

An Open-Source Speech Recognizer for Brazilian Portuguese with a Programming Interface

Nelson Neto, Carlos Silva, Pedro Batista e Aldebaro Klautau
Federal University of Pará (UFPA)
Signal Processing Laboratory (LaPS)
Belém-PA, CEP: 66075-110, Brazil
www.laps.ufpa.br
{nelsonneto,patrickalves,pedro,aldebaro}@ufpa.br

August 18, 2010

1 Proposed Programming Interface

While promoting the widespread development of applications based on speech recognition, the authors noted that it was not enough to make available resources such as language models. These resources are useful for speech scientists but most programmers demand an easy-to-use application programming interface (API). Hence, it was necessary to complement the documentation and code that is part of the Julius package [1]. Julius is an open-source high-performance decoder for speech-related applications.

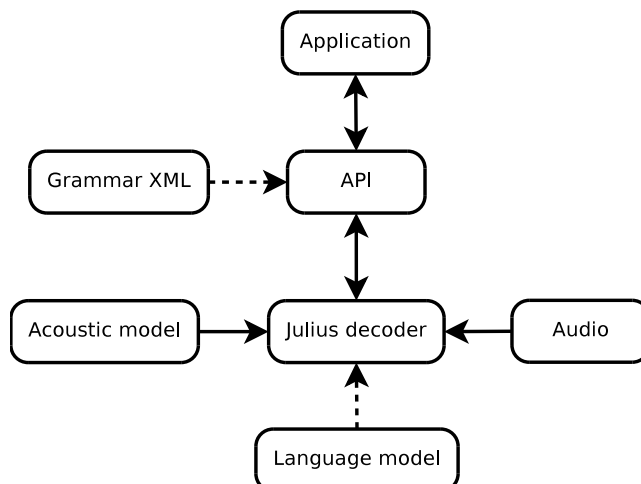


Figure 1: The designed API for easing the task of driving the speech recognizer.

In order to use the Julius speech decoder in practical applications, an API was developed in the C++ programming language with the common language

runtime specification, which enables communication between the languages supported by the Microsoft .NET platform. The proposed API allows the real-time control of the Julius speech recognition engine and the audio interface. As shown in Figure 1, the applications interact with the Julius decoder through the API. The API hides from the user the low level details of the engine operation.

The proposed API is part of the FalaBrasil Project. The free license allows the products and/or derivatives to be commercially sold. This way the community can get the open-source developed resources [2] and share the knowledge gained through the FalaBrasil mailing list [3].

In the sequel the API methods and events are described. More details can be found in [4].

2 Methods and Events

Since the API supports the component object model (COM) automation, it is possible to access and manipulate (i.e. set properties of or call methods on) shared automation objects that are exported by other applications. From a programming point of view, the API consists of a main class referred to as *SREngine*. This class exposes to the applications a set of methods and events that are described in Table 1.

Methods and Events	Basic Description
SREngine	Method to setup and control aspects of the engine
loadGrammar	Method to load a SAPI XML grammar file
addGrammar	Method to load native Julius grammar files
startRecognition	Method to start the recognition process
stopRecognition	Method to stop the recognition process
OnRecognition	Event to receive the recognition results
OnSpeechReady	Event to indicate that the engine is active

Table 1: Main API methods and events.

The *SREngine* class enables applications to control aspects of the Julius decoder, from which the application can load the acoustic and language models to be used, start and stop recognition, and receive events and recognition results.

A speech application must create, load, and activate a grammar, which essentially indicates what type of utterances to recognize, i.e., dictation or context-free grammar. The dictation grammar is implemented via language models, which define a large set of words that may be spoken in a relatively unrestricted way. The context-free grammar acts as the language model. It provides the recognizer with rules that define what the user is expected to say.

The *loadGrammar* method loads a context-free grammar file specified in Microsoft Speech API (SAPI) XML format. To make this possible, a flexible converter was developed by the authors. This toolkit allows users to convert a recognition grammar specified in XML, according to the SAPI Text Grammar Format [5], into the Julius format¹. The conversion procedure uses the SAPI grammar rules to find the allowed connection of words, using word category

¹Julius supports both language model and context-free grammars.

names as terminal symbols. It also defines word candidates in each category, with their pronunciation information.

It should be noted that the converter does not support recursive rules in the grammar, a feature that is supported by Julius. To do this, you may use the *addGrammar* method. So, the *addGrammar* method was developed to load the Julius grammar files. The grammar text format for the Julius decoder is specified in [6].

The *startRecognition* method, responsible for start recognition, basically activates the grammar rules and opens the audio stream. Similarly, the *stopRecognition* method deactivates the rules and closes the audio stream.

In addition to the methods, some events were also built. The *OnSpeechReady* event signals that the engine is active to recognize. In other words, it occurs whenever the *startRecognition* method is invoked. Now the *OnSRRecognition* event occurs whenever a recognition result is available with an associated confidence measure.

A confidence measure of the recognition results is essential to real applications because there are always recognition errors and therefore the recognition results need to be accepted or rejected. The utterance and confidence score are passed from API to application through the *RecoResult* class.

With the limited set of methods and events presented above it is easy to build compact speech recognition applications using the Julius decoder. The next section presents a speech application using the proposed API.

3 PPTController

The PPTController software, illustrated in Figure 2, makes use of own resources built in the Signal Processing Laboratory of the Federal University of Pará.



Figure 2: Main screen of the PPTController software.

The PPTController was developed in the C# programming language. The

user can control a Microsoft Office PowerPoint 2007 slide presentation via voice. The available commands are:

- **Mostrar:** first command to be sent, it opens the slide show.
- **Próximo or Avançar:** go to the next slide in the presentation.
- **Anterior or Voltar:** back to the previous slide in the presentation.
- **Primeiro:** go immediately to the first slide.
- **Último:** go immediately to the last slide.
- **Fechar:** close the presentation and return to the application's main screen.

The API distribution package has four DLL's: *julius.dll*, *sent.dll*, *mkfa.dll*, and *LapsAPI.dll*. The first three libraries contain the Julius package and must be copied to the system folder: `\Windows\system\`. The last library is the API by itself, and has to be included as a reference in the project.

The recognizer is initialized by the *SREngine* constructor that takes as argument a configuration file ("jConf"). This particular file contains the specifications of all resources used by Julius during the speech recognition process such as acoustic and language models. The speaker independent acoustic model was built with own resources [7]. The used sampling rate was 22,050 Hz and each sample was represented with 16 bits. The context-free grammar was developed following the documentation found in [6]. See Section 5 for extra details about the configuration file.

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

An alternative way of specifying the context-free grammar to be used during the recognition process is through the *loadGrammar* method. Firstly it is necessary save the SAPI XML code below into a file (e.g. 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>
```

Then, the *loadGrammar* method is used to load the grammar file. Actually, special characters are not permitted.

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

Note that the code below has the same effect as the *loadGrammar* method already cited. The difference is the input files specified in Julius format.

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

The grammar files used by the PPTController software can be found in the folder: */PPTController/pptGrammar/*.

After instantiating the constructor, the functions that will represent its events should be delegated.

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

A function without arguments should be passed to the *OnSpeechReady* event, stating that the engine is active to recognize. Now the *OnRecognition* event receives the *RecoResult* class as an argument. The *RecoResult* class loads information about the utterance and confidence score reported by the Julius decoder.

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

The *Actions* function, seen in the code above, is responsible to control the Microsoft PowerPoint program features. For this, it makes use of automation objects. This communication is not documented here, but your code is easy to understand and is available in the PPTController package.

Finally, the *startRecognition* method can be invoked to start the recognition process, as well as the *stopRecognition* method to stop it. These methods are called from actions of two buttons shown below.

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

A compact programming interface for speech application development using Julius decoder was briefly presented. All resources, including the proposed API, can be found in the FalaBrasil Project web site [2].

4 Installation and Operating Requirements

The API was originally designed for the Microsoft Windows 32-bit (x86) operational systems. Table 2 shows the main API required settings.

Currently the API also supports Linux 32-bit. This version works the same way as the Windows version, described in Section 1. The main difference is that the actions are now passed to the API through function pointer, using the *setOnRecognizeAction* method.

Features	Specification
Hard disk	50 MB
Random-access memory (RAM)	2 GB
Operacional system	Windows XP, Vista ou 7
Processor	Intel(R) Pentium Dual Core 1.80GHz
Sound card	16-bits
Particular hardware	Microphone
PPTController: particular software	Microsoft Office PowerPoint 2007

Table 2: Main API required settings.

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

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

The shared library (*libLapsAPI.so*) must be copied to the */usr/lib* system folder and passed as parameter to the compiler. The code bellow shows how to compile the *main* function.

```
08080002701@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*
```

Besides the *gcc* compiler (version 4.2.4), it is necessary that the *g++* compiler (version 4.2.4) is also installed. The *g++* compiler is required to manipulate the library data. The access to the libraries folder could be denied. In this case, it's possible to overwrite the *LD_LIBRARY_PATH* variable.

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:"path/to/libLapsAPI.so"
```

5 Extras

While building the PPTController solution, a common error is address wrong references in the project. The DLL's below must be added:

- *LapsAPI*: can be found in the API package folder: */LaPSAPI/*.
- *Microsoft.Office.Core* and *Microsoft.Office.Interop.PowerPoint*: for example, using the Microsoft Visual Studio software package, the next steps must be followed to add the DLL's:

1. Right-click on *References* item in the *Solution Explorer* window (see Figure 3);

2. Choice the *Add References* option;
3. On a new tabbed window, inside the COM tab, search and select the *Microsoft PowerPoint 12.0 Object Library* object.

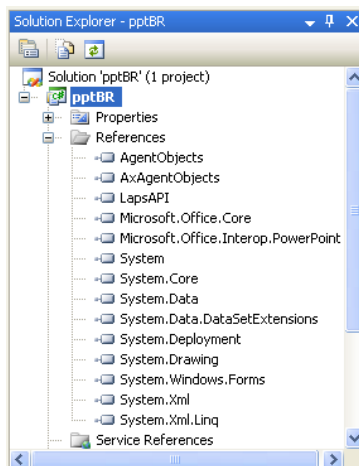


Figure 3: Solution Explorer window.

The configuration file used by PPTController can be found in the folder: */PPTController/pptConf/*. Before load the settings, verify the acoustic and language models address inside the configuration file. Maybe it will be necessary to change the path of the files below:

```
##### 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"
```

The log file */PPTController/pptConf/JuliusLog* can be useful as a way to find the error source.

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.