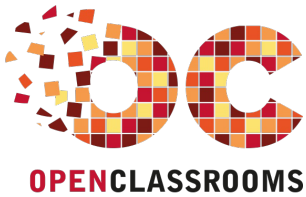
La reconnaissance vocale dans son application

Par janulrich00001



www.openclassrooms.com

Licence Creative Commons 7 2.0
Dernière mise à jour le 6/11/2009

Sommaire

La reconnaissance vocale dans son application

Par  janulrich00001

Mise à jour : 06/11/2009
Difficulté : Intermédiaire

(cc) BY-SA

La reconnaissance vocale, au sens informatique du terme, est un procédé permettant d'analyser la parole et de la transformer en mots et en phrases. Elle permet donc le passage d'un son en un texte. Le terme généralement utilisé en anglais est Automatic Speech Recognition (ASR).

J'ai décidé de créer ce tutoriel suite à mon travail de Bachelor pour devenir ingénieur des médias spécialisé en technologies de l'information. Ce travail consistait à développer une application pour apprendre le braille avec Mouskie au moyen de la synthèse et de la reconnaissance vocale et donc en s'affranchissant totalement du clavier et de l'écran (pas très utilisables par des personnes handicapées de la vue).

Au début de ce travail, je dois dire que j'ai galéré pour trouver des informations concernant la reconnaissance vocale et surtout comment l'ajouter dans une application. C'est une science obscure de l'informatique et personne n'en parle. Alors, je me suis dit que j'allais apporter ma petite contribution...

Je tiens à signaler que j'ai fait des recherches sur la reconnaissance vocale et les différents moteurs qui existent. Le moteur qui est utilisé dans ce tutoriel provient de Windows, plus précisément *Windows Speech Recognition* et est disponible en français dans Windows Vista et Windows 7, dans la version en français bien sûr .

Sommaire du tutoriel :

Avant de commencer

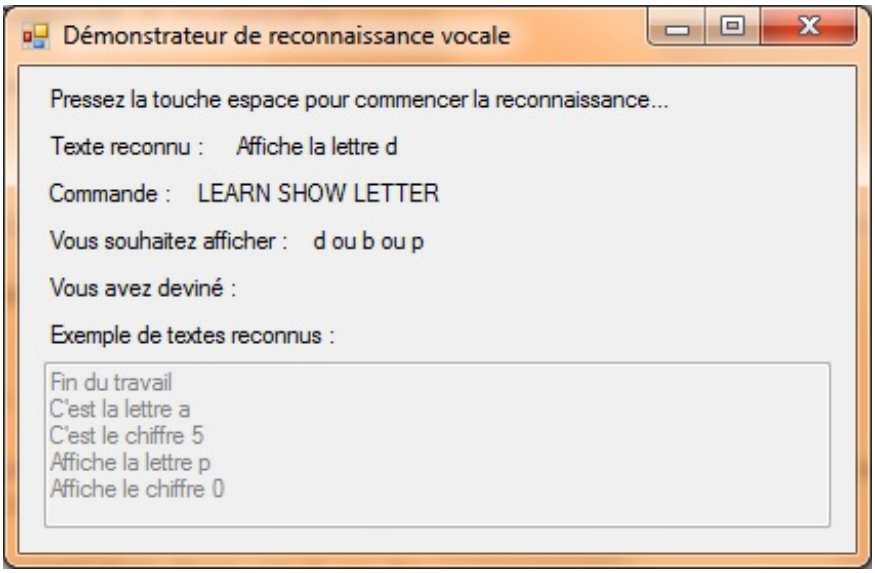
Avant d'enter dans le vif du sujet, voici une liste du matériel et des logiciels nécessaires avant de commencer à sérieusement travailler :

- Windows Vista ou Windows 7 en français
- Visual C# Express Edition
- Un microphone
- Quelques notions en C#
- Quelques notions en XML

L'utilisation d'un autre IDE est également possible (je me suis forcé à retourner de Visual Studio 2008 aux éditions Express pour ce tuto!), mais je ne garantis pas que tout se fasse de la même manière. Aussi comme un niveau intermédiaire est demandé pour suivre ce tuto, je ne m'attarderai pas sur les détails genre : comment configurer un microphone, où trouver l'IDE, c'est quoi Windows et j'en passe et des meilleures.

Sinon, pour allécher le peuple, voici une super capture d'écran qui montre ce qui sera obtenu à la fin de cette lecture. Il s'agira d'un démonstrateur permettant d'utiliser diverses commandes dans le but d'afficher des lettres, des chiffres et faire quitter l'application. Après, si tout a été compris, il sera possible de créer des applications qui fonctionnent avec l'ASR très facilement.

En passant, j'ai développé ce démonstrateur juste avant mon travail de Bachelor pour montrer à mon professeur que je n'allais pas me planter pendant le projet. Donc, oui, c'est minimaliste, mais ça marche et ça fait bien ce qu'on demande :



Base du projet

Le graphisme ce n'est pas ce qui est intéressant dans ce tutoriel. Je pense que c'est mieux de se concentrer sur le fond du problème et donc voici les quelques étapes et copier-coller à faire pour atteindre le même résultat que présenté ci-dessus. Rien de spécial n'est utilisé ici. A noter l'utilisation de WinForms et pas de WPF au niveau graphique. Alors, voici la liste de A à Z de ce qu'il faut faire en détail :

- Démarrer Visual C# 2008 Express Edition
- Créer un nouveau projet (File > New Project > Windows Forms Application => Renommer en DemoRecoVocale et OK)
- Cliquer sur le "+" de Form1.cs
- Faire un clic-droit sur Form1.Designer.cs et cliquer sur "View code"
- Copier-coller le code ci-dessous dans Form1.Designer.cs =>

Secret (cliquez pour afficher)

Code : C#

```
namespace DemoRecoVocale
{
    partial class Form1
    {
        /// <summary>
        /// Variable nécessaire au concepteur.
        /// </summary>
        private System.ComponentModel.IContainer
components = null;

        /// <summary>
        /// Nettoyage des ressources utilisées.
        /// </summary>
        /// <param name="disposing">true si les
ressources managées doivent être supprimées ; sinon,
false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Code généré par le Concepteur Windows
Form
```

```
        /// <summary>
        /// Méthode requise pour la prise en charge du
concepteur - ne modifiez pas
        /// le contenu de cette méthode avec l'éditeur
de code.
        /// </summary>
        private void InitializeComponent()
        {
            this.recoTextLabel = new
System.Windows.Forms.Label();
            this.afficheLabel = new
System.Windows.Forms.Label();
            this.devineLabel = new
System.Windows.Forms.Label();
            this.recoText = new
System.Windows.Forms.Label();
            this.affiche = new
System.Windows.Forms.Label();
            this.devine = new
System.Windows.Forms.Label();
            this.exemplesLabel = new
System.Windows.Forms.Label();
            this.exemples = new
System.Windows.Forms.TextBox();
            this.commandLabel = new
System.Windows.Forms.Label();
            this.commandText = new
System.Windows.Forms.Label();
            this.helpLabel = new
System.Windows.Forms.Label();
            this.SuspendLayout();
            //
            // recoTextLabel
            //
            this.recoTextLabel.AutoSize = true;
            this.recoTextLabel.Location = new
System.Drawing.Point(12, 32);
            this.recoTextLabel.Name = "recoTextLabel";
            this.recoTextLabel.Size = new
System.Drawing.Size(85, 13);
            this.recoTextLabel.TabIndex = 3;
            this.recoTextLabel.Text = "Texte reconnu :
";
            //
            // afficheLabel
            //
            this.afficheLabel.AutoSize = true;
            this.afficheLabel.Location = new
System.Drawing.Point(12, 78);
            this.afficheLabel.Name = "afficheLabel";
            this.afficheLabel.Size = new
System.Drawing.Size(123, 13);
            this.afficheLabel.TabIndex = 5;
            this.afficheLabel.Text = "Vous souhaitez
afficher :";
            //
            // devineLabel
            //
            this.devineLabel.AutoSize = true;
            this.devineLabel.Location = new
System.Drawing.Point(12, 101);
            this.devineLabel.Name = "devineLabel";
            this.devineLabel.Size = new
System.Drawing.Size(98, 13);
            this.devineLabel.TabIndex = 6;
            this.devineLabel.Text = "Vous avez deviné
:";
            //
            // recoText
```

```
            //
            this.recoText.AutoSize = true;
            this.recoText.Location = new
System.Drawing.Point(103, 32);
            this.recoText.Name = "recoText";
            this.recoText.Size = new
System.Drawing.Size(27, 13);
            this.recoText.TabIndex = 7;
            this.recoText.Text = "vide";
            //
            // affiche
            //
            this.affiche.AutoSize = true;
            this.affiche.Location = new
System.Drawing.Point(141, 78);
            this.affiche.Name = "affiche";
            this.affiche.Size = new
System.Drawing.Size(27, 13);
            this.affiche.TabIndex = 8;
            this.affiche.Text = "vide";
            //
            // devine
            //
            this.devine.AutoSize = true;
            this.devine.Location = new
System.Drawing.Point(116, 101);
            this.devine.Name = "devine";
            this.devine.Size = new
System.Drawing.Size(27, 13);
            this.devine.TabIndex = 9;
            this.devine.Text = "vide";
            //
            // exemplesLabel
            //
            this.exemplesLabel.AutoSize = true;
            this.exemplesLabel.Location = new
System.Drawing.Point(12, 124);
            this.exemplesLabel.Name = "exemplesLabel";
            this.exemplesLabel.Size = new
System.Drawing.Size(146, 13);
            this.exemplesLabel.TabIndex = 10;
            this.exemplesLabel.Text = "Exemple de textes
reconnus :";
            //
            // exemples
            //
            this.exemples.Enabled = false;
            this.exemples.Location = new
System.Drawing.Point(12, 144);
            this.exemples.Multiline = true;
            this.exemples.Name = "exemples";
            this.exemples.Size = new
System.Drawing.Size(384, 81);
            this.exemples.TabIndex = 11;
            this.exemples.Text = "Fin du
travail\r\nC'est la lettre a\r\nC'est le chiffre
5\r\nAffiche la lettre p\r\nAffiche le chiffre 0";
            //
            // commandLabel
            //
            this.commandLabel.AutoSize = true;
            this.commandLabel.Location = new
System.Drawing.Point(12, 55);
            this.commandLabel.Name = "commandLabel";
            this.commandLabel.Size = new
System.Drawing.Size(66, 13);
            this.commandLabel.TabIndex = 12;
            this.commandLabel.Text = "Commande :";
            //
            // commandText
```

```
            //
            this.commandText.AutoSize = true;
            this.commandText.Location = new
System.Drawing.Point(84, 55);
            this.commandText.Name = "commandText";
            this.commandText.Size = new
System.Drawing.Size(27, 13);
            this.commandText.TabIndex = 13;
            this.commandText.Text = "vide";
            //
            // helpLabel
            //
            this.helpLabel.AutoSize = true;
            this.helpLabel.Location = new
System.Drawing.Point(12, 9);
            this.helpLabel.Name = "helpLabel";
            this.helpLabel.Size = new
System.Drawing.Size(309, 13);
            this.helpLabel.TabIndex = 14;
            this.helpLabel.Text = "Pressez la touche
espace pour commencer la reconnaissance...";
            //
            // MainWindow
            //
            this.AutoScaleDimensions = new
System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode =
System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new
System.Drawing.Size(408, 237);
            this.Controls.Add(this.helpLabel);
            this.Controls.Add(this.commandText);
            this.Controls.Add(this.commandLabel);
            this.Controls.Add(this.exemples);
            this.Controls.Add(this.exemplesLabel);
            this.Controls.Add(this.devine);
            this.Controls.Add(this.affiche);
            this.Controls.Add(this.recoText);
            this.Controls.Add(this.devineLabel);
            this.Controls.Add(this.afficheLabel);
            this.Controls.Add(this.recoTextLabel);
            this.Name = "MainWindow";
            this.Text = "Démonstrateur de reconnaissance
vocale";
            this.ResumeLayout(false);
            this.PerformLayout();

        }

        #endregion

        public System.Windows.Forms.Label recoTextLabel;
        public System.Windows.Forms.Label afficheLabel;
        public System.Windows.Forms.Label devineLabel;
        public System.Windows.Forms.Label recoText;
        public System.Windows.Forms.Label affiche;
        public System.Windows.Forms.Label devine;
        public System.Windows.Forms.Label exemplesLabel;
        public System.Windows.Forms.TextBox exemples;
        public System.Windows.Forms.Label commandLabel;
        public System.Windows.Forms.Label commandText;
        public System.Windows.Forms.Label helpLabel;
    }
}
```

- Pressez la touche F5 et normalement la fenêtre devrait s'afficher correctement.

Bon j'espère que ça marche ! Sinon, il faut recommencer ces petites étapes. Donc, actuellement en ce qui concerne la reconnaissance vocale il n'y a rien du tout.

Préparation de la structure

Voilà, on y arrive gentiment. Il faut encore réaliser quelques opérations et après tout sera prêt. Tout d'abord, il faut importer les bibliothèques de reconnaissance vocale dans le projet. La bibliothèque que l'on va utiliser se nomme System.Speech et elle est disponible en standard, donc pas de manipulations compliquées à effectuer. Pour cela il suffit de faire un clic-droit sur References > Add Reference => sélectionner la bonne et cliquer sur OK.

Dans le programme, nous allons avoir besoin d'un fichier de grammaire XML pour gérer les commandes vocales ; alors nous allons en créer un. Pour cela, il faut faire un clic-droit sur DemoRecoVocale puis Add > New Item => sélectionner XML File, le nommer Grammaire.grxml et cliquer sur OK. Ensuite, faire un clic-droit sur ce nouveau fichier puis Properties et aller à l'option Copy to Output Directory pour sélectionner Copy always.

Maintenant, au tour de la classe qui gérera la reconnaissance vocale. Pour cela, il faut faire un clic-droit sur DemoRecoVocale puis Add > Class => renommer en ASR.cs et cliquer sur OK. Cette classe s'occupera de tout ce qui concerne la reconnaissance vocale et les actions qui s'en suivent (modifications des textes affichés et fermeture de l'application). Elle contiendra aussi l'instance du moteur de reconnaissance vocale.

Le moteur de reconnaissance vocale

Dans cette partie se trouve l'initialisation du moteur de reconnaissance vocale et de ses fonctions les plus basiques. Aussi, dans un souci de qualité, le fonctionnement de l'application sera modifié pour que la reconnaissance vocale ne fonctionne qu'après la pression de la touche Espace et jusqu'à la fin de la reconnaissance vocale. Cela permet d'éviter que l'application s'emballe à cause du bruit ambiant notamment (souffle, travaux, téléphone, etc.).

Pour commencer, copiez-collez le code ci-dessous pour remplacer celui actuellement contenu dans votre classe ASR. Ce code effectue une série d'actions : l'initialisation du moteur de reconnaissance vocale, le choix de l'entrée du microphone, le chargement de la grammaire XML et le choix des méthodes pour les divers cas de reconnaissance (reconnaissance effective, reconnaissance en cours, reconnaissance échouée). Je conseille de bien lire le code car il y a pas mal de commentaires qui expliquent bien ce que fait le code et je pense qu'il est assez verbeux pour ne pas le détailler encore ici.

Code : C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Speech.Recognition.SrgsGrammar;
using System.Speech.Recognition;
using System.Windows.Forms;


namespace DemoRecoVocale
{
    class ASR
    {
        public SpeechRecognitionEngine ASREngine;
        private Label recoText;
        private Label commandText;
        private Label devine;
        private Label affiche;

        /// <summary>
        /// Constructeur de l'ASR (Automatic Speech Recognition)
        /// </summary>
        public ASR(ref Label recoText, ref Label commandText, ref
Label devine, ref Label affiche)
        {
            //Les 4 labels dont le texte devra être changé
            //en fonction de ce qui est reconnu
            this.recoText = recoText;
            this.commandText = commandText;
```

```
            this.devine = devine;
            this.affiche = affiche;
            //Démarrage du moteur de reconnaissance vocale
            StartEngine();
        }

        /// <summary>
        /// Démarrage du moteur de reconnaissance vocale et
chargement du
        /// fichier de grammaire Grammaire.grxml
        /// </summary>
        private void StartEngine()
        {
            //Création d'un document de la norme SRGS à partir du
fichier grxml
            SrgsDocument xmlGrammar = new
SrgsDocument("Grammaire.grxml");
            //Création d'une grammaire depuis le fichier de
grammaire
            Grammar grammar = new Grammar(xmlGrammar);
            //Création de l'objet traitant la reconnaissance vocale
            ASREngine = new SpeechRecognitionEngine();
            //Récupération du son du microphone
            ASREngine.SetInputToDefaultAudioDevice();
            //Chargement de la grammaire
            ASREngine.LoadGrammar(grammar);
            //Link des fonctions a appeler en cas de reconnaissance
d'un texte
            ASREngine.SpeechRecognized +=
ASREngine_SpeechRecognized;
            ASREngine.SpeechRecognitionRejected +=
ASREngine_SpeechRecognitionRejected;
            ASREngine.SpeechHypothesized +=
ASREngine_SpeechHypothesized;
            //Spécification du nombre maximum d'alternatives
            //Par exemple : b ou p ou d ou t, t ou d, i ou j, etc.
            //Utile pour les sons qui se ressemblent
            ASREngine.MaxAlternates = 4;
        }

        /// <summary>
        /// Méthode utilisée lorsque la reconnaissance vocale est
en cours
        /// </summary>
        private void ASREngine_SpeechHypothesized(object sender,
SpeechHypothesizedEventArgs e)
        {
        }

        /// <summary>
        /// Méthode utilisée lorsque la reconnaissance vocale a
échoué
        /// </summary>
        private void ASREngine_SpeechRecognitionRejected(object
sender, SpeechRecognitionRejectedEventArgs e)
        {
        }

        /// <summary>
        /// Méthode utilisée lorsque la reconnaissance vocale est
réussi
        /// </summary>
        private void ASREngine_SpeechRecognized(object sender,
SpeechRecognizedEventArgs e)
        {
        }
    }
}
```

Il faut également modifier la classe Form1 pour que son code ressemble à celui ci-dessous. La seule chose qui y est effectuée est l'initialisation de la classe ASR et la gestion de l'activation de la reconnaissance vocale par la touche Espace. Il est également possible d'activer la reconnaissance vocale en permanence en changeant simplement la commande `ASR.ASREngine.RecognizeAsync(RecognizeMode.Single);` par `ASR.ASREngine.RecognizeAsync(RecognizeMode.Multiple);`

Code : C#

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Speech.Recognition;

namespace DemoRecoVocale
{
    public partial class Form1 : Form
    {
        private ASR ASR;
        public Form1()
        {
            InitializeComponent();
            ASR = new ASR(ref this.recoText, ref this.commandText,
ref this.devine, ref this.affiche);
            this.KeyPress += ActivateASR;
        }

        /// <summary>
        /// Active la reconnaissance vocale lors de la pression sur
la touche
        /// espace pour une séquence de commande
        /// </summary>
        private void ActivateASR(object sender, KeyPressEventArgs e)
        {
            if (e.KeyChar == (char)Keys.Space)
            {
                try
                {
                    //Activation de la reconnaissance vocale pour
une commande

ASR.ASREngine.RecognizeAsync(RecognizeMode.Single);
                }
                catch { }
            }
        }

    }
}
```

Les commandes vocales

Pour le moment, l'application plante car la grammaire XML (le fichier qui contient les commandes vocales) n'est pas encore créée. Nous allons donc voir ce que c'est. Cette grammaire permet de lister les commandes vocales qui seront disponibles dans l'application avec une flexibilité assez impressionnante. Ce fichier sera écrit d'après la spécification Speech Recognition Grammar Specification édictée par le W3C. Voici à quoi ressemble un de ces fichiers (celui-ci permet de reconnaître les termes a, b quitter et fermer) :

Code : XML

```
<?xml version="1.0" encoding="UTF-8"?>
<grammar version="1.0" xml:lang="fr-FR" mode="voice" tag-
format="semantics-ms/1.0"
root="mouskie" xmlns="http://www.w3.org/2001/06/grammar">
  <rule id="mouskie" scope="public">
    <ruleref special="GARBAGE" />
    <one-of>
      <item>
        a
        <tag>$.mouskie={}; $.mouskie._value="A";</tag>
      </item>
      <item>
        b
        <tag>$.mouskie={}; $.mouskie._value="B";</tag>
      </item>
      <item>
        Quitter
        <tag>$.mouskie={}; $.mouskie._value="QUIT";</tag>
      </item>
      <item>
        Fermer
        <tag>$.mouskie={}; $.mouskie._value="QUIT";</tag>
      </item>
    </one-of>
    <ruleref special="GARBAGE" />
  </rule>
</grammar>
```

Tout d'abord, il contient les en-têtes XML normales, puis une série d'informations sur la version de la grammaire, la langue, la règle racine (mouskie), la visibilité et le format des tags. Ensuite, les commandes vocales sont listées avec des informations particulières. Donc maintenant qu'est-ce que tout cela veut bien dire?

Le tag-format c'est la définition du format utilisé pour le texte contenu dans la balise tag. Cette balise tag est utilisée pour renvoyer des informations sémantiques à l'application. En gros au lieu de recevoir des phrases dans le style "Jean a mangé à 13h une pomme verte savoureuse." on aurait plutôt "Qui => Jean, Action => Manger, Nourriture => Pomme, Couleur => Verte". Bref, des informations beaucoup plus simples à interpréter dans un programme informatique.

La règle racine (mouskie dans ce cas) c'est là où le programme va entrer en premier. Cette règle contiendra toutes les commandes vocales et leur manière de fonctionner comme c'est le cas ici.

Le GARBAGE est une règle spéciale qui permet de dire "parle tant que tu veux, je n'écouterai pas avant que tu dises quelque chose que je peux comprendre". En gros, ici on peut voir que les textes *a, b, Quitter et Fermer* peuvent être reconnus. Cette règle permettrait de dire à l'application "J'aimerais bien QUITTER l'application maintenant" et la reconnaissance vocale se focaliserait sur le terme "QUITTER" qu'elle comprend. Le reste qui vient avant et après, elle ne connaît pas, alors elle jette.

Les termes reconnus sont comme dit juste avant a, b, Quitter et Fermer et toutes les phrases qui peuvent les contenir grâce au garbage. Ces termes qui peuvent être reconnus sont dans des balises one-of, et item. Le one-of permet de dire "Tu peux reconnaître UN de ces termes". Le item permet de définir un terme. A noter qu'un item peut contenir un one-of et ainsi de suite cela permet de faire de la construction de phrase.

La balise tag intimement liée au tag-format permet de renvoyer des commandes à l'application. Ici on peut voir que lorsque "Quitter" ou "Fermer" sont reconnus l'application recevra la commande "QUIT" en retour. Cette technique est pratique pour que plusieurs commandes entraînent la même action dans l'application. Bref, cela simplifie la programmation.

Avec le fichier de grammaire XML ci-dessus, notre application peut déjà fonctionner.

Développement des acquis

Maintenant que le moteur de reconnaissance vocale tourne et que la grammaire est en place, il faut d'un côté développer la grammaire pour que le nombre de commandes vocales corresponde à ce que l'on souhaite faire et de l'autre développer l'application pour qu'elle puisse traiter les commandes qu'elle reçoit. Ici, on ne va développer qu'une toute petite grammaire dont voici les fonctions qu'elle comportera :

- Quitter l'application

- Affiche une lettre ou un chiffre
- Distinguer les lettres des chiffres sémantiquement
- Offrir plusieurs manières d'exprimer une commande
- Utiliser un vocabulaire naturel

Pour ce faire, tout d'abord la grammaire avec les règles que voici :

Code : XML

```
<?xml version="1.0" encoding="UTF-8"?>
<grammar version="1.0" xml:lang="fr-FR" mode="voice" tag-
format="semantics-ms/1.0"
root="mouskie" xmlns="http://www.w3.org/2001/06/grammar">
	<rule id="mouskie" scope="public">
		<ruleref special="GARBAGE" />
		<one-of>
			<item>
				<one-of>
					<item>quitter</item>
					<item>fermer</item>
				</one-of>
				<tag>$.mouskie={};
$.mouskie._value="QUIT";</tag>
			</item>
			<item>
				<one-of>
					<item>affiche</item>
					<item>montre</item>
					<item>c'est</item>
				</one-of>
				<ruleref uri="#data_type" />
				<tag>$.data_type=$data_type; $.mouskie={};
$.mouskie._value="LEARN";</tag>
			</item>
		</one-of>
		<ruleref special="GARBAGE" />
	</rule>
	<rule id="data_type" scope="public">
		<one-of>
			<item>
				la lettre<tag>$._value="LETTER";</tag>
				<ruleref uri="#letters" />
				<tag>$.letters=$letters;</tag>
			</item>
			<item>
				le chiffre<tag>$._value="NUMBER";</tag>
				<ruleref uri="#numbers" />
				<tag>$.numbers=$numbers;</tag>
			</item>
		</one-of>
	</rule>
	<rule id="letters" scope="public">
		<one-of>
			<item>a</item>
			<item>b</item>
		</one-of>
		<tag>$._value = $recognized.text;</tag>
	</rule>
	<rule id="numbers" scope="public">
		<one-of>
			<item>0</item>
			<item>1</item>
		</one-of>
		<tag>$._value = $recognized.text;</tag>
	</rule>
</grammar>
```

Plusieurs choses sont utilisées ici : les règles internes (data_type, letters, numbers) et divers tags. Les règles internes sont assez simples à comprendre car elles sont appelées avec la balise ruleref et une URI. Par exemple : **<ruleref** uri="#numbers" **/>** permet de faire appel à la règle des nombres et avec le tag associé **<tag>**$.numbers=$numbers;**</tag>** les informations dans les tag de la règle numbers pourront être transmises.

Un autre tag est utilisé dans les règles des lettres et des nombres **<tag>**$._value = $recognized.text;**</tag>** . Ce tag permet de dire que le texte reconnu dans cette règle sera directement mis dans value. Pour les lettres et les chiffres c'est pratique car sinon il aurait fallu à chaque fois faire **<tag>**$._value = "a";**</tag>** ,**<tag>**$._value = "b";**</tag>** , etc.

Maintenant que la grammaire est créée il faut passer à son utilisation au niveau du C#. Pour cela, seule la méthode ASREngine_SpeechRecognized doit être assez bien modifiée car c'est elle qui s'occupe des traitements lors d'une reconnaissance vocale qui se déroule avec succès. Les deux autres méthodes (reconnaissance vocale en cours et échouée) sont plutôt basiques dans cet exemple. Comme d'habitude je donne les codes et donne les explications ensuite.

Code : C#

```
/// <summary>
/// Méthode utilisée lorsque la reconnaissance vocale est en cours
/// </summary>
private void ASREngine_SpeechHypothesized(object sender,
SpeechHypothesizedEventArgs e)
{
    recoText.Text = "Hypothèse : " + e.Result.Text;
    devine.Text = "";
    affiche.Text = "";
    commandText.Text = "";
}

/// <summary>
/// Méthode utilisée lorsque la reconnaissance vocale a échoué
/// </summary>
private void ASREngine_SpeechRecognitionRejected(object sender,
SpeechRecognitionRejectedEventArgs e)
{
    recoText.Text = "Reconnaissance impossible";
    devine.Text = "";
    affiche.Text = "";
    commandText.Text = "";
}

/// <summary>
/// Méthode utilisée lorsque la reconnaissance vocale est réussi
/// </summary>
private void ASREngine_SpeechRecognized(object sender,
SpeechRecognizedEventArgs e)
{
    recoText.Text = e.Result.Text;
    devine.Text = "";
    affiche.Text = "";
    //Récupération de la commande de base utilisée (QUIT ou LEARN)
    string baseCommand = e.Result.Semantics["mouskie"].Value.ToString();
    commandText.Text = baseCommand;
    if (baseCommand.Equals("QUIT"))
        Environment.Exit(0);
    else if (baseCommand.Equals("LEARN"))
    {
        string dataType = e.Result.Semantics["data_type"].Value.ToString();
        commandText.Text += " " + dataType;
        string node = "";
        //Choix du noeud en fonction de la commande trouvée
        if (dataType.Equals("NUMBER"))
            node = "numbers";
        else if (dataType.Equals("LETTER"))
            node = "letters";
        try
        {   //Parcours des alternatives pour toutes les afficher
```

```
                for (int i = 0; i < e.Result.Alternates.ToArray().Length; i++)
                {
                    string found =
    e.Result.Alternates.ToArray()[i].Semantics["data_type"][node].Value.ToString();
                    if (i != 0)
                        affiche.Text += " ou ";
                    affiche.Text += found;
                }
            }
            catch { }
        }
    }
```

La première chose qui est faite, c'est l'extraction de la commande **mouskie** qui peut prendre la valeur QUIT ou LEARN. Pour ce faire, il faut décortiquer l'objet SpeechRecognizedEventArgs qui est transmis et c'est là qu'on retrouve la sémantique. Toute la sémantique des diverses règles peut être retrouvée de cette manière.

Deuxième point, la multiple reconnaissance vocale. Dans le code pour démarrer le moteur de reconnaissance vocale, se trouvait ce petit bout de code :

Code : C#

```
//Spécification du nombre maximum d'alternatives
//Par exemple : b ou p ou d ou t, t ou d, i ou j, etc.
//Utile pour les sons qui se ressemblent
ASREngine.MaxAlternates = 4;
```

Grâce à lui, le moteur de reconnaissance vocale ne retourne non pas un résultat mais jusqu'à 4. Pourquoi autant? Et bien simplement parce que la différence sonore entre un p, un b, un t et un d est tellement minime que de cette manière il est possible d'avoir les 4.
Tout au long de ce tutoriel, de nouvelles notions pas très courantes ont été abordées. En effet, la reconnaissance vocale est encore très peu présente dans l'informatique. J'espère que ce tutoriel aura servi à plus d'un, car, je l'ai expérimenté moi-même, les informations sur le sujet sont plutôt maigres. Si vous avez des questions concernant la grammaire XML ou d'autres choses en rapport avec ce tutoriel, n'hésitez pas à me contacter par MP.

Partager