresentar suas características e tipos.

**Capítulo 3**. Ferramentas utilizadas. O capítulo apresenta as ferramentas que foram necessárias para a construção de todo o projeto. Em relação às tecnologias Mozilla, há uma descrição da linguagem XUL (*XML User Language*) e do Javascript. Na área de voz são descritas as ferramentas para a síntese, que são basicamente o FreeTTS e o MBROLA, além do sistema Coruja para o reconhecimento. Por fim, há a descrição de como foi construído o conversor gramatical para a JLaPSAPI.

**Capítulo 4**. A Ferramenta Desenvolvida. Há uma descrição dos passos realizados para a construção da extensão em si, bem como a explanação de como os componentes estão estruturados.

**Capítulo 5**. Resultados Alcançados. Aqui é descrito o processo de ajuste dos parâmetros do decodificador Julius, bem como a especificação da base de dados utilizada. Por fim, são apresentados os resultados alcançados no processo de ajuste dos parâmetros.

**Capítulo 6**. Conclusão e Trabalhos Futuros. Por fim, é apresentado as conclusões sobre a pesquisa, e também quais as perspectivas em relação ao futuro do projeto.

# Capítulo 2

# Tecnologias de Voz

A forma de interação do usuário com o computador tem modificado ao longo dos anos. Atualmente, o desafio é a utilização da fala para intermediar esse processo. Nesse aspecto, o processo de síntese, produção do sinal de voz pela máquina, e o reconhecimento automático de voz (RAV), processo no qual o sinal de voz é interpretado pelo computador [Taylor 2009*a*], são essenciais. Essa forma de interação a alguns anos era vista como futurística [Canny 2006]. Porém, algumas ferramentas já foram desenvolvidas com uso dessa tecnologia [Colen & Batista 2010].

O guia da JSAPI [JSAPI 1998] ressalta que o reconhecimento juntamente com a síntese de voz pode melhorar a acessibilidade dos usuários. Um exemplo tem-se, por exemplo, ao modificar a fonte de um texto, assim o comando "Use 12 pontos, negrito, Arial", pode substituir vários cliques.

Nas próximas secções são apresentadas as características dos processos de Reconhecimento e de Síntese de Voz.

## 2.1 Reconhecimento automático de voz

Um sistema RAV típico adota uma estratégia estatística, baseada em modelos ocultos de Markov (HMMs) [Juang & Rabiner 1991], e é composto por cinco blocos: *front end*, dicionário fonético, modelo acústico, modelo de linguagem e decodificador, como indicado na Figura 2.1. As duas principais aplicações de RAV são comando e controle e ditado [Huang et al. 2001]. O primeiro é relativamente simples, pois o modelo de linguagem é composto por uma gramática que restringe as sequências de palavras aceitas. O último, tipicamente, suporta um vocabulário com mais de 60 mil palavras e exige mais processamento.

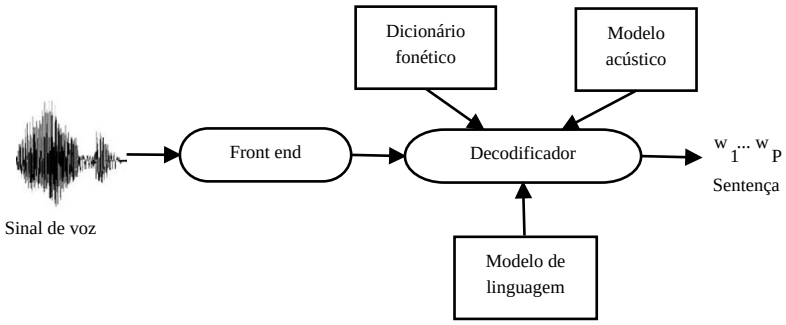


**Figura 2.1:** Principais blocos de um típico sistema RAV.

O processo de *front end* convencional extrai segmentos curtos (janelas, ou *frames*) de um sinal de voz e converte, a uma taxa de *frames* constante (tipicamente, 100 HZ), cada segmento para um vetor **x** de dimensão $L$ (tipicamente, $L = 39$). Assumimos aqui que $T$ *frames* são organizados em uma matriz $\mathbf{X}$ de dimensão $L \times T$, que representa uma sentença completa. Existem várias alternativas no que diz respeito à parametrização do sinal de voz. Apesar da análise dos coeficientes cepstrais de frequência da escala Mel (MFCCs) ser relativamente antiga [Davis & Merlmestein 1980], essa provou ser efetiva e é geralmente usada como entrada para os blocos de *back end* do sistema RAV [Huang et al. 2001].

O modelo de linguagem de um sistema de ditado, fornece a probabilidade $p(\mathcal{T})$ de observar a sentença $\mathcal{T} = [w_1, \ldots, w_P]$ de $P$ palavras. Conceitualmente, o decodificador tem como objetivo achar as sentenças $\mathcal{T}^*$ que maximizam a probabilidade *a posterior* dada por (segundo [Rabiner & Juang 1993])

$$\mathcal{T}^* = \arg\max_{\mathcal{T}} p(\mathcal{T}|\mathbf{X}) = \arg\max_{\mathcal{T}} \frac{p(\mathbf{X}|\mathcal{T})p(\mathcal{T})}{p(\mathbf{X})}, \tag{2.1}$$

onde $p(\mathbf{X}|\mathcal{T})$ é dada pelo modelo acústico. Como $p(\mathbf{X})$ não depende de $\mathcal{T}$, a equação anterior é equivalente a

$$\mathcal{T}^* = \arg\max_{\mathcal{T}} p(\mathbf{X}|\mathcal{T})p(\mathcal{T}). \tag{2.2}$$

Na prática, uma constante empírica é usada para ponderar a probabilidade do modelo de linguagem $p(\mathcal{T})$ antes da mesma ser combinada com a probabilidade dos modelos acústicos $p(\mathbf{X}|\mathcal{T})$.

Dado o grande volume de sentenças concorrentes (hipóteses), a Equação 2.2 não pode ser calculada independentemente para cada hipótese. Portanto, os sistemas RAV usam estruturas de dados como árvores léxicas e são hierárquicos, usando o artifício de separar

as sentenças em palavras, e as palavras em unidades básicas, que aqui chamaremos de fones [Huang et al. 2001]. A busca por $\mathcal{T}^*$ é chamada decodificação e, na maioria dos casos, hipóteses são descartadas ou podadas (*pruning*). Em outras palavras, para tornar viável a busca pela "melhor" sentença, algumas candidatas são descartadas e a Equação (2.2) não é calculada para elas [Deshmukh et al. 1999, Jevtić et al. 2001].

Um dicionário fonético (conhecido também como modelo léxico) faz o mapeamento das palavras em unidades básicas (fones) e vice-versa. Para uma melhor performance, HMMs contínuas são adotadas, onde a distribuição de saída de cada estado é modelada por uma mistura de Gaussianas, como mostrado na Figura 2.2. A topologia típica de uma HMM é a *left-to-right*, onde as únicas transições permitidas são de um estado para ele mesmo ou para o estado seguinte.

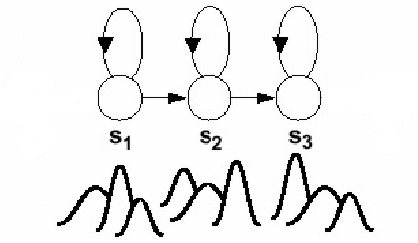


**Figura 2.2:** Representação gráfica de uma HMM contínua *left-to-right* com três estados e uma mistura de três Gaussianas por estado.

Dois problemas clássicos com relação à modelagem acústica são: a inconstância dos fones devido à co-articulação e à insuficiência de dados para estimar os modelos. O método de compartilhamento de parâmetros (*sharing*) visa combater esse último problema, melhorando a robustez dos modelos. Em muitos sistemas, o compartilhamento é implementado no nível de estado, ou seja, o mesmo estado pode ser compartilhado por HMMs diferentes.

O modelo acústico pode conter uma HMM por fone. Isso seria bastante razoável caso um fone pudesse ser seguido por qualquer outro, o que não é verdade, já que os articuladores do trato vocal não se movem de uma posição para outra imediatamente na maioria das transições de fones. Nesse sentido, durante o processo de criação de sistemas que modelam a fala fluente, busca-se um meio de modelar os efeitos contextuais causados pelas diferentes maneiras que alguns fones podem ser pronunciados em sequência [Ladefoged 2001]. A solução encontrada é o uso de HMMs dependentes de contexto, que modelam o fato de um fone sofrer influência dos fones vizinhos. Por exemplo, supondo a notação do trifone *a-b+c*, temos que *b* representa o fone central ocorrendo após o fone *a* e antes do fone *c*.

Segundo [Huang et al. 2001], existem basicamente dois tipos de modelos trifones: *internal-word* e *cross-word*. As diferenças entre os mesmos é que no caso do *internal-word* as co-articulações que extrapolam as durações das palavras não são consideradas, sendo assim, menos modelos são necessários. Já no caso do *cross-word*, que considera a co-articulação entre o final de uma palavra e o início da seguinte, a modelagem é mais precisa, porém o número de modelos trifones gerados cresce muito, o que dificulta o trabalho do decodificador e gera uma necessidade de mais dados para treino. Alguns exemplos de transcrição podem ser conferidos na Tabela 2.1.

| | |
|---|---|
| Sentença | arroz com bife |
| Monofones | sil a R o s k o∼ b i f i sil |
| *Internal-Word* | sil a+R a-R+o R-o+s o-s k+o∼ k-o∼ b+i b-i+f i-f+i f-i sil |
| *Cross-Word* | sil sil-a+R a-R+o R-o+s o-s+k s-k+o∼ k-o∼+b o∼-b+i b-i+f i-f+i f-i+sil sil |

**Tabela 2.1:** Exemplos de transcrições com modelos independentes e dependentes de contexto.

A escassez de dados de treino também afeta a modelagem da língua que estima (segundo [Rabiner & Juang 1993])

$$P(\mathcal{T}) = P(w_1, w_2, \ldots, w_P) \tag{2.3}$$

$$= P(w_1)P(w_2|w_1)\ldots P(w_P|w_1, w_2, \ldots, w_{P-1}). \tag{2.4}$$

É impraticável estimar a probabilidade condicional$P(w_i|w_1, \ldots, w_{i-1})$, mesmo para valores moderados de $i$. Assim, o modelo de linguagem para sistemas RAV consiste de um modelo $n$-gram, que assume que a probabilidade $P(w_i|w_1, \ldots, w_{i-1})$ depende somente das $n-1$ palavras anteriores. Por exemplo, a probabilidade $P(w_i|w_{i-2}, w_{i-1})$ expressa um modelo de linguagem trigrama.

Resumindo, após o treinamento de todos os modelos estatísticos, um sistema RAV na etapa de teste usa o *front-end* para converter o sinal de entrada em parâmetros e o decodificador para encontrar a melhor sentença $\mathcal{T}$.

Os modelos acústicos e de linguagem podem ser fixos durante a fase de teste, porém adaptá-los pode gerar uma melhor performance. Por exemplo, o domínio de aplicação pode ser estimado, e um modelo de linguagem específico usado. Isso é crucial para aplicações com vocabulário técnico, como relatórios médicos de raios X [Antoniol et al. 1993]. A adaptação do modelo acústico possui igual importância [Lee & Gauvain 1993], este pode ser adaptado, por exemplo, a um locutor, ou ao sotaque de uma determinada região.

Sistemas RAV que usam modelos independentes de locutor são convenientes, porém devem ser robustos o suficiente para reconhecer, com boa performance, áudio de qualquer locutor. Com o custo de exigir que o usuário leia algumas sentenças, técnicas de adaptação ao locutor podem melhorar os modelos HMM para um locutor específico. Técnicas de adaptação podem também ser usadas para compensar variações no ambiente acústico, reduzindo o descasamento causado pelo canal ou efeitos de ruído aditivo.

Neste trabalho, nos referimos a um motor ou *engine* RAV, como um decodificador e todos os recursos necessários para sua execução (modelos de linguagem e acústico, etc.).

## 2.2 Medidas de Erro para Reconhecimento de Voz

A avaliação do sistema desenvolvido é fundamental, sobretudo para que se possa dimensionar o impacto de determinada técnica sobre o reconhecimento. Duas bases devem ser usadas. A primeira para o treinamento do modelo e melhoria deste. A segunda é a base de teste sobre a qual será feita a avaliação do sistema adotando determinada técnica. Os erros que podem acontecer em um sistema de reconhecimento são os seguintes:

- Substituição: uma palavra errada é trocado pela correta.
- Inserção: uma palavra extra é adicionada ao resultado do reconhecimento.
- Omissão: uma palavra falada é omitida do resultado.

A taxa de erro por palavra (WER), em percentagem, é dada pela seguinte expressão:

$$WER = \frac{100\%(S+O+I)}{N} \tag{2.5}$$

onde N é o número de palavras da sentença correta, S, O e I são o número de erros de substituição, omissão e inserção na sentença reconhecida,

Quanto menor a taxa de erro por palavra maior acurácia possui o sistema.

O Algoritmo que detecta as palavras erradas deve procurar à máxima sub sequência de palavras que casa com a sentença alvo.

## 2.3 Sistemas *Text-To-Speech*

Um sistema TTS realiza basicamente o trabalho inverso de um reconhecedor de voz, sendo capaz de gerar uma voz semelhante a humana a partir de texto. Há técnicas que sintetizam voz, mas diferem dos sistemas TTS.

Conforme [Dutoit 2001], existem sistemas que geram fala artificial através da concatenação de palavras isoladas, ou sentenças, esses sistemas não podem ser confundidos com os que realizam TTS. Os outros sistemas são limitados no sentido de sintetizar palavras somente usando o vocabulário já armazenado. Já os sistemas TTS possibilitam a construção de sentenças que não foram definidas de antemão, o que permite a geração de um maior número de sentenças distintas.

A principal desvantagem de um sistema TTS é a qualidade da voz. Ainda que passaram por melhorias, ainda é possível identificar uma voz sintetizada. As pesquisas têm evoluído no sentido de produzir uma voz mais natural possível. Dois aspectos analisados, no meio acadêmico, são a **inteligibilidade** e a **naturalidade** [Taylor 2009*b*]. Essas características são analisadas nas vozes produzidas para que se possa avaliar a qualidade destas.

Como a escrita não permite a expressão de sentimentos de uma maneira integral, é de se esperar que os sistemas TTS possuam uma deficiência na transmissão de emoções. O ramo que busca incrementar emoções a fala é conhecido como síntese de voz emotiva ou emocional [Taylor 2009*b*]. Esta área é mais recente, tendo seus primeiros trabalhos publicados no final da década de 90 [Cahn & Cahn n.d., Murray 1989, Murray & Arnott 1995]. Estas técnicas foram aplicadas em sintetizadores de formantes e a Fonix utilizou comercialmente em seu sintetizador, o DECTalk [DECtalk 2005].

### 2.3.1 Arquitetura

A arquitetura mais comum dos sistemas TTS é a composta por dois elementos: o módulo de análise de texto e o motor de síntese [van Santen et al. 1996]. Um sistema TTS genérico é apresentado na Figura 2.3. No diagrama, existe um módulo de Processamento de Linguagem Natural (PLN) que produz a transcrição fonética do texto de entrada, bem como a entonação e o ritmo desejados. Já no módulo de Processamento Digital de Sinais (PDS) produz o sinal da fala de acordo com as informações recebidas pelo módulo de PLN. PLN é o módulo de análise do texto e o PDS é o motor de síntese, desta forma que eles serão identificados no decorrer deste trabalho.

### 2.3.2 Análise do texto

O *front end*, que é o analisador de texto, converte números, símbolos, abreviações e silgas, em formas correspondentes por extenso. Esta etapa também pode ser conhecida como normalização de texto ou pré-processamento de texto [Holmes & Holmes 2001].

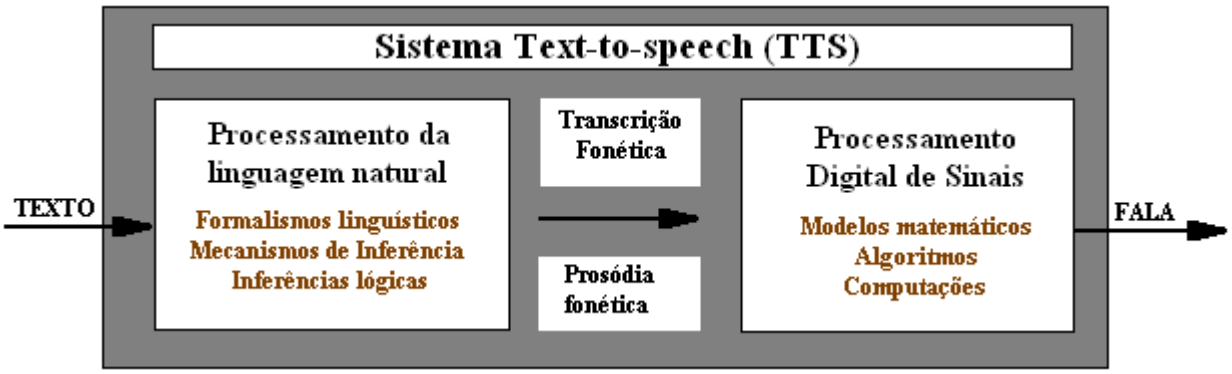


**Figura 2.3:** Diagrama funcional simples de um sistema TTS.

A análise texto é importante porque os módulos predecessores interpretam número reduzido de caracteres, em geral reduzidos ao alfabeto da língua. Por exemplo, suponha que se deseja sintetizar o seguinte texto 2.6:

Segundo o IBGE, 14,5% da população possui alguma deficiência (2.6)

Esse texto, apesar de simples, não pode ser utilizado diretamente em um sistema TTS. Os numerais, a pontuação entre os números e o símbolo % devem sofrer transformações. Além disso, existem diferentes maneiras de leitura do texto. Em nosso caso, o acrônimo *IBGE* pode ser lido propriamente como *ibge* ou ainda como *instituto brasileiro de geografia e estatística*. Esses aspectos são definidos pelo módulo de análise de texto. Uma forma normalizada do texto 2.6 seria igual ao exemplificado no texto 2.7:

segundo o ibge quatorze vírgula cinco por cento da população possui alguma deficiência (2.7)

Tarefas típicas são a conversão de todas as letras de caixa alta para baixa, remoção de sinais de pontuação, expansão de abreviações e tradução de símbolos. O armazenamento do texto original é importante, pois, porventura, etapas posteriores podem necessitar de informações que foram descaracterizadas no módulo de análise do texto. Por exemplo, a identificação de uma determinada sigla, que foi transformada em caixa baixa, além da possibilidade de escrita por extenso, dificultaria a busca de uma sigla.

O etiquetamento gramatical, também conhecido como analisador morfossintático, pode ser incluído nos módulos de análise de texto. O processo objetiva a extração de informações morfossintática do texto original, que podem ser utilizadas para fazer a desambiguação de palavras com pronúncia igual e grafia diferente, homófomas, e aquelas

com pronúncia diferente e grafia igual, heterófonas. O etiquetamento gramatical é conhecido na academia como *POS tagger*[1].

Outra tarefa do *front end* consiste em três partes: associar transcrições fonéticas a cada palavra, divisão das palavras em sílabas e determinar as sílabas tônicas. Respectivamente, esses processos são conhecidos como conversão grafema-fonema (G2P), silabação e marcação da sílaba tônica. Reunidos, eles fornecem a representação linguística interna do sistema.

### 2.3.3 Motor de síntese

A voz sintetizada é produzida pelo motor de síntese. As entradas deste módulo consistem das saídas do módulo de análise de texto. O motor de síntese é também conhecido como *back end*.

Há várias formas utilizadas para produção de voz, dentre elas, destacam-se três [Huang et al. 2001]:

- **Síntese articulatória**: baseia-se em regras que parametrizam o aparelho fonador e o processo de articulação [Holmes & Holmes 2001] - velocidade e pressão do ar, abertura da boca, dentre outras características. Neste método, os parâmetros são modificados de forma a representar os fonemas.
- **Síntese por formantes**: produz voz fazendo uma fonte gerar formas de ondas periódicas que, em seguida, são filtradas e ressoadas por vários módulos em cascata ou paralelo. Na prática, esse paradigma realiza o controle do *pitch*, que é a percepção da frequência fundamental dos sons, e outras frequências. É conhecido também como fonte-filtro. O sintetizador por formantes mais conhecido é o de Klatt [Klatt 1980], inclusive utilizado no DECTalk. O ponto fraco deste modelo é a baixa naturalidade, já que é difícil capturar todas as sutilezas envolvidas no processo, tornando com a voz fique mecanizada.
- **Síntese concatenativa**: atualmente, é a forma de síntese que atingiu maior apelo comercial. Consiste, basicamente, na construção de uma base de dados, geralmente extensa, com vários segmentos de voz, que são rotulados e classificados para que sejam concatenados e reproduzam um novo áudio. O problema concentra-se, sobretudo, no tamanho da base a ser armazenada. Outro problema é a falta de flexibilidade, fazendo necessária a aplicação de algumas técnicas para a modificação da prosódia [Huang et al. 2001].

[1] http://nlp.stanford.edu/software/tagger.shtml

Atualmente, dos métodos citados acima, somente a síntese concatenativa permanece como estado da arte. Entretanto, uma nova abordagem tem surgido, sendo esta a síntese baseada em HMMs. A técnica não chega a ser tão nova, porém, sua difusão acontece somente no fim da década de 90, com trabalhos de pesquisadores japoneses [Masuko et al. 1996, Tokuda, Masuko, Yamada, Kobayashi & Imai 1995, Tokuda, Kobayashi & Imai 1995, Kobayashi et al. 1999].

# Capítulo 3

# Ferramentas Utilizadas

## 3.1 Framework Mozilla

A fundação Mozilla, organização sem fins lucrativos, criada em Março de 1998, destaca-se no mercado mundial pelo desenvolvimento de aplicativos usados por milhões de usuários pelo mundo, tais como o browser Firefox e o gerenciador de emails Thunderbird, entre outros.

Os aplicativos Mozilla possuem a vantagem de execução em várias plataformas. Isto ocorre porque existe uma camada de interpretação entre a aplicação e o sistema operacional [Boswell et al. 2002].

Duas tecnologias principais são utilizadas para o desenvolvimento de tais aplicativos: sendo estes o XML User-Interface Language (XUL) e o JavaScript.

Além de serem usadas para o desenvolvimento dos aplicativos, essas tecnologias também são utilizadas para a construção de extensões. Uma extensão adiciona uma nova funcionalidade aos aplicativos Mozilla, tais como Firefox, SeaMonkey e Thunderbird. Esta pode ser usada para customizar a aplicação conforme as necessidades dos usuários.

Além da construção de extensões, o Mozilla também permite o desenvolvimento de plugins. A diferença entre estes e as extensões é sua característica de ajudar as aplicações a exibirem conteúdos específicos, tais como arquivos multimídia.

O XUL permite a criação de interfaces para múltiplas plataformas, assim sua principal característica é a portabilidade. Além disso, os aplicativos podem ser executados online ou offline.

Para renderizar os objetos gráficos visíveis aos usuários, a Mozilla usa o Gecko, que é um motor de layout. O Gecko interpreta os códigos inscritos em HTML, CSS, XUL ou

Javascript e exibe os elementos especificados, resultando na interface da aplicação [1].

A utilização do XUL com o Gecko apenas apresenta os elementos gráficos, entretanto, estes componentes não possuem funcionalidades. As funções podem ser adicionadas através de códigos em Javascript. Assim, essas tecnologias colaboram entre si no desenvolvimento de ferramentas mais completas.

## 3.2 XUL - XML User Language

O arquivo XUL é um simples arquivo de texto que contém uma sintaxe XML e possui extensão .xul. As aplicações Mozilla interpretam o texto desenhando os elementos na tela de acordo com a especificação do arquivo. Tags (rótulos) de outras linguagens também podem ser incluídas nos arquivos .xul, sendo elas HTML e XML.

Certa ordem deve ser seguida para que se tenha um arquivo XUL válido. A primeira linha requerida é a seguinte declaração XML:

<?xml version="1.0"?>

Caso o desenvolvedor deseje adicionar algum comentário ao código, sendo que este deve estar circundado por <!– e – >, o poderá fazer somente depois da declaração da linha especificada acima.

Todo rótulo que for aberto deve ser fechado com a barra. Alguns rótulos podem ser fechados ao fim da especificação do elemento, como exemplificado a seguir:

<label value="Getting Started"/>

A linguagem XUL é *case sensitive*. Assim, se o rótulo <window> foi aberto e este for fechado com algum caractere com capitulação diferente, como por exemplo, </Window> a renderização não ocorrerá.

## 3.3 JavaScript

Javascript é uma linguagem de script orientada a objetos. Sua principal característica é ser multi plataforma. Ela é usada juntamente com outras tecnologias ou aplicativos, já que esta não foi projetada para ser usada sozinha. O Javascript é embarcada facilmente a outras linguagens, sendo este o objetivo de criação da linguagem.

Um determinado objeto pode conectar-se ao Javascript para adicionar funcionalidades a este. Para isto o Javascript contém um conjunto de elementos como: operadores, estruturas de controle e expressões. Contém ainda um conjunto de objetos tais como Array,

[1] https://developer.mozilla.org/en/Gecko

Date e Math.

Através das funcionalidades do Liveconnect, o Javascript pode se comunicar com o Java e vice versa. Na próxima seção veremos as características do Liveconnect.

## 3.4 Liveconnect

Liveconnect é o nome da API que permite a chamada de métodos escritos em Java através de código em Javascript e vice versa. Esse processo utiliza um objeto *wrapper*, que nada mais é do que um objeto com o tipo de dados da linguagem alvo que inclui um objeto da linguagem fonte. Estes objetos são usados pelo Javascript para acessar métodos e campos dos objetos Java. Assim, a chamada a um método ou acesso a um campo em um objeto *wrapper* corresponde a uma chamada em um objeto Java. No lado Java, os objetos Javascript são encapsulados em uma instância da classe netscape.javascript.JSObject e então passado ao Java.

Ao ser enviado um objeto Javascript para o Java, em tempo de execução, é criado um objeto *wrapper* Java do tipo JSObject. Quando ocorre o processo inverso, o Java envia um objeto JSObject para o Javascript, que em tempo de execução é transformado em um objeto Javascript. A classe JSObject provê uma interface para chamar os métodos e examinar as propriedades do JavaScript.

Existem tipos de objetos específicos do *Liveconnect* para a manipulação de objetos ou vetor Java, ou para referenciar classes ou pacotes. Todo o acesso do Javascript para o Java pode ser feito por esses objetos, que são apresentados na tabela a seguir:

| **Objeto** | **Descrição** |
|---|---|
| JavaArray | Contém um vetor Java encapsulado. |
| JavaClass | Referência a uma classe Java. |
| JavaObject | Contém um objeto Java encapsulado. |
| JavaPackage | Uma referência do Javascript a um pacote Java. |

**Tabela 3.1:** Tipos suportados pelo Liveconnect

Como o Java é fortemente tipado, cada variável deve ser declarado como pertencente a um tipo específico. Já o JavaScript é fracamente tipada, não há necessidade de declarar o tipo pertinente a cada uma, cabendo assim ao motor de execução a conversão da variável para o tipo apropriado.

Para usar os objetos do Javascript no Java deve-se realizar a importação das seguintes classes do pacote netscape.javascript:

- netscape.javascript.JSObject : permite ao código Java acessar os métodos e propriedades dos objetos do Javascript.
- netscape.javascript.JSException : permite a manipulação de erros do Javascript através do código Java.

As classes acima descritas estão dispostas no plugin.jar que deve ser adicionado ao projeto, ou ao classpath. O referido .jar encontra-se dentro da pasta de instalação do Java no diretório "jre/lib".

## 3.5 JSAPI

O Java Speech API (JSAPI) foi um padrão criado pela Sun Microsystems para facilitar a construção de aplicativos adicionando reconhecimento e síntese. Tais aplicativos, segundo o [JSAPI 1998], possibilitam a entrada de dados mais rápida do que através do teclado. Outro aspecto ressaltado é a idéia que sistemas com interação via voz melhoram a acessibilidade de pessoas com limitações físicas.

O desenvolvimento dessa interface contou com a participação de companhias que manipulavam as tecnologias de processamento de fala. Este processo se deu de forma aberta, sendo passível de revisão e comentário público. Desta forma, chegou-se a uma especificação com alto nível e excelente técnica, [JSAPI 1998].

JSAPI é uma extensão da plataforma Java. Por isso, ela estende uma funcionalidade do Java, seja através de pacotes ou classes.

O projeto do JSAPI possui os seguintes objetivos:

- Prover suporte para síntese e reconhecimento seja no modo comando controle e ditado.
- Prover uma interface multi plataforma para síntese e reconhecimento.
- Habilitar o acesso ao estado da arte das tecnologias de voz.
- Suporte integrado com outras funcionalidades da plataforma Java, incluindo a Java Media APIs.
- Ser simples, compacta e fácil de aprender.

Ao usar o JSAPI, como pacote de extensão, no desenvolvimento de aplicativos com o Java agrega-se a estes as vantagens advindas da linguagem de programação. Dentre as características do Java destacam-se:

- Portabilidade: a máquina virtual Java roda em vários tipos de hardware e sistemas operacionais.

- Poderosa e com ambiente compacto: segue o padrão de orientação a objetos que facilita a reutilização do código, implicando no desenvolvimento rápido e eficiente. Segundo o [JSAPI 1998], os passos típicos para o reconhecimento de voz são:
- Projeto da gramática: define que palavras os usuários poderão falar.
- Processamento do sinal: análise do espectro característico do áudio introduzido.
- Reconhecimento de fonemas: compara espectro de fonemas padrões com o espectro obtido.
- Reconhecimento de palavra: reconhece a sequência de fonemas mais provável para uma determinada entrada.
- Geração de resultado: informa a palavra que foi reconhecida e informações sobre o áudio obtido.

A JSAPI suporta dois tipos básicos de gramáticas: gramáticas de regras e as gramáticas de ditado. A gramática baseada em regras define o que é esperado que o usuário fale. Assim, o número de palavras é reduzido, o que resulta em rapidez e acurácia no processo de reconhecimento. As gramáticas do JSAPI usam o padrão Java Speech Grammar Format (JSGF).

Gramáticas de ditado impõem poucas restrições no que pode ser dito. Essa forma de reconhecimento consome maior quantidade dos recursos computacionais, porém aumenta a gama de palavras a serem reconhecidas.

A JSAPI possui um pacote que define a representação de um software abstrato denominado de *speech engine*. Este pacote é o javax.speech. *Speech engine* é um termo genérico atribuído a sistemas projetados para manipular entradas ou saídas de voz.

O sistema Coruja, descrito na próxima seção, manipula o Julius, através de chamadas por Java Native Interface (JNI). Esse código permite a criação de uma instância do reconhecedor, sendo seguido a especificação do pacote *javax.speech*.

## 3.6 Coruja

O Coruja [Silva et al. 2010] é um motor (*engine*) para reconhecimento de voz para o PB. O sistema é produto dos esforços do grupo FalaBrasil da Universidade Federal do Pará, que desde 2009, o disponibilizou a comunidade de desenvolvedores.

O Coruja é composto pelos seguintes componentes: o decodificador Julius [Lee & Kawahara 2009]; modelos acústico e de linguagem para o PB; e uma API chamada de LaPSAPI, que é implementada em C/C++. Desta forma, o Coruja pode ser entendido como um sistema que possibilita o reconhecimento de fala para o PB, utilizando o deco-

dificador Julius, sendo este controlado por uma API que facilita sua manipulação.

A LaPSAPI encapsula as operações do Julius, assim os detalhes de baixo nível são omitidos ao programador. Desta forma, a LaPSAPI realiza o controle em tempo real das funcionalidades do Julius e ainda da interface de áudio do sistema. Em relação a plataforma, essa API pode ser utilizada tanto no Linux quanto no Windows, no primeiro utilizando código em C++, e no outro através de qualquer linguagem da plataforma Microsoft.NET. Essa plataforma inclui as linguagens C#, Visual Basic, J#, entre outros. O processo de interação entre uma aplicação e o Coruja é ilustrado na Figura 3.1.

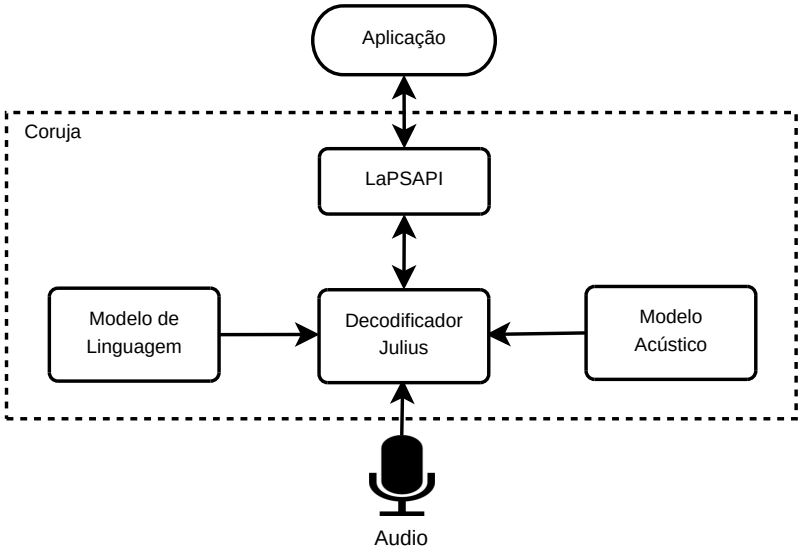


**Figura 3.1:** Modelo de interação entre uma aplicação e o Coruja.

O projeta visa alcançar o maior número possível de desenvolvedores. Por isso, os idealizadores do projeto decidiram incluir suporte a linguagem Java, para isso a LaPSAPI deveria sofrer alterações. Outra deficiência da LaPSAPI é que esta não segue nenhuma das especificações consagradas que regem o desenvolvimento de aplicativos com suporte a reconhecimento de fala. Assim, o Coruja não suporta uma mudança do motor de reconhecimento, caso seja necessário ao longo do projeto.

## 3.7 JLaPSAPI: Uma API em Java para o Coruja

A JLaPSAPI é uma API escrita em Java para o Coruja, compatível com a especificação JSAPI, que disponibiliza os recursos para utilização com a linguagem Java, que é amplamente utilizada pela comunidade de desenvolvedores [Oliveira et al. 2011].

Para não ser necessário re-implementar as funcionalidades disponibilizadas pelo sistema Coruja através da LaPSAPI, a JLaPSAPI opera sobre esta. A comunicação entre

as duas, tendo em vista que a LaPSAPI é escrita em C/C++ e a JLaPSAPI em JAVA, é realizada através Java Native Interface (JNI) [Liang 1999].

O acesso ao Coruja, em sua nova arquitetura, é feita através da utilização do código especificado na JSAPI. O programador agora pode alternar entre o Coruja e outro motor de reconhecimento que respeite as especificações do JSAPI, como por exemplo o Sphinix-4. Desta forma, o desenvolvedor não precisa alterar o código da sua aplicação, caso haja mudança no motor de reconhecimento, como ilustrado na Figura 3.2.

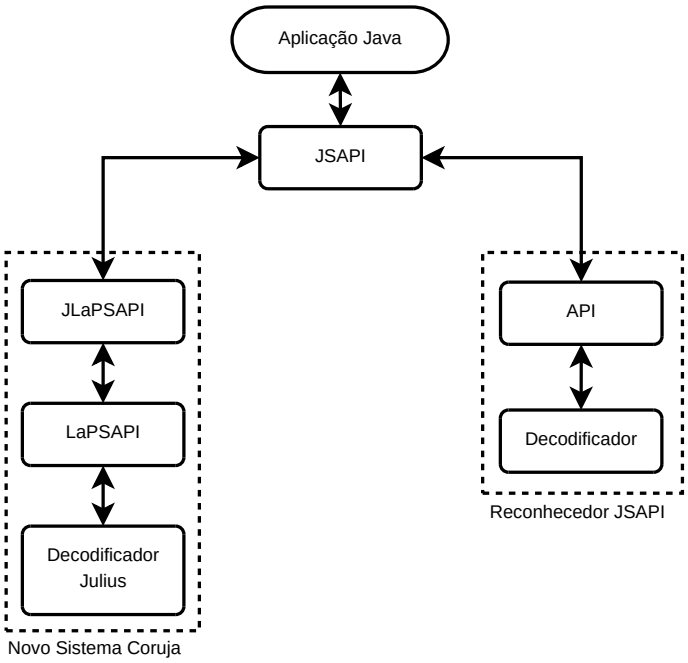


**Figura 3.2:** Modelo de interação entre uma aplicação Java e o Coruja através da especificação JSAPI.

Os métodos e eventos contidos no momento na JLaPSAPI estão relacionados na Tabela 3.2. Apesar de contar com um número reduzido, os recursos disponíveis são suficientes para a construção de aplicativos com suporte a reconhecimento de fala em PB.

A classe *Recognizer* especificado pela JSAPI é a principal na tarefa de reconhecimento. Uma instância desta pode ser criada através do método *createRecognizer*. A instância da classe permite o controle dos aspectos, propriedades e métodos, do decodificador utilizado, no caso do Coruja, o Julius. Os métodos *allocate* e *deallocate* são utilizados, respectivamente, para alocar os recursos do decodificador Julius e desalocá-los.

Para carregar uma gramática livre de contexto, que fornece ao reconhecedor um conjunto específico de palavras e frases a serem ditas, servindo assim como modelo de linguagem, usa-se o método *loadJSGF*. O arquivo deve estar no formato especificado pela

| Métodos e Eventos | Descrição Básica |
|---|---|
| createRecognizer | Cria uma instância do *engine* |
| allocate | Aloca os recursos do *engine* |
| deallocate | Desaloca os recursos do *engine* |
| loadJSGF | Carrega uma gramática de um arquivo |
| setEnabled | Ativa ou desativa reconhecimento de uma gramática |
| addResultListener | Escolhe classe ouvinte do resultado por *engine* ou por gramática |
| resume | Inicia o reconhecimento do *engine* |
| pause | Pausa o reconhecimento do *engine* |
| resultAccepted | Recebe o resultado do reconhecimento |

**Tabela 3.2:** Métodos e eventos suportados pela JLaPSAPI.
Mozilla

Java Speech Grammar Format (JSGF) [JSGF 2011]. O conversor necessário para ler o arquivo no formato JSGF e transformá-lo no formato suportado pelo Julius é uma contribuição desta pesquisa e será melhor explicado na próxima seção. O método *setEnabled* habilita ou desabilita o reconhecimento usando a gramática, dependendo da variável *booleana* que este receberá em seu argumento.

O método que inicia o reconhecimento em si é o *resume*, caso seja necessário pausar o processor o método *pause* pode ser utilizado. O resultado do reconhecimento é capturado pela instância da classe *ResultAdapter*, que é invocada através do método *addResultListener*. Por fim, o método *resultAccepted* retorna a sentença reconhecida. No Anexo B.3, é apresentando um exemplo de uso do Coruja a partir da JSAPI.

### 3.7.1 Adicionando Suporte a Gramática ao JLaPSAPI

Um dos blocos principais, onde ocorre o reconhecimento, é a JLaPSAPI. No início do projeto a JLaPSAPI realizava reconhecimento de fala contínua usando somente o modelo de linguagem, também conhecido como modo ditado. Porém, como o sistema alvo poderia ser manipulado melhor através de um sistema que trabalhasse com gramática controlada, trabalhou-se para o acréscimo desta a API.

O reconhecimento através de uma gramática controlada, para ser possível, deve receber um arquivo de entrada com as sentenças ou palavras relevantes ao sistema. Assim, em nosso projeto, era imprescindível a construção de um conversor que gerasse os arquivos necessários ao Julius a partir de um arquivo no formato JSGF. Para que o Julius possa trabalhar no modo gramática deve-se fornecer dois arquivos para que este compare a sentença de entrada ao conjunto de palavras relevantes ao sistema, sendo que este retornará a com maior probabilidade. Os dois arquivos de entrada para o Julius são o .grammar e

o .voca. O primeiro descreve a estrutura da sentença e a categoria das regras. O segundo contém a lista de palavras respectivas as categorias contidas no .grammar e a sequência fonética desta.

O desafio então era receber a entrada de um arquivo no JSGF e gerar os arquivos no formato .voca e no .grammar. A Figura 3.3 mostra o trabalho do conversor, que gera os arquivos apropriados para o Julius, tendo como base um arquivo no formato JSGF.

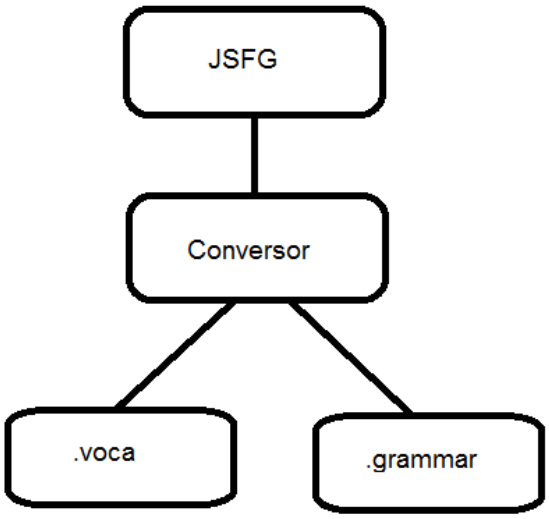


**Figura 3.3:** Processo de conversão entre a gramática no formato JSGF e o do Julius.

Por exemplo, temos o seguinte arquivo no formato JSGF:

```
\#JSGF V1.0;tal

grammar mail;

public <comandos> = abrir | ler | escrever | salvar | anterior |
    próxima | fechar | cima | baixo | lidas;
```

Caso o usuário entre com um arquivo especificado acima a saída deveria ser a seguinte para o arquivo .grammar do Julius:

```
S : NS\_B comandos NS\_E
```

O NS_B e o NS_E são usados para especificar o silêncio antes e depois da palavra a ser reconhecida. O S indica a sentença a ser reconhecida. A palavra “comandos“ é uma categoria que será especificada no .voca. A saída do arquivo .voca correspondente é exibida abaixo.

```
\% NS\_B
<s>sil
\% NS\_E
</s>sil
\% comandos
abrir a b r i X
ler l e X
escrever e s k r e v e X
salvar s a w v a X
anterior a~ t e r i o X
próxima p r O S i~ m a
salvar s a w v a X
fechar f e S a X
cima s i~ m a
baixo b a j S u
lidas l i d a s
```

Para a construção do arquivo .voca utilizou-se o conversor grafema fonema, já que o arquivo é formado pela palavra (grafema) e sua correspondente transcrição fonética. O conversor usado é o descrito em [Siravenha et al. 2008], que recebe uma palavra e seguindo determinadas regras retorna sua grafia fonética, conforme o padrão adotado pelo alfabeto SAMPA [SAMPA 2011].

Os fonemas contidos nesses arquivos são usados para definir a sequência da HMM do modelo acústico. Se houver variação na forma que a palavra é pronunciada, estas podem ser expressas em cada linha acrescentando as palavras variantes.

Por fim, o conversor foi adicionado no projeto da JLaPSAPI. O desenvolvedor pode fornecer um arquivo no formato JSGF e usar os métodos especificados na JSAPI para adicionar uma gramática na aplicação.

## 3.8 Ferramentas de Síntese

### 3.8.1 MBROLA

O MBROLA é um projeto da Faculdade Politécnica de Mons (Bélgica). O objetivo do projeto é a criação de um conjunto de vozes sintetizadas, de várias línguas e gêneros, livres para utilização em aplicações sem fins militares ou comerciais.

O projeto MBROLA é um sintetizador de voz baseado em concatenação de difones. Um banco de dados com difones é transformado no formato MBROLA. O projeto incentiva outros pesquisadores compartilharem seus banco de dados, com difones, em suas línguas.

Todos os difones da base são organizados em uma lista de entrada, onde cada difone é associado com sua respectiva informação prosódica. Essas informações consistem na duração e na amplitude da curva da frequência fundamental. O sinal de saída tem 16 bits por amostra. Assim, o projeto MBROLA não consiste em um sistema TTS, mas fornece as informações prosódicas necessárias para a geração de um voz.

Os bancos de dados com as vozes do projeto MBROLA, não constituem em si um sistema TTS, mas podem ser usados no FreeTTS, já que este não inclui suporte ao PB, em sua configuração padrão. O projeto MBROLA disponibiliza 4 bases de difones para o PB, sendo elas a BR1, BR2, BR3 e a BR4 [MBROLA 2012].

### 3.8.2 FreeTTS

O FreeTTS é um sistema do tipo *Text-to-Speech* (ver descrição no capítulo 2) criado pelo *Speech Integration Group* da *Sun Microsystems*, totalmente desenvolvido em Java. O FreeTTS é baseado no Flite [Black & Lenzo 2001] que é um motor de síntese de voz desenvolvido pela *Carnegie Mellon University* (CMU). Por sua vez, o Flite é baseado no Festival [Festival 2008] e no FestVox, que pertencem, respectivamente, a Universidade de Edinburgh e a CMU. Esses projetos provêem ferramentas, *scripts* e documentação para construção de novas vozes sintetizadas.

O Flite é inscrito em C, e projetado para ser utilizado em várias plataformas. Flite é somente uma biblioteca de síntese, que inclui duas vozes, mas ainda permitem a adição de novas vozes.

Segundo a documentação do FreeTTS [FreeTTS 2012], ele inclui as seguintes funcionalidades:

- Núcleo com um motor de síntese;
- Suporte a vozes que sintetizam números, para o idioma americano, sendo elas:
  - Voz masculina de 8khz, difone, US *English*
  - Voz masculina de 16khz, difone, US *English*
  - Voz masculina de 16khz, domínio limitado, US *English*
- Suporte a vozes importadas do FestVox (somente para vozes do Inglês Americano);
- Suporte a vozes do projeto MBROLA;

- Suporte parcial a JSAPI 1.0;
- Documentação específica da API;
- Várias aplicações demonstrativas.

# Capítulo 4

# A Ferramenta Desenvolvida

Tendo como base as ferramentas existentes, pertinentes ao reconhecimento e a síntese de voz para o PB, bem como o conhecimento das Tecnologias usadas pela Mozilla para o desenvolvimento de seus aplicativos, a construção da extensão baseia-se na união dessas ferramentas.

## 4.1 Arquitetura da Extensão

A arquitetura geral da extensão desenvolvida segue o esquema apresentado na Figura 4.1. Nesta, primeiro o usuário fornece um sinal para a entrada do sistema, que consiste em um sinal de voz, sendo interpretado pelo modo de reconhecimento automático de voz. Em seguida, o Java fornece, via Liveconnet, o resultado do reconhecimento a parte do código escrito em Javascript e XUL. A extensão, assinalada pela linha tracejada, interage com as funcionalidades do Thunderbird, através da chamada das sobreposições, que estão dispostas dentro da extensão. Isto ocorre porque toda vez que a página do TB é invocada sua sobreposição também o será. E esta, por sua vez, possui o código que invoca o reconhecedor e o sintetizador.

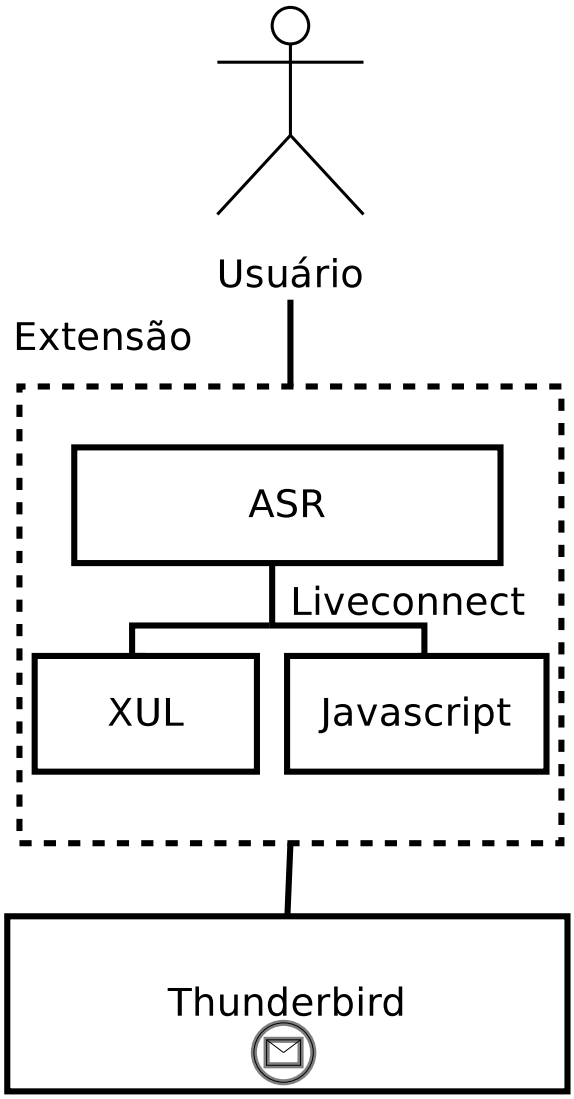


**Figura 4.1:** Esquema geral da extensão desenvolvida.

A estrutura da extensão pode ser estudada em componentes, sendo que cada um realiza uma determinada tarefa. Assim como em um sistema, que é um conjunto de partes que desempenham suas funções tendo em vista um objeto comum, as partes da extensão desenvolvida desempenham funções específicas que ao final contribuem para a adição de reconhecimento de voz ao Thunderbird.

O primeiro componente a ser descrito é o de reconhecimento de voz. Este possui como base o Coruja. Porém, faz-se necessário a criação de um código que una o Coruja e o plugin.jar, que permita o encapsulamento dos objetos do Java em um tipo interpretável pelo Javascript. Assim, a palavra reconhecida através do uso do Coruja é enviada como um JSObject para o código escrito em Javascript.

Outro componente fundamental é o da extensão em si onde se encontram as sobreposições das páginas com funcionalidades do Thunderbird. As páginas são agrupamentos de códigos em XUL, com um cabeçalho específico que as identificam, sendo elas responsáveis por mostrar uma determinada interface gráfica que fornece uma funcionalidade ao TB. Por exemplo, existe uma página com o código da tela de abertura, envio de mensagens, caixa de entrada, entre outras. As sobreposições possuem a mesma estrutura das páginas, porém, elas adicionam ou modificam a estrutura da página a qual estão associadas, assim, os dois códigos são executados quando a página é invocada, o pertinente a esta e o da sua sobreposição, ver Figura 4.2. A referida associação, entre a página e a sobreposição, é definida em um arquivo da extensão que é o chrome.manifest (ver Apêndice B.1). Ao ser removidas, a aplicação retorna ao estado anterior.

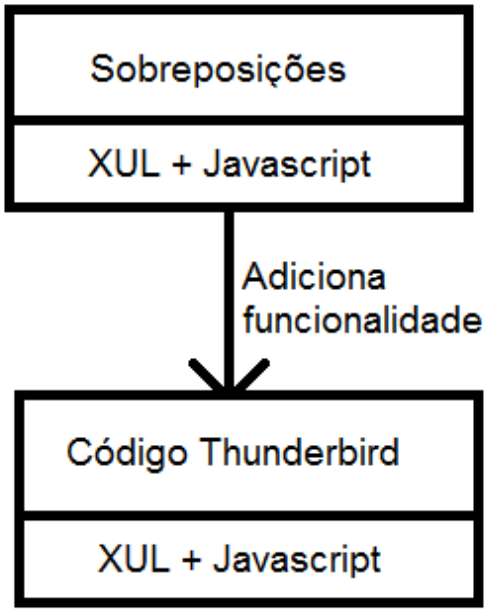


**Figura 4.2:** Interação entre a extensão e o TB.

A comunicação entre os blocos que contêm a extensão com o reconhecedor e o sintetizador e o TB acontece da forma descrita na Figura 4.3. Esta mostra a entrada do usuário com um sinal de voz, que será processado pelo bloco que contém o reconhecedor, que gerará a palavra reconhecida. A palavra é encapsulada dentro de um JSObject e é enviada a sobreposição, que por sua vez, invocará determinada funcionalidade de acordo com a palavra.

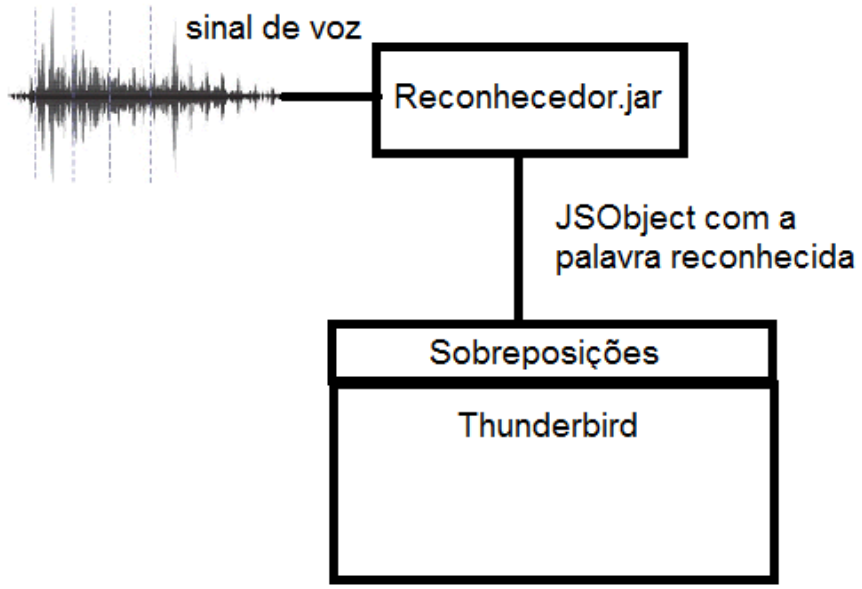


**Figura 4.3:** Interação do reconhecedor com o Thunderbird.

Nas próximas seções serão apresentados alguns detalhes da construção dos módulos acima apresentados.

## 4.2 Criação do .jar com o Reconhecedor

Para a criação do módulo de reconhecimento faz-se necessário seguir os passos descritos na página do projeto da JLaPSAPI [1]. Se os passos forem seguidos corretamente, será criada uma classe chamada *SimpleRecognition*. Essa classe, que é escrita em Java, contém um exemplo do código necessário para realizar o reconhecimento de voz.

Para realizar o reconhecimento usando uma gramática é necessário que a classe passe por modificações . Para isso, é necessária a construção de um arquivo no formato JSFG. As especificações para construção desse arquivo pode ser encontrado em [JSGF 2011]. Após a criação do arquivo o próximo passo é invocar o código necessário para carregar a gramática. Assim, deve-se adicionar o código abaixo, na classe SimpleRecognition, após invocar o método *allocate()*.

```
FileReader reader = new FileReader("./mail.grammar");
gram = rec.loadJSGF(reader);
```

[1] https://www.laps.ufpa.br/falabrasil/jlapsapi

No código o objeto da classe FileReader recebe, como argumento do construtor, o caminho para o arquivo que contém a especificação da gramática. A variável de referência que aponta para o caminho da gramática deve ser passada como argumento ao método loadJSGF. Esse método invoca o código que faz a conversão para os tipos especificados pelo Julius e carrega o reconhecimento de voz usando a gramática.

O próximo passo é adequar o código de forma que ele receba um objeto do código escrito em Javascript. Nesse objeto está encapsulada uma função do JS, que recebe como parâmetro uma variável com o texto, ou seja, do tipo *String*, sendo que esta função será invocada dentro do código do Java e o parâmetro a ser devolvido será a palavra reconhecida. Para isso, é necessário adicionar ao projeto o plugin.jar, que encontra-se dentro da pasta de instalação do Java no diretório jre/lib/. A seguir, realiza-se a importação da classe JSObject e da JSException, que auxilia na detecção dos erros que porventura venham a acontecer na manipulação do objeto nativo do JS. O código que realiza o importe das classes é especificado abaixo.

```
import netscape.javascript.JSObject;
import netscape.javascript.JSException;
```

No código escrito em JS, que vai dentro da extensão, deve haver uma chamada a um método contido na classe SimpleRecognition. Isto pode acontecer graças ao uso das especificações do Liveconnect. O método deve receber como argumento uma variável do tipo JSObject, que contém uma função do JS encapsulada, como já foi citado no parágrafo anterior. No escopo do método, o objeto que é passado por parâmetro é atribuído a uma variável estática de classe. O fato desta ser de classe implica que pode ser manipulada em qualquer um dos métodos que a classe possui e por ser estática a variável sempre apontará para o mesmo endereço de memória independente do objeto que a acessa. A variável deve ser assim declarada porque ela vai ser invocada pelo método que fica analisando a entrada do microfone, e caso um sinal seja injetado nesta, o método dispara um evento que retorna a palavra reconhecida. O código concernente ao método que é invocado pelo código escrito em JS é apresentado a seguir.

```
 public static void Atualiza(JSObject temp){
try{
func = temp;
Chamar();
}catch(JSException e){
System.out.println(e.getMessage());
```

```
}
}
```

No código acima, há a chamada a um método estático, o Chamar(), que não precisa de um objeto da classe para ser invocado, já que a chamada de um classe pelo Liveconnect não implica na execução do método principal, conhecido como *main*, o nome dele foi alterado para Chamar() e este é invocado quando o Atualiza é invocado. Por fim, a palavra reconhecida é armazenada em uma variável e enviada a função do JS que esta dentro da variável de classe. A classe JSObject possui o método *call* que permite a chamada da função do Javascript. A chamada ocorre conforme a linha de código abaixo.

```
func.call("call", new Object[] {null,resultado });
```

O objeto *func* é o que aponta ao objeto JS com a função. A *String* "*call*"especifica o tipo de operação a ser realizada, que é a chamada da função. Então, é criado um objeto que encapsula a variável resultado, sendo este enviado ao código em JS. Esse processo é sintetizado na Figura 4.4.

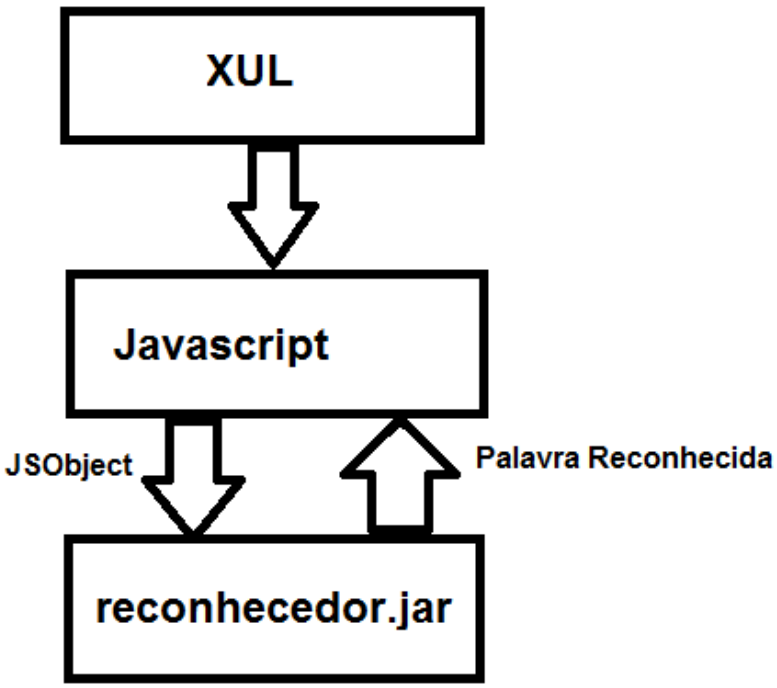


**Figura 4.4:** Papel do liveconnect na extensão.

A classe *SimpleRecognition* com modificações pode ser encontrada no Apêndice B.3.

O último passo consiste em encapsular o projeto em um .jar. A classe principal deve ser a *SimpleRecognition*. O .jar criado deve colocado dentro da pasta que contêm o projeto da extensão.

Feito isso, agora o projeto concentra-se em estruturar o código que irá manipular as funcionalidades do TB de acordo com a palavra que foi reconhecida. Esse processo é melhor descrito nas seções posteriores.

## 4.3 Criação do .jar com o Sintetizador

A criação do Sintetizador baseou-se em um tutorial que apresenta os passos para utilizar o FreeTTS com o MBROLA [2]. Nesse tutorial, o autor ensina como usar o FreeTTS usando uma das vozes do MBROLA para o PB. O autor também criou um código que gera uma interface que permite o usuário entrar com um texto e então este é sintetizado. Em nossa extensão essa interface gráfica não seria necessária, por isso foram realizadas alterações no código, de forma a simplificar a aplicação. Assim, exclui-se todos os elementos gráficos, e a classe ficou com apenas um método que recebe uma *String* e devolve a síntese desta.

Desta forma, encapsulou-se as classes dentro de um .jar e este foi adicionado a pasta do projeto. Na extensão, quando se deseja sintetizar algo é só invocar o método, através do Liveconnect, e passar o texto.

## 4.4 Chrome List

Como o Thunderbird faz uso do protocolo *chrome*, que serve para fazer chamadas aos arquivos em XUL, assim, ao se especificar um caminho que inicie com a palavra *chrome* os aplicativos Mozilla interpretam que existe código em XUL neste arquivo, e também em JS. É necessário saber qual arquivo deve ser manipulado, de forma a acrescentar funcionalidade a alguma interface do TB. Para auxiliar os desenvolvedores nesse sentido uma extensão é disponibilizada pela Mozilla, sendo esta o Chrome list (Figura 4.5). Esta extensão permite a visualização do código do Thunderbird.

O Chrome list exibe todo o código pertinente ao Thunderbird. Para usá-lo é necessário fazer o download na página de Add-ons da Mozilla [3]. A seguir, deve-se instalá-la no TB.

[2] https://code.google.com/p/gtext2vox/

[3] https://addons.mozilla.org

Ao instalar o Chrome List um problema de compatibilidade de versão pode ser apresentado. Para solucionar o problema é só abrir o arquivo install localizado na raiz da extensão e modificar a versão do aplicativo nos rótulos <em:minVersion> <em:maxVersion>.

Após a instalação aparece um novo item no menu Ferramentas, que é o Explore Chrome.

Quando o item especificado acima é selecionado uma tela igual a mostrada na figura 4.5 é exibida. Na guia à esquerda aparece as pastas com os componentes do aplicativo. Ao clicar nas pastas os arquivos com os códigos são dispostos na tela a esquerda e ao clicar no arquivo uma nova janela é aberta mostrando o código.

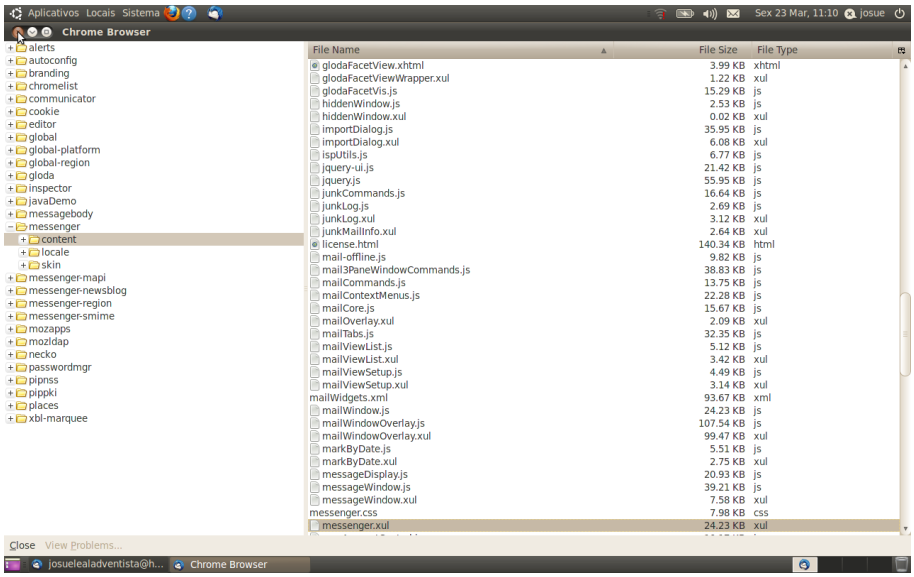


**Figura 4.5:** Tela do Chrome List.

## 4.5 Como Criar um Extensão ao TB

Para iniciar a construção do código pertinente a extensão em si é importante seguir o tutorial disponível no site da Mozilla [Mozilla 2012], pois este auxilia na tomada dos primeiros passos na construção de uma extensão.

Após seguí-lo, entende-se acerca da estrutura padrão de pastas que deve ser obedecida, além da criação de dois arquivos: o install e o chrome.manifest (ver Apêndice B.2). No primeiro, como o nome indica, informações da versão, autor, aplicativo alvo, entre outras, são definidas. No chrome.manifest tem-se a relação entre a página que deseja-se adicionar uma sobreposição e o caminho do arquivo com as alterações.

Cada aplicativo da Mozilla possui um código de identificação. O Thunderbird possui o seguinte Id 3550f703-e582-4d05-9a08-453d09bdfdc6. Este código deve ser inserido

entre a tag <em:id>, do arquivo install.

O Chrome.manifest mostra qual sobreposição se relaciona com qual página, como a mostrada a seguir:

```
overlay chrome://messenger/content/messenger.xul
chrome://primicia/content/voiceOverlay.xul
```

Nota-se que por padrão a linha deve começar com a palavra *overlay* (sobreposição), indicando que a segunda *chrome* URL sobrepõe a primeira.

Para saber quais páginas serão sobrepostas é necessária a análise do código do TB. Esse trabalho é facilitado pela ferramenta supracitada, o Chrome List. Ao utilizá-lo encontramos as seguintes páginas:

- MsgAccountCentral.xul - É a página que contém o código da tela inicial do TB que apresenta atalhos para a caixa de entrada, página de composição, configurações e outras páginas.
- Messenger.xul - É a página que apresenta o código da caixa de entrada, que lista as mensagens recebidas e permite a navegação e manipulação dessas.
- Messengercompose - Contêm os códigos pertinentes a tela de composição das mensagens.

Identificada as páginas relevantes, inicia-se a construção do código das sobreposições. Nesse sentido, a Mozilla escolheu tecnologias para a construção de seus aplicativos que facilitassem o desenvolvimento de plugins e extensões. Assim, para o desenvolvimento das extensões necessita-se somente de um editor de texto, sendo que a análise do código é feita pelo próprio aplicativo e caso ocorra um erro este pode ser visualizado no menu ferramentas selecionando o item console de erros.

Como a tarefa de ficar instalando e removendo uma determinada extensão pode constituir incomodo a alguns, existe um plugin chamado foxbeans que pode ser utilizado para o desenvolvimento na Integrated Development Environment (IDE) Netbeans. Ao executar o projeto o aplicativo alvo é aberto e a extensão adicionada automaticamente, juntamente com os arquivos necessários.

Embora o desenvolvedor utilize o Netbeans, é necessário que uma propriedade do Thunderbird seja alterada no config editor, plugin.disable, que por default encontra-se com o valor true. Há a necessidade de mudar o valor da variável para false, senão o seguinte erro será exibido no console “Java is not defined”.

O config editor está localizado no menu Editar na opção Preferências. Ao selecioná-lo aparecerá uma janela. Deve-se clicar na aba Geral e depois no botão Config Editor,

então outra janela será exibida informando que o usuário deve ter cuidado ao alterar as configurações. Então, o próximo passo é procurar a propriedade plugin.disable e mudar seu valor para false. Assim o plugin Java funcionará no aplicativo.

As extensões do TB, seguindo o padrão da Mozilla, devem ser arquivos do tipo .xpi. Para gerar uma pasta compactada com este formato é necessário primeiro compactar a pasta com os arquivos para o formato .zip, utilizando uma ferramenta para este fim, e ao final renomeá-lo com a extensão exigida, o .xpi. A extensão em si contém as bibliotecas java, no formato .jar, encapsuladas dentro do seu pacote, tanto do sintetizador como do reconhecedor.

## 4.6 Instalação da Ferramenta

Para instalar a extensão no Thunderbird é necessário que se tenha em mãos o arquivo .xpi que contenha o projeto. Futuramente, esse arquivo será disponibilizado na página do LaPS.

Além da extensão, deve-se seguir as instruções dispostas no site LaPS, na seção sobre a JLaPSAPI[4], para que se possa adquirir os modelos acústicos e de linguagem que são indispensáveis para a aplicação.

Depois de realizado o passo acima e tendo posse do arquivo .xpi da extensão, o próximo passo a ser realizado, no Thunderbird, é a instalação em si da extensão. Com o TB aberto vá ao menu ferramentas e clique no item complementos, o que mostrará uma janela que permite a instalação através da especificação de um determinado arquivo. Procure o arquivo em seu computador e depois é só clicar em instalar.

Ao final do processo de instalação o TB irá informar que para que as alterações sejam efetivadas é necessário seu reinício. Após o reinício, se tudo aconteceu conforme o esperado, uma voz sintetizada será ouvida informando quais os possíveis comandos poderão ser acionados.

[4]http://www.laps.ufpa.br/falabrasil/jlapsapi

# Capítulo 5

# Resultados Alcançados

Para analisarmos o desempenho e ajustar os parâmetros do decodificador, fez-se o uso da base de dados de áudio denominado LaPSMail. A construção de um corpus de voz LapsMail, foi projetado para representar um conjunto básico de comandos necessários para controlar uma aplicação de correio eletrônico. A idéia é estabelecê-lo como referência para a avaliação de sistemas RAV para PB dentro desse contexto.

Atualmente, o corpus LapsMail consiste de 86 sentenças, incluindo 43 comandos e 43 nomes falados por 25 voluntários (21 homens e 4 mulheres), o que corresponde a 84 minutos de áudio. Ao todo são observadas 95 palavras distintas. Segue abaixo três sentenças que fazem parte do corpus. A lista completa das sentenças pode ser visualizada no Apêndice C.

(1) <s> abrir caixa de entrada </s>
(2) <s> responder ao remetente </s>
(3) <s> ana carolina </s>

O corpus LapsMail foi gravado usando um microfone de alta qualidade (Shure PG30), amostrado em 16.000 Hz e quantizado em 16 bits. O ambiente acústico não foi controlado, existindo a presença de ruído ambiente nas gravações. A base de áudio LapsMail encontra-se publicamente disponível [FalaBrasil 2009].

Além da base de dados, outros elementos foram utilizados nos testes, sendo estes os mencionados a seguir. Primeiramente, o decodificador Julius em sua versão 4.1. O modelo acústico e o de linguagem, respectivamente, em suas versões *LAPSAM1.7.1.am.bin* e *LAPSLM1.7.1.lm.bin*. O dicionário fonético utilizado foi o *dictionary_ssp.dic*. Esses elementos, estão disponíveis também no site do [FalaBrasil 2009].

Os testes foram realizados em uma máquina com as seguintes configurações:

- Processador Core 2 Duo E6420, com frequência de *clock* igual a 2.13 GHz.
- Memória RAM (*Random Access Memory*) de 1 Gigabytes, DDR2.
- Disco rígido SATA com capacidade para 160 Gigabytes.
- Sistema Operacional Ubuntu 10.04.

## 5.1 Ajuste de Parâmetros

O Julius possui algumas dezenas de parâmetros que podem receber, porventura, um determinado intervalo de valores. Esses valores, apresentam comportamento específico dependendo do tipo aplicação a ser usada, gramática ou ditado, bem como o tamanho do dicionário. Como a ferramenta proposta baseia-se, sobretudo, na utilização de determinada gramática, priorizou-se o ajuste desses parâmetros para este tipo específico.

[Lee et al. 2001], ressalta alguns parâmetros de relevância no processo de ajuste, sendo estes, não em ordem de importância, o *Beam*, o *Word Insertion Penalty* e os parâmetros do Modelo de linguagem. O autor, afirma ainda que uma decodificação eficiente usa o menor valor possível de Beam. Em relação aos outros parâmetros nenhuma afirmação é encontrada.

Em [Lee 2009] tem a explicação de cada flag que compõe o arquivo de configuração do Julius. A seguir tem-se uma explanação dos parâmetros utilizados nas simulações.

- Beam - marcado com a *flag* -b no passo um e -b2 para o segundo passo. Representa o número de nós da HMM usados na classificação do primeiro passo. Quanto menor for o valor mais rápido será o decoder pelo fato de diminuir o número de nós a serem pesquisados, mas se o número for pequeno demais haverá uma perda na acurácia.
- Modelo de linguagem - o decodificar verifica a probabilidade em relação ao modelo acústico e o de linguagem. Os parâmetros pertencentes ao LM, peso e *penalty*, implicam em um peso maior ou não a probabilidade deste.
- *Word Insertion Penalty* (WIP) - Usado para balancear os erros de inserção e deleção.

No Julius dois passos são realizados no processo de decodificação, sendo que cada passo possui os parâmetros supracitados. Esses parâmetros sofrem influência mútua, assim por tentativa e erro verificou-se qual relação existente entre eles e os valores ótimos para estes.

Nos experimentos realizados, notou-se que as variáveis do primeiro passo possuem relevância maior no desempenho do reconhecedor. O objetivo foi encontrar valores que diminuíssem a WER, bem como o *Real Time Factor* (xRT). Em relação ao segundo passo,

notou-se que o parâmetro que afetava o resultado do reconhecimento é o Beam. A seguir, são mostrados gráficos que melhor elucidam o exposto aqui.

O primeiro parâmetro a ser avaliado foi o Beam. No gráfico da Figura 5.1, verifica-se a relação entre o valor do Beam e a WER. Nota-se que acima de 550 a taxa da WER permanece invariável.

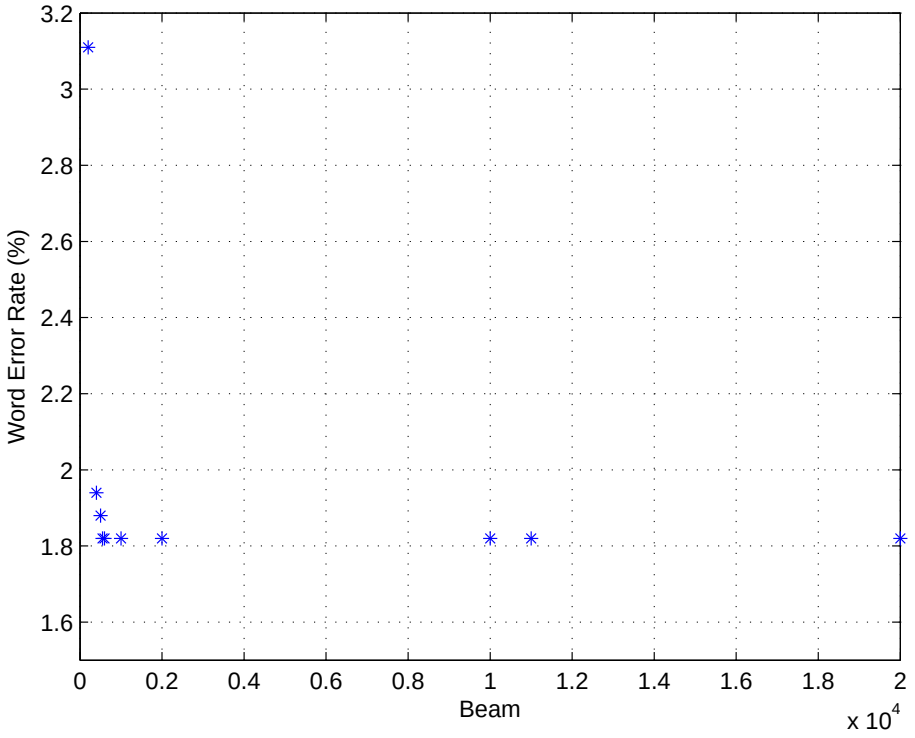


**Figura 5.1:** Comportamento do Beam em relação com a WER.

Os experimentos corroboram a afirmação de [Lee et al. 2001], pois, no gráfico da Figura 5.2, conclui-se que apesar de possuírem o mesmo valor de WER quanto menor o valor do Beam mais rápido será o reconhecimento, ou seja, menor xRT. Isso acontece porque a tarefa de decoder acontecerá tendo em vista um menor número de nós na HMM.

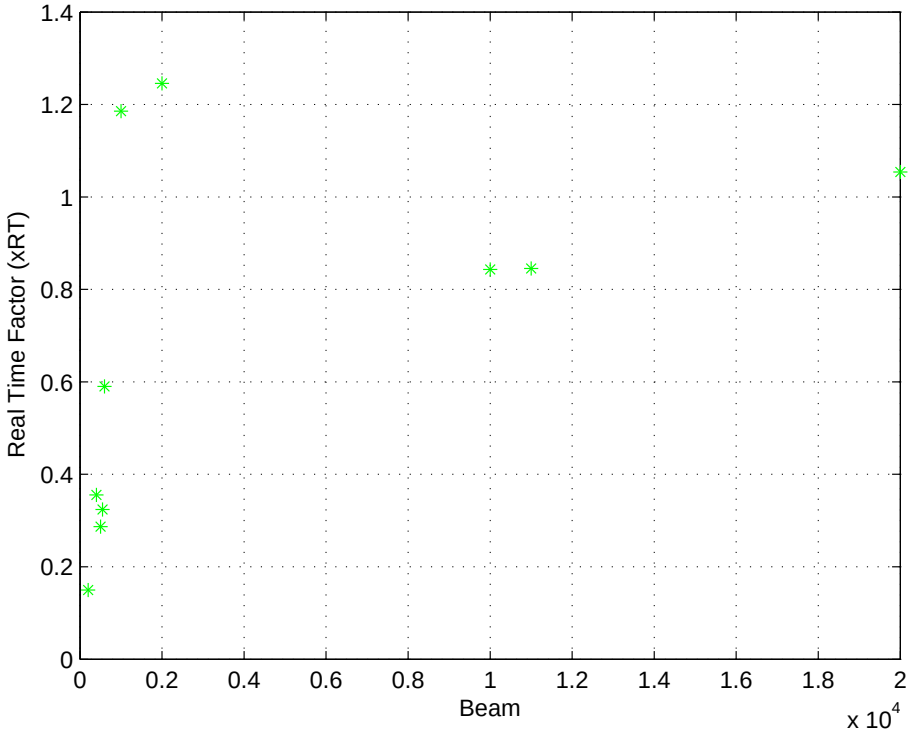


**Figura 5.2:** Comportamento do Beam em relação com o xRT.

O gráfico da Figura 5.3 mostra o comportamento do peso do modelo de linguagem, no que diz respeito a WER. Nota-se que esta não apresenta variação com o crescimento do peso do modelo de linguagem.

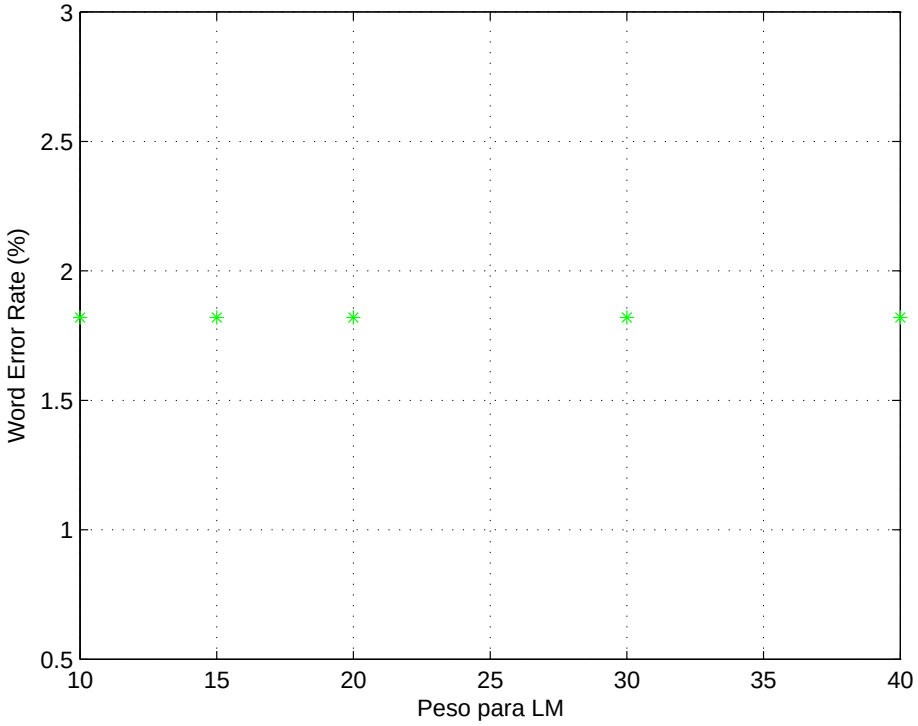


**Figura 5.3:** Comportamento do peso do ML em relação com a WER.

Em relação ao xRT há uma variação de acordo com o valor utilizado. A variação é não linear, pois o acréscimo não é necessariamente acompanhado de um aumento no valor do xRT. Os menores valores encontrados foram com peso igual a 15 e a 20, sendo que por tentativa, o valor de peso igual a 15 se comportou melhor com a variação de outros valores.

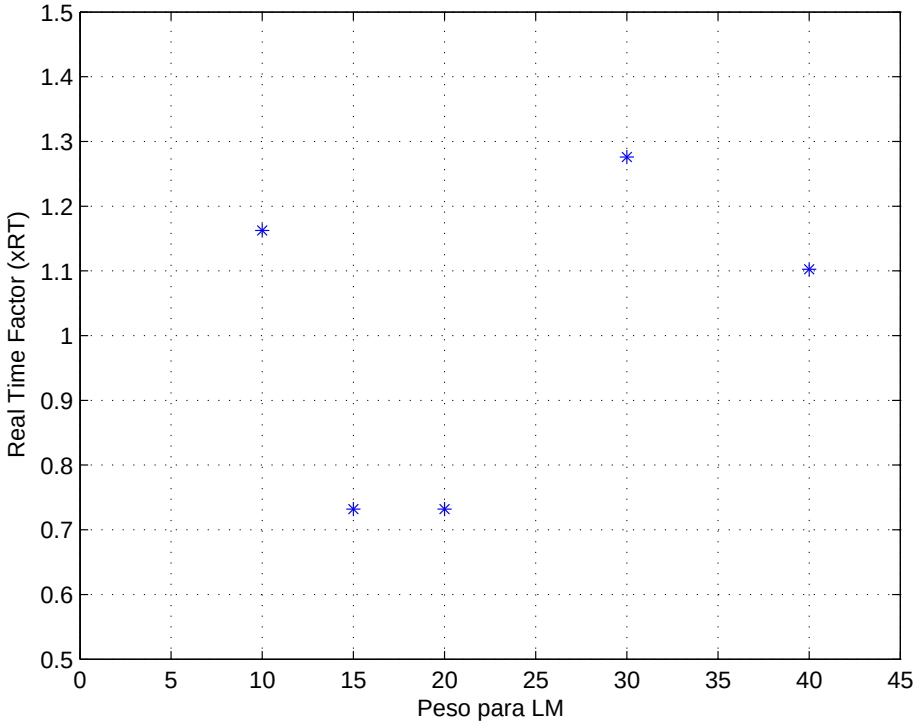


**Figura 5.4:** Comportamento do peso do ML em relação com o xRT.

Em [Alghamdi et al. 2009], tem-se que o valor do Word Insertion Penalty (WIP) varia entre 0,2 a 0,9. Os experimentos realizados cor roboraram com a afirmação do referido autor. Nota-se que o aumento do valor para 0,7 e 0,9 diminui a WER.

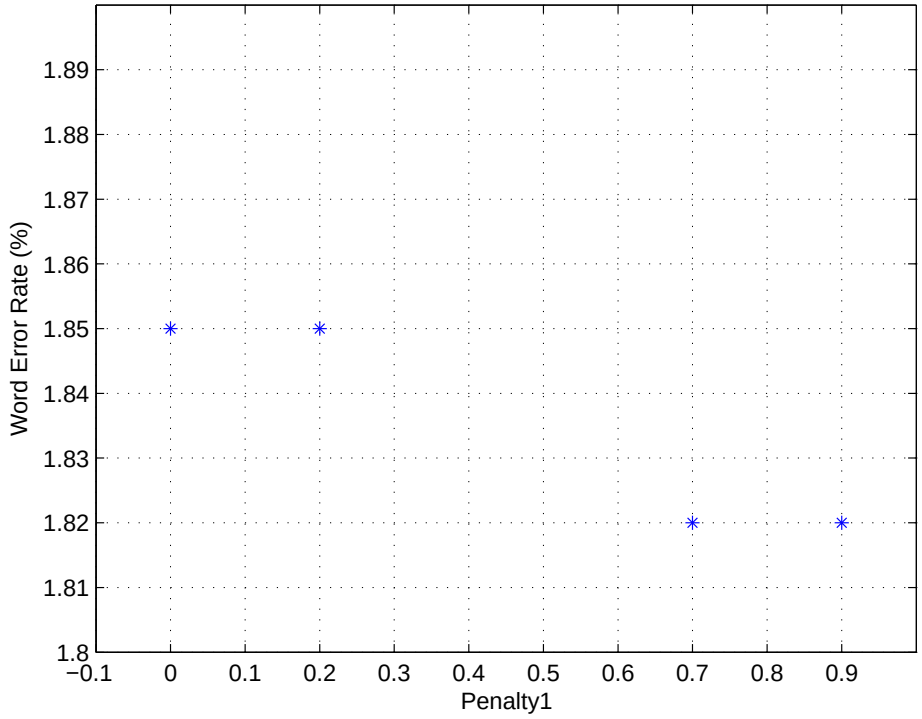


**Figura 5.5:** Comportamento do WIP em relação com a WER.

Quando se analisa o gráfico da Figura 5.4 cruzado com o xRT nota-se que este possui menor valor quando o WIP é 0,7. Assim, embora o 0,9 e 0,7 tenham o mesmo valor de WER, este possui menor xRT, sendo portanto o utilizado na configuração.

| **Parâmetro** | **Valor** |
|---|---|
| Beam | 550 |
| Peso do Modelo de linguagem | 15 |
| Penalty do Modelo de linguagem | 30 |
| Word Insertion Penalty | 0,7 |

**Tabela 5.1:** Melhores valores encontrados para os parâmetros do Julius

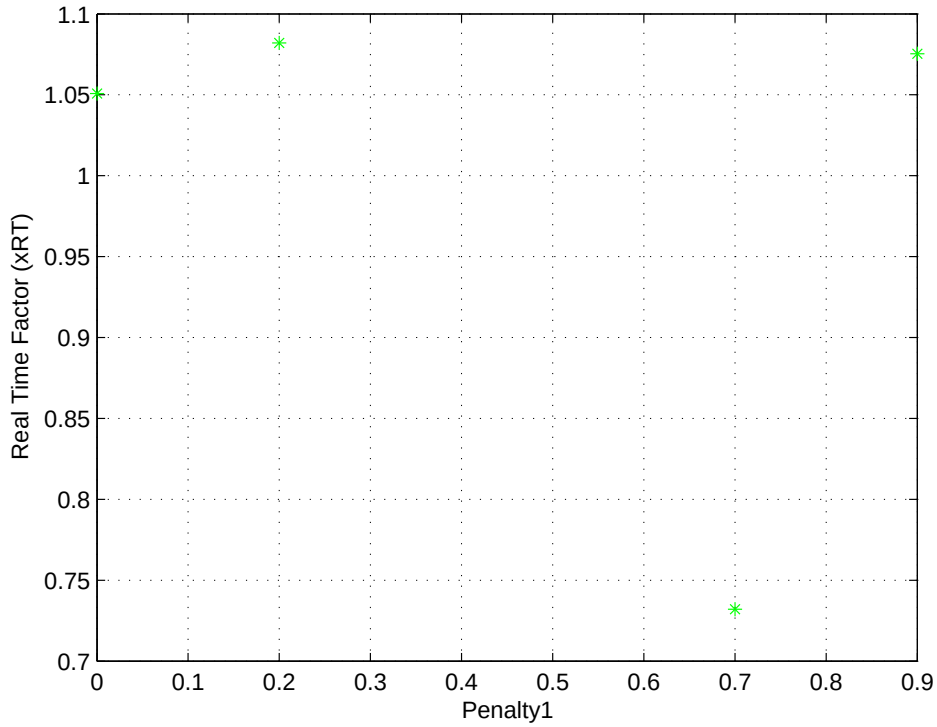


**Figura 5.6:** Comportamento do WIP em relação ao xRT.

Outros gráficos podem ser consultados no Apêndice A.1. Assim, por tentiva e erro os melhores valores encontrados para os parâmetros do arquivo de configuração do Julius estão especificados na Tabela 5.1.

O processo de ajuste dos parâmetros é de extrema importância para a melhoria de desempenho da ferramenta proposta. O principal ganho que tem-se é em relação ao tempo de resposta do decoder. Com os parâmetros padrões o xRT é por volta de 1 segundo, depois do ajuste esse valor caiu para 0,3 segundos. Melhoria de aproximadamente 70% na velocidade do reconhecimento.

# Capítulo 6

# Conclusão e Trabalhos Futuros

Os recursos existentes para o reconhecimento de voz para o PB, sobretudo os disponibilizados pelo LaPS, avançaram, no sentido de permitir uma gama maior de aplicativos fazerem uso de suas funcionalidades. Nesse contexto, temos o Thunderbird que é um dos gerenciadores de emails mais usados no mundo, no entanto a Mozilla faz uso de tecnologias específicas em seu desenvolvimento, mesmo em face disso os recursos podem ser aplicados aos aplicativos que esta desenvolve.

Em relação ao sintetizador, MBROLA utilizada em conjunto com o FreeTTS, ainda há muito o que melhorar em relação a naturalidade da voz utilizada. Deve-se, portanto, estudar meios para melhorar, ou gerar, uma voz mais inteligível. No entanto, entende-se que a voz utilizada atualmente, mesmo com suas limitações, pode auxiliar deficientes visuais, pois o texto sintetizado, embora não pareça natural, é compreensível.

O Thunderbird possui muitas funcionalidades para a manipulação dos emails recebidos. As funcionalidades, são manipuladas mais facilmente usando uma gramática comando controle. Para utilizar esse tipo de reconhecimento no Coruja, foi construído um conversor, que é uma etapa para adicionar ao projeto da JLaPSAPI o modo gramática. O conversor faz a transição entre o formato JSFG para o .voca e . grammar específicos do Julius. O modo foi construído com sucesso, bem como sua integração com o TB.

Como o TB possui várias funcionalidades que podem ser controladas por uma única palavra ou sentença curta, a extensão se beneficia da utilização de uma gramática controlada. O LaPSMailBenchMark mostra qualitativamente essa afirmação. Na manipulação usando comandos do aplicativo o uso da gramática é perfeitamente aplicável, porém na composição dos emails faz-se necessário o uso do modo ditado. O problema é que a acurácia deve ser melhorada. Segundo o experimento, mais da metade das palavras reconhecidas, nesse modo, estão incorretas. Assim, é fundamental a melhoria desse modo.

Nesse processo de busca de melhores taxas de reconhecimento, no menor tempo possível, foram realizados os experimentos tendo em vista o ajuste dos parâmetros do decodificador Julius. O processo alcançou uma melhoria, sobretudo na redução do xRT. Essa melhoria foi notável durante o desenvolvimento da aplicação no que consideramos dois momentos, antes e depois do ajuste. Nota-se que há uma melhoria substancial no tempo de resposta o que mudou o futuro foco da aplicação, pois via-se esta como auxiliadora de pessoas com deficiência, porém agora vislumbra-se que esta poderá auxiliar todos os usuários na tarefa de leitura de seus emails.

Assim, as ferramentas existentes já permitem a construção de uma interface controlada por fala. Neste trabalho, as utilizamos para manipular as funcionalidades TB, apresentando boa eficiência, desde que seja usado o modo de gramática controlada. Porém, para que se possa enviar mensagens o modo de reconhecimento usando ditado ainda precisa avançar muito, bem como a inteligibilidade do sintetizador.

## 6.1 Trabalhos Futuros

- Melhoria da acurácia do modelo de linguagem. Para a composição das mensagens é fundamental o aprimoramento deste modo.
- Possibilitar a manipulação total do Thunderbird através da fala. Como o Thunderbird possui milhares de linhas de código, a abrangência das funcionalidades deve exigir um esforço razoável.
- Procurar ou gerar uma voz que melhor sintetize as sentenças necessárias e mesmo as mensagens do Thunderbird.

## 6.2 Limitações

- A extensão é compatível com o Linux, distribuição ubuntu em sua versão 10.10. Não suporta versões superiores.
- É compatível com o Thunderbird 3.1.18. Não apresenta compatibilidade com versões superiores.
- O sintetizador não reconhece palavras com acento, ou siglas. Assim, para a leitura das mensagens deve haver um pré-processamento destas, retirando a acentuação e convertendo as siglas.

# Referências Bibliográficas


Alghamdi, Mansour, Moustafa Elshafei & Husni Al-Muhtaseb (2009), 'Arabic broadcast news transcription system', *Springer* pp. 183–195.

Antoniol, G., R. Fiutem, R. Flor & G. Lazzari (1993), 'Radiological reporting based on voice recognition', *Human-computer interaction. Lecture Notes in Computer Science* **753**, 242–253.

Barley, Stephen, Debra Meyerson & Stine Grodal (2010), 'E-mail as source and symbol of stress', *Articles in Advance* .

Black, A. W. & K. A. Lenzo (2001), 'Flite: a small fast run-time synthesis engine', *In Proceedings of the 4th ISCA Workshop on Speech Synthesis, page 204, Scotland* .

Boswell, David, Brian King, Ian Oeschger, Pete Collins & Eric Murphy (2002), *Creating Applications with Mozilla*, O'Reilly Media.

Cahn, Janet E. & Janet E. Cahn (n.d.), Generating expression in synthesized speech, Relatório técnico, MIT.

Canny, John (2006), 'The future of human-computer interaction', *Queue - HCI* **4**, 24–36.

Colen, W. D. & P. Batista (2010), 'Veja mamãe, sem as mãos! SpeechOO, uma extensão de ditado para o BrOffice.org', *11th Fórum Internacional Software Livre* .

Davis, S. & P. Merlmestein (1980), 'Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences', *IEEE Trans. on ASSP* **28**, 357–366.

DECtalk (2005), 'Dectalk'. `http://dectalk.com`, Visitado em Julho de 2012.

Deshmukh, N., A. Ganapathiraju & J. Picone (1999), 'Hierarchical search for large-vocabulary conversational speech recognition', *IEEE Signal Processing Magazine* pp. 84–107.

Dutoit, Thierry (2001), *An Introduction to Text-To-Speech Synthesis*, Kluwer.

FalaBrasil (2009), 'Grupo falabrasil'. `http://www.laps.ufpa.br/falabrasil`, Visitado em Junho de 2012.

Festival (2008), 'Festival project'. `http://www.cstr.ed.ac.uk/projects/festival`, Visitado em Março de 2008.

FreeTTS (2012), 'Freetts project'. `http://freetts.sourceforge.net`, Visitado em Junho de 2012.

Holmes, J. N. & W. J. Holmes (2001), *Speech Synthesis and Recognition*, T & F STM.

Huang, X., A. Acero & H. Hon (2001), *Spoken Language Processing*, Prentice-Hall.

Jevtić, N., A. Klautau & A. Orlitsky (2001), Estimated rank pruning and Java-based speech recognition, *em* 'Automatic Speech Recognition and Understanding Workshop'.

JSAPI (1998), *Java Speech API Programmer's Guide*, Sun Microsystems.

JSGF (2011), '*Java Speech Grammar Format*'. `http://www.w3.org/TR/jsgf/`, Visitado em Abril de 2011.

Juang, H. & R. Rabiner (1991), 'Hidden Markov models for speech recognition', *Technometrics* **33**(3), 251–272.

Klatt, D. (1980), 'Software for a cascade / parallel formant synthesizer', *Journal of the Acoustical Society of America* **67**, 971–95.

Kobayashi, Takao, Keiichi Tokuda, Takashi Masuko, Takayoshi Yoshimura, Tadashi Kitamura, Keiichi Tokuda, Takashi Masuko, Takao Kobayashi & Tadashi Kitamura (1999), 'Simultaneous modeling of spectrum, pitch and duration in hmm-based speech synthesis'.

Ladefoged, P. (2001), *A Course in Phonetics*, 4ª edição, Harcourt Brace.

Lee, Akinobu (2009), *The Julius Book*, 0.0.2 ed. - rev 4.1.2.

Lee, Akinobu & Tatsuya Kawahara (2009), 'Recent development of open-source speech recognition engine julius', *Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)* .

Lee, Akinobu, Tatsuya Kawahara & Kiyoshiro Shikano (2001), 'Julius - an open source real-time large vocabulary recognition engine', *Proc. European Conference on Speech Communication and Technology* pp. 1691–1694.

Lee, Chi Hui & Jean Luc Gauvain (1993), 'Speaker adaptation based on MAP estimation of HMM parameters', *IEEE ICASSP* pp. 558–561.

Liang, Sheng (1999), *The Java$^{TM}$ Native Interface Programmer's Guide and Specification*, Addison-Wesley.

Masuko, T., K. Tokuda, T. Kobayashi & S. Imai (1996), Speech synthesis using HMMs with dynamic features, *em* 'ICASSP '96: Proceedings of the Acoustics, Speech, and Signal Processing, 1996. on Conference Proceedings., 1996 IEEE International Conference', IEEE Computer Society, Washington, DC, USA, pp. 389–392.

MBROLA (2012), 'Mbrola project'. `http://tcts.fpms.ac.be/synthesis`, Visitado em Maio de 2012.

Mozilla (2012), 'Building a thunderbird extension'. `https://developer.mozilla.org/en/Extensions/Thunderbird/Building\_a\_Thunderbird\_extension`, Visitado em Abril de 2012.

Murray, I. R. & J. L. Arnott (1995), 'Implementation and testing of a system for producing emotion-by-rule in synthetic speech', *Speech Communication* **16**, 369–390.

Murray, Ian R. (1989), Simulating Emotion in Synthetic Speech, Tese de doutorado, University of Dundee, UK.

Neto, Nelson, Carlos Patrick, Aldebaro Klautau & Isabel Trancoso (2010), 'Free tools and resources for brazilian portuguese speech recognition', *The Brazilian Computer Society* **16**, 53–68.

Oliveira, Rafael, Pedro Batista, Nelson Neto & Aldebaro Klautau (2011), 'Recursos para desenvolvimento de aplicativos com suporte a reconhecimento de voz para desktop e sistemas embarcados', *12º Fórum Internacional de Software Livre* .

Outlook (2011), 'Microsoft outlook'. `http://office.microsoft.com/pt-pt/outlook-help/acerca-do-reconhecimento-de-voz-HP003084099.aspx`, Visitado em Junho de 2012.

Rabiner, L. & B. Juang (1993), *Fundamentals of Speech Recognition*, PTR Prentice Hall, Englewood Cliffs, N.J.

SAMPA (2011), 'Sampa - computer readable phonetic alphabet'. `www.phon.ucl.ac.uk/home/sampa/`, Visitado em Maio de 2011.

Silva, Patrick, Pedro Batista, Nelson Neto & Aldebaro Klautau (2010), 'An open-source speech recognizer for Brazilian Portuguese with a windows programming interface', *The International Conference on Computational Processing of Portuguese (PROPOR)* .

Siravenha, Ana, Nelson Neto, Valquíria Macedo & Aldebaro Klautau (2008), 'Uso de regras fonológicas com determinação de vogal tônica para conversão grafema-fone em português brasileiro', *7th International Information and Telecommunication Technologies Symposium* .

Taylor, Paul (2009*a*), *Text-To-Speech Synthesis*, Cambridge University Press.

Taylor, Paul (2009*b*), *Text-to-Speech Synthesis*, Cambridge University Press.

Tokuda, K., T. Kobayashi & S. Imai (1995), 'Speech parameter generation from HMM using dynamic features', *Acoustics, Speech, and Signal Processing, IEEE International Conference on* **1**, 660–663.

Tokuda, Keiichi, Takashi Masuko, Tetsuya Yamada, Takao Kobayashi & Satoshi Imai (1995), 'An Algorithm For Speech Parameter Generation From Continuous Mixture HMMs With Dynamic Features'.

van Santen, J.P.H., J. Hirschberg, J. Olive & R. Sproat, eds. (1996), *Progress in Speech Synthesis*, Springer-Verlag, New York.

# Apêndice A

# Gráficos

## A.1 Gráficos do ajuste dos parâmetros do Julius

### A.1.1 Word Insertion Penalty do 2º passo

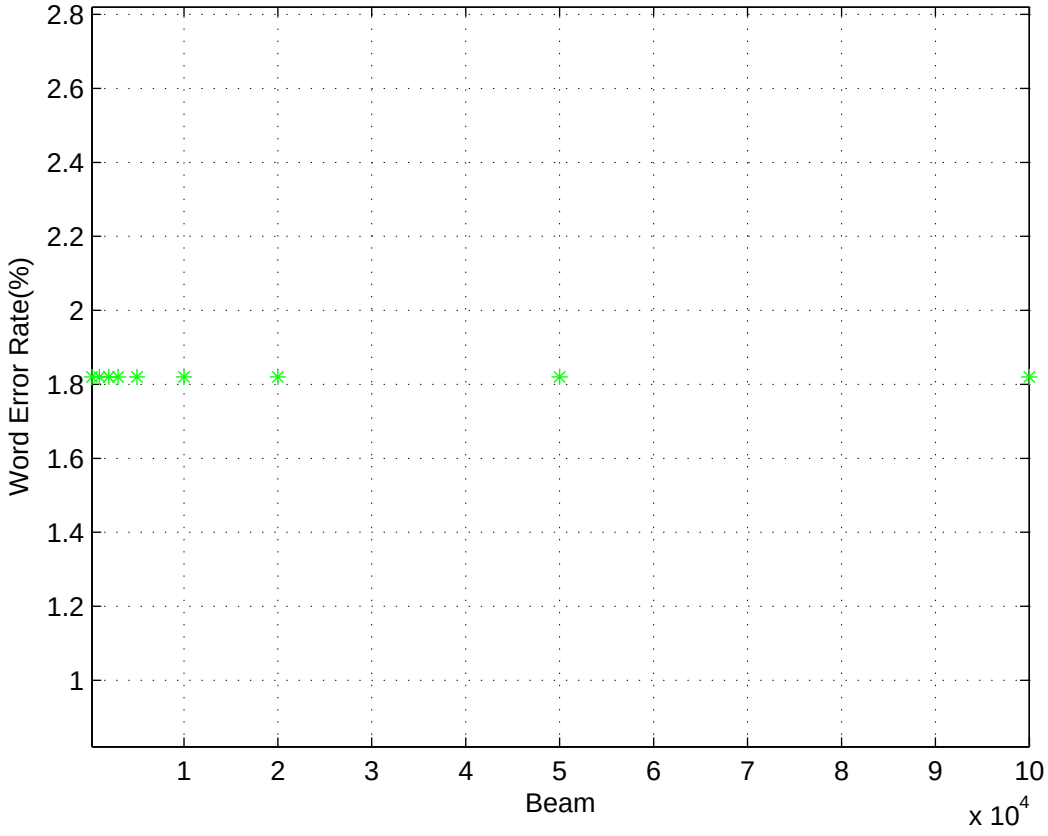


**Figura A.1:** Comportamento do Beam do segundo passo em relação ao WER

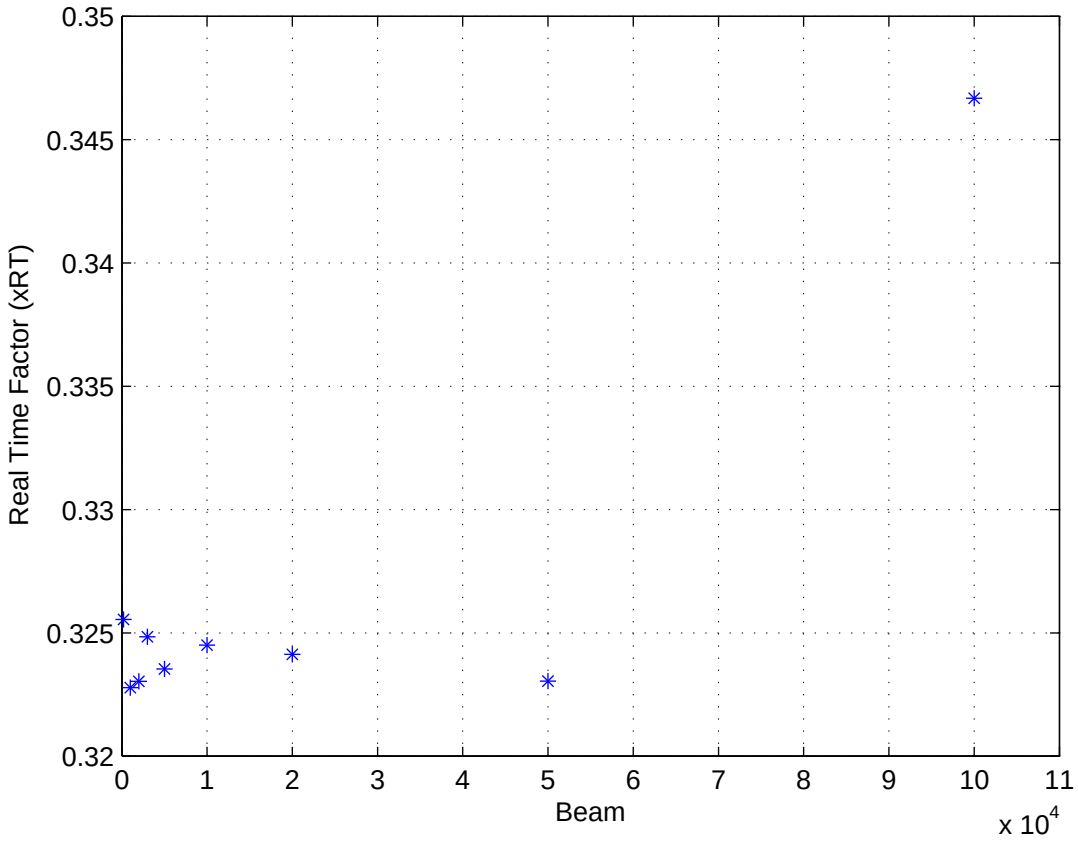


**Figura A.2:** Comportamento do Beam do segundo passo em relação ao xRT.

### A.1.2 Word Insertion Penalty do 1º passo para o Modelo de Linguagem

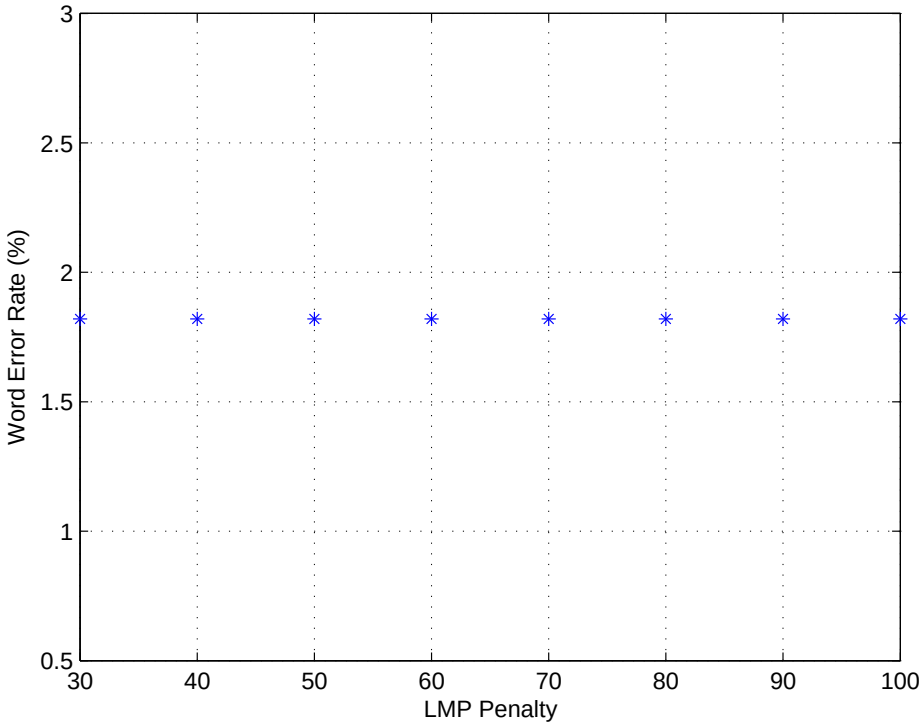


**Figura A.3:** Comportamento do Penalty do ML em relação a WER.

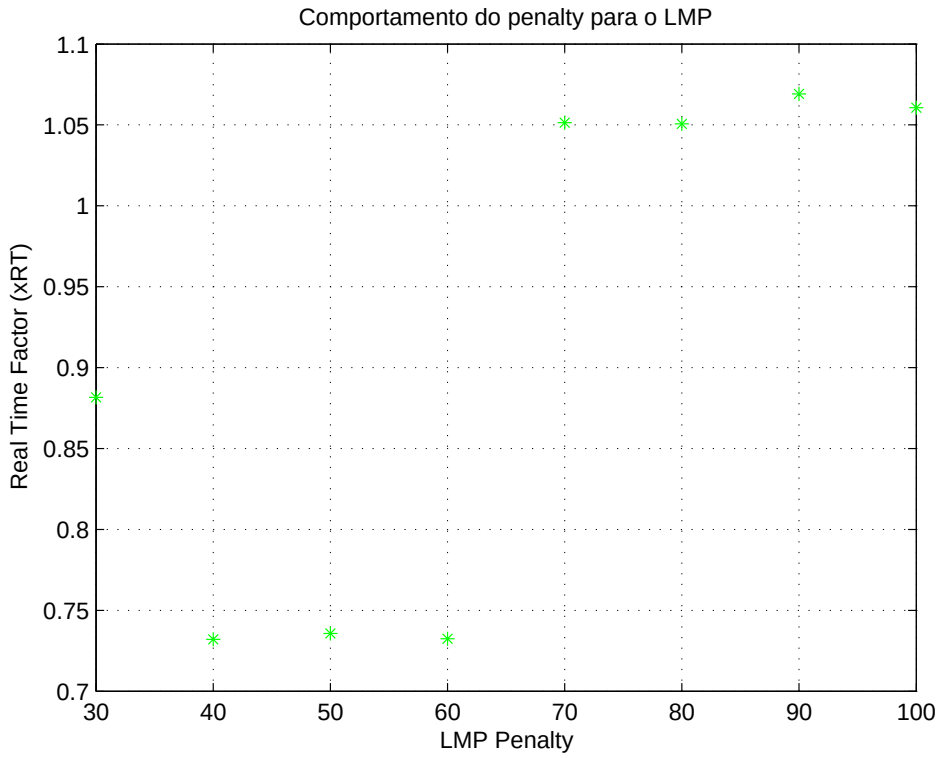


**Figura A.4:** Comportamento do Penalty do LM em relação a xRT.

# Apêndice B

# Tecnologias Mozilla

## B.1 Chrome.manifest

```
 content    primicias content/
locale    primicias    en-US    locale/en-US/
skin    primicias    classic/1.0    skin/
overlay chrome://messenger/content/msgAccountCentral.xul
chrome://primicias/content/msgoverlay.xul
overlay chrome://messenger/content/messenger.xul
chrome://primicias/content/voiceOverlay.xul
overlay chrome://messenger/content/messenger.xul
chrome://primicias/content/replyto_col_overlay.xul
overlay chrome://messenger/content/messenger.xul
chrome://primicias/content/teste.xul
overlay chrome://messenger/content/messengercompose/messengercompose.xul
chrome://primicias/content/composeoverlay.xul
resource primicias modules/
resource content content/
```

## B.2 Install.rdf

```
 <?xml version="1.0"?>
<RDF
    xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:em="http://www.mozilla.org/2004/em-rdf#">\hyphenation{A-PRO-VA-DA}
```

```
\hyphenation{Chro-me}
\hyphenation{cria-do}
\hyphenation{ou-tros}
\hyphenation{pa-râ-me-tros}
\hyphenation{geo-gra-fia}
\hyphenation{pe-ri-ó-di-cas}
\hyphenation{fa-ci-li-tar}
\hyphenation{de-co-di-fi-ca-dor}
\hyphenation{re-co-nhe-ci-men-to}
\hyphenation{ca-mi-nho}
\hyphenation{re-co-nhe-ci-da}
\hyphenation{PROPOR}
\hyphenation{Science}
\hyphenation{pró-xi-mo}
\hyphenation{es-pe-ra-do}
\hyphenation{cons-ti-tu-ir}
\hyphenation{des-cri-tos}
\hyphenation{fun-cio-na-li-da-de}
\hyphenation{rea-li-za}
\hyphenation{setEnabled}
\hyphenation{Microsoft}
\hyphenation{re-co-nhe-ce-dor}
\hyphenation{co-nhe-ci-do}
\hyphenation{pro-ble-ma}
\hyphenation{re-fe-ren-tes}
\hyphenation{La-bo-ra-tó-rio}
```

```
  <Description about="urn:mozilla:install-manifest">
    <em:name>Primicias</em:name>
    <em:id>josue@ufpa.br</em:id>
    <em:version>0.1</em:version>
    <em:creator>Josué Dantas</em:creator>
    <em:description>Manipulação do TB via voz</em:description>
    <em:homepageURL>http://www.laps.ufpa.br</em:homepageURL>
    <em:targetApplication>
      <Description>
```

```
        <em:id>{3550f703-e582-4d05-9a08-453d09bdfdc6}</em:id>
        <em:minVersion>1.5</em:minVersion>
        <em:maxVersion>13.1.*</em:maxVersion>
      </Description>
    </em:targetApplication>
  </Description>
</RDF>
```

## B.3 Classe SimpleRecognition modificada

Abaixo está a classe SimpleRecognition modificada para que ocorra a interação desta com o código da extensão que esta em escrito em JS.

```
import br.ufpa.laps.jlapsapi.recognizer.Listener2;
import javax.speech.Central;

import javax.speech.recognition.Recognizer;
import javax.speech.recognition.RecognizerModeDesc;
import javax.speech.recognition.Result;
import javax.speech.recognition.ResultAdapter;
import javax.speech.recognition.ResultEvent;
import javax.speech.recognition.ResultToken;
import javax.speech.recognition.RuleGrammar;
import javax.speech.recognition.DictationGrammar;

import netscape.javascript.JSObject;
import netscape.javascript.JSException;

import java.awt.Font;
import java.io.File;
import java.io.FileReader;

public class SimpleRecognition extends ResultAdapter {

static Recognizer rec;
static RuleGrammar gram;
```

```
static DictationGrammar dic;
static JSObject func;

public void resultAccepted(ResultEvent e) {
try {
Result r = (Result) (e.getSource());
ResultToken tokens[] = r.getBestTokens();
String resultado ="";
for (int i = 0; i < tokens.length; i++)
resultado += tokens[i].getSpokenText();
func.call("call", new Object[] {null,resultado });

}catch (JSException e1) {
e1.printStackTrace();
}
}

public static void Atualiza(JSObject temp){
try{
func = temp;
Chamar();
}catch(JSException e){
System.out.println(e.getMessage());
}
}
public static void Chamar() {


try {
Runtime.getRuntime().exec("rm -rf /tmp/jlapsapi");
RecognizerModeDesc rmd = (RecognizerModeDesc) Central
.availableRecognizers(null).firstElement();
rec = Central.createRecognizer(rmd);
rec.allocate();
FileReader reader = new FileReader(System.getProperty("user.home")+"/mail.grammar
gram = rec.loadJSGF(reader);
```

```
dic = rec.getDictationGrammar("dicSr");
dic.setEnabled(false);
gram.addResultListener(new SimpleRecognition());
} catch (Exception e) {
e.printStackTrace();
}
}
}
```

# Apêndice C

# Lista de sentenças da LaPSMail

abrir email
ler mensagens
abrir caixa de entrada
criar mensagem
criar nova mensagem
responder ao remetente
responder
responder a todos
enviar
salvar anexo
adicionar contato
inserir contato
inserir destinatario
anexar arquivo
salvar mensagem
salvar rascunho
próxima
próxima mensagem
anterior
mensagem anterior
organizar por ordem alfabética
organizar por data
organizar por remetente
exibir mensagens não lidas

próxima não lida
primeira mensagem
primeira não lida
última mensagem recebida
última mensagem não lida
não lidas
mover mensagem
encaminhar mensagem
encaminhar
excluir mensagem
deletar
enviar para lixeira
fechar
sair
descartar
procurar
spam
ler
pára
gravar
andrey
anderson
agnaldo
aldebaro
bruno
adalbery
ana carolina
fernanda
nelson
josué
renan
danilo
jonathas
lailson
gustavo
mariana

muller
diego
diogo
sílvia
nagib
marcos
kelly
mônica
guilherme
william
jefferson
claudomir
lucila
pedro
rodrigo
leonardo
claudio
fabiola
pelaes
rafael
suane
charles
marcel
ericson
igor
cleyton
hugo