Ecole d'ingénieurs et d'architectes de Fribourg
Hochschule für Technik und Architektur Freiburg

Interface multimodale

Rapport Travail pratique

Logiciel de reconnaissance vocale Sphinx-4

Philippe Galley, Blaise Grand & Samuel Rossier

19 mai 2006

Table des matières

1 Introduction

La reconnaissance vocale est une technologie informatique permettant à un logiciel d'interpréter une langue naturelle humaine. Le principe est simple : un enregistrement de quelques mots prononcés par un locuteur est interprété en texte. Cela permet entre autres la transcription automatique et le contrôle de systèmes par la voix. Cette technologie utilise des méthodes informatiques des domaines du traitement du signal et de l'intelligence artificielle.

Une phrase enregistrée et numérisée est donnée au programme de reconnaissance vocale. Celui-ci catégorise chaque phonème qu'il distingue selon un modèle de probabilités statistiques.

Une fois catégorisés, les phonèmes[1] sont interprétés pour former des mots encore une fois selon un modèle statistique.

Il y a plusieurs logiciels de reconnaissance vocale, parmi les plus connus :

- Dragon Naturally Speaking
- Via voice
- Crescendo
- Sphinx-4

Ce rapport traîte de l'installation, du fonctionnement et l'utilisation de l'API[2] Sphinx-4. Un exemple d'intégration de Sphinx dans une application sera également présenté.

2 Informations générales sur Sphinx-4

2.1 Introduction

Sphinx-4 est un système de reconnaissance vocale entièrement écrit dans le langage de programmation Java. Il a été créé conjointement par le groupe *Sphinx* à l'université *Carnegie Mellon*, les laboratoires *Sun Microsystems* et *Hewlett-Packard*.

2.2 Présentation de Sphinx-4

Sphinx-4 est une librairie de reconnaissance vocale écrite entièremenent en Java. Les buts de Sphinx sont d'avoir une reconnaissance vocale hautement flexible, d'égaler les autres produits commerciaux et de mettre en collaboration les centres de recherche de diverses universités, des laboratoires Sun, des laboratoires HP et du MIT.

Sphinx-4 est hautement configurable. La reconnaissance de Sphinx-4 supporte notamment les mots isolés et les phrases (utilisation de grammaires). L'architecture de Sphinx-4 est modulable pour permettre de nouvelles recherches et pour tester de nouveaux algorithmes.

La qualité de la reconnaissance dépend directement de la qualité des données vocales[3] (*voice data*). C'est pour cela que l'équipe de recherche de Sphinx-4 développe actuellement un « entraîneur »pour rassembler le plus de données vocales possibles. Les données vocales sont la clé d'une bonne reconnaissance vocale.

[1] Un phonème est la plus petite unité discrète ou distinctive que l'on puisse isoler par segementation dans la chaîne parlée.

[2] Interface de programmation. Offre des fonctions supplémentaires

[3] Les données vocales sont les informations relatives à une voix propre. Ce sont par exemple les différents phonèmes, les différents mots (lexique), les différents façons de prononciation. Plus ces informations seront importantes et connues par le système, meilleure sera sa réaction et ses choix à faire.

2.2.1 Fonctionnement de la reconnaissance vocale

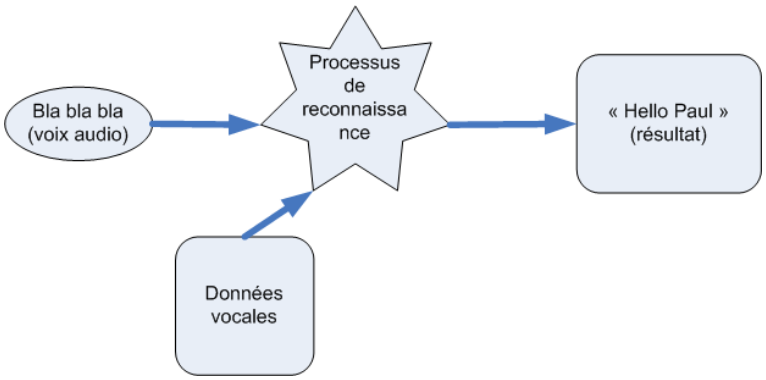

FIG. 1 – Fonctionnement de la reconnaissance vocale

La figure 1 illuste le fonctionnement de la reconnaissance vocale. Le principe est simple : une voix audio est enregistrée par le système de reconnaissance. Ce dernier, avec l'aide des données vocales, détermine les mots ou les phrases prononcés.

2.2.2 Architecture de Sphinx-4

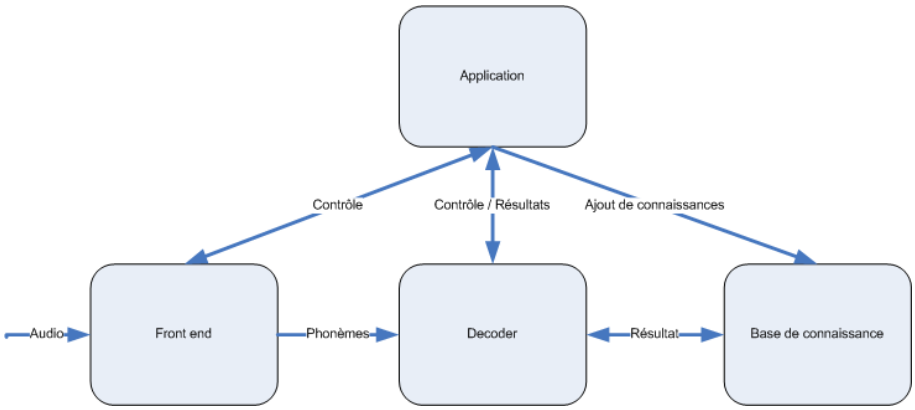

FIG. 2 – Architecture de Sphinx-4

Le graphique 2 illustre l'architecture générale de Sphinx-4.

Front-End Le Front-End découpe la voix enregistrée en différentes parties et les prépare pour le décodeur.

Base de connaissances La base de connaissance est l'information qu'utilise le décodeur pour déterminer les mots et les phrases prononcées. La base de connsaissance est composée :

- D'un dictionnaire.
  - Classification des mots.
  - Prononciation des mots (un mot peut avoir plusieurs prononciations).
  - Prononciation représentée comme des sons ou dans d'autres unités.

  - Peu varier en taille, de quelque mots à plusieurs centaines de milliers.
- D'un modèles acoustique.

- D'un modèle de langage.

  - Décrit ce qui peut être dit dans un contexte bien spécial.
  - Aide à rétrécir l'espace de recherche.

  Il y a trois sortes de modèle de language : le plus simple est utilisé pour les mots isolés, le deuxième pour les applications basées sur des commandes et des contrôles et le dernier pour le langage courant.

Décodeur Le décodeur est le coeur de Sphinx-4. C'est lui qui traîte les informations reçues depuis le Front-End, les analyse et les compare avec la base de connaissances pour donner un résultat à l'application.

3 Installation

3.1 Logiciels requis

Sphinx-4 a été compilé et testé sur Solaris, Mac OS X, Linux et Windows. L'exécution, la compilation et les tests de Sphinx-4 demandent des logiciels supplémentaires. Les logiciels suivants doivent être installés sur la machine :

- Java 2 SDK, Standard Edition 5.0. `http://java.sun.com`.
- Les différentes librairies qui composent Sphinx-4 (sphinx4-1.0beta-bin.rar). `http://cmusphinx.sourceforge.net/sphinx4/`
  Cette archive contient les différentes libraires (lib), la javadoc de Sphinx-4 et des démonstrations (sources et exécutables).

3.2 Implémentation de la librairie avec Eclipse

L'implémentation de Sphinx-4 dans une application quelconque se fait relativement facilement. La première état consiste à créer un nouveau projet (Menu Fichier - Nouveau - Projet). La figure 3 illustre la création d'un nouveau projet dans Eclipse.

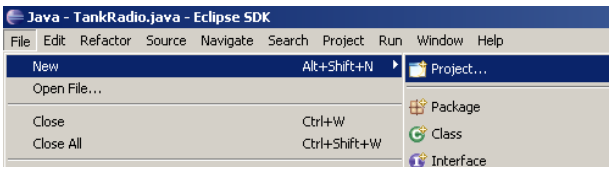

FIG. 3 – Création d'un nouveau projet

Il faut, dans la deuxième étape, insérer les librairies Sphinx-4 dans le projet. Pour ceci, on fait un clique droit sur le projet et on va dans les propriétés du projet. On choisit ensuite le menu « Java Build Path ». On clique enfin sur « Add External JARs »pour ajouter les différentes librairies fournies par Sphinx (figure 4).

Les librairies à ajouter sont les suivantes :

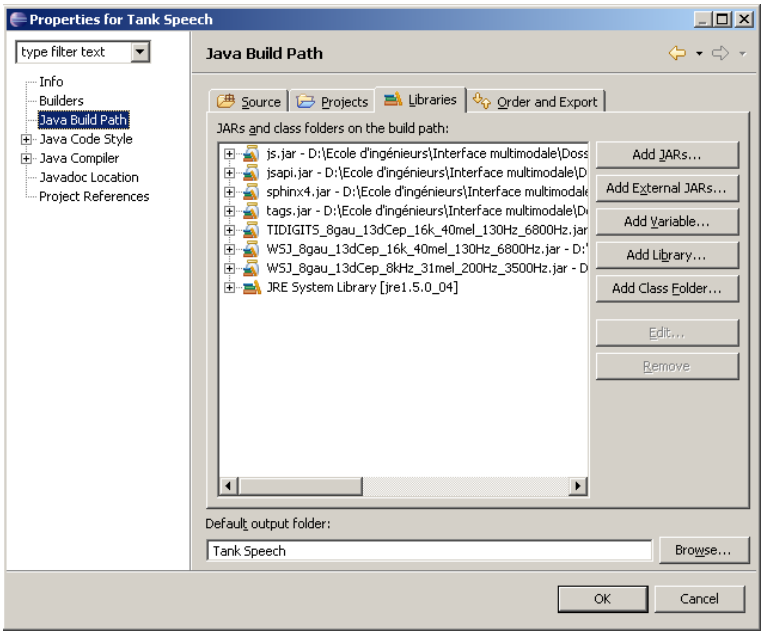

FIG. 4 – Insertion des librairies Sphinx-4 dans le projet

- js.jar.
- jsapi.jar (Celle-ci doit être créée en lançant l'application jsapi.exe qui se trouve dans le répertoire lib de l'archive téléchargée).
  Cette librairie est utilisé entre autre par Java pour enregistrer le son.
- sphinx4.jar.
- TIDIGITS_8gau_13dCep_16k_40mel_130Hz_6800Hz.jar.
  Pour la reconnaissance des nombres uniquement.
- WSJ_8gau_13dCep_8kHz_31mel_200Hz_3500Hz.
- WSJ_8gau_13dCep_16k_40mel_130Hz_6800Hz.

Ces librairies sont fournies dans l'archive télélchargeable sur le site officiel de Sphinx-4 (`http://cmusphinx.sourceforge.net/sphinx4/`).

Il faut créer ensuite un fichier *monFichier.gram*. Ce fichier contient la grammaire utilisée par l'application, c'est-à-dire les mots ou les phrases qui sont potentiellement prononçables.

Exemple de grammaire 1

```
1 grammar test;
2
3 public <simple> = hello world ;
```

Listing 1 – Exemple 1 de fichier grammaire

Le fichier grammaire ci-dessus permet de comprendre la phrase suivante

```
1 hello world
```

Exemple de grammaire 2

```
1 public <basicCmd> = <startPolite> <command> <endPolite>;
2
3 <command> = <action> <object>;
4 <action> = /10/ open |/2/ close |/1/ delete |/1/ move;
5 <object> = [the | a] (window | file | menu);
```

```
6
7  <startPolite> = (please | kindly | could you | oh mighty computer) *;
8  <endPolite> = [ please | thanks | thank you ];
```

Listing 2 – Exemple 2 de fichier grammaire

Le fichier grammaire ci-dessus permet de comprendre toutes les phrases suivantes :

```
1  move a menu thanks
2  kindly open the menu thanks
3  could you delete a menu
4  delete a menu thanks
5  delete the window thanks
6  could you oh mighty computer could you open file please
7  oh mighty computer please could you open menu thanks
8  could you could you could you could you kindly please close the file please
9  close file
10 move a file please
11 please please move window
12 please oh mighty computer open window please
13 oh mighty computer delete file
14 could you please please open window thanks
15 kindly open menu
16 kindly close the menu please
17 oh mighty computer could you move file thanks
18 could you close the menu
19 open the menu
20 oh mighty computer please kindly delete menu thanks
```

La figure 5 illustre la grammaire ci-dessus de façon graphique.

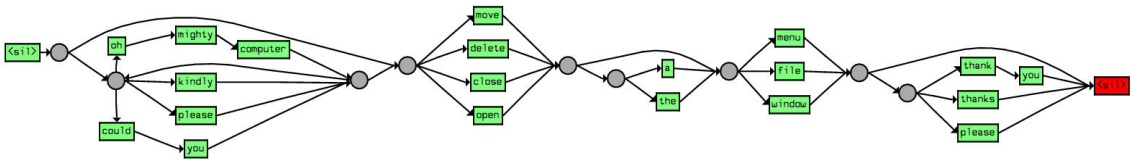

FIG. 5 – Graphe correspondant à une grammaire

Après avoir écrit le fichier de grammaire, il faut créer le fichier de configuration monFichier.config.xml. Le plus simple est d'utiliser un fichier de configuration d'une des différentes démonstrations fournies dans l'archive téléchargée. Ce fichier permet de spécifier entre autre le dictionnaire utilisé et la grammaire utilisée.

```
1  ...
2  <!-- ******************************************************* -->
3     <!-- The Grammar configuration                            -->
4     <!-- ******************************************************* -->
5
6     <component name="jsgfGrammar" type="edu.cmu.sphinx.jsapi.JSGFGrammar">
7        <property name="dictionary" value="dictionary"/>
8        <property name="grammarLocation"
9            value="resource:/tankspeech.TankRadio!/tankspeech/"/>
10       <property name="grammarName" value="grammaireTank"/>
11    <property name="logMath" value="logMath"/>
12    </component>
13 ...
14 <!-- ******************************************************* -->
15    <!-- The Dictionary configuration                        -->
16    <!-- ******************************************************* -->
17
18    <component name="dictionary"
19       type="edu.cmu.sphinx.linguist.dictionary.FastDictionary">
20       <property name="dictionaryPath"
21     value="resource:/edu.cmu.sphinx.model.acoustic.WSJ_8gau_13dCep_16k_40mel_130Hz_6800Hz.Model!/edu/cmu/sphinx/model/acoustic/
           WSJ_8gau_13dCep_16k_40mel_130Hz_6800Hz/dict/cmudict.0.6d"/>
22       <property name="fillerPath"
23     value="resource:/edu.cmu.sphinx.model.acoustic.WSJ_8gau_13dCep_16k_40mel_130Hz_6800Hz.Model!/edu/cmu/sphinx/model/acoustic/
           WSJ_8gau_13dCep_16k_40mel_130Hz_6800Hz/dict/fillerdict"/>
24       <property name="addSilEndingPronunciation" value="false"/>
25       <property name="allowMissingWords" value="false"/>
26       <property name="unitManager" value="unitManager"/>
27    </component>
28 ...
```

Listing 3 – Extrait du fichier de configuration

La dernière étape de l'implémentation de Sphinx-4 est d'écrire le code Java qui va enregistrer la voix, l'interprêter et exécuter le code en conséquence. Le code ci-dessous illustre ceci.

```
try {
    // Identification du fichier de configuration.
    URL url = TankRadio.class.getResource("tankspeech.config.xml");

    // Chargement de la configuration.
    ConfigurationManager cm = new ConfigurationManager(url);

    // Création de l'objet de reconnaissance.
    Recognizer recognizer = (Recognizer) cm.lookup("recognizer");
    // Création de l'objet microphone.
    Microphone microphone = (Microphone) cm.lookup("microphone");

    // Allocation des ressources pour la reconnaissance.
    recognizer.allocate();

    // On teste si le microphone est prêt à enregistrer.
    if (microphone.startRecording()) {
        // On reconnait un mot ou une phrase.
        Result result = recognizer.recognize();
        // On teste si le résultat est NULL. Si il n'y a aucun résultat, on affiche un
        // message d'erreur dans la barre de titre de l'application.
        if (result != null) {
            // On récupère la chaîne reconnue si le résultat enregistré est non NULL.
            String resultText = result.getBestFinalResultNoFiller();
            // On appelle la méthode ou le code correspondant à la commande reçue.
            if(resultText.compareTo("left")==0)this.monTank.tournerAGauche();
        } else {
            this.monTank.afficherMessage("Répéter s'il vous plaît");
        }
    } else {
        recognizer.deallocate();
    }
} catch (Exception e) {
    System.out.println("Problème: " + e);
}
```

4 Application *Tank Speech*

4.1 Introduction

L'application présentée pour illustrer l'intégration de Sphinx est un petit jeu nommé *Tank Speech*. Le but de ce jeu est de diriger un char d'assaut en lui donnant des ordres orals tel que *left*, *right*, *slow*, *quick*, *start*, *stop* ou *attack*.

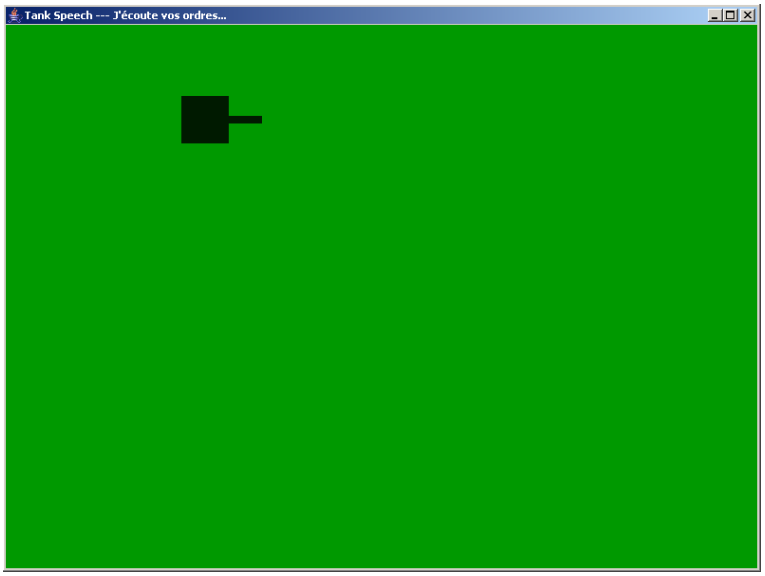

FIG. 6 – Interface du jeu *Tank Speech*

La figure 6 présente l'interface du jeu *Tank Speech*. Celle-ci est très sommaire. En effet, seul le char est représenté dans la fenêtre. Par ailleurs, lorsque la commande *attack* est appelée, une

bombe est envoyée et explose dans la direction dans laquelle elle a été lancée.
Lorsqu'une commande est reçue par le tank, le nom de cette commande est inscrite dans la barre de titre de la fenêtre.

4.2 Implémentation de la reconnaissance vocale

4.2.1 Code Java pour l'implémentation de la reconnaissance vocale

Le code ci-dessous présente l'implémentation de la reconnaissance pour notre application.

```
1  /*
2   * Auteur:    Philippe Galley, Blaise Grand, Samuel Rossier.
3   * Date:      8 mai 2006.
4   * Description: Cette classe implémete la reconnaisance vocale pour
5   *            le jeu Tank Speech. Elle écoute et interprête les
6   *            les commandes données par l'utilisateur et exécute
7   *            les méthodes en conséquence.
8   */
9
10 package tankspeech;
11
12 import tankspeech.TankRadio;
13 import edu.cmu.sphinx.frontend.util.Microphone;
14 import edu.cmu.sphinx.recognizer.Recognizer;
15 import edu.cmu.sphinx.result.Result;
16 import edu.cmu.sphinx.util.props.ConfigurationManager;
17 import java.net.URL;
18
19 public class TankRadio extends Thread {
20
21    Tank monTank;
22
23    public TankRadio(Tank monTank){
24        this.monTank = monTank;
25    }
26
27    public void run() {
28        try {
29            // Identification du fichier de configuration.
30            URL url = TankRadio.class.getResource("helloworld.config.xml");
31
32            // Chargement de la configuration.
33            ConfigurationManager cm = new ConfigurationManager(url);
34
35            // Création de l'objet de reconnaissance.
36            Recognizer recognizer = (Recognizer) cm.lookup("recognizer");
37            // Création de l'objet microphone.
38            Microphone microphone = (Microphone) cm.lookup("microphone");
39
40            // Allocation des ressources pour la reconnaissance.
41            recognizer.allocate();
42
43            // On teste si le microphone est prêt à enregistrer.
44            if (microphone.startRecording()) {
45
46                // On démarre le tank.
47                this.monTank.start();
48                // On indique que l'on est prêt à recevoir des ordres vocaux.
49                this.monTank.afficherMessage("J'écoute vos ordres...");
50                // On écoute indéfiniment les commandes.
51                while (true) {
52                    // On reconnait un mot ou une phrase.
53                    Result result = recognizer.recognize();
54                    // On teste si le résultat est NULL. Si il n'y a aucun résultat, on affiche un
55                    // message d'erreur dans la barre de titre de l'application.
56                    if (result != null) {
57                        // On récupère la chaîne reconnue si le résultat enregistré est non NULL.
58                        String resultText = result.getBestFinalResultNoFiller();
59                        // On affiche la chaîne reconnue.
60                        this.monTank.afficherMessage(resultText);
61                        // On appelle la méthode du tank correspondante à la commande reçue.
62                        if(resultText.compareTo("left")==0)this.monTank.tournerAGauche();
63                        if(resultText.compareTo("right")==0)this.monTank.tournerADroite();
64                        if(resultText.compareTo("slow")==0)this.monTank.ralentir();
65                        if(resultText.compareTo("quick")==0)this.monTank.accelerer();
66                        if(resultText.compareTo("stop")==0)this.monTank.stopper();
67                        if(resultText.compareTo("attack")==0)this.monTank.attaquer();
68                    } else {
69                        this.monTank.afficherMessage("Répéter s'il vous plaît");
70                    }
71                }
72            } else {
73                recognizer.deallocate();
74            }
75        } catch (Exception e) {
76            System.out.println("Problème: " + e);
77        }
78    }
79 }
```

Listing 4 – Code utilisé pour implémenter la reconnaissance vocale

4.2.2 Fichier de grammaire

Le fichier sert à déclarer toutes les commandes possibles pour diriger et actionner le Tank.

```
#JSGF V1.0;

grammar grammaireTank;

public <greet> = ( left | right | quick | slow | stop | restart | attack);
```

Listing 5 – Fichier de grammaire du jeu Tank Speech

4.2.3 Fichier de configuration

```
    <!-- ******************************************************** -->
    <!-- The Grammar configuration                                -->
    <!-- ******************************************************** -->

    <component name="jsgfGrammar" type="edu.cmu.sphinx.jsapi.JSGFGrammar">
        <property name="dictionary" value="dictionary"/>
        <property name="grammarLocation"
             value="resource:/tankspeech.TankRadio!/tankspeech/"/>
        <property name="grammarName" value="grammaireTank"/>
    <property name="logMath" value="logMath"/>
    </component>


    <!-- ******************************************************** -->
    <!-- The Dictionary configuration                        -->
    <!-- ******************************************************** -->

    <component name="dictionary"
        type="edu.cmu.sphinx.linguist.dictionary.FastDictionary">
        <property name="dictionaryPath"
     value="resource:/edu.cmu.sphinx.model.acoustic.WSJ_8gau_13dCep_16k_40mel_130Hz_6800Hz.Model!/edu/cmu/sphinx/model/acoustic/
           WSJ_8gau_13dCep_16k_40mel_130Hz_6800Hz/dict/cmudict.0.6d"/>
        <property name="fillerPath"
     value="resource:/edu.cmu.sphinx.model.acoustic.WSJ_8gau_13dCep_16k_40mel_130Hz_6800Hz.Model!/edu/cmu/sphinx/model/acoustic/
           WSJ_8gau_13dCep_16k_40mel_130Hz_6800Hz/dict/fillerdict"/>
        <property name="addSilEndingPronunciation" value="false"/>
        <property name="allowMissingWords" value="false"/>
        <property name="unitManager" value="unitManager"/>
    </component>
```

Listing 6 – Fichier de configuration du jeu Tank Speech

4.3 Résultats

Le résultat de l'application est très satisfaisant. La majorité des ordres sont compris par le Tank qui les exécutes correctement. Même en prononçant plusieurs commandes à la suite, celles-ci sont en principe toutes exécutées.

On peut voir dans ce petit exemple que l'implémentation de la reconnaissance vocale est relativement simple et vite fait dans la mesure où ce sont des mots ou des phrases relativement simples. En effet, la compréhension de phrases plus complexe demanderait un travail plus conséquent au niveau du fichier de grammaire.

5 Conclusion

L'étude de cette librairie était très intéressante. Malgré le peu d'informations et d'exemples sur Internet, nous avons pu, grâce aux exemples fournis par l'équipe Sphinx-4, implémenter notre propre application utilisant la reconnaissance vocale. Notre application ne reconnait que

mots isolés (grammaire simple), il serait intéressant maintenant de développer une plus grosse application avec une grammaire plus complexe, pouvant reconnaître des phrases plus ou moins longues.

Références

[1] *http ://cmusphinx.sourceforge.net/sphinx4.*