Programmer le traitement de la parole

Jean-Marc Farinone
(Maître de Conférences CNAM)

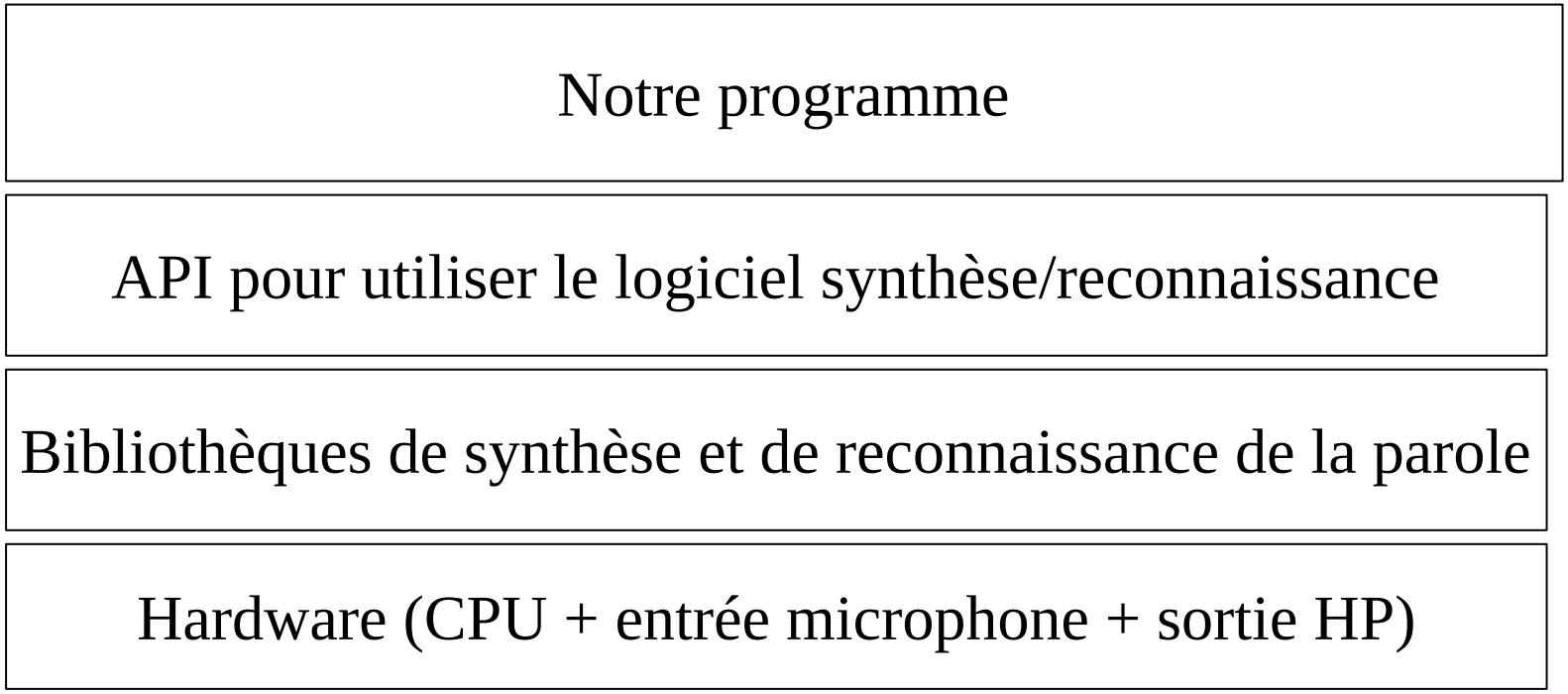
Plan de l'exposé

- Présentation
- La "pile" pour traiter la parole
- Synthèse de la parole : JSML
- Reconnaissance de la parole : JSGF
- Exemples d'applications
- Bibliographie

Présentation

- Intégrer la parole dans les interfaces homme-machine
- En sortie => synthèse
- En entrée => reconnaissance
- Ecrire des programmes qui "parlent correctement" (emphase, fin de phrase, ...) qui lance du code lors de phrases reconnues.
- Un langage de programmation : Java

La "pile" pour traiter la parole



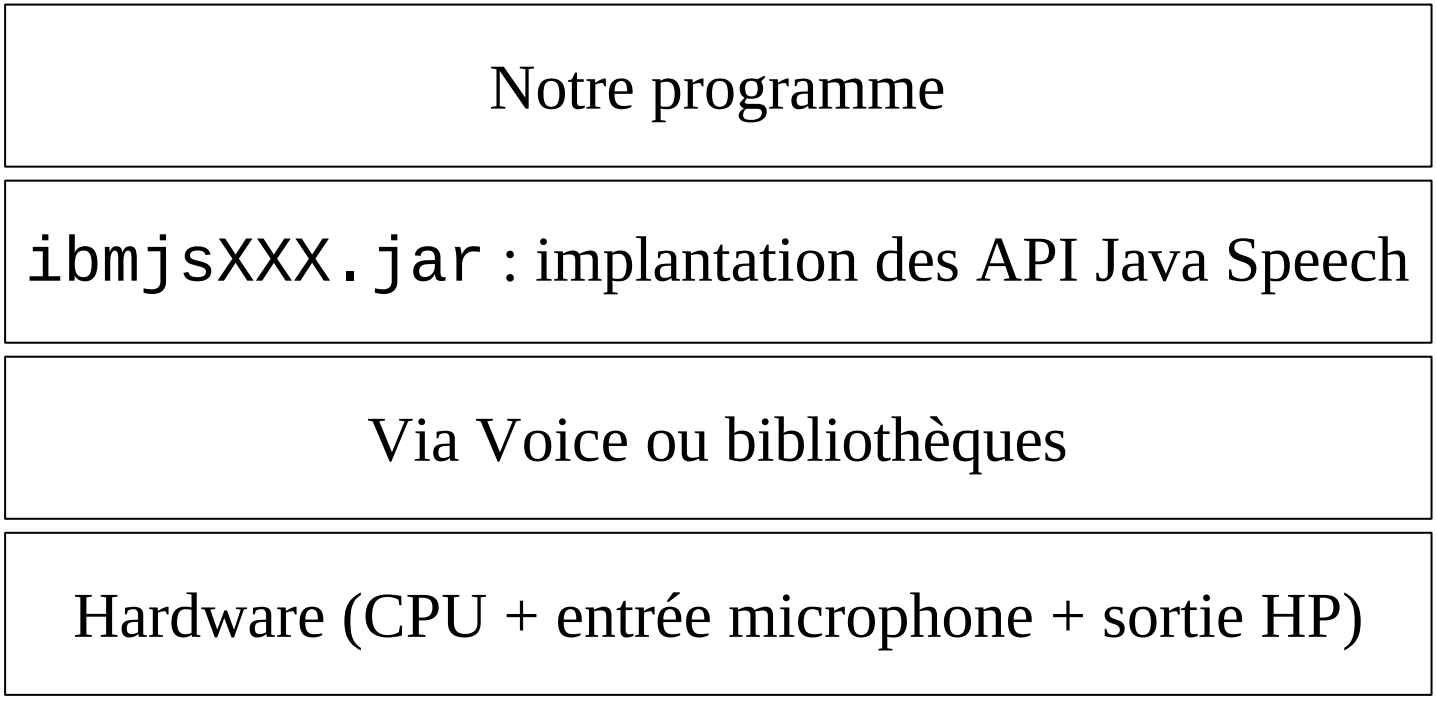
Java Speech

- Java Speech API (JSAPI) = essentiellement des spécifications pour utiliser la parole dans des programmes
- Voir à `http://java.sun.com/products/java-media/speech`
- Interfaces homme-machine pour :
  - les dictées vers l'ordinateur, la reconnaissance, la synthèse de la parole
- Défini par SUN, Apple, AT&T, Dragon Systems, IBM, Novell, Philips et Texas.

Implantation de Java Speech

- IBM's "Speech for Java"
  (`http://www.alphaworks.ibm.com/tech/speech`)
  - avec Via Voice. Pour l'anglais US et UK, français, allemand, italien, espagnol, japonais.
  - Windows 9x, NT et Linux (RedHat Linux 6)
- Lernout & Hauspie racheté par ScanSoft
  (`http://www.lhsl.com/default2.htm`)
  - Sun Solaris OS version 2.4 et plus + Win + Mobile + automobile

La "pile" Java Speech d'IBM



Implantation de Java Speech (suite)

- Festival (U. Edimbourg, …) sur les Unix
- Conversa Web 3.0 : browser avec de la parole pour Win32
- voir à : `http://java.sun.com/products/java-media/speech/forDevelopers/jsapifaq.html`

Installation Java Speech (Win32)

- `http://www.alphaworks.ibm.com/tech/speech` la technologie JavaSpeech chez IBM
- Après ouverture de l'archive dans un répertoire *dir*, vous avez dans ce répertoire : `README.html`, `install.bat` (script d'installation), `lib/ibmjs.jar` (implantation des interfaces Java Speech), `lib/*.dll` le code natif de ces implantations, dans le répertoire `hello/` l'application Hello, `ref/` répertoire de la documentation de référence Java Speech.
- Vous devez ensuite :
  - modifier la variable `CLASSPATH` : y inclure *dir*`\lib\ibmjs.jar`,
  - modifier la variable `PATH` pour y inclure *dir*`\lib`
  - lancer, dans le répertoire *dir*, `install.bat`

Installation bibliothèques natives de synthèse et reconnaissance

- 1er cas IBM Via Voice:
  - Ce n'est pas gratuit
    - La version de base convient
    - Suivre les instructions d'installation
- 2ieme cas bibliothèques de reconnaissance et synthèse gratuite 90 jours d'utilisation à : `http://www6.software.ibm.com/dl/viavoice/runtime-p` sous réserve d'inscription.

Bibliothèques de reconnaissance et synthèse (pour windows)

Edition V1.0 (Pre-req ViaVoice 98 or Via Voice Run Time)	ibmjs100r13a.exe (1MB)
ViaVoice TTS Run Time V5.1 - 90 Day Trial Brazilian Portuguese	tts5bp.zip (3MB)
Finnish	tts5fn.zip (3MB)
French	tts5fr.zip (3MB)
German	tts5gr.zip (3MB)
Italian	tts5it.zip (3MB)
Spanish	tts5sp.zip (3MB)
U.K. English	tts5uk.zip (3MB)
U.S. English	tts5us.zip (3MB)
ViaVoice Dictation Run Time, incl C&C V8.0 - 90 Day Trial Brazilian Portuguese	vvrtdbr8.zip (73MB)
French	vvrtdfr8.zip (115MB)
German	vvrtdgr8.zip (237MB)
Spanish	vvrtdsp8.zip (88MB)
Italian	vvrtdit8.zip (132MB)

- `tts5fr.zip` = (synthèse) 3Mo
- `vvrtdfr8.zip` = (reconnaissance) 115 Mo !!

Un exercice d'installation

- Installer IBM Voice (ou les bibliothèques 90 jours d'utilisation gratuites : bien lancer les 2 `Setup.exe` celui de la reconnaissance et celui de la synthèse).
- Installer JavaSpeech d'IBM (gratuit), i.e. ouvrir l'archive `ibmjsXXXwin.zip`
- Lancer `hello.bat` : ça marche !!

Synthèse de la parole (TTS)

Synthèse de la parole (TTS)

- Texte => phrases parlées : Text To Speech
- Il faut :
  - 1) Analyser le texte d'entrée en paragraphes, phrases, début et fin de phrase pour une meilleure intonation.
  - 2) Repérer les constructions idiomatiques de la langue (abréviation, lecture des dates, des sommes d'argent, ...) et savoir les différencier :
    - St. Mathews hospital is on Main St. -> "Saint Mathews hospital is on Main street"
  - 3) Convertir chaque mot en suite de phonèmes (unité sonore élémentaire)
  - 4) Traiter la prosodie i.e. le rythme, la "mélodie" de l'élocution, ...
  - 5) La production sonore

Démonstration

- `1litValCCAM.bat`

Le synthétiseur de la parole

- Est fonction d'une langue (français, allemand, anglais) et d'une voix.
- Les divers états d'un synthétiseur sont l'activation, la mise en pause, la reprise et l'arrêt de la lecture.

Rappel Java

- Une référence d'interface peut repérer tout objet
- Non pas d'une interface (pourquoi ?)
- D'une classe qui implémente cet interface.

```
public interface MonInterface { …}
public class MaClasse implements  MonInterface { …}

…

   MaClasse refObj = new MaClasse(); // ou bien un objet récupéré
   MonInterface ref = refObj;
```

Programme (complet) de synthèse de la parole


```
import javax.speech.*;
import javax.speech.synthesis.*;
import java.util.Locale;

public class HelloWorld2 {
   public static void main(String args[]) {
      try {
         // Récupérer le synthesiseur francais
         Synthesizer synth = Central.createSynthesizer(
            new SynthesizerModeDesc(Locale.FRENCH));

         // Prepare le synthesiseur pret a parler
         synth.allocate();
         synth.resume();

         // Prononce une phrase "C'est la valeur C ... "
         String phraseAProno ncer = "C'est la valeur C, ";
         for (int i=0; i < args.length; i++)
            phraseAProno ncer += args[i];
         synth.speakPlainText(phraseAProno ncer, null);

         // Attend jusqu'a la fin de la lecture
         synth.waitEngineState(Synthesizer.QUEUE_EMPTY);

         // Désalloue le synthesiseur
         synth.deallocate();
      } catch (Exception e) { e.printStackTrace(); }
   }
}
```

Explication du programme

- Recherche du synthétiseur français
- Positionne le synthétiseur prêt à parler.
- Utilisation de `speakPlainText()`
- MAIS, on aimerait :
  - lire des phrases avec une certaine voix, une certaine prosodie, en insistant sur certains mots, etc..

Le synthétiseur de la parole (suite)

- On peut, dynamiquement, changer certaines caractéristiques comme :
  - le volume sonore du silence au volume maximum.
  - la vitesse d'élocution
  - la tessiture (hauteur moyenne et intervalle autour de cette moyenne) de la voix
  - le style de voix (homme, femme, robot, agé, jeune, enrhumé, heureux, ...)

Synthèse de la parole : JSML

- JSML = langage de balisage (DTD XML)
- Annotation du texte voir à
  `http://java.sun.com/products/java-media/speech/forDevelopers/JSML/index.html`
- Exemple de balises JSML
  - <BREAK MSECS="1000"/> arrêt de la lecture pendant 1 seconde.
  - <EMP attributs> ... </EMP> insister sur le texte compris entre les balises.
    <EMP LEVEL="strong">mesdames</EMP>
  - <SAYAS attributs> ... </SAYAS>
    <SAYAS SUB="S S deux zI">SSII</SAYAS>
  - <PROS attributs> ... </PROS> caractéristique de prosodie (tessiture de la voix, vitesse d'élocution, ...)
    <PROS RATE="150">texte dit à 150 mots par minute</PROS>
    <PROS PITCH="100">voix grave</PROS>
    <PROS PITCH="200">voix aiguë</PROS>
    <PROS VOL="0.1">chuchotement</PROS>
    <PROS VOL="0.9">je parle très fort</PROS>

Démonstration JSML

```
"Bienvenue à ce séminaire. Bonjour
  <EMP LEVEL="strong">mesdames</EMP>, bonjour
 <EMP LEVEL="reduced">, messieurs</EMP>
 <BREAK MSECS="1000" />
 avec la balise SAYAS, le sigle<SAYAS SUB="S S deux zI">SSII</SAYAS>
 <BREAK MSECS="1000" />
  sans balise SSII
  <BREAK MSECS="1000" />
 <PROS PITCH="100">essai d'une voix grave</PROS>
 <PROS PITCH="200">essai d'une voix aiguë</PROS>
 <BREAK MSECS="1000" />
 <PROS VOL="0.1">en parlant très doucement</PROS>
 <BREAK MSECS="1000" />
  <PROS VOL="0.9">ou encore très fort</PROS>
 <BREAK MSECS="1000" />
  fin du message"
```

- `2bienvenue.bat`

Programmation JSML

```
import javax.speech.*;
import javax.speech.synthesis.*;

 public class Bienvenue {
    public static void main(String args[]) {
       try {
          Synthesizer synth = Central.createSynthesizer(
             new SynthesizerModeDesc(Locale.FRENCH));
          synth.allocate();
          MonSpeakable monSpeak = new MonSpeakable();
          synth.speak(monSpeak, null);
          synth.speak("fin du message", null);
     }
```

Programmation JSML (suite)

- avec :

```
class MonSpeakable implements Speakable {
   public String getJSMLText() {
     StringBuffer buf = new StringBuffer();
        buf.append("Bienvenue à ce séminaire ");
 buf.append("<EMP LEVEL=\"strong\">" + "mesdames" + "</EMP>");
 buf.append("<EMP LEVEL=\"reduced\">" + ", messieurs" +
   "</EMP>");
        ...
        return buf.toString();
  }
}
```

Programmation JSML (fin)

- Utilisation de la méthode `speak(Speakable, SpeakableListener)` sur l'objet synthétiseur.
- `Speakable` est une interface qui spécifie la méthode `public String getJSMLText()` qui doit renvoyer une `String` contenant un texte formaté en JSML.

Reconnaissance de la parole

Reconnaissance de la parole

- Java Speech se "restreint à" :
  - une seule langue
  - une seule entrée audio
- Java Speech peut s'adapter à une voix quelconque
- L'ensemble des phrases à reconnaître = les grammaires peuvent être dynamiquement chargées.

Travail d'un reconnaisseur de parole

- Traitement grammatical : créer et activer une grammaire pour connaître ce qui est significatif d'écouter
- Analyse du signal : Faire une analyse spectrale de l'entrée audio
- Reconnaissance des phonèmes : comparer les motifs spectraux aux phonèmes du langage

Travail d'un reconnaisseur de parole (suite)

- Reconnaissance des mots : comparer les suites de phonèmes aux mots à reconnaître spécifiés par la grammaire active.
- Générer le résultat : avertir l'application des phrases de la grammaire qui ont été reconnues.

Démonstration

- Dialogue entre l'humain et l'ordinateur

ordinateur> Bonjour humain, mon nom est ordinateur, quel est votre nom ?
humain> Je m'appelle *prénom nom*
ordinateur> Bonjour *prénom nom*
humain> Répétez après moi
ordinateur> Je t'écoute

humain> dictée terminée par "C'est fini"
ordinateur> répète la dictée

humain> au revoir
ordinateur> A bientôt

- `3demoJS`

Remarques sur la démo

- reconnaissance de C'est fini, au revoir, ... ainsi que des non-terminaux *prénom*, *nom*
- reconnaissance d'une dictée quelconque
- synthèse de la parole :
  - traitement associé (bonjour *prénom nom*)
  - synthèse de la dictée
- Remarque : programme très facilement transposable en allemand, anglais, etc.

Reconnaissance de la parole

- Demande d'indiquer la langue, l'empreinte vocale de l'utilisateur
- Demande d'indiquer une grammaire :
  - de dictée continue (reconnaissance du langage naturel)
  - ou de commandes vocales

Les "Grammars"

- Une grammaire (`Grammar`) doit être créée et activée pour un reconnaisseur (`Recognizer`) afin qu'il sache ce qu'il doit écouter principalement.
- *DictationGrammar* : grammaire de dictée = ce qui permet de reconnaitre les mots d'une langue naturelle (pour les dictées dans les mails, les traitements de texte, ...)
- *RuleGrammar* = grammaire de règles = grammaire hors contexte (construite à l'aide de JSGF) donnée par un programmeur décrivant des phrases susceptibles d'être dites par un utilisateur.

Reconnaissance de la parole (suite)

- Les états d'un reconnaisseur sont l'activation, la désactivation.
- Il gère les diverses grammaires qu'il utilise
- Java Speech utilise la technique des listeners lorsqu'une phrase a été reconnue.

Listener en Java

- Une des idées en Java : programmation par délégation
- cf. cours IHM sur les événements
- A un objet susceptible d'être utilisé à n'importe quel moment (par l'utilisateur ou autre ...) est associé des objets "listener" qui sont avertis lorsque l'objet source a été activé.

Listener en Java

- Ces listeners lance alors une méthode appropriée spécifiée à l'avance
- Question :
  - comment Java le permet il ?
- Réponse :
  - interface, méthodes spécifiées dans cet interface, classe qui doit implanter cette interface, `Vector` de listeners dans l'objet.

Grammaire de dictée (Dictation Grammar)

- Une telle grammaire impose peu de contraintes i.e. s'intéresse à tout ce qui peut être dit dans une langue donnée
- Dictation Grammar = "grammaire" d'un langage naturel (français, anglais, etc.)
- Nécessite plus de ressources système que les rule grammar

Grammaire de dictée (Dictation Grammar)

- On ne créée pas dans le programme de dictation grammar : ce sont des grammaires données par les couches basses
- Peut être optimisées pour certains domaines (médecine, commercial, etc.)

Grammaire de règles (Rule Grammar)

- Java Speech permet :
  - de définir des grammaires (de règles) qui décrivent ce qu'un utilisateur peut dire
  - d'associer un traitement aux phrases reconnues.
- Grammaires de reconnaissance sont définies par JSGF (Java Speech Grammar Format)
- Spécifications de JSGF à : `http://java.sun.com/products/java-media/speech/forDevelopers/JSGF/index.html`

Programmer la reconnaissance de la parole

- Les diverses notions vues précédemment sont traduites par des objets de classes.
- Ces objets sont construits (par `new MaClasse()`) ou récupérés car donnés par la bibliothèque Java Speech.

Un objet pour la reconnaissance de la parole

- Un reconnaisseur de parole est une "machine" qui convertit de la parole en format texte.
- Les classes et interfaces utiles se trouvent dans le paquetage `javax.speech.recognition`.
- dont l'interface `Recognizer` (qui hérite de `javax.speech.Engine`).

Reconnaissance : un premier exemple simple

- Voir à : `http://java.sun.com/products/java-media/speech/forDevelopers/jsapi-guide/Recognition.html`

```
grammar javax.speech.demo;

public <phrase> = bonjour à tous
       | meilleurs voeux
       | merci mon cher ordinateur;
```

- `4RecSimple.bat`

Reconnaissance : un premier exemple simple (et complet) 1/3

```
import javax.speech.*;
import javax.speech.recognition.*;
import java.io.FileReader;
import java.util.Locale;

public class HelloWorld extends ResultAdapter {
  static Recognizer rec;
  public static void main(String args[]) {
    try {
       // Récupère un reconnaisseur pour la langue française.
       rec = Central.createRecognizer(new
EngineModeDesc(Locale.FRENCH));

       // Lance ce reconnaisseur
       rec.allocate();

       // Charge un fichier contenant une rule grammar, construit
une RuleGrammar et la rend utilisable.
       FileReader reader = new FileReader(args[0]);
       RuleGrammar gram = rec.loadJSGF(reader);
       gram.setEnabled(true);
```

Reconnaissance : un premier exemple simple (et complet) 2/3

```
      // Associe un listener à ce reconnaisseur
      rec.addResultListener(new HelloWorld());

      // Avertit des changements d'états du reconnaisseur
      // ici sa création
      rec.commitChanges();

      // Demande le focus et lance l'écoute
      rec.requestFocus();
      rec.resume();
    } catch (Exception e) {
      e.printStackTrace();
    }
}
```

Reconnaissance : un premier exemple simple (et complet) 3/3

```
// Reçoit uniquement les événements RESULT_ACCEPTED :
// Ecrit ce qui a été reconnu,
// désalloue les ressources et termine le programme

public void resultAccepted(ResultEvent e) {
        Result r = (Result)(e.getSource());
        ResultToken tokens[] = r.getBestTokens();

        for (int i = 0; i < tokens.length; i++)
           System.out.print(tokens[i].getSpokenText() + " ");
        System.out.println();

        rec.deallocate();
         System.exit(0);
}
}
```

Explication de demoJS

- grammaire de texte à reconnaître

`hello_fr.gram`

```
grammar hello;
  <first> = Henri {Henri}
          | Maurice {Maurice}
          | Victor {Victor}
  ;
  <last>  = Matisse {Matisse}
          | Chevalier {Chevalier}
          | Hugo {Hugo}
 ;
  <name> = <first> <last>;
  public <nameis> = Je m'appelle {name} <name>;
  public <begin> = Répétez après moi {begin};
  public <stop> = C'est fini {stop};
  public <bye> = Au revoir {bye};
```

Une Rule grammar

- A une telle grammaire sera associé un objet `RuleGrammar`
- Une règle `public` est une règle qui est activée lorsqu'elle a été matchée.
- La balise (tag) `{}` est retourné sous forme de `String` lorsque la règle a été matchée.
- < > = non terminaux
- terminaux écrits en texte plein

Rule grammar

- | = ou bien
- [ ] optionnel (i.e. 0 ou 1)
- () = regroupement
- * = 0 ou plusieurs

Une Rule grammar : exercice

- Donner des phrases construites par la grammaire :

```
grammar SimpleCommandes;
public <Commande> = [<Politesse>]
<Action> <Objet> (et <Action>
<Objet>)*;
<Action> = ouvrez | fermez;
<Objet> = la fenêtre | la porte;
<Politesse> = s'il vous plaît;
```

Programme reconnaissance de la parole

- Chargement d'un reconnaisseur de parole
- Chargement de la grammaire
- Associer un listener à cette grammaire
- Le listener lance sa méthode `resultAccepted()` lorsqu'une règle a été reconnue.

Reconnaissance/synthèse de la parole

- Le fichier de configuration (de propriétés) entre autre pour les réponses de l'ordinateur.

`res_fr.properties`

```
# file for grammar
grammar   = hello_fr.gram

# things we say
greeting  = Bonjour Humain. Mon nom est Ordinateur. Quel
est <EMP/>votre nom?
hello     = Bonjour
listening = Je t'écoute.
eh        = Pardon? Eh? Quoi?
bye       = À bientôt!
```

Multithreading Java

- Un programme Java lance plusieurs threads à l'exécution.
- Un programme Java se termine lorsque toutes les threads non démons sont terminées
- Ce qui explique que le programme suivant ne se termine pas.

Multithreading Java (suite)

```
import java.awt.*;

public class ButtonTestApp extends Frame {
  public ButtonTestApp () {
    add("Center", new Button("Premier bouton"));
    pack();
    setVisible(true);
  }
  public static void main(String args[ ]) {
    Frame fr = new ButtonTestApp ();
  }
}
```

- Une fenêtre reste affichée. Pourquoi ?
- Car thread d'affichage (AWT)

Classe anonyme en Java

- C'est un raccourci syntaxique permettant de créer "à la volée" un objet d'une classe sans indiquer le nom de la classe
- Syntaxe :

```
new ClasseDeBaseOuInterfaceAImplémenter() { …
   // corps de la classe anonyme }
```

- Souvent mis comme argument d'une méthode ce qui donne :

```
ref.meth(new
ClasseDeBaseOuInterfaceAImplémenter() { …
// corps de la classe anonyme });
```

Programme traitant la parole : syntaxe (1/7)

```
import java.io.*;
import java.util.*;
import javax.speech.*;
import javax.speech.recognition.*;
import javax.speech.synthesis.*;

public class Hello {
    static RuleGrammar ruleGrammar;
    static DictationGrammar dictationGrammar;
    static Recognizer recognizer;
    static Synthesizer synthesizer;
    static ResourceBundle resources;

  // Le listener pour la rule grammar. Sa méthode
  // resultAccepted() est appelée lorsqu'une
  // règle publique a été reconnue.
  // On teste alors les balises associées à cette grammaire
  // et on lance l'action correspondante à la balise
  // (filtre sur les balises).
  // Utiliser les balises plutôt que de tester
  // directement les phrases dites permet d'être
  // indépendant de la langue. On peut alors changer la
  // langue sans changer ce programme.
  // Remarque : on utilise une classe interne


  static ResultListener ruleListener = new ResultAdapter() {
```

Programme traitant la parole : syntaxe (2/7)

```
public void resultAccepted(ResultEvent e) {
    try {
    // On récupère les balises à l'aide de l'événement
FinalRuleResult result = (FinalRuleResult) e.getSource();
String tags[] = result.getTags();
  // L'utilisateur à dit la phrase correspondant à la
  //  règle "name" ("Je m'appelle ...", "my name is ...")
    if (tags[0].equals("name")) {
      // On construit la réponse ("bonjour …")
      // On récupère ce qui correspond à "hello" dans
      // le fichier de configuration des phrases locale s
      String s = resources.getString("hello");
      for (int i=1; i<tags.length; i++)
        s += " " + tags[i];
        // on lance la synthèse vocale
        speak(s);

  // L'utilisateur a dit "répéter après moi"
    } else if (tags[0].equals("begin")) {
      // l'ordinateur répond et se prépare pour la dictée
      speak(resources.getString("listening"));

      // la Rule grammar est désactivée (hormis sa règle
      // <stop>) au profit de la dictation grammar.
      ruleGrammar.setEnabled(false);
      ruleGrammar.setEnabled("<stop>", true);
      dictationGrammar.setEnabled(true);
      recognizer.commitChanges();
```

Programme traitant la parole : syntaxe (3/7)

```
    // L'utilisateur a dit "c'est fini"
    } else if (tags[0].equals("stop")) {
      // la dictation grammar est désactivée
      // au profit de la rule grammar.
      dictationGrammar.setEnabled(false);
      ruleGrammar.setEnabled(true);
      recognizer.commitChanges();

    // L'utilisateur a dit "au revoir"
    } else if (tags[0].equals("bye")) {
      // L'ordinateur dit à bientot
      speak(resources.getString("bye"));
      if (synthesizer!=null)
      synthesizer.waitEngineState(Synthesizer.QUEUE_EMPTY);
      Thread.sleep(1000);
       System.exit(0);
     }
   } catch (Exception ex) { ex.printStackTrace();}
  }};

   // Le listener pour la dictation grammar.
   // Sa méthode resultUpdated() est appelé
   // pour chaque mot reconnu.
   // Sa méthode method resultAccepted() est appelé
   // lorsque l'objet dictation grammar est désactivé i.e.
   //  quand l'utilisateur a dit "c'est fini".
   static ResultListener dictationListener = new
ResultAdapter() {
```

Programme traitant la parole : syntaxe (4/7)

```
  int n = 0; // nombre de mots lus

  public void resultUpdated(ResultEvent e) {
    Result result = (Result) e.getSource();
    for (int i=n; i<result.numTokens(); i++)
System.out.println(result.getBestToken(i).getSpokenText());
      n = result.numTokens();
    }

  public void resultAccepted(ResultEvent e) {
    // On récupère tout ce qui a été dit
    Result result = (Result) e.getSource();
    String s = "";
    // L'ordinateur le répète
    for (int i=0; i<n; i++)
      s += result.getBestToken(i).getSpokenText() + " ";
      speak(s);
      n = 0;
    }
  };

  // L'audio listener imprime des traces du volume d'entrée
  static RecognizerAudioListener audioListener =
    new RecognizerAudioAdapter(){
      public void audioLevel(RecognizerAudioEvent e) {
        System.out.println("volume " + e.getAudioLevel());
      }
  };
```

Programme traitant la parole : syntaxe (5/7)

```
    // La méthode qui "fait parler l'ordinateur".
    // Si le synthesizer n'est pas disponible
    // elle écrit les mots à l'écran.
    static void speak(String s) {
      if (synthesizer != null) {
        try {
          synthesizer.speak(s, null);
        } catch (Exception e) {e.printStackTrace();}
      } else {
        System.out.println(s);
      }
    }

    //
    // la méthode main()
    //
```

Programme traitant la parole : syntaxe (6/7)

```
  public static void main(String args[]) {
    try {
      // Chargement de ressources locales
// la "culture" locale (exemple le français métropolitain fr_FR)
      System.out.println("locale is " + Locale.getDefault());

// recherche d'un fichier propre à cette culture res_fr.properties
      resources = ResourceBundle.getBundle("res");

// récupération et allocation du reconnaisseur par défaut :
// celui correspondant à la culture locale (null)
      recognizer = Central.createRecognizer(null);
      recognizer.allocate();
// ajout d'un audio listener (qui testera le volume sonore)
      recognizer.getAudioManager().addAudioListener(audioListener);

// récupération de la dictation grammar
      dictationGrammar = recognizer.getDictationGrammar(null);
// ajout d'un listener à cette dictation grammar
      dictationGrammar.addResultListener(dictationListener);

// crée une rule grammar à partir du fichier hello_fr.gram
// (lu dans le fichier des propriétés locales). Lui associe
// un listener et active cette grammaire.
     Reader reader = new FileReader(resources.getString("grammar"));
     ruleGrammar = (RuleGrammar) recognizer.loadJSGF(reader);
     ruleGrammar.addResultListener(ruleListener);
     ruleGrammar.setEnabled(true);
```

Programme traitant la parole : syntaxe (7/7)

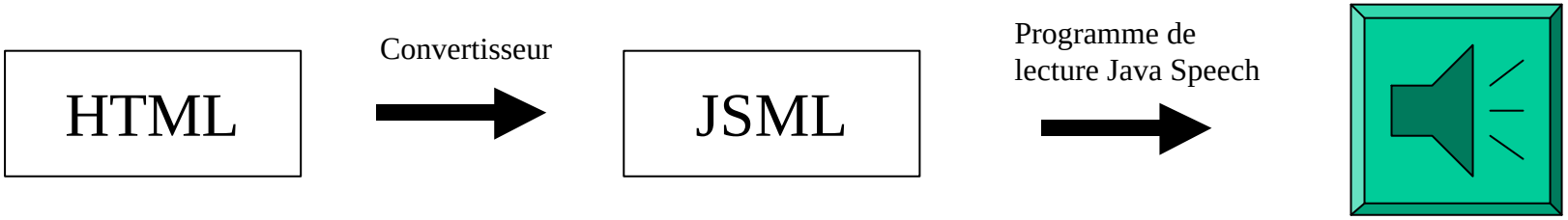
```
  // informe le reconnaisseur des états des grammaires et le lance.
    recognizer.commitChanges();
    recognizer.requestFocus();
    recognizer.resume();

  // Synthèse de la parole
  // crée un synthesizer correspondant à la culture locale (null),
  // lance un message de bienvenu (méthode statique de cette classe)
    synthesizer = Central.createSynthesizer(null);
    if (synthesizer!=null) synthesizer.allocate();
    speak(resources.getString("greeting"));
    } catch (Exception e) { e.printStackTrace();System.exit(-1);}
  }
}
```

Exemples d'applications

- "Ecouter les pages Web" (Hervé Declipeur)
- Interroger une base de données par la parole (Patrice Gangnard projet valeur C IAGL 1999) : exercice C CAM 2001

Lecture de pages HTML

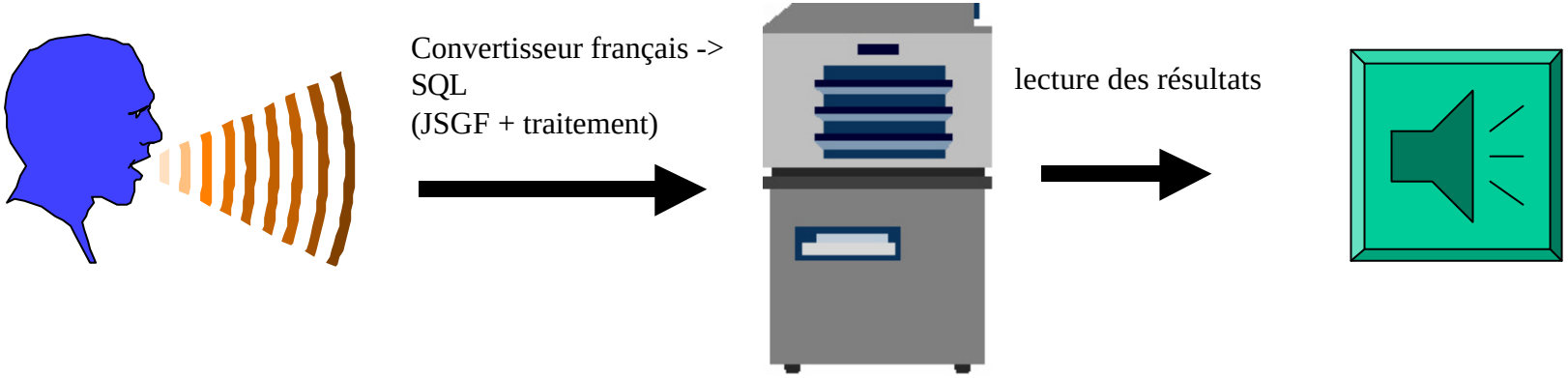
- Hervé Déclipeur (valeur C CAM du CNAM) RVDesign, `declipeur@noos.fr`



- Démonstration : `5litWeb.bat`

Interroger BD par le parole

- Patrice Gangnard projet valeur C IAGL 1999



- Démonstration : `6BDParole.bat`

Bibliographie

- `http://java.sun.com/products/java-media/speech` : page d'accueil Java Speech.
- IBM's "Speech for Java" (`http://www.alphaworks.ibm.com/tech/speech`)
- Spécifications de JSGF à : `http://java.sun.com/products/java-media/speech/forDevelopers/JSGF/index.html`

Bibliographie

- Patrice Gangnard Interrogation de bases de données par la parole val C IAGL CNAM 1999.
- `http://java.sun.com/products/java-media/speech/forDevelopers/jsapi-guide/index.html` : le guide de programmation Java Speech