SpeechOO: Uma Extensão de Ditado para o LibreOffice

Pedro Batista[1], **William Colem**[2], **Rafael Oliveira**[1],
Hugo Santos[1], **Welton Araújo**[1], **Nelson Neto**[1], **Aldebaro Klautau**[1]

[1] Laboratório de Processamento de Sinais (LaPS) – Universidade Federal do ará (UFPA)
http://www.laps.ufpa.br

[2]Centro de Competencia em Software Livre (CCSL) – Universidade de São Paulo (USP)
http://ccsl.ime.usp.br/
pedro@ufpa.br,william.colen@gmail.com,
{rafaelso,hugosantos,waraujo,nelsonneto,aldebaro}@ufpa.br

Abstract. *People with special needs encounter serious difficulties in having access to computers. In this scenario, speech technologies may be seen as adaptive and contribute to increased autonomy and social inclusion of people with disabilities. The use of computers by physically impaired people has been a problem. This work presents SpeechOO, an extension for the Writer application in the LibreOffice office suite. Using this plug-in, physically impaired people can control the Writer software, dictate and edit text via speech commands in Brazilian Portuguese.*

Resumo. *Pessoas com necessidades especiais encontram uma série de dificuldades no acesso a computadores. Nesse cenário, tecnologias de fala podem ser vistas como acessivas e contribuem para o crescimento de autonomia e inclusão social de pessoas com essas necessidades. Esse trabalho apresenta o SpeechOO, uma extensão para o aplicativo Writer do pacote de escritório LibreOffice. Usando esta extensão, pessoas com deficiência física podem controlar o aplicativo Writer, ditar e editar texto através de comandos de voz em Português Brasileiro.*

1. Introdução

Este trabalho apresenta o SpeechOO, uma extensão para o programa Writer do pacote LibreOffice, que permite sua utilização a partir de interface de voz. Atualmente não é de conhecimento dos autores nenhum trabalho similar para o LibreOffice. Exite porém trabalhos da Microsoft [win 2010] e Nuance [dra 2010] para o editor de texto Word do pacote Microsoft Office, porém ambos possuem código fechado.

É importante notar o papel social desse trabalho, no sentido de acessibilidade, pois a partir do SpeechOO, pessoas com dificuldades ou impossibilidades de usar o teclado do computador podem editar textos no computador. É chamada atenção também para o potencial de tecnologias de fala mundialmente, para acessibilidade e comodidade. No Brasil a questão acessiva recebe dedicação especial, visto que, segundo o Instituto Brasileiro de Geografia Estatística (IBGE), existe mais de 9 milhões de pessoas com dificuldades físicas no Brasil [IBGE 2010]. A seguir é discutido como o trabalho será apresentado.

Inicialmente, será descrita a Java Speech API, uma especificação para o uso de tecnologias de voz em ambiente Java. Em seguida, é descrita a especificação Universal

Network Objects (UNO), que permite o desenvolvimento de extensões para o LibreOffice. Finalmente, é apresentada a JLaPSAPI, interface desenvolvida para permitir o uso do Coruja [Silva et al. 2010] através da especificação JSAPI.

2. JSAPI: Java Speech API

A Java Speech API (JSAPI) é uma especificação que permite aos desenvolvedores incorporarem tecnologia de voz aos seus aplicativos escritos no ambiente de programação Java. A JSAPI surgiu de uma parceria entre grandes empresas, como a Apple Computer, AT&T, Dragon Systems, IBM Corporation, dentre outras. Dessa cooperação surgiram, além da especificação, *engines* de voz implementando o padrão proposto, ambos visando facilitar o desenvolvimento de aplicativos baseados em voz na plataforma Java.

A JSAPI é independente de plataforma com suporte a sistemas para ditado, comando e controle e síntese de voz. Desse ponto em diante, o termo JSAPI será utilizado como referência a parte ASR dessa API. Basicamente, a JSAPI é utilizada para fazer a interface entre a aplicação Java e o reconhecedor, disponibilizando um conjunto de classes que representam a visão do programador sobre o *engine*, conforme a Figura 1. A maior parte dos produtos que implementam JSAPI são comerciais, e a *Sun* (líder do projeto), não possui nenhuma implementação de referência. A única implementação para ASR de código livre encontrada é o Sphinx-4.

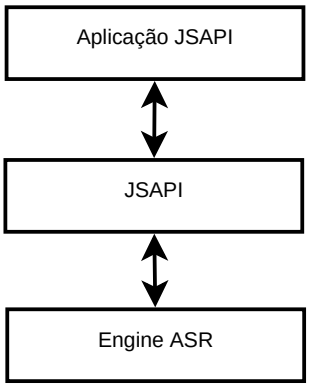

Figura 1. Arquitetura de alto nível de uma aplicação usando JSAPI.

2.1. Usando um *engine* ASR através da JSAPI

Para uma melhor compreensão da especificação JSAPI, serão descritos a seguir seus principais recursos que possibilitam a utilização de reconhecimento de voz em aplicativos Java.

A classe *Central* da JSAPI é usada para encontrar e criar o reconhecedor. Essa classe espera encontrar o arquivo `speech.properties`, provido pelo *engine*, nos diretórios `user.home` ou `java.home/lib` do sistema operacional. Seu método *createRecognizer* é responsável por instanciar um reconhecedor, descrito pela classe *RecognizerModeDesc* que é o responsável por receber as características que o reconhecedor deve possuir.

A classe *RecognizerModeDesc* é responsável por exemplo, por definir o idioma para o qual queremos o reconhecedor, e se este deve suportar ou não o modo de ditado.

Caso a aplicação não especifique a língua do reconhecedor, a língua padrão do sistema, retornada por *java.util.Locale.getDefault()*, será utilizada. Se mais de uma *engine* suportar a língua padrão, a classe *Central* dará preferência a um *engine* que esteja *runni ng* (em espera), e então a um que dê suporte a língua definida no *locale*. O objeto *EngineList*, descrito a seguir, surge como um possibilidade de visualização dos reconhecedores disponibilizados, facilitando o estudo e a escolha do *software* que melhor se adequá à aplicação.

```
RecognizerModeDesc required = new RecognizerModeDesc();
required.setLocale(new Locale("en", ""));
required.setDictationGrammarSupported(Bollean.TRUE);
EngineList engineList =
    Central.availableRecognizers(required);
for (int i = 0; i < engineList.size(); i++) {
    RecognizerModeDesc desc =
        (RecognizerModeDesc) engineList.get(i);
    System.out.println(desc.getEngineName() +
            desc.getLocale());
}
```

Escolhido o reconhecedor o próximo passo é criá-lo, através do método *createRecognizer* da classe *Central*. Os métodos *allocate* e *resume* alocam todos os recursos necessários e prepararam o reconhecedor para o reconhecimento, respectivamente. O código abaixo ilustra a criação do *engine*.

```
static Recognizer rec;
rec = Central.createRecognizer(required);
rec.allocate();
```

Uma gramática de regras suporta o diálogo discreto entre uma aplicação e o usuário. O que pode ser dito é definido explicitamente através de um conjunto de regras descritas, utilizando-se uma linguagem denominada Java Speech Grammar Format (JSGF) [jsg 1998]. Outra alternativa se dá através de objetos e métodos da própria JSAPI. As regras de uma gramática podem ser carregadas em tempo de execução através de um arquivo de texto, uma URL ou uma regra por vez. O código abaixo mostra uma gramática sendo carregada a partir de um arquivo contendo regras no formato da JSGF.

```
FileReader reader = new FileReader("myGrammar.jsgf");
RuleGrammar gram = rec.loadJSGF(reader);
```

O trecho de código a seguir ilustra a criação de uma gramática e adição de uma regra com a palavra “iniciar”, semelhante ao que foi feito acima, porém através de métodos que permitem a especificação de uma gramática via código fonte:

```
RuleGrammar gram = rec.newRuleGrammar("myGrammar");
RuleToken word = new RuleToken("iniciar");
gram.setRule("ruleName", word, true);
```

A linha de código a seguir cria uma instância de gramática para ditado. Tal estrutura suporta a entrada de texto sem formato pré-determinado. O usuário pronuncia as

palavras uma após a outra em qualquer sequência desde que pertençam ao domínio em uso. Palavras não pertencentes ao conjunto, geram resultados incorretos ou são simplesmente ignorados.

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

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

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

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

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

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

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

O sucesso (*resultAccepted*) ou fracasso (*resultRejected*) do reconhecimento não é determinado por um patamar único. De fato, este índice pode ser estabelecido através do método *setConfidenceLevel* que aceita um valor variando de *0.0* até *1.0*, sendo o valor padrão igual a *0.5*. Assim, o nível de confiabilidade é o limite entre o que é aceito e o que é rejeitado. Por exemplo, em um nível de confiabilidade mínimo, todos os resultados são tratados como *resultAccepted*. O nível de sensibilidade é o limite entre o som que deve ser processado e o que deve ser ignorado. É classificado com um valor que vai de *0.0* até *1.0*, respectivamente referente aos sons mais baixos e mais altos. Seu valor padrão é *0.5*, e quanto mais baixo for esse parâmetro, mais alto o usuário terá que falar.

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

Os reconhecedores geram, além dos resultados referentes às gramáticas, eventos de áudio (*RecognizerAudioEvent*) de forma que as aplicações possam receber informações sobre a entrada do som. São três eventos gerados: um com informações sobre o volume da entrada do som, e outros dois quando se tem o início ou o fim de um diálogo. Estes eventos são gerados automaticamente e, para serem utilizados, basta que se associe um ouvidor de áudio (*RecognizerAudioListener*) ao reconhecedor.

3. Desenvolvimento de extensões para o LibreOffice

As extensões (*plug-in*) são cada vez mais importantes no universo de aplicativos, pois elas possibilitam que qualquer programador adicione funcionalidades a um determinado programa sem grandes esforços. Outro motivo do sucesso das extensões é a sua fácil e rápida instalação, pois elas são apenas acopladas a um *software* principal.

Podemos observar o quanto extensões são importantes visitando o *site* da comunidade LibreOffice[1], onde podemos encontrar mais de 100 extensões para esse aplicativo. Outro exemplo de sucesso com extensões é o navegador de internet Mozilla Firefox[2], para o qual podemos encontrar milhares de extensões, com algumas sendo utilizadas por milhões de usuários.

As próximas seções descrevem o pacote de ferramentas para escritório LibreOffice e a especificação Universal Network Objects (UNO), exemplificando o uso de componentes UNO no desenvolvimento de extensões.

3.1. A suíte de escritório LibreOffice

O LibreOffice é um pacote de ferramentas livre e de código aberto para escritório, desenvolvido pela The Document Foundation surgindo de um projeto paralelo ao OpenOffice [lib 2011]. Com o LibreOffice é possível editar textos, fazer planilhas, gráficos, cálculos, apresentação de *slides*, dentre diversas outras funcionalidades. Este é compatível com os principais sistemas operacionais (Windows, MAC OS X, Linux, BSD e UNIX) e também com arquivos de outras ferramentas de escritório como o Microsoft Office, OpenOffice, iWork e GoogleDocs.

O LibreOffice foi escrito principalmente em linguagem de programação C++. No entanto, é possível desenvolver extensões em diversas linguagens, como por exemplo, Java, C++ e Python. Essa diversidade é possível graças a especificação UNO, que define componentes que são implementados em diversas linguagens e são capazes de se comunicar entre si.

3.2. UNO: Universal Network Objects

O LibreOffice fornece uma API para desenvolvimento de extensões independente de linguagem de programação. Isso é possível graças a especificação UNO, que permite a comunicação entre componentes implementados em qualquer linguagem, para a qual existe uma implementação UNO. Atualmente, é possível encontrar extensões em Java, C++, Python, entre outras.

O LibreOffice foi construído a partir de bibliotecas em C++, então uma mudança nessa base exigiria que grande parte do programa fosse recompilado, o que levava cerca

[1]http://extensions.libreoffice.org/extension-center
[2]https://addons.mozilla.org/en-US/firefox/

de 2 dias, isso se nenhum problema fosse encontrado. Esse é apenas um dos problemas que levaram a criação de um modelo de componentes.

O UNO veio então para permitir a adição de componentes por meio de interfaces. O UNO é composto principalmente por uma especificação binária, onde é definido o *layout* de memória de seus tipos. Cada objeto possui seu próprio ambiente, e a comunicação com o restante do programa é feita através de uma ponte UNO. A ponte UNO permite a comunicação inclusive de programas escritos em diferentes linguagens de programação, visto que a especificação de memoria é única. Como ilustrado na Figura 2, um componente se comunica com o ambiente UNO que fará a ponte entre esse e qualquer outro componente. Componentes UNO são implementações da especificação UNO, que pode ser o programa principal (principal funcionalidade do programa) e extensões, que são *features* adicionais ao programa principal.

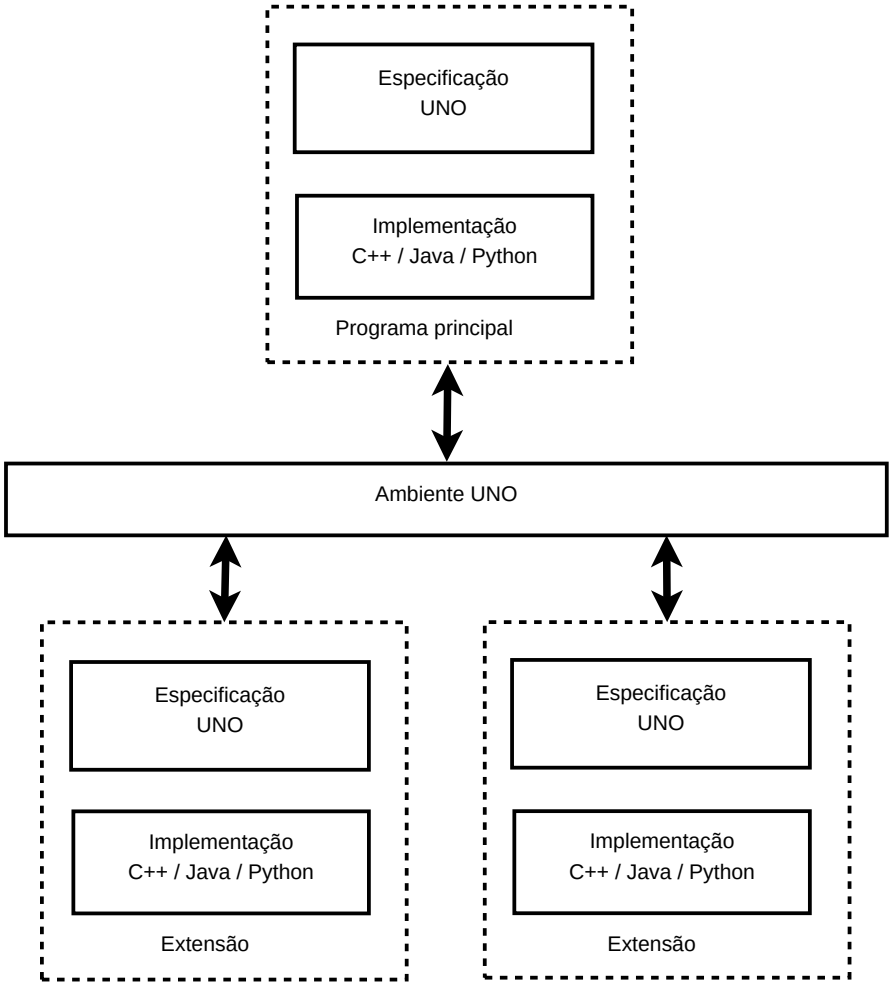

Figura 2. Arquitetura do UNO.

A seguir é apresentado um código em Java usando a especificação UNO. Esse código tem como principal objetivo mostrar na tela do processador de texto Writer do LibreOffice, a *String* passada como argumento para o método *insertNewSentence* da classe *InputSentence* também definida abaixo.

```
package org.speechoo.inputText;
```

```
import com.sun.star.frame.XController;
import com.sun.star.text.XText;
import com.sun.star.text.XTextDocument;
import com.sun.star.text.XTextViewCursor;
import com.sun.star.text.XTextViewCursorSupplier;
import com.sun.star.uno.UnoRuntime;
import org.speechoo.SpeechOO;
public class InputSentence {
  public void insertNewSentence(String sentence) {
    XTextDocument xDoc = (XTextDocument) UnoRuntime.queryInterface(
        XTextDocument.class, SpeechOO.m_xFrame.getController().getModel());
    XText xText = xDoc.getText();
    XController xController = xDoc.getCurrentController();
    XTextViewCursorSupplier xViewCursorSupplier =
        (XTextViewCursorSupplier) UnoRuntime.queryInterface(
            XTextViewCursorSupplier.class, xController);
    XTextViewCursor xCursor2 = xViewCursorSupplier.getViewCursor();
    xText.insertString(xCursor2, sentence, true);
    xCursor2.gotoRange(xCursor2.getEnd(), false);
  }
}
```

4. JLaPSAPI: Utilizando o Coruja pelo padrão JSAPI

A JLaPSAPI é uma API proposta, desenvolvida a partir da LaPSAPI, que permite o uso do decodificador Julius (e consequentemente do *engine* Coruja) em aplicativos Java. Apesar da LaPSAPI descrita em [Silva et al. 2010] ser suficiente e robusta para o desenvolvimento de aplicativos, esta não segue nenhuma especificação reconhecida, ou seja, um programa escrito em cima dessa API, não poderia trocar seu *engine* ASR. Outro problema é que não há suporte a uma linguagem *cross-platform* como a linguagem de programação Java.

Assim, surgiu a necessidade de se construir uma nova API em cima da LaPSAPI seguindo a especificação JSAPI, a qual chamamos JLaPSAPI. A JLaPSAPI é de grande importância para o SpeechOO, pois, dessa forma, este pode usar diferentes *engines*, inclusive em diferentes línguas. Além de ser importante para que o SpeechOO possa ser utilizado em diferentes sistemas operacionais, já que seu programa principal (LibreOffice) também é *cross-platform*.

A JLaPSAPI exigiu a comunicação entre código Java e C++ (LaPSAPI). Essa comunicação foi provida pela Java Native Interface (JNI)[Liang 1999]. Nessa nova arquitetura o acesso ao *engine* é feito através de código especificado pela JSAPI. Como mostrado na Figura 3, o programador agora tem a possibilidade de alternar entre o Coruja e qualquer outro *engine* que siga a especificação JSAPI, sem a necessidade de alteração no código de sua aplicação Java.

A JLaPSAPI vem sendo constantemente desenvolvida para oferecer todas as funcionalidades demandadas pelo SpeechOO, uma aplicação conceitualmente mais complexa quanto à avaliação do texto reconhecido.

5. SpeechOO

O SpeechOO possui dois modos de funcionamento: ditado e controle. No modo ditado, a fala do usuário é convertida em texto, que então é adicionado ao corpo do documento no

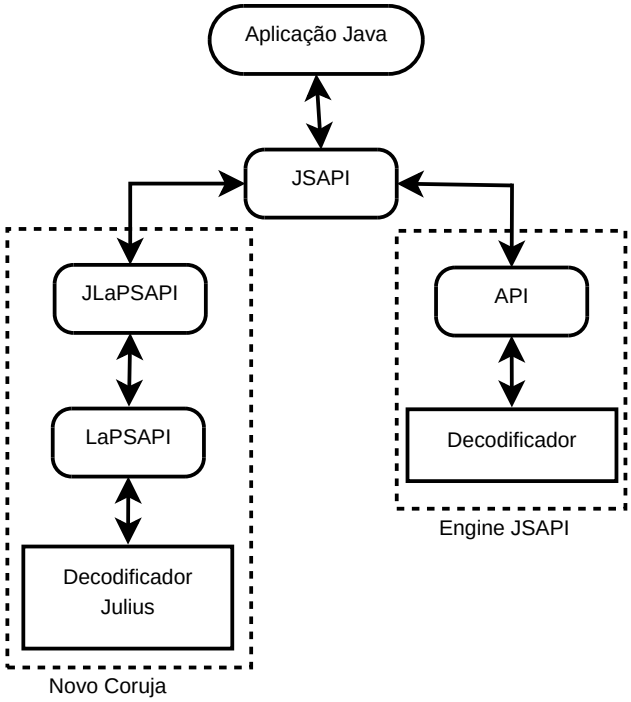

Figura 3. Arquitetura da JLaPSAPI.

Tabela 1. Métodos e eventos suportados pela JLaPSAPI.

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

LibreOffice Writer. Já no modo comando, a fala do usuário é interpretada como comandos para o LibreOffice Writer, através dos quais é possível desde mudar a formatação do texto até salvar o documento. Um vídeo de demonstração e instalação do SpeechOO está disponível em [spe 2011]. Nas próximas seções a arquitetura do SpeechOO é discutida (Seção 5.1) bem como cada um dos seus modos de funcionamento (Seções 5.2 e 5.3).

5.1. Arquitetura do SpeechOO

A Figura 4 ilustra a arquitetura do SpeechOO. Como é possível observar, a comunicação entre o SpeechOO e o LibreOffice é efetuada através da especificação UNO. Já a interface com o reconhecedor de voz é realizada através da especificação JSAPI. Note que o *engine* de reconhecimento utilizado é o Coruja, que pode ser substituído por qualquer outro *engine* que implemente a especificação JSAPI.

Ainda na Figura 4, são mostrados os modelos acústico e de linguagem, que po-

dem ser substituídos sem a necessidade de alteração do código fonte do SpeechOO. Esta facilidade possibilita a utilização do SpeechOO em diversos idiomas, como por exemplo, o inglês. Para adicionar suporte a qualquer idioma ao SpeechOO, basta criar para este modelos acústicos e de linguagem compatíveis com o decodificador Julius.

A divisão dos dois modos de funcionamento do SpeechOO (ditado e comando) foi implementada com o uso de *threads*. Para cada modo de funcionamento, uma *thread* é criada, sendo que apenas uma por vez pode estar ativa em um determinado momento do uso da aplicação. Este modelo de implementação é usado para evitar a ocorrência de conflitos entre os modos de funcionamento do SpeechOO. Por exemplo, quando o usuário fala a palavra “negrito”, não é possível que a extensão escreva o texto “negrito”, e simultaneamente ative a função negrito do Writer. Neste caso, apenas a ação referente ao modo de funcionamento ativo seria realizada.

No início da execução, as duas *threads* referentes aos modos de funcionamento do SpeechOO são criadas, sendo que inicialmente, a *thread* do modo ditado fica ativa por padrão. Para a alternância entre os modos de ditado e comando, um *keyboard listener* foi utilizado. Este é uma *thread* sempre ativa que captura os eventos disparados quando uma tecla é pressionada. Assim que uma tecla pré-definida para *switch*, neste caso a ALT GRAPH, é pressionada, o modo comando é ativado, permanecendo assim até que a tecla seja liberada. A tecla ALT GRAPH foi escolhida como tecla *switch* pois possui localização favorável (parte de baixo do teclado) e não possui nenhuma função predefinida no LibreOffice Writer.

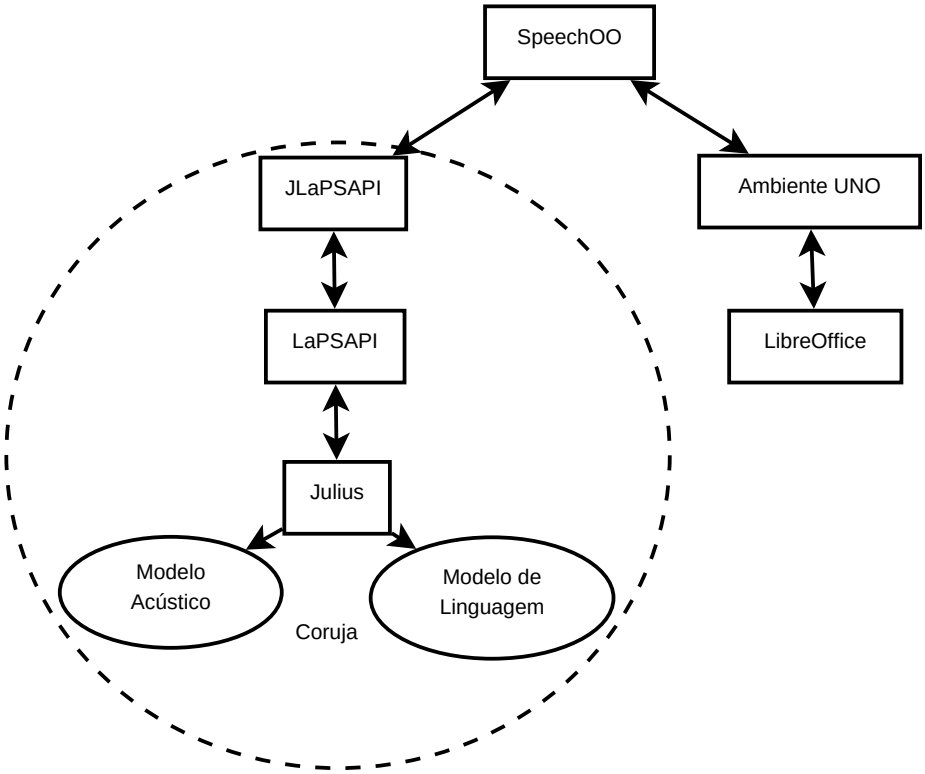

Figura 4. Arquitetura do SpeechOO.

5.2. SpeechOO: modo ditado

O modo padrão de funcionamento do SpeechOO é o modo ditado. Neste modo, a fala do usuário é convertida em texto e então é apresentada na tela de edição do aplicativo LibreOffice Writer. O trecho de código abaixo ilustra em linhas gerais como essa funcionalidade foi implementada no SpeechOO.

```
package org.speechoo.recognized;
import javax.speech.recognition.Result;
import javax.speech.recognition.ResultAdapter;
import javax.speech.recognition.ResultEvent;
import javax.speech.recognition.ResultToken;
import org.speechoo.inputText.InputSentence;
public class FreeDictationListener extends ResultAdapter {
  InputSentence inputSentence = new InputSentence();
  @Override
  public void resultAccepted(ResultEvent e) {
    StringBuffer returnTokens = new StringBuffer();
    Result r = (Result) (e.getSource());
    ResultToken tokens[] = r.getBestTokens();
    for (int i = 0; i < tokens.length; i++) {
      if (i > 0)
        returnTokens.append(' ');
      returnTokens.append(tokens[i].getSpokenText());
    }
    inputSentence.insertNewSentence(returnTokens.toString());
  }
}
```

A classe *FreeDictationListener* implementa um *ResultAdapter* que de acordo com o padrão JSAPI recebe os resultados do reconhecimento. Dessa forma o método *resultAccepted* é chamado quando uma sentença está disponível. O método *resultAccepted* transforma a sentença em uma *String* e utiliza a classe apresentada na Seção 3.2 para mostrar na tela do Writer o texto reconhecido.

5.3. SpeechOO: modo comando

Como comentado na Seção 5, o modo comando é utilizado para manipular as funcionalidades do LibreOffice Writer, como por exemplo, formatar o texto. A lista com todos os possíveis comandos é extensa, por isso nos parágrafos seguintes apresentamos apenas alguns dos mais importantes. Lista completa dos comandos pode ser encontrada em [spe 2011].

Atualmente existem comandos básicos para formatação de texto, como colocar o texto em negrito, itálico, sublinhado e para mudar o alinhamentos de parágrafos. Para estes comandos, o funcionamento ocorre da mesma forma como acontece quando usando um *mouse*, onde apenas clicando no botão referente ao comando ele pode ser ativado ou desativado.

Também foram criados comandos que permitem a navegação no texto. Esta navegação pode ser realizada por palavra ou por parágrafo, sendo possível avançar ou voltar no texto. Os comandos que permitem essa navegação são: `voltar palavra`,

avançar palavra, voltar parágrafo e avançar parágrafo. Os comandos backspace, espaço e enter também foram implementados e funcionam tal como quando se usa um teclado. É possível também selecionar partes do texto para edições ou exclusões. A seleção é feita sempre onde o cursor está presente, podendo esta selecionar uma palavra ou parágrafo.

O SpeechOO é desenvolvido em parceria com Centro de Competência em Software Livre da Universidade de São Paulo, e um projeto para seu desenvolvimento foi aprovado pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) segundo o edital 09/2010 [cnp 2011].

Referências

(1998). Java speech grammar format specification. Technical report, Sun microsystems.

(Visitado em Dezembro , 2011). Openoffice.org community announces the document foundation. http://web.archive.org/web/20100930085933/http://www.documentfoundation.org/contact/tdf_release.html.

(Visitado em Dezembro , 2011). Resultado edital CNPq nº 09/2010 - PDI - grande e pequeno porte. http://www.cnpq.br/resultados/2010/009.htm.

(Visitado em Julho , 2010). Dragon NaturallySpeaking. http://www.nuance.com/naturallyspeaking.

(Visitado em Julho , 2010). Windows Speech Recognition. http://www.microsoft.com/speech.

(Visited in June, 2011). SpeechOO: A dictation pad for LibreOffice. http://code.google.com/p/speechoo.

IBGE (2010). Census demographic profiles. Visited in January.

Liang, S. (1999). *The Java™ Native Interface Programmer's Guide and Specification*. Addison-Wesley.

Silva, P., Batista, P., Neto, N., and Klautau, A. (2010). An open-source speech recognizer for Brazilian Portuguese with a windows programming interface. *The International Conference on Computational Processing of Portuguese (PROPOR).*