Um Reconhecedor de Voz Livre para Português Brasileiro com Interface de Programação

Nelson Neto, Carlos Silva, Pedro Batista e Aldebaro Klautau
Universidade Federal do Pará (UFPA)
Laboratório de Processamento de Sinais (LaPS)
Belém-PA, CEP: 66075-110, Brasil
www.laps.ufpa.br
{nelsonneto,patrickalves,pedro,aldebaro}@ufpa.br

August 18, 2010

1 A Interface de Programação

Ao promover o amplo desenvolvimento de aplicações baseadas em reconhecimento de voz, os autores observaram que não era suficiente apenas tornar os recursos disponíveis, tais como modelos de linguagem. Esses recursos são úteis para pesquisadores, mas o que a maioria dos programadores desejam é a praticidade oriunda de uma interface de programação de aplicativos (API). Por isso, foi necessário complementar o código que faz parte do pacote de distribuição do Julius [1]. O Julius é um reconhecedor *open-source* de alta performance para grandes vocabulários.

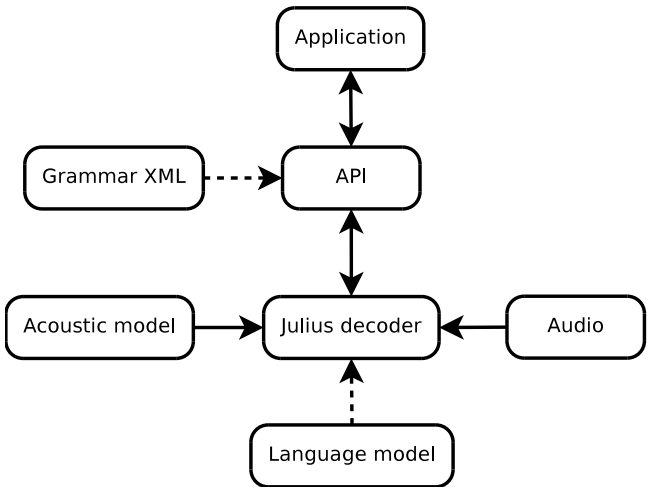

Figure 1: Modelo de interação com a API.

Assim, uma API foi desenvolvida na linguagem de programação C++ com a especificação *common language runtime*, que permite comunicação entre as lin-

guagens suportadas pela plataforma Microsoft .NET. A API proposta permite o controle em tempo-real do *engine* para reconhecimento de voz, Julius, e da interface de áudio do sistema. Como pode ser visto na Figura 1, as aplicações interagem com o reconhecedor Julius através da API. Basicamente, a API "esconde" do programador detalhes de baixo nível relacionados à operação do *engine*.

A API proposta é parte do Projeto FalaBrasil. Esse projeto não restringe o uso de seus recursos para fins comerciais e não-comerciais. Todos recursos desenvolvidos podem ser adquiridos na página do projeto [2]. Os usuários podem relatar problemas e sugestões através da lista de discussão do projeto [3].

A seguir os métodos e eventos da API serão apresentados. Mais detalhes podem ser encontrados em [4].

2 Métodos e Eventos

Visto que a API suporta objetos compatíveis com o modelo de automação *component object model* (COM), é possível acessar e manipular (i.e. ajustar propriedades, invocar métodos) objetos de automação compartilhados que são exportados por outras aplicações. Do ponto de vista da programação, a API consiste de uma classe principal denominada *SREngine*. Essa classe expõe à aplicação um conjunto de métodos e eventos descritos na Tabela 1.

Métodos/Eventos	Descrição Básica
SREngine	Método para carregar e inicializar o reconhecedor
loadGrammar	Método para carregar gramática SAPI XML
addGrammar	Método para carregar gramática nativa do Julius
startRecognition	Método para iniciar o reconhecimento
stopRecognition	Método para pausar/parar o reconhecimento
OnRecognition	Evento chamado quando alguma sentença é reconhecida
OnSpeechReady	Evento chamado quando o reconhecimento é ativado

Table 1: Principais métodos e eventos da API.

A classe *SREngine* permite que a aplicação controle aspectos do reconhecedor Julius. Essa classe possibilita que a aplicação carregue os modelos acústico e de linguagem a serem utilizados, inicie e pare o reconhecimento e receba eventos e resultados provenientes do *engine* de reconhecimento.

Uma aplicação baseada em voz precisa criar, carregar e ativar uma gramática, que essencialmente indica o método de reconhecimento empregado, ou seja, ditado ou livre de contexto. A gramática para ditado é implementada via modelo de linguagem, que define um extenso conjunto de palavras. Por sua vez, essas palavras podem ser pronunciadas de uma forma relativamente irrestrita. Já a gramática livre de contexto age como um modelo de linguagem. Ela provê ao reconhecedor regras que definem o que pode ser dito.

Através do método *loadGrammar* é possível carregar uma gramática livre de contexto especificada no formato Microsoft Speech API (SAPI) XML. Para tornar isso possível, um conversor gramatical foi desenvolvido e integrado ao método *loadGrammar*. Essa ferramenta permite que o sistema converta automaticamente uma gramática de reconhecimento especificada no padrão SAPI

Text Grammar Format [5] para o formato suportado pelo Julius[1]. O procedimento de conversão usa as regras gramaticais SAPI para encontrar as conexões permitidas entre as palavras, usando o nome das categorias como nós terminais. Isso também define as palavras candidatas em cada categoria, juntamente com as suas respectivas pronúncias.

É importante salientar que o conversor ainda não suporta regras gramaticais recursivas, facilidade suportada pelo Julius. Para esse tipo de funcionalidade deve-se carregar a gramática nativa do Julius através do método *addGrammar*. A especificação para esse tipo de gramática pode ser encontrada em [6].

O método *startRecognition*, responsável por iniciar o processo de reconhecimento, basicamente ativa as regras gramaticais e abre o *stream* de áudio. Similarmente, o método *stopRecognition* desativa as regras e fecha o *stream* de áudio.

Adicionalmente aos métodos, alguns eventos também foram implementados. O evento *OnSpeechReady* sinaliza que o *engine* está ativado para reconhecimento. Em outras palavras, ele surge toda vez que o método *startRecognition* é invocado. Já o evento *OnRecognition* acontece sempre que o resultado do reconhecimento encontra-se disponível, juntamente com o seu nível de confiança.

A medida de confiança do que foi reconhecido pelo *engine* é essencial para aplicações reais, dado que sempre ocorrerá erros de reconhecimento e, portanto, os resultados podem ser aceitos ou rejeitados. A seqüencia de palavras reconhecidas e o seu nível de confiança são passados da API para a aplicação através da classe *RecoResult*.

A seguir será comprovado que, fazendo uso do conjunto limitado de métodos e eventos apresentados acima, é viável construir compactas aplicações baseadas em voz com a API proposta.

3 PPTController

Para ilustrar a funcionalidade e a possibilidade de comunicação da API com outros programas, será apresentado o PPTController, ilustrado na Figura 2.

O PPTController é uma aplicação escrita na linguagem de programação C# que faz uso de recursos próprios construídos no Laboratório de Processamento de Sinais da Universidade Federal do Pará. Com esse aplicativo é possível controlar uma apresentação de *slides* do *Microsoft Office PowerPoint 2007* via comandos de voz. O usuário controla o documento através de comandos específicos:

- **Mostrar:** primeiro comando que deve ser enviado, pois abre o *slide show*.
- **Próximo ou Avançar:** ir para o próximo *slide* da apresentação.
- **Anterior ou Voltar:** voltar para o *slide* anterior na apresentação.
- **Primeiro:** ir imediatamente para o primeiro *slide* da apresentação.
- **Último:** ir imediatamente para o último *slide* da apresentação.
- **Fechar:** fechar a apresentação e voltar para a tela principal do aplicativo.

[1] O decodificador Julius suporta tanto modelos de linguagem, como gramáticas livre de contexto.

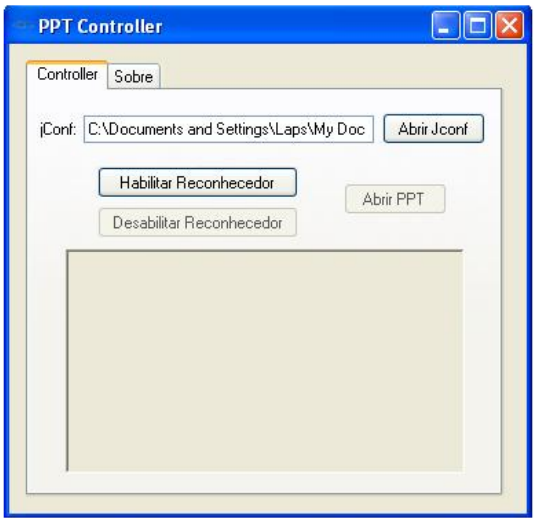

Figure 2: Tela principal do *software* PPTController.

O pacote de distribuição da API contém quatro DLL's: *julius.dll*, *sent.dll*, *mkfa.dll* e *LapsAPI.dll*. As três primeiras possuem códigos do Julius e devem ser copiadas para a seguinte pasta do sistema operacional: $\backslash WINDOWS\backslash system\backslash$. Já a última biblioteca é a API, propriamente dita, que deve ser incluída como referência no código fonte do aplicativo.

O reconhecedor é inicializado através do construtor *SREngine* que recebe como argumento o arquivo de configuração do Julius ("jConf"). Esse arquivo contém as especificações de todos os recursos utilizados pelo Julius durante o processo de reconhecimento de voz, como os modelos acústico e de linguagem. O modelo acústico independente de locutor foi construído com recursos próprios [7], usando arquivos de áudio re-amostrados para 22,050 Hz com 16 bits. A gramática livre de contexto foi elaborada seguindo a documentação do Julius [6]. Mais detalhes sobre o arquivo de configuração podem ser encontrados na Seção 5.

```
SREngine re;
re = new SREngine("C:/PPTController/pptConf/ppt.jconf");
```

Uma forma alternativa de especificar a gramática livre de contexto a ser utilizada durante o processo de reconhecimento é através do método *loadGrammar*. Para isso, primeiramente, é preciso salvar o código SAPI XML abaixo em um arquivo (p.e. comandos.xml).

```
<?xml version="1.0" encoding="utf-8" ?>
<GRAMMAR LANGID="416">
  <RULE NAME="COMMANDS">
    <LIST>
      <P>MOSTRAR</P>
      <P>ANTERIOR</P>
      <P>VOLTAR</P>
      <P>SAIR</P>
      <P>PRIMEIRO</P>
```

```
      <P>FECHAR</P>
    </LIST>
  </RULE>
</GRAMMAR>
```

Em seguida, basta utilizar o método *loadGrammar*, tendo como parâmetro de entrada o código SAPI XML. No momento, não é permitido o uso de caracteres especiais no arquivo XML.

```
re.loadGrammar("comandos.xml");
```

Lembrando que o método *addGrammar* aceita como entrada os arquivos que formam a gramática de reconhecimento do Julius. Ou seja, o código abaixo tem o mesmo efeito do método *loadGrammar* já citado, a diferença é que a gramática está no formato do Julius.

```
re.addGrammar("comandos", "comandos.dict", "comandos.dfa");
```

Todos os arquivos de gramática testados no PPTController podem ser encontrados na pasta */PPTController/pptGrammar/*.

Com o reconhecedor instanciado deve-se delegar para o *SREngine* quais funções vão representar os eventos, ou seja, o que acontecerá quando o reconhecimento for ativado ou quando alguma sentença for reconhecida, eventos *OnSpeechReady* e *OnRecognition*, respectivamente.

```
SREngine.OnRecognition += sr_onRecognition;
re.OnSpeechReady += sr_speechReady;
```

Ao evento *OnSpeechReady* deve ser passada uma função sem argumentos, informando que o Julius está ativado para reconhecimento. Já o evento *OnRecognition* recebe uma função com o argumento *RecoResult*. A *RecoResult* é uma classe da API produzida sempre que uma sentença é reconhecida. Ela possui informações sobre o reconhecimento, como a própria sentença e o seu nível de confiança informado pelo reconhecedor. O uso das funções pode ser conferido no código abaixo.

```
void sr_onRecognition(RecoResult result){
  if (result.getConfidence() > 0.7)
    Actions(result.getUterrance());
}
void sr_speechReady(){
  recogetionRichTextBox1.AppendText("Reconhecendo\n");
}
```

A função *Actions*, presente no código acima, é responsável por controlar as funcionalidades do programa *Microsoft PowerPoint*. Para isso, ela faz uso de componentes integrados dentro de uma arquitetura *object linking and embedding* (OLE). Essa comunicação não será aqui documentada, porém seu código é de fácil compreensão e encontra-se disponível no pacote do PPTController.

Assumindo que o reconhecedor está instanciado, o método *startRecognition* pode ser invocado para iniciar o processo de reconhecimento, assim como o método *stopRecognition* para parar o reconhecimento. No PPTController, especificamente, esses métodos são chamados a partir de ações de dois botões mostrados abaixo.

```
void button1_Click(object sender, EventArgs e){
  re.startRecognition();
}
void button2_Click(object sender, EventArgs e){
  re.stopRecognition();
}
```

O aplicativo desenvolvido, PPTContoller, é um *software* livre e encontra-se disponível na página do Projeto FalaBrasil [2], juntamente com as versões mais atualizadas da API e dos recursos construídos.

4 Condições de Instalação e Operação

A API foi originalmente projetada para o sistema operacional Microsoft Windows 32 bits (x86). A Tabela 2 mostra as configurações recomendadas para o correto funcionamento da API.

Características	Especificação
Espaço em disco rígido	50 MB
Memória RAM	2 GB
Sistema operacional	Windows XP, Vista ou 7
Processador	Intel(R) Pentium Dual Core 1.80GHz
Placa de som	16-bits
Hardware especial	Microfone
PPTController: software especial	Microsoft Office PowerPoint 2007

Table 2: Principais condições de instalação e operação da API.

Contudo, uma versão da API para ambiente Linux 32-bit também foi desenvolvida. Seu princípio de funcionamento é o mesmo da versão para Windows, descrita na Seção 1. A diferença básica é que as ações são passadas para a API através de ponteiro de função, usando o método *setOnRecognizeAction*.

```
void reconheceu(RecoResult *result){
   cout << result->getUterrance() << " Confiança "
      << result->getConfidence() << endl;
}

re->setOnRecognizeAction(&reconheceu);
```

Para usar a biblioteca compartilhada (*libLapsAPI.so*) deve-se copiá-la para */usr/lib* e passá-la como parâmetro para o compilador. O código abaixo mostra como compilar a função *main*.

```
080800027O1@laps02:/Coruja_Linux\$ gcc -ILapsAPI/include/
-Ijulius-4.1.3/libjulius/include/ -Ijulius-4.1.3/libsent/include/
-o main main.cpp -LLapsAPI/Release/ -lLapsAPIJulius4.1.3

main.cpp: In function int main():
main.cpp:15: warning: deprecated conversion from string constant to char*
```

Além do compilador *gcc* (versão 4.2.4), é fundamental que o compilador *g++* (versão 4.2.4) também esteja instalado na máquina. O compilador *g++* é necessário para manipular os dados da biblioteca. Caso não se tenha acesso a pasta de bibliotecas, pode-se sobrescrever a variável *LD_LIBRARY_PATH*, através do comando *export*.

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:"caminho/para/libLapsAPI.so"
```

5 Extras

Um erro comum de compilação do programa PPTController é o endereçamento errado das referências. Dito isso, as DLL's abaixo precisam estar adicionadas ao projeto:

- *LapsAPI*: encontrada na pasta */LaPSAPI/* presente no pacote da API.
- *Microsoft.Office.Core* e *Microsoft.Office.Interop.PowerPoint*: por exemplo, para adicionar as DLL's no ambiente de programação Microsoft Visual Studio, siga os passos abaixo:

  1. Click com o botão direito no item *References* presente na janela *Solution Explorer*, que pode ser visualizada na Figura 3;
  2. Seleciona a opção *Add References*;
  3. Na janela que se segue, dentro da aba COM, procure e selecione o objeto *Microsoft PowerPoint 12.0 Object Library*.

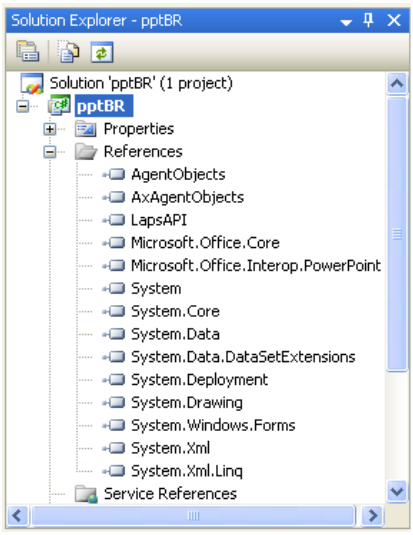

Figure 3: Exemplo da janela *Solution Explorer*.

O arquivo de configuração para executar o PPTController pode ser encontrado na pasta */PPTController/pptConf/*. Caso ocorra algum erro durante o carregamento das configurações, verifique o endereçamento dos modelos acústico

e de linguagem dentro do arquivo de configuração, pois dependendo do diretório que o sistema tome como base para a aplicação, será preciso alterar o caminho dos arquivos abaixo:

```
################### grammar path ########################
-dfa "C:/Coruja0.2/Gramatica/comandos.dfa"
-v "C:/Coruja0.2/Gramatica/comandos.dict"

################ acoustic model path ###################
-h "C:/Coruja0.2/LaPSAM1.5/LaPSAM1.5.am.bin"
-hlist "C:/Coruja0.2/LaPSAM1.5/tiedlist1.5.txt"
-htkconf "C:/Coruja0.2/LaPSAM1.5/edaz.conf"
```

O arquivo de log */PPTController/pptConf/JuliusLog* pode ajudar a encontrar a origem do erro.

References

[1] "http://julius.sourceforge.jp/en/," Visited in May, 2010.

[2] "http://www.laps.ufpa.br/falabrasil," Visited in June, 2010.

[3] "groups.google.com/group/coruja-users?hl=pt-br," Visited in June, 2010.

[4] P. Silva, P. Batista, N. Neto, and A. Klautau, "An open-source speech recognizer for brazilian portuguese with a windows programming interface," *The International Conference on Computational Processing of Portuguese*, 2010.

[5] "msdn.microsoft.com/en-us/ms723632(vs.85).aspx," Visited in June, 2010.

[6] "julius.sourceforge.jp/en_index.php?q=en_grammar.html," Visited in May, 2010.

[7] P. Silva, N. Neto, and A. Klautau, "Novos recursos e utilização de adaptação de locutor no desenvolvimento de um sistema de reconhecimento de voz para o Português Brasileiro," *In XXVII Simpósio Brasileiro de Telecomunicações, Blumenau, Brasil*, 2009.