

*Master 2 Informatique
Spécialité Professionnel*

Laboratoire d'Informatique, de Robotique
et de Microélectronique de Montpellier
(LIRMM)

AOUAD Chakir

Rapport de Stage

Tuteur : Jean-Yves Delort
Année 2005 - 2006

Université Montpellier 2



Remerciements

Je tiens tout particulièrement à remercier mon tuteur de stage, Mr Delort Jean-Yves, grâce à qui j'ai pu effectuer ce stage et découvrir le monde de la téléphonie mais également celui de la reconnaissance vocale. Mes remerciements lui sont aussi adressés pour ses précieux conseils et pour avoir bien voulu me faire part de son expérience.

Un grand merci à Dirk Schnelle, leader du projet JVoiceXML, pour avoir répondu à mes nombreuses questions et pour sa collaboration.

Sommaire

Introduction		
1.1	<i>Présentation de l'établissement d'accueil</i>	9
1.2	<i>Présentation du projet</i>	10
 Objectifs de la mission technique		
2.1	<i>Sujet du stage</i>	13
2.2	<i>Cahier des charges</i>	13
2.3	<i>Contrainte</i>	14
 VoiceXML		
3.1	<i>Présentation</i>	17
3.2	<i>Analogie HTML et VoiceXML</i>	18
3.3	<i>Architecture des applications VoiceXML</i>	19
3.4	<i>JVoiceXML</i>	20
 Reconnaissance vocale		
4.1	<i>Les Modèles de Markov Cachés</i>	23
4.2	<i>Théorie de la reconnaissance vocale</i>	25
4.2.1	<i>Extraction de caractéristiques</i>	28
4.2.2	<i>Modèle acoustique</i>	28
4.2.3	<i>Modèle de langage</i>	31
4.3	<i>Etude des logiciels existants</i>	32
4.3.1	<i>HTK</i>	32
4.3.2	<i>Sphinx 4</i>	33
4.4	<i>Création d'un modèle acoustique français</i>	35
4.4.1	<i>Utilisation du modèle créé par le LIUM</i>	35
4.4.2	<i>Création d'un nouveau modèle</i>	36
4.4.2.1	<i>Collecte des données</i>	36
4.4.2.2	<i>Apprentissage du modèle acoustique</i>	38
 Synthèse vocale		
5.1	<i>Théorie de la synthèse vocale</i>	41
5.1.1	<i>Types de synthèses vocales</i>	41
5.1.2	<i>Fonctionnement d'un synthétiseur vocal</i>	44
5.2	<i>Utilisation de MBrola et intégration à FreeTTS</i>	46

Serveur téléphonique	
6.1	<i>Présentation d'Asterisk</i> 49
6.2	<i>Etude et installation</i> 51
6.2.1	<i>Choix de la version</i> 51
6.2.2	<i>Tests et mise en place des services</i> 52
6.3	<i>Intégration à Sphinx 4 et réception en streaming</i> 54
6.3.1	<i>Couplage Asterisk – Sphinx 4</i> 54
6.3.1.1	<i>Customisation de Sphinx 4</i> 55
6.3.1.2	<i>Création du module Asterisk</i> 56
6.3.1.3	<i>Mise en place d'un serveur tiers</i> 57
6.3.2	<i>Transfère sur un appel d'un streaming audio</i> 58
6.4	<i>Intégration à JVoiceXML</i> 59
6.4.1	<i>Asterisk-Java</i> 59
6.4.2	<i>Connectivité Asterisk – JVoiceXML</i> 60
Au-delà du stage	
7.1	<i>Demandes et proposition</i> 63
7.2	<i>Projet de création d'entreprise</i> 64
Conclusion	65
Annexes	
<i>Algorithmes</i>	68
<i>L'algorithme du forward</i>	68
<i>L'algorithme de Viterbi</i>	70
<i>L'algorithme du forward-backward</i>	71
<i>Création d'un modèle acoustique avec SphinxTrain</i>	74
<i>Introduction</i>	74
<i>Installation</i>	74
<i>Configuration de SphinxTrain</i>	75
<i>Création du modèle acoustique</i>	79
<i>FAQ</i>	80
<i>Ecriture d'un module pour Asterisk 1.2</i>	84
<i>Récupération des sources Asterisk</i>	84
<i>Ecrire son propre module</i>	86
<i>Explication sur les différentes fonctions de base</i>	86
<i>Ecriture de l'application HelloWorld</i>	87
<i>Accélérer le développement</i>	90
<i>Eléments du langage VoiceXML</i>	92
<i>Bibliographie/Références</i>	95
<i>Glossaire</i>	96

Introduction

Dans le cadre du stage de fin d'études du Master 2 Informatique, j'ai été accueilli au sein du Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM). Ce stage s'attaque à plusieurs fronts qui s'articulent autour du langage VoiceXML à savoir la reconnaissance et la synthèse vocale d'une part et la téléphonie (traditionnelle ou sur IP) d'autre part, avec pour objectif leur intégration.

1.1 Présentation de l'établissement d'accueil

Le LIRMM est une unité mixte de recherche, dépendant conjointement de l'Université Montpellier II et du Centre National de la Recherche Scientifique.

Les activités de recherche du LIRMM le positionnent pleinement dans les "Sciences et Technologies de l'Information et de la Communication" (STIC) qui constituaient, en 2000, l'une des priorités du gouvernement français.

En effet, les recherches actuelles et en émergence au LIRMM couvrent un large spectre des STIC

- l'informatique fondamentale,
- l'interaction entre les systèmes informatiques et les utilisateurs,
- le développement de machines communicantes d'intervention, de production ou de service,
- le développement des composants matériels et logiciels des systèmes informatiques et de communication.

Regroupant près de 300 personnes (dont un peu plus de la moitié sont des permanents), le LIRMM pilote deux formations doctorales, l'une en Informatique, l'autre en Systèmes automatiques et microélectroniques.

Aux soutiens du CNRS et de l'Université Montpellier II, s'ajoutent ceux d'une quinzaine de programmes de recherche nationaux et d'une dizaine de programmes de recherche européens auxquelles le LIRMM participe.

1.2 Présentation du projet

Au cours de ces dernières années, la révolution de la technologie de la voix sur IP a bouleversée l'utilisation du réseau Internet et de la téléphonie. En effet, la voix sur IP (VoIP) présente de nombreux avantages, elle permet non seulement de téléphoner à moindre coût, mais aussi de combiner voix, données et vidéo sur le même réseau de transport.

Parallèlement, l'apparition du langage standard pour la création d'applications vocales, VoiceXML, a permis l'accès aux systèmes d'informations de l'entreprise à l'aide de dispositifs multimodaux.

Le monde du logiciel libre connaît un succès croissant qui compte parmi les outils les plus prisés Asterisk pour la téléphonie et JVoiceXML pour le standard VoiceXML.

Asterisk en plus de proposer toutes les fonctions classiques d'un standard téléphonique (PBX) suscite un engouement sans précédent qui s'explique par sa richesse en terme de fonctionnalités et le faible coût matériel nécessaire.

JVoiceXML quant à lui, est une implémentation partielle du standard VoiceXML qui s'attelle à fournir une plateforme complète et fiable.

En outre, d'autres domaines tournent autour de VoiceXML. Citons particulièrement la reconnaissance automatique de la parole (RAP) qui est une des composantes fondamentales de tout système de réponse automatique (IVR – Interactive Voice Response).

Le couplage de ces différentes technologies présente des avantages indéniables et c'est ce que ce stage se propose de mettre en lumière.

Objectifs de la mission technique

Bien entendu, avant de commencer quelque étude que ce soit, il est important de définir l'objectif à atteindre et les moyens pour y parvenir.

Aussi, dans cette première partie, est présenté le sujet même du stage qui a permis de mettre en avant les points les plus importants et en partie la contrainte.

2.1 Sujet du stage

Le sujet du stage a été défini de la sorte :

« L'objectif de ce stage est d'explorer l'ensemble des possibilités offertes par VoiceXML puis de concevoir un logiciel permettant d'exécuter du code VoiceXML sur une machine distante.

Pour la première partie on regardera comment mettre bout à bout un appel téléphonique avec une application Java.

Pour la seconde partie, le logiciel sera développé sur une architecture client serveur. Le serveur interprétera le code VoiceXML et communiquera les éléments du dialogue à un applet Java qui prononcera le texte et effectuera la reconnaissance vocale. On réutilisera les mêmes composants utilisés par JVoiceXML pour réaliser ces tâches.

Au cours de ce stage vous acquérez de nouvelles compétences sur la reconnaissance vocale, la synthèse vocale, la téléphonie, Java, XML, architecture client serveur.

Conditions requises : excellent niveau de programmation, Java. »

2.2 Cahier des charges

Ce stage s'appuyant sur diverses technologies, divers axes de recherches et de développement ont du être définis :

En rapport avec la téléphonie

- Etudier comment utiliser la téléphonie en Java
- Trouver une solution permettant un CTI (Couplage Téléphonie Informatique)
- Etudier et mettre en place Asterisk

En rapport avec la reconnaissance et synthèse vocale

- Etudier les principes de la reconnaissance vocale (abrégié par ASR ou RAP)
- Créer un modèle acoustique permettant la reconnaissance vocale du français
- Créer un modèle acoustique adapté à la téléphonie
- Intégrer la reconnaissance vocale sur un réseau téléphonique
- Etudier la synthèse vocale (abrégié par TTS)

En rapport avec le langage VoiceXML

- Etudier VoiceXML
- Coupler JVoiceXML à Asterisk
- Franciser JVoiceXML (au travers du TTS et de l'ASR)

2.3 Contrainte

Peu de contraintes ont été définies, puisqu'en effet, une certaine liberté dans le choix des méthodes m'a été laissée. L'unique contrainte étant que les logiciels utilisés devaient être gratuits (avec une forte préférence pour ceux étant open-source).

VoiceXML

Depuis quelques années déjà, l'informatique se veut plus abordable et plus séduisant par l'intégration d'interfaces homme-machine évoluées.

Pourtant, ces interfaces ne sont pas naturelles pour l'homme qui doit s'habituer (pour certains avec peines) à un monde visuel et fenêtré.

L'avancée technologique en matière de traitement du signal, les connaissances du signal vocal et la puissance croissante des processeurs permettent actuellement de fournir une nouvelle interface aux humains: la voix. Ce choix n'est nullement arbitraire: la parole est le moyen de communication le plus naturel pour l'homme.

VoiceXML illustre un exemple d'une tendance très actuelle : la convergence de l'informatique et de la téléphonie (CTI, Computer Telephony Integration) pour l'intégration de services web vocaux.

3.1 Présentation

VoiceXML est un langage destiné à la création d'interfaces purement vocales : reconnaissance vocale en entrée, audio préenregistré et/ou synthèse de la parole en sortie. Les applications sont multiples, tant pour le Web (navigation à la voix, aide aux malvoyants) que pour la téléphonie (fixe ou mobile).

Il utilise la téléphonie car elle est quasi-omniprésente, dans le monde entier.

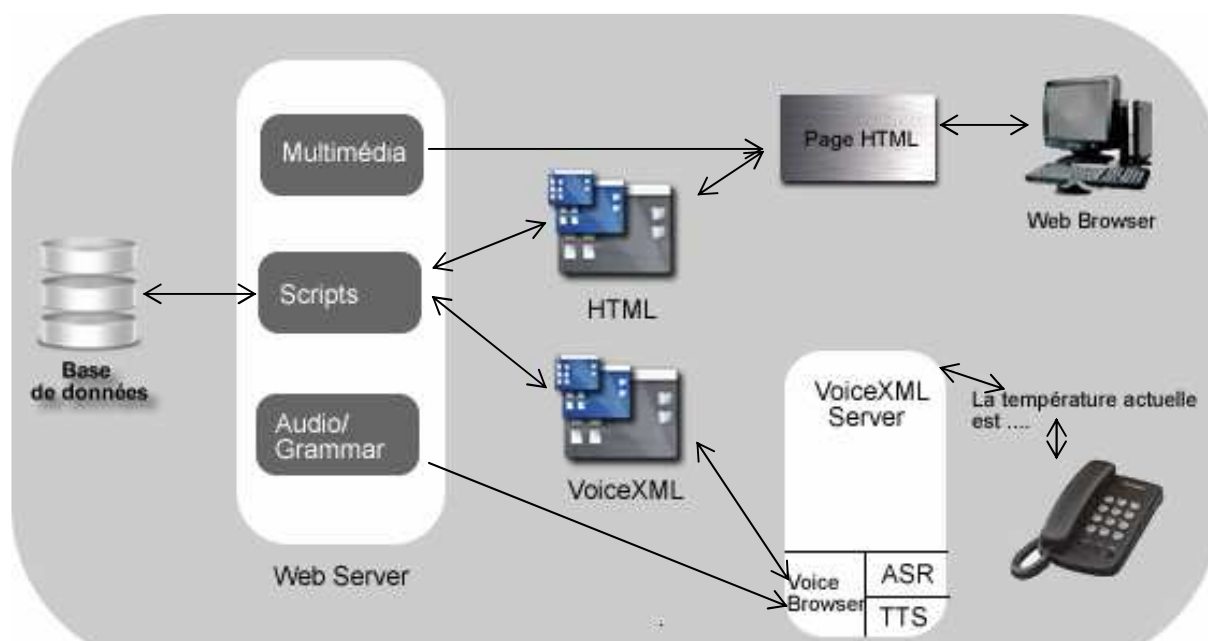
La voix est un moyen de communication puissant et maîtrisé dans les applications mobiles, contrairement au WAP qui apporte un service pratique mais avec des moyens inappropriés. En effet, l'écran est trop petit et la navigation n'est pas intuitive. Les interfaces visuelles sont cependant importantes, et les dispositifs multimodaux qui sont capables de traiter du graphisme et/ou du vocal ("dites le nom de la ville ou choisissez-le à partir de la liste") seront les plus prometteurs.

Par ailleurs, VoiceXML est aussi une technologie orientée Web. Son langage est de haut niveau et le développement en est ainsi facilité. Les applications peuvent être développées à l'aide d'applications web puissantes et bon marché. L'utilisation d'un serveur web facilite la mise à jour et permet de structurer l'information.

Enfin, VoiceXML est également une spécification du VoiceXML Forum, un consortium d'industrie de plus de 300 compagnies. Le forum est actif dans les tests de conformité, l'enseignement, le marketing et a donné le contrôle du développement ultérieur de langage au World Wide Web Consortium (W3C). Puisqu'il s'agit d'une spécification, les applications qui fonctionnent sur une plate-forme devraient fonctionner aussi bien sur une autre.

3.2 Analogie HTML et VoiceXML

Tout comme HTML, VoiceXML est un langage à balises utilisable sur le réseau internet. Le langage HTML est un langage de création de pages web alors que VoiceXML est un langage d'écriture de scénarios vocaux. HTML et VoiceXML ont quelques points en commun, comme le montre la figure ci-dessous, le navigateur vocal (voice browser) vient demander la page VoiceXML sur le serveur http de la même manière qu'un navigateur classique va chercher une page HTML.



Lorsqu'un utilisateur navigue sur des pages Web classiques, codées en HTML, le mode prédominant d'interaction est le pointage souris et saisie de données au clavier. Le sens le plus sollicité est donc la vision. En revanche, lorsque nous utilisons VoiceXML, le mode d'interaction est la voix (les entrées pouvant éventuellement se faire avec les touches du pavé numérique du téléphone).

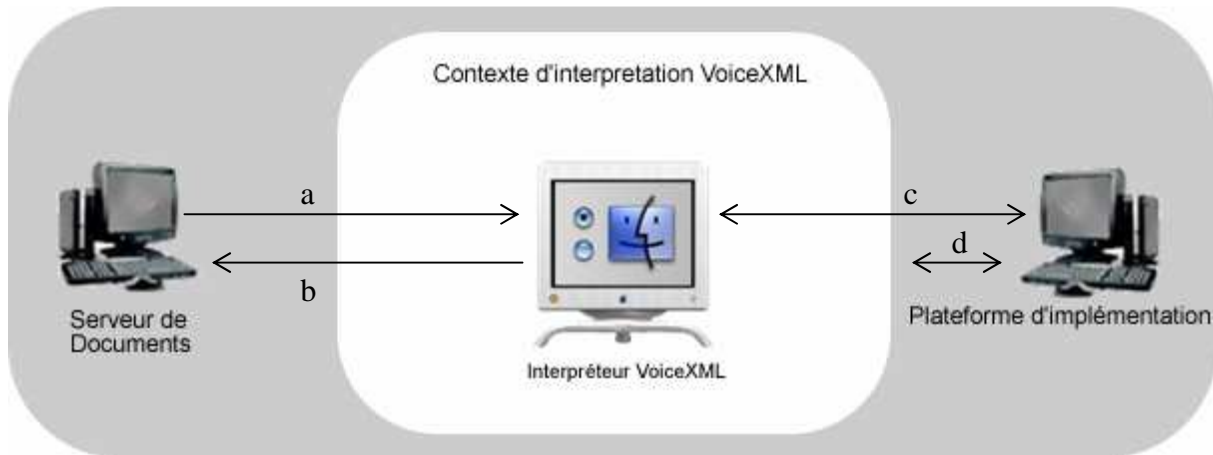
Nous remarquons que le serveur comporte deux entités (sur lesquels nous revenons plus loin dans ce rapport) : TTS et ASR.

TTS (Text To Speech) permet la synthèse vocale de la parole. Les serveurs de synthèse de la parole génèrent de la voix (signal audio) à partir d'un texte. Cette technique permet à un système informatique de restituer sous forme orale, une information fournie sous forme textuelle.

ASR (Automatic speech recognition – RAP en français pour Reconnaissance Automatique de la Parole) permet la reconnaissance de la parole. Couplé à un serveur vocal, la personne consultant un service pourra "naviguer" au sein de ce dernier par l'utilisation de sa voix à la place des touches (dîtes DTMF) de son téléphone.

3.3 Architecture des applications VoiceXML

La figure suivante représente une architecture VoiceXML classique.



a - Un serveur de documents reçoit les requêtes issues d'une application cliente (interpréteur VoiceXML).

b - En réponse, le serveur fournit des documents VoiceXML qui seront traités par l'interpréteur VoiceXML. La plateforme d'implémentation est commandée par le contexte d'interprétation VoiceXML et par l'interpréteur VoiceXML (c et d) :

c - Par exemple, dans une application de réponse vocale interactive, le contexte d'interprétation VoiceXML peut être responsable de la détection d'un appel entrant, de l'acquisition du document VoiceXML initial et de la réponse à cet appel.

d - L'interpréteur VoiceXML conduit le dialogue après l'appel. La plateforme d'implémentation provoque des événements en réponse aux actions de l'utilisateur (par exemple, aller chercher un nouveau document VoiceXML dès réception d'une entrée vocale ou d'une saisie de caractères) et aux événements du système (par exemple, l'expiration d'un temporisateur).

3.4 JVoiceXML

JVoiceXML est une implémentation open-source de VoiceXML écrite en langage Java. Elle permet de créer simplement des documents VoiceXML qui pourront également être traités par l'interpréteur fourni. Ce dernier utilisant des APIs Java standards tels que JSAPI ou JTAPI.

Le langage VoiceXML s'appuyant sur la synthèse (TTS) et sur la reconnaissance vocale (ASR), le logiciel JVoiceXML utilise ainsi par défaut FreeTTS pour la première et Sphinx 4 pour la seconde. Ces deux outils étant également écrits en Java et entièrement libres.

Néanmoins, JVoiceXML n'est pas une implémentation totalement opérationnelle, puisqu'elle n'en est qu'à sa version 0.5. Ainsi, de nombreuses balises ne sont pas supportées et quelques bugs subsistent.

Reconnaissance vocale

Comme défini dans le cahier des charges, une partie du travail a consisté en l'étude du fonctionnement des systèmes de reconnaissance vocale, pour s'attacher ensuite au développement d'un modèle acoustique permettant de reconnaître des mots français.

C'est pourquoi, nous nous proposons dans un premier temps de présenter les Modèles de Markov Cachés, notion mathématique qui nous permettra d'aborder le schéma du fonctionnement des systèmes de reconnaissance automatique de la parole (RAP). Et dans un second temps, nous traiterons du travail effectué dans ce domaine, au cours du stage.

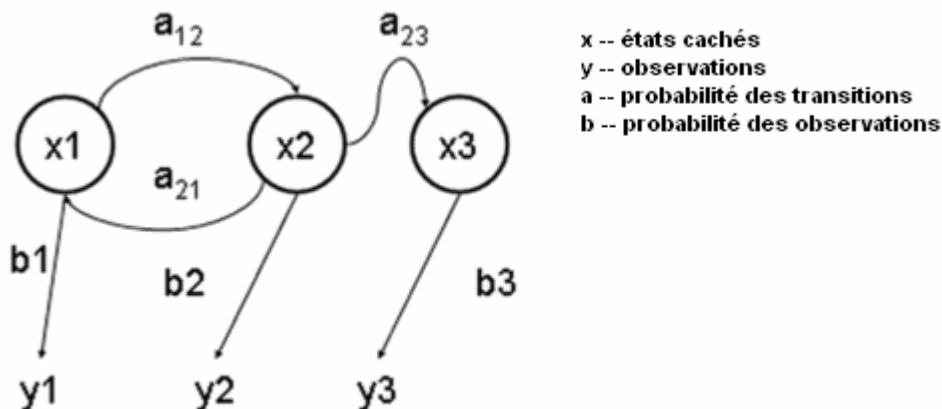
4.1 Les Modèles de Markov Cachés

Un processus de Markov est un système à temps discret qui se trouve à chaque instant dans un état pris parmi N états distincts.

Les transitions entre les états se produisent entre deux instants discrets consécutifs, selon une certaine loi de probabilité. La probabilité de chaque état ne dépend que de l'état qui le précède immédiatement.

Un modèle de Markov caché (MMC ou en anglais HMM pour Hidden Markov Model) représente de la même façon qu'une chaîne de Markov, un ensemble de séquence d'observations dont l'état de chaque observation n'est pas observé, mais associé à une fonction de densité de probabilité.

Il s'agit donc d'un processus stochastique, dans lequel les observations sont une fonction aléatoire de l'état et dont l'état change à chaque instant en fonction des probabilités de transition issues de l'état antérieur.



Plus formellement, un automate à état caché de Markov est caractérisé par un quadruplet des ensembles décrits ci-dessous

- S_i l'état i
- π_i la probabilité que S_i soit l'état initial
- a_{ij} la probabilité de la transition de $S_i \rightarrow S_j$
- $b_i(k)$ la probabilité d'émettre le symbole k étant dans l'état S_i

A condition que

- la somme des probabilités des états initiaux est égale à 1

$$\sum_i \pi_i = 1$$

- la somme des probabilités des transitions partant d'un état est égale à 1

$$\forall i, \sum_j a_{ij} = 1$$

- la somme des probabilités des sorties partant d'un état est égale à 1

$$\forall i, \sum_k b_i(k) = 1$$

On peut donc décrire un modèle de Markov caché comme le jeu de paramètre

$$\lambda = (\mathbf{\Pi}, \mathbf{A}, \mathbf{B})$$

Avec

- $\mathbf{\Pi}$ l'ensemble des probabilités initiales
- \mathbf{A} l'ensemble des probabilités de transition entre les états
- \mathbf{B} l'ensemble des lois (ou densités) de probabilités associées à un état

4.2 Théorie de la reconnaissance vocale

Que l'on veuille utiliser la reconnaissance vocale pour composer un numéro de téléphone, naviguer parmi les fenêtres sur notre PC, entrer des données dans un logiciel ou dicter une lettre dans un traitement de texte, le problème de base reste le même : identifier le sens d'un flux de paroles prononcées souvent dans un bruit de fond plus ou moins important.

Cette tâche est rendue difficile non seulement par les déformations induites par l'usage d'un micro mais aussi par une série de facteurs inhérents au langage humain :

- les homonymies où une même séquence de sons peut correspondre à plusieurs mots (comme le son "s-en" dans cent, sans, san [Francisco], cents, sang, [je] sens)
- les accents locaux
- les habitudes du langage (comme certaines élisions qui rendent difficile la séparation des mots : « j'veis l'chercher » pour « je vais le chercher »)
- les différences de vitesse entre les locuteurs
- les imperfections d'un microphone...

Pour l'oreille humaine, ces facteurs ne représentent généralement pas de difficultés. Le cerveau jongle avec ces déformations de la parole en prenant en compte, quasi inconsciemment, des éléments non verbaux et contextuels qui nous permettent de lever les ambiguïtés.

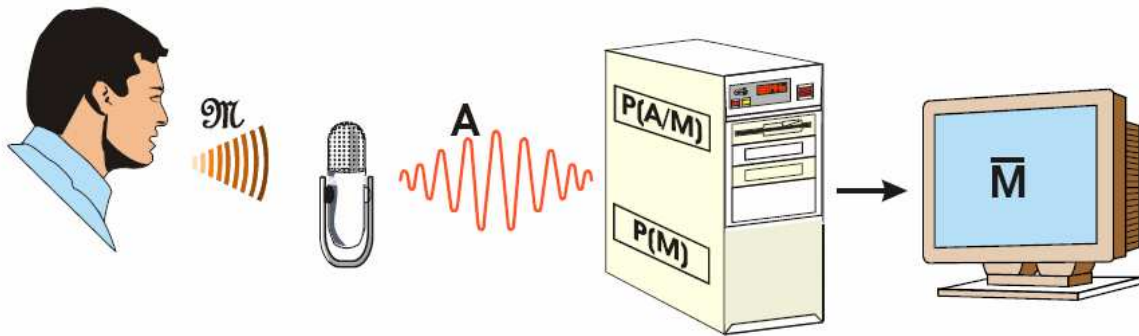
Ce n'est qu'en prenant en compte ces éléments environnants au son proprement dit que les logiciels de reconnaissance vocale pourront atteindre des degrés élevés de fiabilité.

Aujourd'hui, les logiciels qui donnent les meilleurs résultats sont tous basés sur une approche probabiliste.

Le but de la reconnaissance vocale est de reconstituer une séquence de mots \mathcal{M} à partir d'un signal acoustique enregistré \mathbf{A} .

Dans l'approche statistique, on va considérer toutes les suites de mots \mathbf{M} qui pourraient correspondre au signal \mathbf{A} . Dans cet ensemble de suites possibles, on choisira alors celle qui est la plus probable, c'est-à-dire celle qui maximise la probabilité $P(\mathbf{M}/\mathbf{A})$ que \mathbf{M} soit l'interprétation correcte de \mathbf{A} , ce que l'on note

$$\bar{\mathbf{M}} = \arg \max_M P(\mathbf{M}/\mathbf{A})$$



Notons que $P(A/B)$ représente la probabilité de l'événement **A** si l'événement **B** a eu lieu. L'axiome de Bayes permet de calculer la probabilité du concours de deux événements **A** et **B** par les égalités suivantes

$$P(A \text{ et } B) = P(A/B) P(B) = P(B/A) P(A)$$

Où $P(A)$ représente la probabilité que l'événement **A** ait lieu.

Ainsi, l'axiome de Bayes permet de réécrire l'expression :

$$\bar{M} = \arg \max_M \frac{P(A/M) P(M)}{P(A)}$$

Et comme $P(A)$ est une constante dans la recherche du meilleur **M**, on a finalement l'équation (que nous appellerons *équation 1*)

$$\boxed{\bar{M} = \arg \max_M P(A/M) P(M)}$$

équation 1

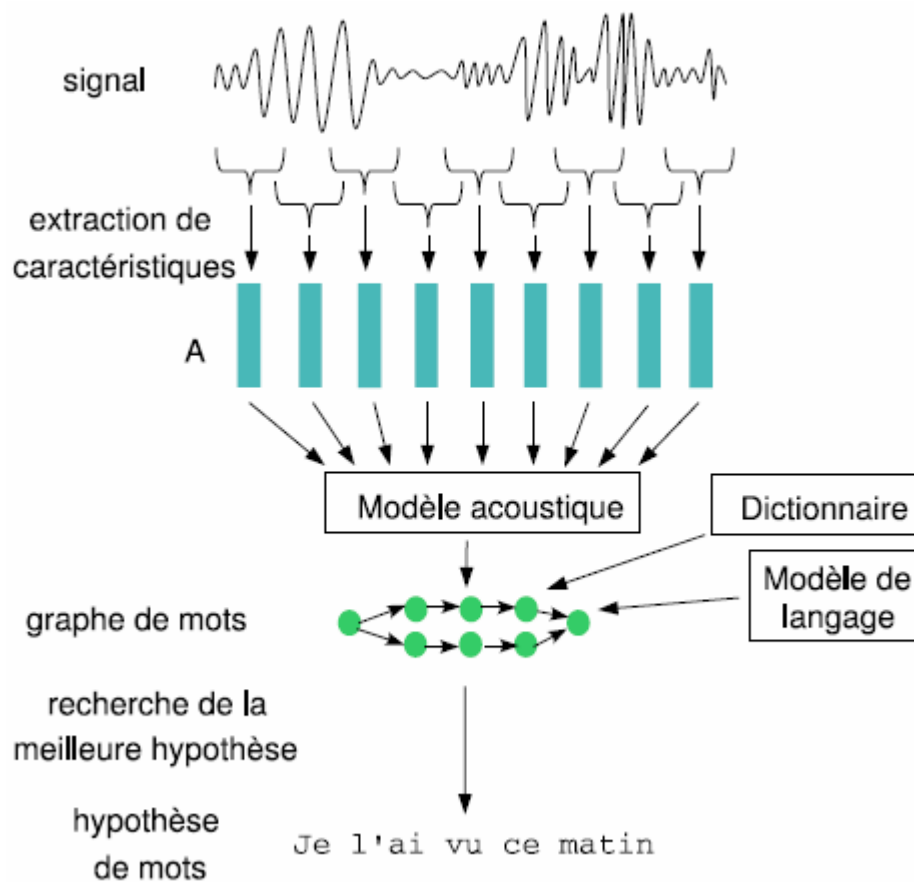
Cette dernière équation est la clé de l'approche probabiliste de la reconnaissance vocale. En effet, le premier terme $P(A/M)$ représente la probabilité d'observer le signal acoustique **A** si la séquence de mots **M** a été prononcée : c'est un problème purement acoustique.

Le second terme $P(M)$ représente la probabilité que c'est la suite de mots **M** qui a été effectivement énoncée : c'est un problème de nature linguistique. L'équation ci-dessus nous enseigne donc que l'on peut scinder le problème de reconnaissance vocale en deux parties indépendantes : on modélisera séparément les aspects acoustiques et les problèmes linguistiques.

Ainsi, la transcription se décompose en plusieurs modules

- l'extraction de caractéristiques produisant **A**
- l'utilisation du modèle acoustique calculant $P(\mathbf{A}/\mathbf{M})$ et cherchant les hypothèses **M** qui sont vraisemblablement associées à **A**
- l'utilisation du modèle de langage calculant $P(\mathbf{M})$ pour choisir une ou plusieurs hypothèses sur **M** en fonction de connaissances sur la langue

Le schéma suivant représente les constituants d'un système de transcription.



4.2.1 Extraction de caractéristiques

Le signal sonore à analyser se présente sous la forme d'une onde dont l'intensité varie au cours du temps. La première étape du processus de transcription consiste à extraire une succession de valeurs numériques suffisamment informatives sur le plan acoustique pour décoder le signal par la suite.

Le signal est susceptible de contenir des zones de silence, de bruit ou de musique. Ces zones sont tout d'abord éliminées afin de n'avoir que des portions du signal utiles à la transcription, c'est-à-dire, celles qui correspondent à de la parole. Le signal sonore est ensuite segmenté en ce que l'on qualifie de groupes de souffle, en utilisant comme délimiteurs des pauses silencieuses suffisamment longues (de l'ordre de 0,3 s). L'intérêt de cette segmentation est d'avoir un signal sonore continu de taille raisonnable par rapport aux capacités de calculs des modèles du système de RAP. Dans la suite du processus de transcription, l'analyse se fera séparément pour chaque groupe de souffle.

Pour repérer les fluctuations du signal sonore, qui varie généralement rapidement au cours du temps, le groupe de souffle est lui-même découpé en fenêtres d'étude de quelques millisecondes (habituellement de 20 ou 30 ms). De manière à ne pas perdre d'informations importantes se trouvant en début ou fin de fenêtres, on fait en sorte que celles-ci se chevauchent, ce qui conduit à extraire des caractéristiques toutes les 10 ms environ.

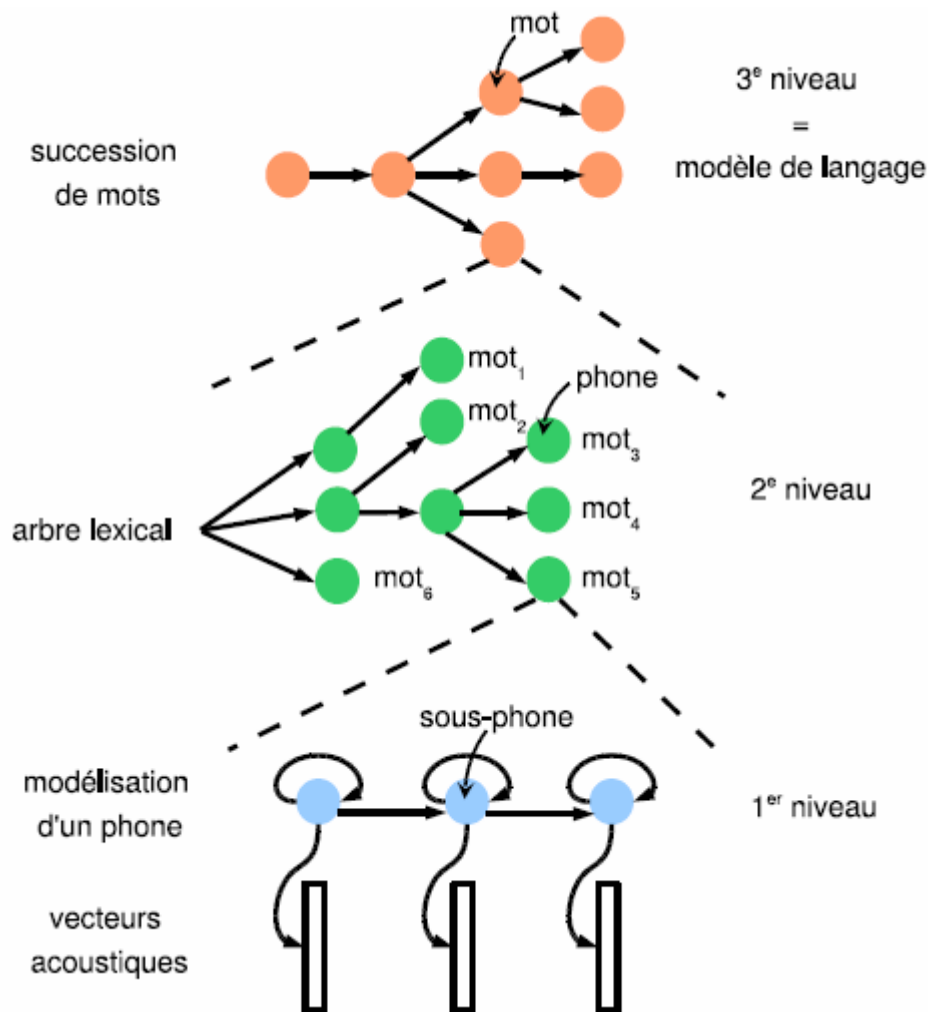
A partir du signal contenu dans chaque fenêtre d'analyse sont calculées des valeurs numériques caractérisant la voix humaine. A l'issue de cette étape, le signal devient alors une succession de vecteurs dits acoustiques, de dimension souvent supérieure ou égale à 39.

4.2.2 Modèle acoustique

L'étape suivante consiste à associer aux vecteurs acoustiques, qui sont, comme nous venons de le voir, des vecteurs numériques, un ensemble d'hypothèses de mots (des symboles). En se référant à l'équation 1 de la modélisation statistique, cela revient à estimer $P(A/M)$. Les techniques qui permettent de calculer cette valeur forment ce qu'on appelle le modèle acoustique.

L'outil le plus utilisé pour la modélisation du modèle acoustique est le Modèle de Markov Cachée présenté précédemment. Les HMMs ont en effet montrés dans la pratique leur efficacité pour reconnaître la parole. Même s'ils présentent quelques limitations pour modéliser certaines caractéristiques du signal, comme la durée ou la dépendance des observations acoustiques successives, les HMMs offrent un cadre mathématique bien défini pour calculer les probabilités $P(A/M)$.

Les modèles acoustiques font intervenir trois niveaux de HMM présentés dans la figure qui suit.



Ils cherchent dans un premier temps à reconnaître les types de son, autrement dit à identifier des phones (qui sont des sons prononcés par des locuteurs et définis par des caractéristiques précises). Pour ce faire, ils modélisent un phone par un HMM, généralement à trois états représentant ses début, milieu et fin. La variable cachée est alors un sous-phone et les observations sont des vecteurs acoustiques.

Pour calculer les probabilités d'observation dans chaque état, deux approches sont souvent envisagées, l'une basée sur la représentation des densités de probabilité par des gaussiennes et l'autre reposant sur des réseaux de neurones. Ces différentes méthodes établissent des hypothèses sur la probabilité des phones prononcés. Or, l'objectif des modèles acoustiques est de déterminer une succession de mots. Les modèles acoustiques utilisent à cette fin un dictionnaire de prononciations, qui effectue la correspondance entre un mot et ses prononciations. Comme un mot est susceptible d'être prononcé de différentes manières, selon son prédécesseur et son successeur, ou tout simplement selon les habitudes du locuteur, il peut y avoir plusieurs entrées dans ce lexique pour un même mot. Les indications sont données au moyen des phonèmes caractéristiques de la prononciation. Sur la figure ci-dessous, les phonèmes sont transcrits dans le système de représentation SAMPA.

adorateurs	a d O R a t 9 R z
adorateurs	a d O R a t 9 R
adoration	a d O R a s j o~
adore	a d O R @
adore	a d O R

Le deuxième niveau de HMMs modélise les mots à partir des HMMs représentant des phones et du lexique de prononciations. Il se présente sous la forme d'un arbre lexical contenant initialement tous les mots du vocabulaire, progressivement élagué au fur et à mesure que sont reconnus des phones. Puisque les HMMs de premier niveau modélisent des phones et non des phonèmes, les phonèmes disponibles dans le dictionnaire de prononciations sont convertis en phones afin de reconnaître des mots. Des règles de transformation dépendant du contexte d'apparition du phonème sont alors utilisées.

Le troisième niveau modélise enfin la succession des mots M au sein d'un groupe de souffle et peut alors incorporer les connaissances apportées par le modèle de langage sur M . Pour établir ce HMM équivalent à un graphe de mots, le HMM correspondant à l'arbre lexical est dupliqué à chaque fois que le modèle acoustique effectue l'hypothèse qu'un nouveau mot a été reconnu.

Le fonctionnement du modèle acoustique qui vient d'être décrit se heurte à un problème majeur : l'espace de recherche du HMM de plus haut niveau devient fréquemment considérable, surtout si le vocabulaire est important et si le groupe de souffle à analyser contient plusieurs mots. Des algorithmes issus de la programmation dynamique permettent de calculer efficacement les probabilités. Il s'agit principalement de l'algorithme de Viterbi et le décodage par pile, appelé aussi décodage A*. De plus, il est fait recours très régulièrement à l'élagage pour ne conserver que les hypothèses susceptibles d'être les plus intéressantes.

Le rôle du modèle acoustique consiste ainsi à aligner le signal sonore avec des hypothèses de mots en utilisant uniquement des indices d'ordre acoustique. Il inclut dans son dernier niveau de modélisation les informations sur les mots apportées par le modèle de langage.

4.2.3 Modèle de langage

Le modèle de langage a pour objectif de trouver les séquences de mots les plus probables, autrement dit celles qui maximisent la valeur $\mathbf{P}(\mathbf{M})$ de l'équation 1. Si l'on se réfère au HMM de plus haut niveau du modèle acoustique (voir figure précédente), les valeurs $\mathbf{P}(\mathbf{M})$ correspondent aux probabilités de succession de mots.

Fonctionnement d'un modèle de langage

En posant $\mathbf{M} = \mathbf{m}_1^n = \mathbf{m}_1 \dots \mathbf{m}_n$, où \mathbf{m}_i est le mot de rang i de la séquence \mathbf{M} , la probabilité $\mathbf{P}(\mathbf{M})$ se décompose comme suit :

$$P(\mathbf{m}_1^n) = P(\mathbf{m}_1) \prod_{i=2}^n P(\mathbf{m}_i | \mathbf{m}_1 \dots \mathbf{m}_{i-1})$$

L'évaluation de $\mathbf{P}(\mathbf{M})$ se ramène alors au calcul des valeurs $\mathbf{P}(\mathbf{m}_i)$ et $\mathbf{P}(\mathbf{m}_i | \mathbf{m}_1^{i-1})$ qui s'obtiennent respectivement à l'aide des égalités

$$P(\mathbf{m}_i) = \frac{C(\mathbf{m}_i)}{\sum_{\mathbf{m} \in \mathbf{V}} C(\mathbf{m})} \quad P(\mathbf{m}_i | \mathbf{m}_1^{i-1}) = \frac{C(\mathbf{m}_1^i)}{\sum_{\mathbf{m}_i} C(\mathbf{m}_1^i)}$$

Où \mathbf{V} est le vocabulaire utilisé par le système de RAP, et $C(\mathbf{m}_i)$ et $C(\mathbf{m}_1^i)$ représentent les nombres d'occurrences respectifs du mot \mathbf{m}_i et de la séquence de mots \mathbf{m}_1^i dans le corpus d'apprentissage.

Malheureusement, pour prédire la suite de mots \mathbf{m}_1^i , le nombre de paramètres $\mathbf{P}(\mathbf{m}_i)$ et $\mathbf{P}(\mathbf{m}_i | \mathbf{m}_1^{i-1})$ du modèle de langage à estimer, augmente de manière exponentielle avec n . Dans le but de réduire ce nombre, $\mathbf{P}(\mathbf{m}_i | \mathbf{m}_1^{i-1})$ est modélisé par un modèle N -gramme, c'est-à-dire, une chaîne de Markov d'ordre $N-1$ (avec $N > 1$), à l'aide de l'équation :

$$P(\mathbf{m}_i | \mathbf{m}_1^{i-1}) \approx P(\mathbf{m}_i | \mathbf{m}_{i-N+1}^{i-1})$$

Cette équation indique que chaque mot \mathbf{m}_i peut être prédit à partir des $N-1$ mots précédents.

Pour $N = 2, 3$ ou 4 , on parle respectivement de modèle *bigramme*, *trigramme* ou *quadrigramme*. Pour $N = 1$, le modèle est dit *unigramme* et revient à estimer $\mathbf{P}(\mathbf{m}_i)$.

Généralement, ce sont les modèles bigrammes, trigrammes et quadrigrammes qui sont utilisés dans les modèles de langage des systèmes de RAP.

4.3 Etude des logiciels existants

Pour rentrer dans le cadre pratique de la reconnaissance de la parole, l'étude de deux logiciels open-sources a été effectuée. En premier, je me suis intéressé à HTK pour ensuite passer à l'utilisation d'un des logiciels les plus connus dans le monde de la reconnaissance vocale, à savoir Sphinx (plus particulièrement la version 4).

4.3.1 HTK

Hidden Markov Model Toolkit (HTK) est un ensemble d'outils portable permettant la création et la manipulation de modèles de Markov cachés. HTK est principalement utilisé dans le domaine de la recherche de la reconnaissance vocale bien qu'il soit tout à fait utilisable dans de nombreuses autres applications telles que la synthèse vocale, la reconnaissance de l'écriture ou la reconnaissance de séquences d'ADN.

Il est composé d'un ensemble de modules et outils écrits en langage C. Ces différents outils facilitent l'analyse vocale, l'apprentissage des HMMs, la réalisation de tests et l'analyse des résultats. Il est à noter, que ce qui a contribué au succès de HTK, est qu'il est accompagné d'une assez bonne documentation. Cette dernière manquant cruellement à d'autres systèmes de RAP tel que Sphinx4 (voir section suivante).

C'est ainsi que dans un premier temps j'ai créé de petits modèles acoustiques à base de mots. Cela m'a permis de me familiariser avec les divers outils et vocabulaire utilisé dans le RAP.

Malgré tout, j'ai abandonné HTK au profit de Sphinx-4 et cela pour plusieurs raisons

- De nombreuses applications utilisent Sphinx, en particulier JVoiceXML
- Sphinx bénéficie d'une plus grande communauté et par conséquent il est plus facile d'obtenir de l'aide
- Les droits de HTK sont détenus par Microsoft qui propose une licence un peu plus restrictive que Sphinx
- Une étude indienne a cherché à comparer HTK et Sphinx. Il s'est avéré que ces deux systèmes sont approximativement équivalents en terme d'erreur de reconnaissance, mais Sphinx a l'avantage d'être légèrement plus rapide

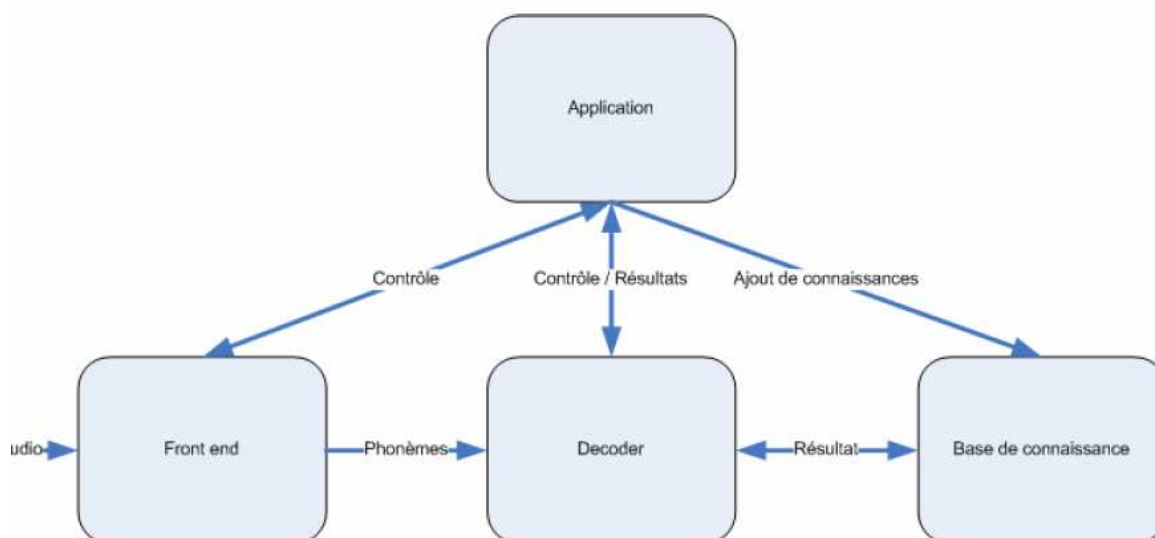
4.3.2 Sphinx 4

Sphinx 4 est un logiciel de reconnaissance vocale écrite entièrement en Java. Les buts de Sphinx sont d'avoir une reconnaissance vocale hautement flexible, d'égaliser les autres produits commerciaux et de mettre en collaboration les centres de recherche de diverses universités, des laboratoires de Sun et de HP mais aussi du MIT.

Tout en étant hautement configurable, la reconnaissance de Sphinx 4 supporte notamment les mots isolés et les phrases (utilisation de grammaires). Son architecture 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. Ces dernières étant les informations relatives à une voix propre. Ce sont par exemple les différents phonèmes, les différents mots (lexique), les différentes façons de prononciation. Plus ces informations seront importantes et connues par le système, meilleure sera sa réaction et ses choix à faire.

Comme le montre la figure suivante qui représente son architecture, Sphinx 4 s'articule autour de trois modules.



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 prononcés. La base de connaissance 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 quelques 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 langage : 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 traite les informations reçues depuis le Front-End, les analyse et les compare avec la base de connaissances pour donner un résultat à l'application.

4.4 Création d'un modèle acoustique français

Parmi les objectifs définis dans le cahier des charges, figure l'étude du fonctionnement d'un système de reconnaissance automatique de la parole. Ce savoir permettant par la suite d'utiliser ou de créer un modèle acoustique ouvrant la voie à la reconnaissance de la langue française.

4.4.1 Utilisation du modèle créé par le LIUM

Sur le site du groupe CMU Sphinx, quelques modèles sont fournis gratuitement (chacun étant "spécialisé" dans un contexte particulier) mais seulement pour la langue anglaise (qui plus est, avec un accent américain).

La création d'un nouveau modèle acoustique réclamant beaucoup de temps et de ressources, je me suis d'abord tourné vers la recherche d'un modèle français. A cet effet, se trouve sur l'Internet un modèle acoustique fourni par le Laboratoire d'Informatique de l'Université du Maine (LIUM). Ce modèle a été créé suite à la participation à la campagne d'évaluation ESTER dont le but est de retranscrire automatiquement la radiophonie française.

Le modèle acoustique ayant été créé pour la version 3 de Sphinx, j'ai été amené à le convertir pour qu'il puisse fonctionner avec Sphinx4. Malheureusement, après plusieurs essais, la reconnaissance était très mauvaise, voire quasi-nulle. J'ai donc échangé plusieurs courriels avec l'équipe du LIUM qui m'a garanti la compatibilité du modèle avec Sphinx 4, bien qu'elle ne l'ait jamais utilisé avec cette version-là.

Finalement j'ai préféré abandonner ce modèle acoustique pour en créer un nouveau. Notons cependant, qu'il semblerait que l'équipe du LIUM ait par la suite rencontré le même problème et se soit résolu à mettre un avertissement sur leur site web.

4.4.2 Création d'un nouveau modèle

La création d'un nouveau modèle acoustique se fait en deux grandes étapes. La première consistant en la collecte des données. La seconde effectuant l'apprentissage du modèle à partir de ces mêmes données.

Pour cela j'ai utilisé le logiciel SphinxTrain, développé (tout comme Sphinx4) par le CMU (Carnegie Mellon University).

4.4.2.1 Collecte des données

Une des tâches les plus longues et les plus fastidieuses concerne la collecte des données. En effet, nous devons fournir à SphinxTrain tout un ensemble de données sonores (sous un format spécifique) et de données textuelles. Ces dernières étant les transcriptions des premières.

Ainsi, il est recommandé d'avoir en sa possession un corpus audio avec transcription d'environ 10 - 15 heures au minimum pour avoir une reconnaissance d'un vocabulaire moyen. Il existe quelques corpus disponibles sur Internet mais qui sont assez coûteux (certains étant vendus quelques milliers d'euros). C'est pourquoi, il est dans certains cas préférable de réaliser sa propre base de données sonore.

Deux contraintes s'ajoutent alors

- Pour que la reconnaissance soit indépendante du locuteur, il faut faire des enregistrements avec plusieurs personnes (une centaine est raisonnable)
- La qualité du modèle acoustique est fonction des données sonores fournies pour l'apprentissage. Une durée de 10 secondes maximum améliore sensiblement la qualité de la reconnaissance (certains recommandent 3-4 secondes par enregistrement)

N'ayant pas le temps et les ressources nécessaires pour réaliser des enregistrements de plusieurs heures avec différentes personnes, j'ai essayé de créer automatiquement un corpus français. En effet, il est de plus en plus fréquent de trouver sur Internet des bénévoles qui effectuent des enregistrements de lectures de livres pour personnes déficientes visuellement.

L'avantage de cette méthode est que l'on est en présence de plusieurs voix, et d'une grande quantité d'enregistrements avec leurs transcriptions.

Néanmoins, pour répondre à la deuxième contrainte il faudrait faire un prétraitement sur chaque fichier audio détectant les moments de silence (en général à la fin d'une phrase) pour extraire des morceaux de lecture. Le problème sous-jacent, est de savoir comment associer la transcription de ces morceaux automatiquement, puisqu'il n'existe aucune donnée temporelle...

Je me suis donc tourné vers une autre solution qui consiste à n'exploiter que les livres où interagissent plusieurs personnes au travers de dialogues (pièces de théâtres, débats, etc.). Il suffirait donc de détecter qui parle, extraire ce qu'elle dit et faire l'alignement avec la transcription.

Je pensais ainsi, qu'à travers une transformée de Fourier rapide (FFT pour Fast Fourier Transform) je pourrais savoir qui parle pour ensuite faire le traitement cité précédemment. Cependant après mûre réflexion les deux problèmes suivants se sont posés

- avec une FFT il faudrait trouver la fréquence fondamentale et le timbre d'une voix sachant qu'elles sont difficilement distinguables
- comment attribuer à une seule personne les couples fréquence/timbre, sachant que les derniers sont différents selon ce qu'on dit et les premiers varient tout le temps ?

De plus, quelques recherches m'ont permis de m'apercevoir qu'il s'agissait d'un problème complexe, celui de l'empreinte vocale (domaine de la biométrie). En plus subsisteraient des confusions lorsque par exemple deux personnes parlent simultanément.

Enfin, j'ai également essayé d'exploiter les sous-titres présents dans les DVD vidéo. Bien que l'extraction des sous-titres est tout à fait faisable à partir de logiciels basés sur la reconnaissance d'écriture (les sous-titres étant en fait de simples images JPEG) le problème est qu'ils sont fait indépendamment de l'enregistrement audio des films. Par conséquent, ils diffèrent sensiblement de ce qui est dit. Pour remédier à cela il aurait fallut à la fin de la constitution du corpus, réécouter chaque extrait sonore et corriger manuellement la transcription...

A l'heure actuelle, la constitution d'une base de données sonore avec transcription est difficilement réalisable automatiquement. La meilleure solution est alors d'en créer une à la main.

A l'instar du modèle acoustique TIDIGITS (fournit par Texas Instruments) j'ai créé un corpus qui permet de reconnaître des chiffres français de zéro à neuf. Ce corpus est mono locuteur (puisque, enregistré que par moi-même) et constitué de 100 enregistrements de courtes phrases comme « *deux huit neuf six un cinq huit trois* » ou « *un un un deux huit un* ».

Pour cela, j'ai créé un petit outils générant aléatoirement une suite (de taille variable) de nombres en essayant de garder une distribution égale des phonèmes (meilleure est la distribution, meilleure sera la reconnaissance).

Puisque destiné à la reconnaissance des dix premiers chiffres avec en entrée ma voix, il n'est pas nécessaire de disposer d'un grand corpus audio.

4.4.2.2 Apprentissage du modèle acoustique

Une fois que l'on possède un corpus, on peut passer à l'étape de la création du modèle acoustique. Pour cela, il faut savoir quelle base sonore utiliser (phonèmes, syllabes, mots). J'ai opté pour l'utilisation des phonèmes. Cela a permis par la suite de pouvoir rajouter de nouveaux mots dans le dictionnaire (en spécifiant quels phonèmes interviennent pour les mots donnés) sans même avoir à recréer un nouveau modèle.

La création à proprement parler de ce modèle est définie en page annexe de ce rapport. Il est à noter, que même si Sphinx est pourvu d'une communauté relativement importante, les étapes de la création d'un modèle acoustique sont difficilement trouvable (les informations sont assez dispersés sur l'Internet et on trouve plusieurs bribes dans les différents forums du CMU). C'est la raison pour laquelle, j'ai pris l'initiative d'écrire un tutorial expliquant pas à pas comment, à partir de SphinxTrain, créer un modèle acoustique pour les différentes versions de Sphinx.

Par ailleurs, une certaine expertise est nécessaire avant toute création, puisqu'il faut non seulement définir plusieurs paramètres qui influenceront la qualité de la reconnaissance, mais aussi apporter certaines modifications dans le code des scripts de SphinxTrain corrigeant quelques bugs.

L'apprentissage des modèles acoustiques passe par plusieurs étapes préalables. La durée de traitement étant assez variable. Pour avoir une idée, elle varie de quelques minutes, dans le cas d'un petit corpus, à plusieurs semaines de traitements si l'on souhaite avoir un modèle grand vocabulaire, indépendant du locuteur et à base de phonèmes dépendant du contexte. Ce type de phonème permettant une plus grande flexibilité puisqu'elle permet par exemple de prévenir le phénomène de coarticulation.

Sur le plan du traitement, le plus gros du travail est effectué au niveau de l'estimation des paramètres des différents HMMs où l'algorithme de Baum-Welch est utilisé (cf. pages annexes pour l'explication de cet algorithme).

Synthèse vocale

La synthèse vocale donne à l'ordinateur, la possibilité de convertir du texte arbitraire en un discours audible. Un des buts du TTS est d'être en mesure de fournir de l'information textuel à des personnes, via des messages vocaux. Un TTS fournit une sortie vocale à tout type d'information qui peut être stocké dans des bases de données et des services d'information.

5.1 Théorie de la synthèse vocale

Nous présentons ici les concepts de bases permettant de comprendre le fonctionnement d'un synthétiseur vocal.

5.1.1 Types de synthèses vocales

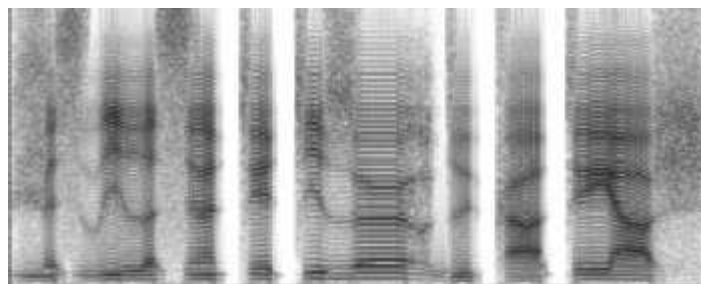
La modélisation du conduit vocal

Cette méthode représente la plus ancienne façon de générer une voix synthétique. Elle était générée par des automates mécaniques en utilisant un ensemble de tubes et de membranes simulant le conduit vocal.

La mise en oeuvre informatique de ce procédé n'a jamais donné aucun résultat probant en raison de son extrême complexité. C'est pour cette raison que cette technologie a été actuellement abandonnée

La synthèse par règles ou par formants (LPC)

Cette technique a été très utilisée entre 1965 et 1985 du fait qu'elle était très peu gourmande en ressource (10 Ko pour les règles décrivant la coarticulation d'une voix). Ce genre de synthétiseurs se base sur le fait que s'il est possible à partir d'une voix d'obtenir un spectrogramme, il est alors possible en toute logique de générer une voix artificielle à partir d'un spectrogramme. Une fois le spectrogramme dessiné, il ne reste plus alors qu'à générer le signal correspondant.



Spectrogramme

Les acousticiens se sont en effet aperçus que les résonances du conduit vocal mettaient en avant certaines plages de fréquence spécifiques au phonème prononcé. Les acousticiens ont nommé ces plages de fréquences « formants ».

Un formant se caractérise par sa fréquence (hauteur) et son énergie (force).

On s'est alors rapidement aperçus que trois à six formants étaient suffisants pour obtenir un phonème de bonne qualité acoustique.

Pour obtenir un synthétiseur vocal, il faut déterminer et stocker les différentes enveloppes spectrales (dont les harmoniques principales sont les formants) des sons de bases (phonèmes) de la voix, ainsi que leur mode d'excitation (suite d'impulsions ou bruit blanc), puis à les recombinaison à volonté pour recréer les mots désirés.

L'un des grands avantages de ce procédé est que très peu de données sont nécessaires pour générer un phonème (la description des formants étant en théorie suffisante) et qu'il est beaucoup plus simple d'apporter quelques modifications à ces données pour obtenir différentes voix.

L'un des inconvénients se retrouve dans le résultat. En effet, le résultat obtenu est généralement moins réaliste que dans le cas d'une voix composée par la mise bout à bout des éléments de la voix

La concaténation mot à mot

Cette synthèse ne sera pas abordée en détail dans cette présentation, mais il suffit de citer l'exemple des annonces dans les gares pour comprendre à quoi elle ressemble. En effet, elle a l'avantage d'être facilement intelligible et très naturel.

Néanmoins, elle donne des fichiers très lourds. Ce handicap empêche son utilisation sur les réseaux tels que l'Internet. De plus, l'interaction qu'elle propose au locuteur est très limitée. Elle ne peut s'exprimer qu'avec des mots qui sont déjà enregistrés et ne peut donc pas lire un texte dont elle ne connaît pas tous les mots. Le changement de cette voix nécessite l'enregistrement complet de tous les mots utilisés pour le fonctionnement du service.

La concaténation de diphtonges (MBrola)

La mise bout à bout des éléments de la voix est une méthode plus moderne qui consiste à mémoriser les phonèmes ou les assemblages de phonèmes prononcés afin de les mettre bout à bout en vue de restituer la voix de la personne. Des algorithmes complexes déforment les phonèmes enregistrés pour leur faire suivre la prosodie de la voix parlée, et donnent d'excellents résultats. Cette technique est utilisée pour les sites d'information comme la météo, l'horoscope ou les résultats sportifs. Ces algorithmes sont cependant mal adaptés aux larges plages de fréquences utilisées dans la voix chantée. De plus, la parole est souvent hyper-articulée, et l'intonation reste peu naturelle. Un système de synthèse par diphtonges de bonne qualité nécessite entre 1 et 5 Mo par voix (pour stocker les quelques 1500 diphtonges correspondants, soit environ 3 minutes de parole).

L'un des inconvénients majeurs se retrouve lors de la définition d'une autre voix, il est nécessaire d'enregistrer une autre personne. L'autre défaut de ce système est que la totalité des phonèmes d'une langue doivent être prononcés. Pour fabriquer un logiciel multilingues avec le même timbre de voix, il est donc nécessaire d'enregistrer une personne parfaitement polyglotte afin d'échantillonner l'ensemble des phonèmes prononçables dans chacune des langues.

La sélection d'unités dans une grande base de données

Cette méthode représente une véritable percée en matière de synthèse vocale. Plutôt que de garder un seul exemplaire de chaque diphone de la langue, on puise dans plusieurs heures de parole, préalablement segmentées phonétiquement. Au moment de choisir les segments à mettre en oeuvre, plusieurs instances d'une même unité phonétique sont alors disponibles, avec des prosodies différentes. Il faut alors choisir les segments dont le contexte est le plus proche de la chaîne phonétique à synthétiser, dont la prosodie est la plus proche de la prosodie à reproduire et dont les discontinuités spectrales sont les plus rapprochées l'une par rapport à l'autre. Cette technique a permis récemment de produire de la parole dont l'intelligibilité et le naturel rendent possible la confusion avec une prononciation humaine. Néanmoins, elle implique un accès très rapide à plusieurs gigaoctets de données.

Cette technologie est encore à l'état expérimental et n'est donc pas encore disponible sur le marché. De plus, cette méthode nécessitant une base de données contenant plusieurs gigaoctets d'information et un accès suffisamment rapide a peu de chance d'aboutir commercialement dans les prochaines années.

5.1.2 Fonctionnement d'un synthétiseur vocal

La qualité de la synthèse vocale est caractérisée par deux facteurs : l'intelligibilité du discours traité, et le naturel de l'ensemble des messages parlés. Sans une grande intelligibilité, un système TTS serait inutile, et sans un naturel, la voix serait ennuyeuse, voire même, pénible à écouter.

Prétraitement

Cette phase consiste dans la transformation d'un texte en une suite de phrases, organisées en mots. Ce prétraitement a pour objectif de retranscrire en toutes lettres les chaînes non orthographiques représentant la reconnaissance des unités de mesure du système international (SI), les symboles non alphanumériques (comme les antislash), les chiffres, les lettres et les motifs (extensions de fichiers, etc.). Il ne faut pas oublier les abréviations, les sigles comme « SVP ». Là encore, il faut apprendre au système à reconnaître les abréviations les plus courantes.

Analyse syntaxique

L'analyse syntaxique découpe le texte en groupes de mots ou tronçons et établit leurs relations de dépendance. Elle permet une meilleure interprétation des mots pour la suite des opérations en découpant chaque mot en lexèmes (unité de base du lexique) et en déterminant leurs appartenances grammaticales.

Calcul prosodique

Le traitement prosodique sert à modéliser l'évolution temporelle de la fréquence fondamentale (vibration des cordes vocales, prédire la durée des sons élémentaires et la durée des pauses). Si la prosodie d'un locuteur réel est recopiée sur la voix de synthèse, le résultat obtenu est sensiblement meilleur. En effet, l'impression de naturel et son intelligibilité s'améliorent.

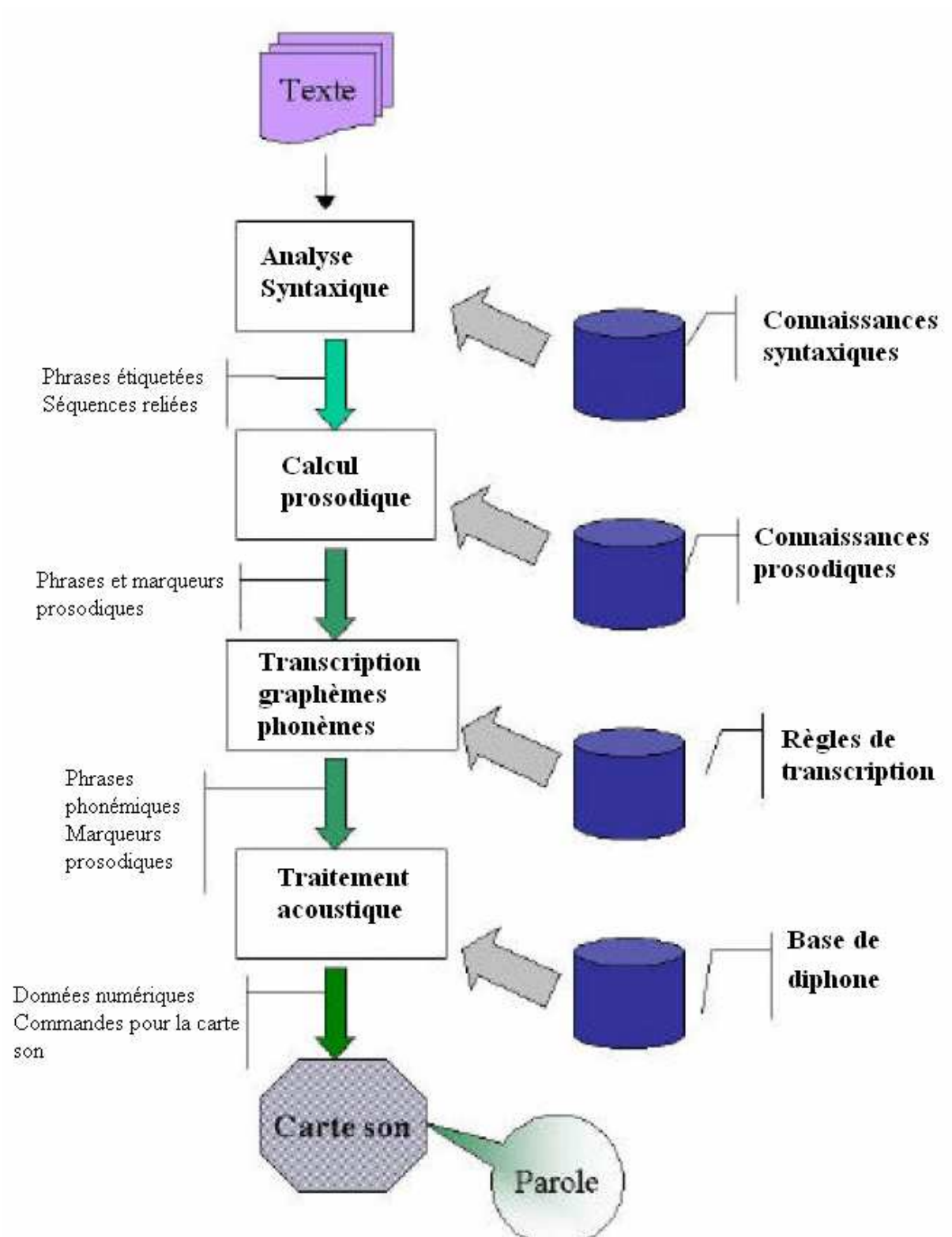
Transcription graphème-phonème

Le but de la transcription graphème-phonème est de passer du texte orthographique (plus ou moins traité par le module précédent) à une suite de symboles phonétiques. Plusieurs niveaux de connaissances sont pris en compte : phonétiques, phonologiques, lexicales, syntaxiques et même sémantiques. La prononciation du français comporte ainsi plus de 1000 règles élémentaires, et plusieurs milliers de règles portant sur les noms propres et mots d'emprunt les plus courants.

Chaque langue est différente et emploie un certain nombre de règles spécifiques.

Traitement acoustique

Le dernier traitement effectué par le synthétiseur est la conversion du texte phonétique en signal de parole. La voix synthétique s'obtient par l'extraction des diphtonges à partir de la voix d'un locuteur, les fragments de signal (entre 1000 et 2000) constituant la bibliothèque seront ensuite concaténés par le synthétiseur pour former le signal de parole. Les paramètres acoustiques les plus utilisés pour représenter ces unités sont le codage par prédiction linéaire.



5.2 Utilisation de MBrola et intégration à FreeTTS

La synthèse vocale en langue française a fait intervenir divers outils disponibles gratuitement. La plateforme JVoiceXML étant en cours de développement, il est actuellement impossible d'utiliser un autre logiciel de synthèse vocale que FreeTTS. Ce dernier, écrit entièrement en Java, est open-source et se base sur FLite (moteur de synthèse vocale développée dans la même université que Sphinx 4 présentée plus haut). Il a l'avantage d'avoir un support partiel de JSAPI. Malheureusement, seules des voix anglaises y sont implémentées et il semblerait que plusieurs tentatives d'implémentations du français aient été un échec.

C'est ainsi, que la solution utilisée consiste à rajouter à FreeTTS une surcouche qui va tout simplement désactiver la sortie audio de ce dernier, pour transférer la phrase à prononcer à un autre moteur de synthèse vocale supportant le français.

Pour cela, deux outils intimement liés sont intervenus. Le premier est MBrola, qui est un synthétiseur vocal assez connu et développé par la Faculté Polytechnique de Mons (Belgique). Comme cela a été dit dans la section 5.1.1, MBrola est basé sur la concaténation des diphtonges venant d'une base de donnée. Il prend une liste de phonèmes comme entrée, ainsi que l'information prosodique (durée des phonèmes et description d'effet, ponctuation, tonalité, vitesse ...), et produit des échantillons de la parole sur 16 bits (linéaires). Il existe différentes tessitures de voix (7 pour le français), et des bases pour plusieurs langage (une trentaine).

Il est tout à fait possible de coupler directement FreeTTS et MBrola, mais cela ne se limite une fois de plus qu'à la langue anglaise...

De fait, il fallait utiliser un logiciel capable d'analyser un texte et de fournir à MBrola une liste de phonèmes. Plusieurs logiciels gratuits existent tels que LLiaPhone, CiceroTTS, ou Vocalyse.

Mon choix s'est porté sur ce dernier parce qu'il est écrit en langage Java et qu'il est possible d'avoir le code source sur simple demande (auprès de Polytech'Nice).

L'intégration produisant des résultats assez satisfaisant, Vocalyse souffrait malgré tout d'une mauvaise phase de prétraitement cité plus haut dans ce rapport (cf. section 5.1.2 expliquant le fonctionnement d'un TTS). J'ai par conséquent ajouté une couche gérant la transformation de différents sigles, ponctuations ou chiffres qui empêchaient le bon fonctionnement.

En outre, l'intégration à FreeTTS, en dehors de pouvoir fonctionner avec JVoiceXML, permet de bénéficier du support de JSAPI.

Serveur téléphonique (Asterisk)

Pour faire communiquer une plateforme JVoiceXML il était nécessaire d'utiliser un outil permettant de gérer les appels téléphoniques. Aujourd'hui il existe de nombreux logiciels visant à remplacer les serveurs téléphoniques, le plus connu et le plus utilisé actuellement étant Asterisk.

6.1 Présentation d'Asterisk

Un PABX (Private Automatic Branch eXchange) est un autocommutateur téléphonique privé. En d'autres termes, il représente l'élément central qui

- Distribue les appels téléphoniques arrivés
- Autorise les appels téléphoniques départs
- Gère les terminaux téléphoniques
- Gère toutes les autres fonctionnalités ou options

Un autocommutateur privé possède sa propre intelligence pour faciliter la commutation des appels voix.

Il existe deux sortes de PABX

- les PABX traditionnels (qui peuvent éventuellement migrer partiellement ou totalement en IP)
- les PABX-IP ou IPBX ou PBXIP (qui nativement offrent une connectivité IP Ethernet)

Asterisk répond à cette deuxième catégorie. En effet, c'est un PBX applicatif open source (écrit en langage C) permettant d'interconnecter en temps réel des réseaux de voix sur IP via plusieurs protocoles (SIP, H323, ADSI, MGCP) et des réseaux de téléphonies classiques via des cartes d'interface téléphonique et tout ceci à moindre coût.

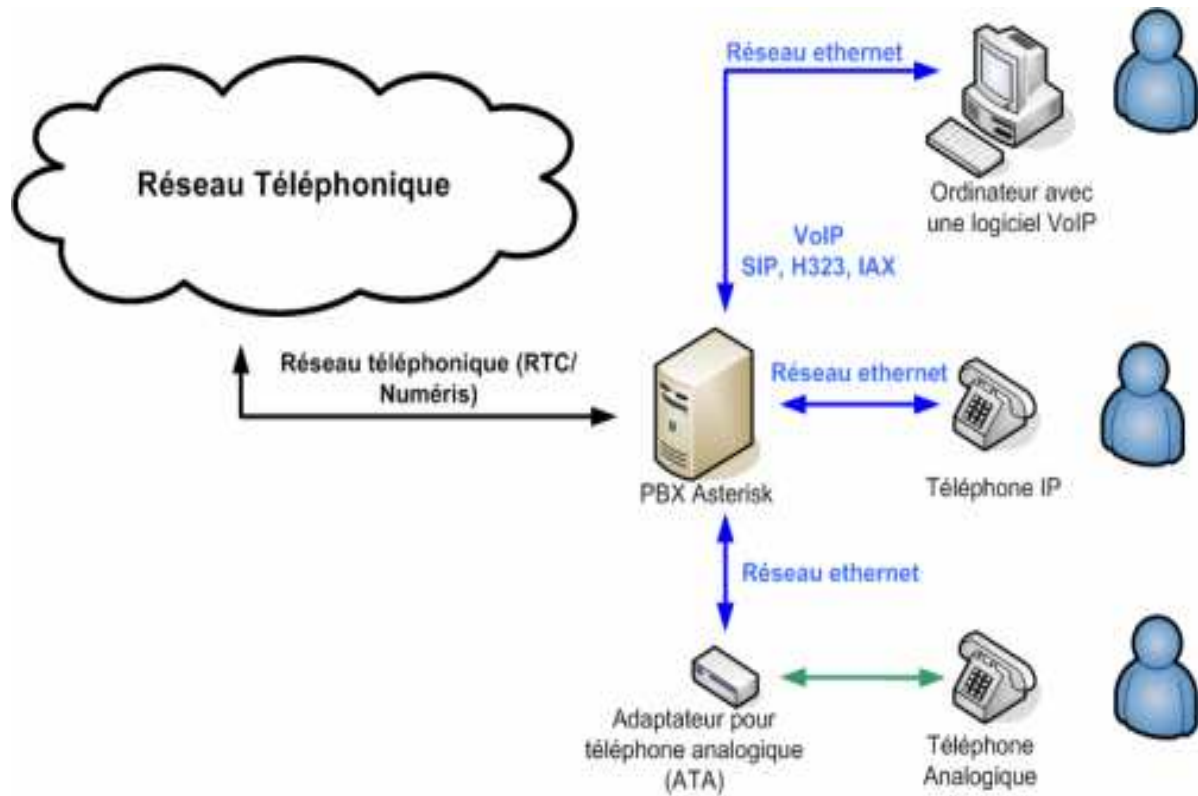
Par ailleurs, il offre toutes les fonctions d'un PBX et ses services associés comme de la conférence téléphonique, des répondeurs interactifs, de la mise en attente d'appels, des mails vocaux, de la musique d'attente, de la génération d'enregistrement d'appels pour l'intégration avec des systèmes de facturation, etc.

Bien qu'étant développé avant tout pour fonctionner sur des systèmes UNIX, il existe des portages plus ou moins officiels vers Windows ou MacOS.

L'avantage que présente Asterisk est qu'il est complètement paramétrable et configurable. De plus, il jouit d'une forte communauté d'utilisateurs, ce qui en facilite la résolution de problèmes lors d'installations ou de mises en place de nouveaux services.

L'inconvénient se situe peut-être du point de vue des développeurs. En effet, bien que tous les codes sources soient assez bien documentés, il n'existe à l'heure actuelle aucune documentation expliquant par exemple la création de nouveaux modules.

Le schéma suivant s'attache à représenter les différentes possibilités de fonctionnement d'un serveur Asterisk.



Comme on peut le voir il est tout à fait possible d'utiliser aussi bien un ordinateur possédant un logiciel adéquate (communément appelé Softphone) qu'un téléphone IP ou un téléphone classique au travers d'interfaces spécifiques (en général des cartes FXO).

Un serveur pouvant soit fonctionner de manière fermée en interconnectant les utilisateurs de son propre réseau. Soit de manière ouverte en se connectant avec le réseau téléphonique classique (RTC ou Numéris) ou avec d'autres serveurs (Asterisk, SIP, ...).

6.2 Etude et installation

6.2.1 Choix de la version

Une des premières étapes a consisté en l'étude du fonctionnement d'Asterisk mais également à en effectuer l'installation et une série de tests.

Mais avant tout, il a été nécessaire de faire un choix quant à la distribution d'Asterisk. En effet, il existe différentes versions dont le dénominateur commun est le noyau développé par l'entreprise fondatrice Digium.

Nous pouvons ainsi citer la version d'Asterisk 1.2 qui est la plus répandue et qui est la base de toutes les autres. Mais nombreuses sont les personnes (plus ou moins néophytes) préférant utiliser Trixbox (anciennement Asterisk@home) qui a su se forger une communauté de par sa facilité d'installation mais également grâce aux nombreux outils préinstallés qui en facilitent grandement l'utilisation.

Contrairement à la version "classique" d'Asterisk cité précédemment, il n'est nul besoin de passer par les lignes de commandes (dans la plupart des cas) et d'avoir une version de Linux installé, puisque Trixbox s'occupe d'installer une version de Linux (CentOS). Notons que l'utilisateur voit toute sa partition effacée avant l'installation de Trixbox.

J'ai donc dans un premier temps procédé à l'installation de Trixbox sur une machine virtuelle (pour éviter toute suppression de partition) en utilisant le logiciel VMWare. Après quelques tests où j'ai particulièrement apprécié les outils d'administration utilisables à partir d'un navigateur Internet et les modules préinstallés, j'ai été déçu par la manière dont le dialplan est géré.

Un dialplan (ou plan de numérotation) est le principal plan de contrôle ou d'exécution de toute opération. Il contrôle la façon de traiter et router tous les appels entrants et sortants. En d'autres termes, il permet de configurer le comportement de toutes les connexions du PBX.

Le contenu d'un dialplan est organisé en sections, qui peuvent être pour des configurations statiques et des définitions, ou alors pour des composants exécutables auquel cas on parle de contexte. Un type spécial de contexte est la macro.

Or, Trixbox utilise énormément les macros ce qui rend la lecture et la modification du dialplan plus difficile. Cela s'explique du fait que toutes les commandes, et les fichiers de configuration sont masqués par les différentes interfaces.

Par conséquent, j'ai préféré réaliser une installation classique d'Asterisk. Celle-ci, se fait sans trop de difficulté, en dépit du fait qu'il faut scrupuleusement suivre un ordre d'installation, et qu'il faille également apporter une petite modification au Makefile pour ajouter une option (le problème étant que cela n'est pas indiqué sur le site Web de Digium, et qu'on peut le découvrir avec un peu de curiosité en éditant ce fichier d'installation).

6.2.2 Tests et mise en place des services

Le choix de la version permettant la meilleure gestion du système et l'installation étant effectué avec succès, je me suis lancé dans l'étude à proprement parler de l'administration du PABX Asterisk.

Cela s'est effectué en plusieurs étapes, allant de la simple création d'un utilisateur à la création d'un routage vers un autre serveur permettant une communication avec "le monde extérieur".

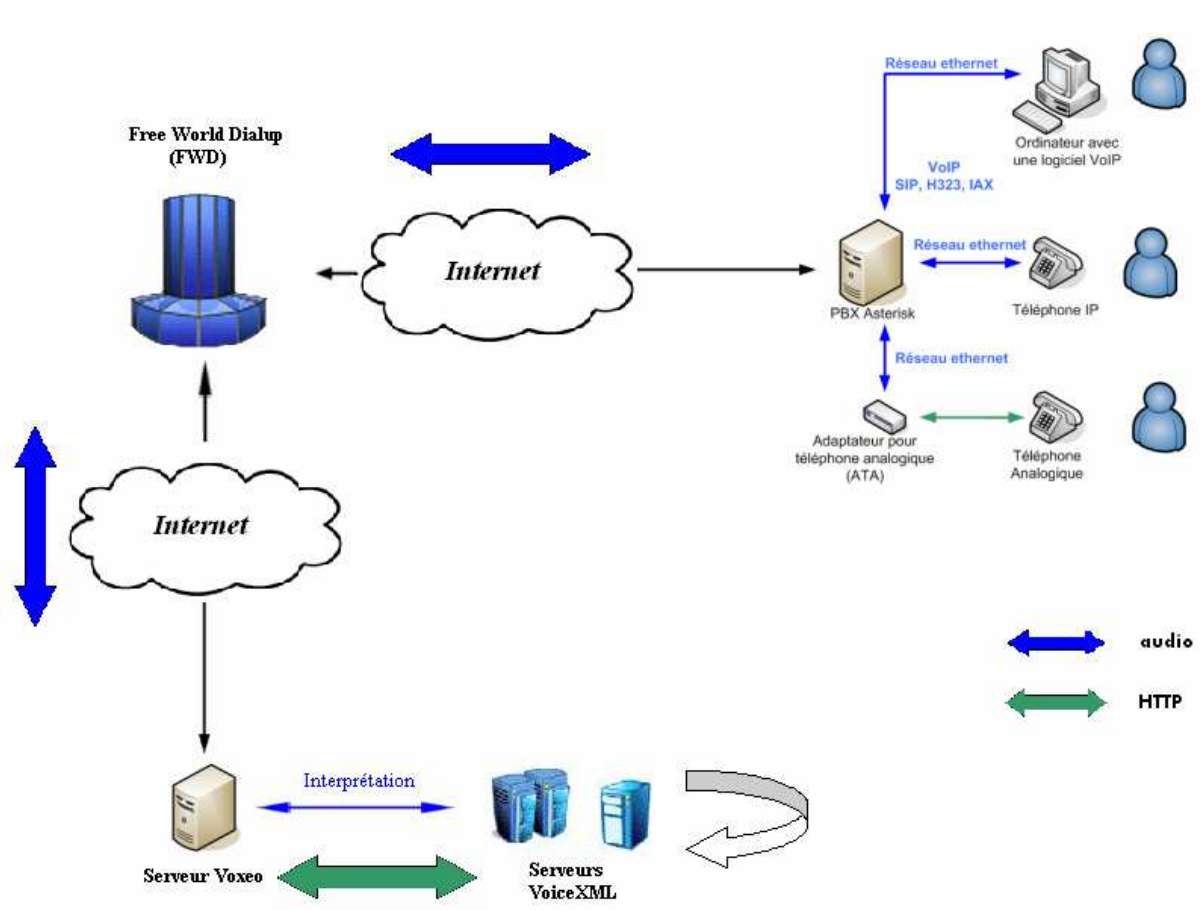
Dans un premier lieu, j'ai créé quelques utilisateurs en définissant certains droits, tels que les numéros pouvant être composés, la gestion de la messagerie vocale, etc.

Ceci a permis la mise en place des concepts de base d'un serveur téléphonique classique, mais également de se familiariser avec les concepts qui se cachent derrière le monde de la téléphonie, avec des standards tels que SIP ou RTP.

Ensuite, en plus de l'administration réseau, mon tuteur m'a demandé d'essayer de faire communiquer Asterisk avec un autre serveur supportant le standard VoiceXML. Il existe en effet sur Internet des fournisseurs qui permettent d'écrire et d'exécuter des applications VoiceXML. Un de ces fournisseurs est Voxeo et permet de pouvoir tester gratuitement (sur une base de 500 minutes de communications par mois) ses propres applications en passant par un logiciel tel que Skype.

Dans le monde de la téléphonie, on appelle « trunk » une ligne de communication entre le central téléphonique et un PBX ou tout simplement entre deux PBX. Il est important de noter que les trunks sont des connexions larges bandes capables de supporter plusieurs conversations simultanément.

Partant de cette notion, et ayant remarqué que Voxeo offre la possibilité de se connecter à un réseau FWD (Free World Dialup, qui est un service possédant des serveurs SIP permettant de passer des appels gratuitement entre utilisateurs inscrits) j'ai créé un trunk entre le serveur Asterisk et le réseau FWD. Ce dernier, jouant l'intermédiaire entre le serveur et le fournisseur Voxeo, a permis de pouvoir créer des IVRs écrits en VoiceXML.



6.3 Intégration à Sphinx 4 et réception en streaming

Parmi les composantes les plus importantes et les plus utiles du langage VoiceXML se trouvent la possibilité de recevoir des entrées utilisateurs à travers la voix (reconnaissance vocale) et l'envoi de messages au travers d'enregistrements (ou de synthèse vocale).

Bien qu'étant riche en terme de services, Asterisk ne possède à l'heure actuelle aucun module lui permettant d'effectuer la reconnaissance vocale (du moins à partir de solutions libres). Par ailleurs, ce PBX peut jouer des fichiers sons, mais aucune implémentation ne lui permet de jouer de l'audio reçu en flux continu (streaming).

Ce qui suit, présente comment l'intégration d'Asterisk avec le logiciel de reconnaissance de la parole Sphinx 4 a été réalisée. Puis est abordé le développement d'un module permettant à Asterisk de recevoir lors d'un appel, du streaming audio.

6.3.1 Couplage Asterisk – Sphinx 4

Jusqu'à présent, la solution prédominante pour effectuer de la reconnaissance automatique de la parole sur un appel téléphonique contrôlé par Asterisk, consistait à effectuer un enregistrement de l'appel.

L'enregistrement se terminait soit au bout d'un laps de temps déterminé (timeout) soit parce que l'appelant (ou l'appelé) avait appuyé sur une touche déterminée (que l'on appelle code DTMF). Asterisk, se chargeait ensuite de faire appel à un logiciel de RAP (par exemple Sphinx 4) qui effectuait la reconnaissance.

Il existe sur Internet une solution reprenant ce principe et qui s'appuie sur Sphinx 2. Cette dernière est une des plus anciennes versions de Sphinx mais qui reste tout de même réputée pour sa rapidité d'exécution.

En dépit des nombreuses demandes sur les forums de développeurs Asterisk, qui pour certaines remontent même à quelques années, aucune intégration de Sphinx 4 et d'Asterisk n'a été faite. Pourtant, au-delà des avantages évidents procurés par ce couplage, Sphinx 4 permet en outre de faire du « Live Decode ». Cela signifie qu'il est possible de faire de la reconnaissance vocale en temps réel sur un flux audio.

Pour permettre à ces deux logiciels de communiquer et fonctionner conjointement, le travail s'est articulé autour de trois pôles, présentés ci-dessous.

- Modification du code source de Sphinx 4 pour la réception en streaming
- Création d'un module Asterisk pour l'envoi des éléments vocaux
- Développement d'un serveur couplant Asterisk et Sphinx

6.3.1.1 Customisation de Sphinx 4

Puisque l'objectif est de faire du décodage en temps réel et que par défaut Sphinx 4 réalise cette tâche en prenant comme entrée le microphone, il a été nécessaire d'étudier et de modifier le code gérant ce dernier.

J'ai ainsi remodelé le code source du microphone, pour qu'il prenne en entrée un flux audio non plus de la carte son, mais à partir d'un serveur au travers de sockets. Malheureusement, cette modification seule n'a pas été suffisante. Il a fallu en outre, prendre en compte toutes les caractéristiques du nouveau signal et ajouter du code de conversion.

Le tableau suivant récapitule les principaux changements techniques qui interviennent.

	Sphinx 4	Téléphonie (Asterisk)
Fréquence	<i>16 KHz</i>	<i>8 KHz</i>
Echantillonnage	<i>16 bits</i>	<i>16 bits</i>
Format	<i>Wav</i>	<i>a-law, μ-law</i>
Bit de lecture	<i>Big-endian</i>	<i>Little-endian</i>

D'autre part, comme le démontre le tableau précédent, le monde de la téléphonie se base sur une fréquence de 8 KHz, alors que pour sa part, Sphinx fonctionne à l'aide de modèles acoustiques à 16 KHz. Ainsi, pour assurer la compatibilité, s'ensuit nécessairement la recréation des modèles appropriés à la téléphonie.

Pour cela, deux solutions se présentent

- Effectuer à nouveau des enregistrements à 8 KHz. Le mieux étant de faire ces enregistrements dans des conditions réelles d'utilisation du téléphone (avec le même bruit environnant et un vrai combiné téléphonique). Cette solution semble néanmoins être coûteuse bien que fournissant les meilleurs résultats
- Réutiliser les enregistrements à 16 KHz. On utilise pour cela la technique du *downsampling* qui vise à modifier "artificiellement" la fréquence d'origine. L'avantage indéniable est le gain de temps et de coûts. L'inconvénient étant que les modèles vont subir les aléas de la téléphonie (mauvaise qualité, bruit, etc.).

J'ai opté pour la seconde solution et ai donc modifié le modèle acoustique que j'avais créé pour l'adapter au modèle téléphonique.

6.3.1.2 Création du module Asterisk

Même si Asterisk se trouve être assez flexible, il ne permet pas à partir de son langage de script AGI (Asterisk Gateway Interface), d'effectuer des opérations complexes de contrôle des flux téléphoniques.

La majorité de la documentation disponible concerne l'écriture de scripts AGI et se trouve donc plutôt axé vers l'utilisateur final (ou plutôt l'administrateur). En contre partie, tout le coté « Asterisk Core » se trouve peu documenté et il faut parcourir les quelques forums de développeurs pour trouver des bribes d'informations.

La communication avec Sphinx ne pouvant se faire qu'à l'aide des fonctions internes d'Asterisk, je me suis attaché à l'écriture d'un module dont le fonctionnement est expliqué ci-après.

Le module développé se trouve sous la forme d'un patch applicable à Asterisk. Cela en facilite la diffusion. Un de ses avantages, est qu'il est tout à fait utilisable au travers d'un script AGI ou plus simplement au travers des commandes classiques que l'on peut trouver dans un dialplan.

Par conséquent, tout comme l'administrateur peut donner des commandes du type « *lorsque l'utilisateur 2304 compose le 666, enregistrer cet appel pendant 60 secondes* » il peut exécuter une commande qui s'interprétera comme « *lorsque l'utilisateur 2304 compose le 333, envoyer en streaming tous les flux vocaux vers le serveur d'adresse IP 192.168.0.52* ».

De cet exemple, on en déduit que le module prend un paramètre qui est l'adresse IP du serveur auquel il faut envoyer les flux vocaux. Il est également possible de définir une option qui mettra fin à l'envoi du streaming si l'utilisateur appuie sur la touche dièse. Si cette option n'est pas définie, par défaut l'envoi s'arrêtera dès que Sphinx 4 détectera la fin de la reconnaissance.

En somme, l'écriture du module consiste à définir sous quel format encoder le flux. Il existe en effet plusieurs codecs parmi lesquels les plus connus sont a-law ou μ -law. Pour faciliter la conversion du coté de Sphinx, le format choisi est le Pulse Code Modulation ou PCM (8 KHz – 16 bits – little endian).

Un traitement est également effectué sur toute information envoyée à partir d'un téléphone (et traitée par Asterisk sous forme de frame). On sépare ainsi toute donnée en trois groupes.

- flux vocal
- informations DTMF (produites à la suite de la saisie d'une touche du téléphone)
- autre information (vidéo par exemple, mais que nous ne traitons pas)

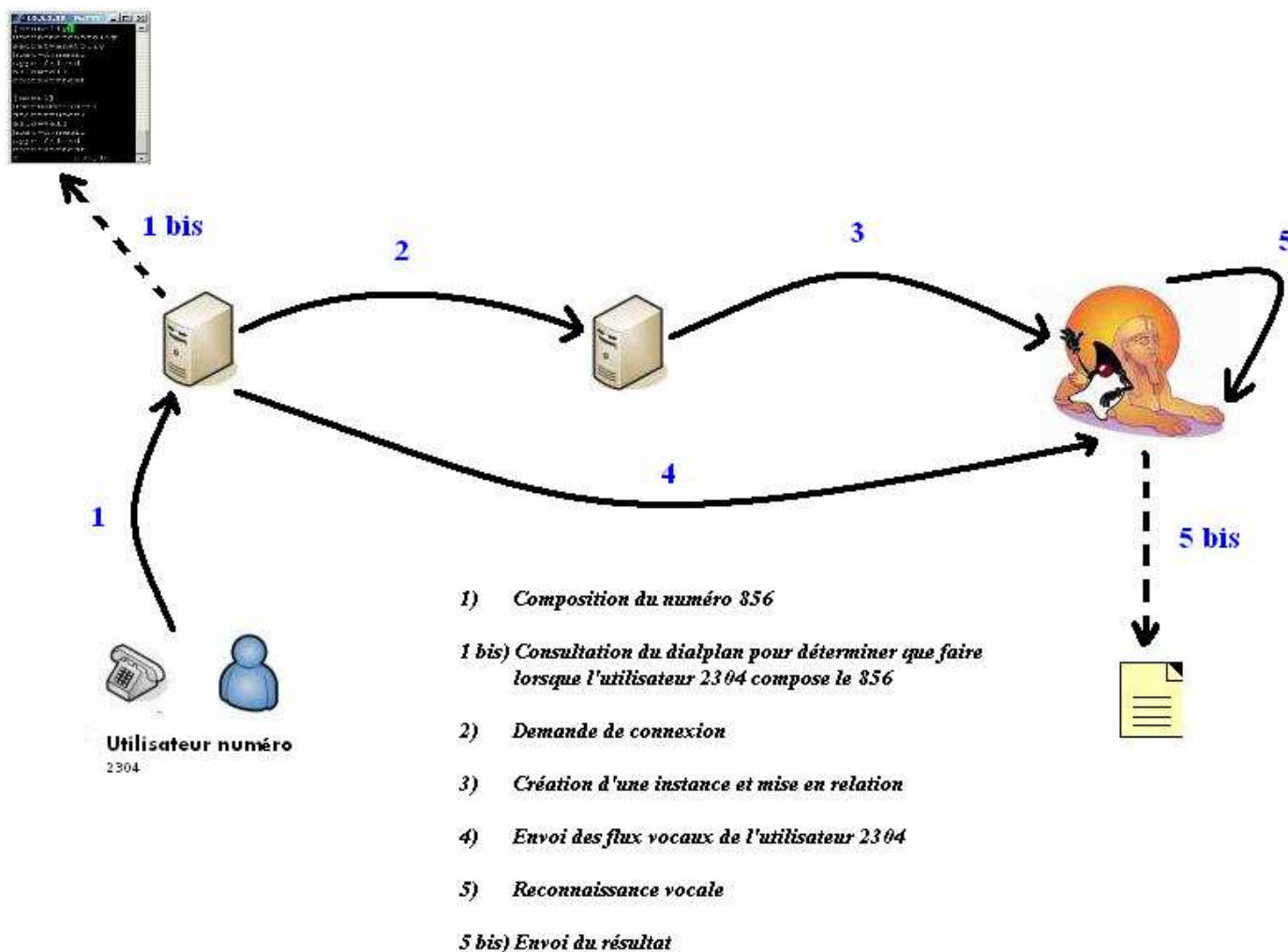
Seules les données de type vocal sont transférées au serveur (en les convertissant préalablement). Quant aux frames qui sont de types DTMF, elles permettent de mettre fin à l'exécution du module comme expliqué plus haut.

6.3.1.3 Mise en place d'un Serveur tiers

Pour coupler les deux entités (Asterisk et Sphinx 4), un serveur qui fonctionne de manière assez classique se met en écoute de nouveaux clients.

A chaque connexion est créée une nouvelle instance de Sphinx 4. Une communication est établie entre celle-ci et l'instance du module Asterisk qui s'est connectée au serveur.

Notons que pour chaque instance de Sphinx 4, existe un fichier de configuration fixant toute une kyrielle de propriétés indispensables au bon fonctionnement. Ce fichier doit être modifié pour répondre aux exigences induites par la téléphonie.



6.3.2 Transfère sur un appel d'un streaming audio

Sur demande du groupe contribuant à l'implémentation du projet JVoiceXML (présenté dans la section 3.4 de ce rapport), j'ai écrit un second module Asterisk qui effectue l'effet inverse du premier. Pour réaliser le couplage avec Sphinx 4, il était nécessaire de récupérer les flux audio envoyés depuis le téléphone pour les transférer vers un serveur tiers. Ici, l'objectif est de recevoir depuis un serveur (ou toute application appropriée) un flux audio et le transférer vers l'appel en cours.

Le module permet bien évidemment d'être utilisable au travers de toute commande classique Asterisk et de réaliser des ordres de type « *lorsque l'utilisateur 2304 appelle le numéro 999 jouer le fichier son musicAttente.wav puis une fois terminé lui jouer le streaming audio depuis le serveur 192.168.0.36 et à la fin raccrocher* ».

Pour mener à bien ce projet, il a été nécessaire d'étudier la manière de procéder des softphones pour transférer des flux vocaux. Il en est ressorti que nombreux sont ces logiciels de téléphonie qui détectent les moments de silence. En effet, pour économiser de la bande passante, si le bruit entrant est inférieur à un certain seuil, les softphones vont par défaut ne rien envoyer.

Or, le principe du module développé, consiste à récupérer d'une part dans une mémoire tampon le flux envoyé par un serveur tiers, et d'autre part, utiliser les frames vocales envoyées par les téléphones.

Ainsi, il suffit de faire un « dumping » de la mémoire tampon vers la frame capturée, et de faire renvoyer cette frame modifiée vers son expéditeur.

Fort heureusement, il est possible dans la plupart des softphones de désactiver la gestion des silences. De plus, même la dernière version d'Asterisk ne gère pas encore cette fonctionnalité ce qui ne pénalise en rien le bon fonctionnement.

Néanmoins, il est nécessaire que les fichiers lus en streaming soient convertis au format approprié à la téléphonie. De plus la qualité est assez médiocre et se trouve être fonction de la charge réseau.

Cette solution doit donc plutôt être vue comme une pré-version posant les bases nécessaires à un futur développement prenant en considération la qualité de service.

6.4 Intégration à JVoiceXML

6.4.1 Asterisk-Java

Asterisk-Java est un package qui contient un ensemble de classes Java permettant de développer des applications Java qui interagissent avec le serveur Asterisk. Asterisk-Java est composé de deux packages : AGI et Manager API.

L'interface AGI permet d'effectuer un ensemble de tâches prédéfinies pour un appel à travers un script extérieur. Asterisk fournit au script les informations sur l'appel (numéro appelé et appelant, CallerID, ...) à travers l'entrée standard.

Asterisk Manager permet au programme client de se connecter à une instance d'Asterisk et d'envoyer des commandes ou de recevoir des requêtes via TCP/IP.

Dans le Manager API, nous pouvons identifier trois concepts :

- Action : quand l'utilisateur veut envoyer une requête au serveur Asterisk, il envoie un message de type Action. Le message de type Action contient le nom de l'opération à exécuter et tous les paramètres requis.
- Response : la réponse envoyée par Asterisk suite à la dernière requête envoyée par l'utilisateur.
- Event : données concernant un événement généré de la part d'Asterisk ou un module d'extension

Généralement, le client envoie des messages de type action au serveur Asterisk, le serveur Asterisk exécute alors la requête du client, et retourne le résultat dans un message de type Response.

Les messages de type Event sont utilisés pour informer le client de l'état du serveur Asterisk (création de nouveaux canaux ...).

6.4.2 Connectivité Asterisk – JVoiceXML

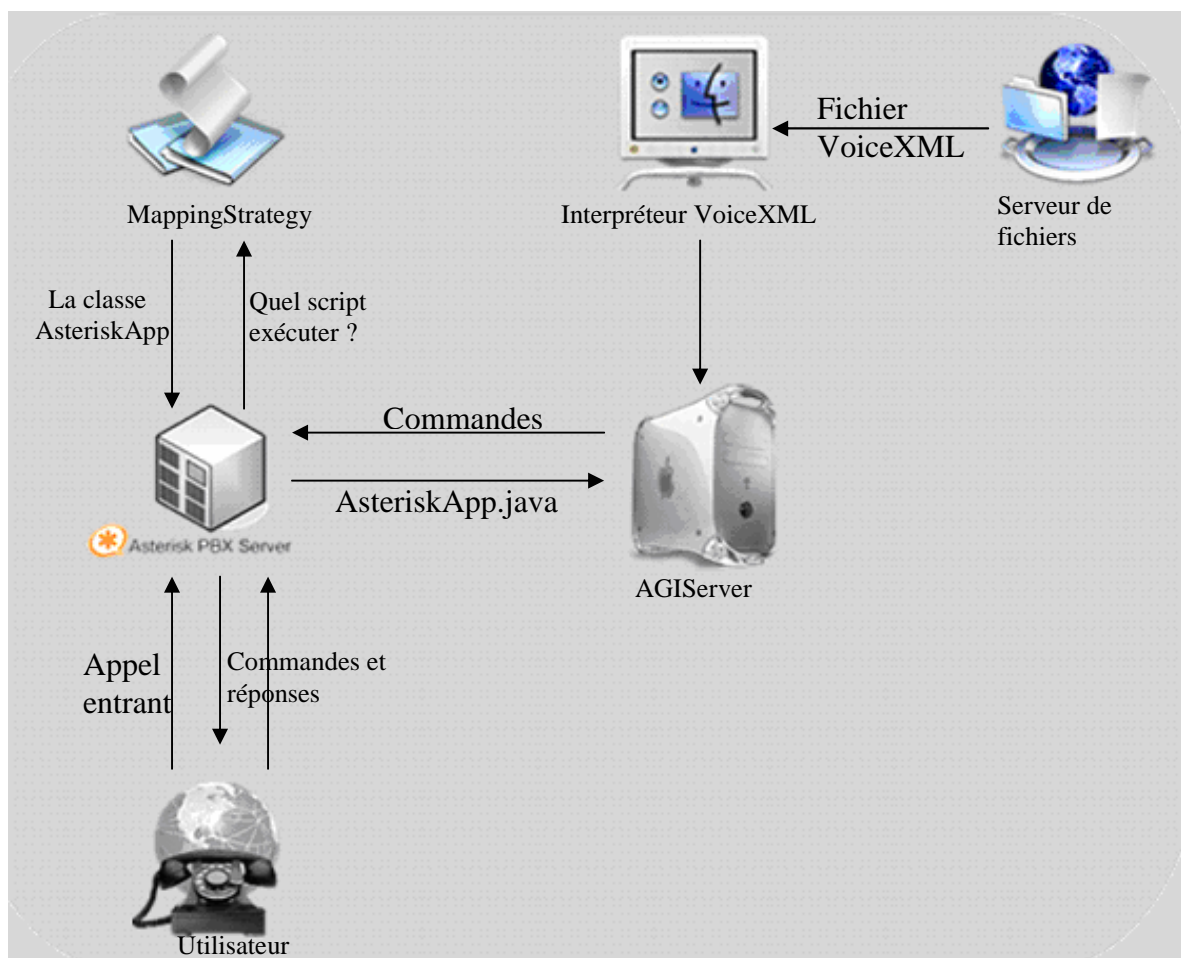
L'intégration de JVoiceXML à Asterisk se fait sur une architecture de type client-serveur. Ainsi, JVoiceXML par le biais de l'API Asterisk-Java présentée précédemment, va commander Asterisk au travers de commandes de type AGI.

Quant à la reconnaissance et la synthèse vocale, elles s'effectuent grâce aux outils développés qui s'intègrent tels quels dans l'environnement de JVoiceXML.

Détaillons le fonctionnement général lors d'un appel classique.

- Un utilisateur identifié par Asterisk appelle un numéro (autorisé)
- Asterisk vérifie au travers du MappingStrategy les commandes à exécuter lorsque l'utilisateur en question a composé le numéro donné.
Dans notre cas, il faut exécuter l'application Java nommée AsteriskApp.
- L'application AsteriskApp, va exécuter le script VoiceXML en appelant l'interpréteur JVoiceXML.

Est présenté ci-dessous un schéma récapitulatif.



Au-delà du stage

Outre l'aspect purement technique, ce stage m'a conduit à collaborer avec différentes personnes et à être parfois contacté. Il m'a aussi apporté l'envie de créer un service répondant à un besoin actuel : la commande par IVR.

7.1 Demandes et proposition

Mes diverses participations sur les forums et les différents échanges effectués au travers de mails, m'ont conduit à être contacté à maintes reprises.

Cela se résume en des demandes

- de partage de mon code source (principalement l'intégration Asterisk – Sphinx 4)
- d'écriture de tutoriaux concernant la création de modèles acoustiques sous Sphinx et de modules sous Asterisk
- de participation au projet JVoiceXML

Concernant les tutoriaux j'ai écrit un "step-by-step" expliquant comment réaliser un modèle acoustique pour Sphinx en précisant les points clés, les modifications à apporter aux scripts, et les divers pré-requis. De plus ayant souvent passé du temps à comprendre certains points indispensables au fonctionnement et à la configuration de Sphinx (et qui s'appliquent de manière générale à tout logiciel de reconnaissance vocale) j'ai écrit une petite Foire Aux Questions (FAQ).

J'ai également créé un document qui permet d'expliquer la mise en place d'un module Asterisk, en partant de l'installation des fichiers sources Asterisk jusqu'à l'écriture d'une petite application (en passant par l'explication des différentes notions propres à Asterisk nécessaires avant tout développement).

Toute cette documentation se trouve en pages annexes de ce rapport.

Pour ce qui est de la participation à JVoiceXML, j'ai été contacté par le leader du projet, à savoir Dirk Schnelle (chercheur Allemand) suite à un post sur le forum de Sphinx et où je parlais de mon intégration avec Asterisk.

Après quelques échanges de mails et de codes, on m'a proposé de rejoindre l'équipe en remplaçant le responsable de la partie téléphonie. J'ai accepté de travailler officieusement sur l'intégration de JVoiceXML avec Asterisk et leur ai dit que je leur donnerai une réponse définitive dès la fin du stage.

Enfin, pour répondre aux diverses demandes et en espérant contribuer au monde de l'open-source, mon code source sera très prochainement mis à disposition sur Internet.

7.2 Projet de création d'entreprise

Lors d'une entrevue avec un ami, nous en sommes venus à parler de téléphonie et des différents services qui s'y rattachent.

L'intérêt qu'il porte à ce domaine couplé aux différentes connaissances acquises durant le stage a fait émerger l'idée de créer un service en destination des restaurants.

Il s'avère en effet, que sur la région parisienne, des services à l'origine créés pour les personnes aveugles, ont été utilisés pour effectuer des commandes. A l'heure du déjeuner, nombreux sont les restaurants (de taille quelconque) qui se retrouvent submergés par une masse de personnes qui viennent manger. Ces personnes, n'ayant pour la plupart guère le temps, ont trouvé une facilité dans l'utilisation des systèmes interactifs téléphoniques, qui leur permettent de passer une commande à l'avance et choisissent de se restaurer sur place ou non. Ces services téléphoniques facilitent en outre la gestion des restaurateurs, qui savent à l'avance l'affluence à prévoir mais qui peuvent aussi préparer les repas.

C'est un service ressemblant à celui-ci que nous souhaitons développer sans se cantonner à la région montpelliéraine. En nous basant entre autre sur Asterisk et Sphinx nous prévoyons de faire une étude de marché, de calculer les différents coûts etc. Si le service semble être viable nous proposerons gratuitement le service à quelques restaurants montpelliérains qui en échange serviront de testeurs.

Nous avons aussi d'autres idées qui s'articulent toujours autour du domaine de la restauration. Dès la mi-septembre, nous prévoyons de faire les démarches nécessaires afin de sonder l'intérêt porté par les restaurateurs et les utilisateurs potentiels. En parallèle, nous commencerons à mettre en place les divers services pour ensuite réaliser quelques batteries de tests, ce qui nous assurera leur fiabilité.

Ce stage, au travers des retours d'expériences et des conseils de Mr Jean-Yves Delort, m'a appris que la partie relationnelle est tout aussi (si ce n'est plus) importante que l'aspect purement technique quant au bon fonctionnement d'une entreprise. Aussi, il nous semble fondamental, avant tout lancement de projet, de s'assurer que l'on possède un véritable sens relationnel.

Conclusion

Au travers de ce stage, j'ai non seulement acquis de nouvelles compétences et connaissances, mais également une certaine maturité.

De nouvelles compétences, car j'ai pu découvrir le merveilleux monde de la téléphonie couplée à l'informatique (communément appelé CTI). Asterisk est un des logiciels open-source qui suscite un important engouement de par sa flexibilité, son faible coût, sa facilité de mise en place, et ses possibilités que certains prétendent infinies.

Savoir l'administrer tout en allant au-delà (en sachant créer des extensions) est une plus-value indéniable.

En parallèle le standard VoiceXML connaît un grand succès. Pouvoir participer à un projet de grande envergure à échelle mondiale tel que JVoiceXML, permet de se situer à la pointe de l'évolution du standard, tout en ayant une prise de conscience de ses limites et de la difficulté à suivre scrupuleusement une spécification.

Enfin, la reconnaissance vocale était un domaine qui m'était totalement inconnu. Contrairement à VoiceXML et Asterisk, la RAP s'inscrit plutôt dans une composante de recherche. Ce fut donc avec appréhension que je me suis lancé dans la théorie de la reconnaissance de la parole, pour y découvrir au final les raisons qui conduisent les chercheurs à s'y investir depuis plus d'une cinquantaine d'années : le Traitement Automatique du Langage Naturel (TALN) est un domaine inépuisable faisant intervenir différentes connaissances. De plus, les applications qui en découlent permettent de faire avancer de manière significative, l'accessibilité et la productivité.

Une maturité dans le sens où j'ai réellement pris conscience que le travail d'un informaticien n'est pas de réaliser des tâches en autarcie sans se préoccuper de l'utilisation que l'on peut faire de son produit. En effet, il est crucial de prendre en considération l'utilisateur final.

Les divers échanges avec mon tuteur d'entreprise m'ont également amené à prendre en considération tout ce qui entoure un projet informatique. En effet, bien souvent, l'informaticien ne s'occupe que de la partie technique sans se soucier de la promotion, de la négociation et de la mise à disposition du produit final.

Créer une entreprise n'est pas chose facile, mais il existe tout un ensemble de subventions et de procédures qui facilitent la tâche. Se lancer dans cette direction, semble avant tout être une aventure riche en expérience, le tout étant de savoir où aller, comment y aller et surtout avoir un brin de passion dans ce que l'on fait.

Annexes

Dans ces annexes, figurent notamment les algorithmes les plus utilisés avec les Modèles de Markov Cachés et les tutoriaux que j'ai écrits et qui concernent l'écriture de modules pour Asterisk et l'écriture d'un modèle acoustique pour Sphinx.

Algorithmes

Trois algorithmes sont généralement associés aux Modèles de Markov Cachés (HMM)

- L'algorithme du forward utilisé pour la reconnaissance de mots isolés
- L'algorithme de Viterbi utilisé pour la reconnaissance de parole continue
- L'algorithme du forward-backward utilisé pour l'apprentissage des HMMs

L'algorithme du forward

Pour effectuer la reconnaissance de mots isolés, il est nécessaire d'évaluer la probabilité qu'un HMM donné (modélisant un mot) a produit une suite d'observations donnée. Ce qui permet par la suite de comparer les résultats de chaque modèle de mot et choisir celui qui a la plus grande probabilité.

Plus formellement, étant donné un HMM noté M composé de

- $\{s\}$ un ensemble d'états
- $\{a_{ij}\}$ un ensemble de probabilités de transitions où a_{ij} est la probabilité de passer de l'état i vers l'état j
- $\{b_i(u)\}$ un ensemble de probabilités d'émissions, où b_i est la distribution de probabilités dans l'espace acoustique décrivant les chances d'émettre chaque son possible u alors qu'on est dans l'état i

Nous devons ainsi calculer la probabilité que M a généré la suite d'observations $y_1^T = (y_1, y_2, y_3, \dots, y_T)$. Puisque chaque état i peut générer une observation u avec la probabilité $b_i(u)$, chaque suite d'état de longueur T contribue plus ou moins à la probabilité totale.

Un algorithme de type brutal chercherait simplement à lister toutes les suites d'état (de longueur T) possible, pour ensuite sommer leur probabilité de générer y_1^T .

On note cependant que c'est un algorithme peu pratique puisque de complexité exponentielle.

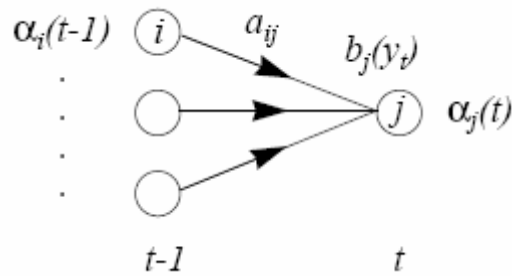
Une solution bien plus efficace est l'utilisation de l'algorithme forward qui est une instance de la classe des algorithmes connue sous le nom de programmation dynamique. Cet algorithme nécessite un traitement et un espace mémoire qui est seulement linéaire en T .

Dans un premier temps, on définit $\alpha_j(t)$ comme étant la probabilité de générer la suite partielle y_1^t , qui se termine dans l'état j à l'instant t .

$\alpha_j(t=0)$ est initialisé à 1.0 à l'état initial et à 0.0 dans tous les autres états.

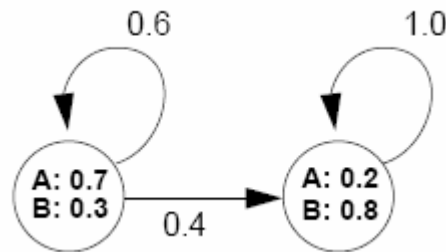
Si on a déjà traité $\alpha_i(t-1)$ pour tout i à l'instant $t-1$, alors $\alpha_j(t)$ peut être calculé récursivement par

$$\alpha_j(t) = \sum_i \alpha_i(t-1) a_{ij} b_j(y_t)$$



Si F est l'état final, alors par induction on peut voir que $\alpha_F(T)$ est la probabilité que le HMM a généré la suite complète d'observations y_1^T .

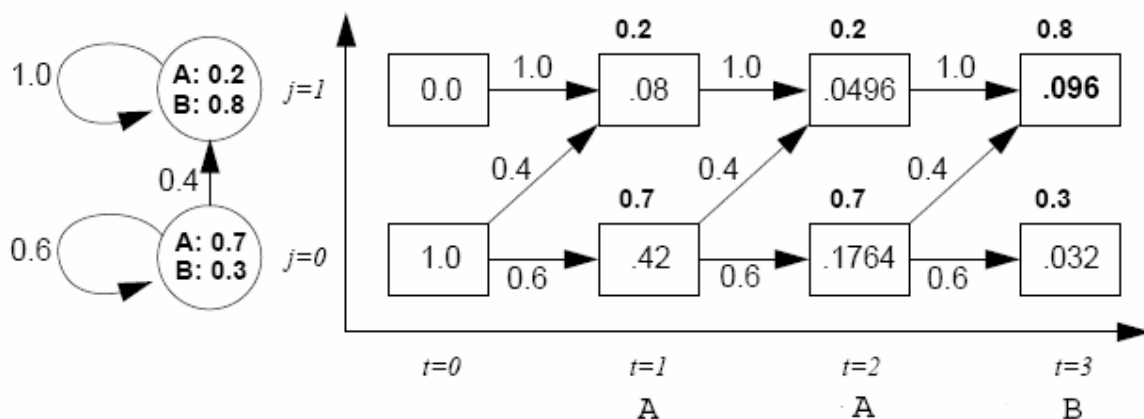
Soit un simple HMM représenté dans la figure suivante.



La figure qui suit, montre un exemple de l'exécution de cet algorithme. Est ici calculé la probabilité que la séquence d'observations $y_1^3 = (A, A, B)$ a pu être générée par le simple HMM précédent.

Chaque case aux coordonnées (t, j) représente la valeur de $\alpha_j(t)$, en utilisant les valeurs données de **a** et **b**.

Dans la dernière case, nous pouvons voir que la probabilité que ce HMM ait généré la suite (A, A, B) est de 0.096.



L'algorithme de Viterbi

Alors que l'algorithme du forward est utile pour la reconnaissance de mots isolés, il ne peut être appliqué avec de la parole continue, car avoir un HMM pour chaque phrase possible est trop coûteux (voir infaisable).

Dans le but d'effectuer de la reconnaissance sur de la parole continue, il vaut mieux déduire la séquence d'états qui a généré la suite d'observations donnée. En effet, à partir de la séquence d'états, on peut facilement retrouver la suite de mots.

Malheureusement, la séquence d'états est cachée (par définition) et ne peut être identifiée de manière unique. En effet, tout chemin peut avoir produit cette suite d'observations avec de faibles probabilités.

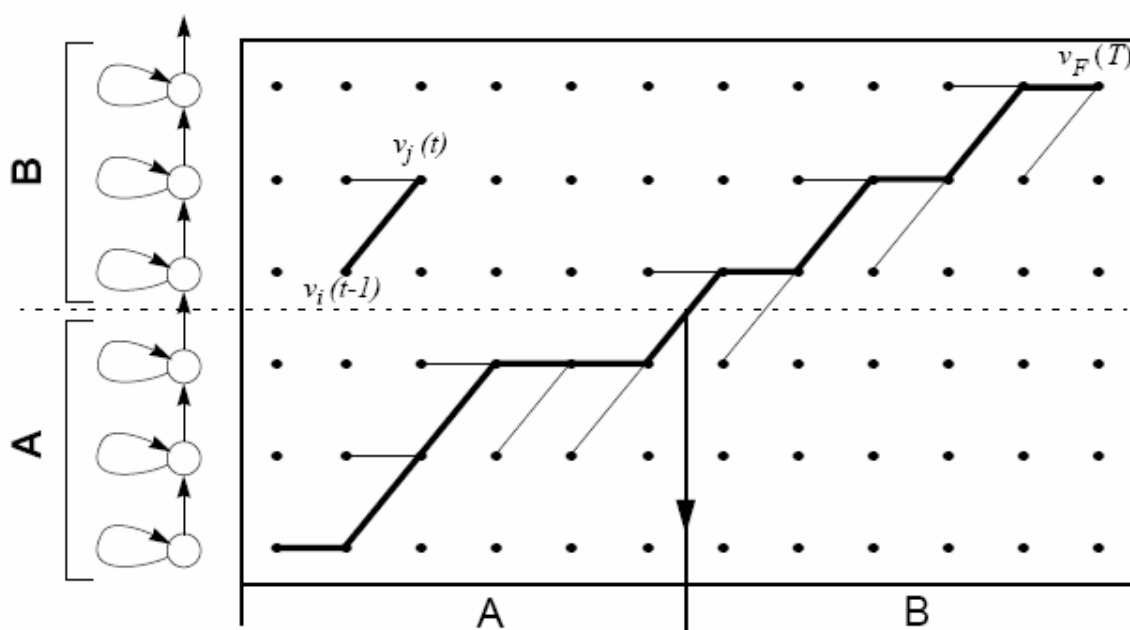
La meilleure chose à faire est donc de trouver l'unique séquence d'états qui a le plus de chance d'avoir produit la suite d'observations. Comme nous l'avons vu précédemment, nous pouvons faire cela en évaluant toutes les séquences d'états possibles et en extraire celle qui a la plus grande probabilité. Le problème est que l'on est face à une complexité exponentielle et par conséquent face à un algorithme infaisable.

L'algorithme de Viterbi est une bien meilleure approche qui est une fois de plus basée sur la programmation dynamique. Identique à l'algorithme du forward, il se différencie par le fait qu'au lieu d'évaluer la somme à chaque case, c'est le maximum qui est ici prit en considération

$$v_j(t) = \max_i [v_i(t-1)a_{ij}b_j(y_t)]$$

Cela identifie implicitement le meilleur état qui précède pour chaque case de la matrice. Si nous identifions explicitement le meilleur prédécesseur, tout en gardant une trace au fur et à mesure, alors lorsque $v_F(T)$ sera évalué (dernier état au dernier instant) nous pouvons grâce à la trace reconstruire la séquence entière d'états (comme la montre la figure ci-dessous).

Une fois que nous avons la séquence d'états, nous pouvons trivialement retrouver la séquence de mots.



L'algorithme du forward-backward

Pour faire l'apprentissage d'un HMM, nous devons optimiser **a** et **b** en tenant compte de la probabilité du HMM de générer toutes les séquences d'observations contenues dans les données d'apprentissage. Cela optimisera les chances du HMM de reconnaître correctement toute nouvelle donnée.

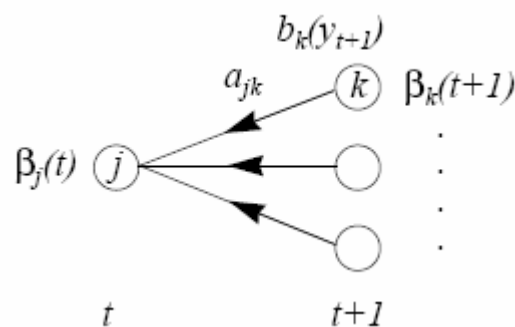
Malheureusement, cela s'avère être un problème difficile qui n'a aucune solution triviale. Par contre, ce que l'on peut faire, c'est commencer avec des valeurs initiales pour **a** et **b** pour ensuite modifier itérativement **a** et **b** en les re-estimant et en les améliorant, jusqu'à ce qu'un critère d'arrêt ait été atteint.

Une instance populaire de cette méthode est l'algorithme du forward-backward (également connu sous le nom de l'algorithme de Baum-Welch) que nous nous proposons de décrire ci-après.

Ayant déjà défini $\alpha_j(t)$ comme étant la probabilité de générer la séquence partielle y_1^t et qui se termine à l'état j à l'instant t , nous allons définir maintenant son image, à savoir $\beta_j(t)$ comme la probabilité de générer le reste de la séquence y_{t+1}^T , qui commence à l'état j à l'instant t . $\alpha_j(t)$ est appelé le terme forward (avant), alors que $\beta_j(t)$ est le terme backward (arrière).

Tout comme $\alpha_j(t)$, $\beta_j(t)$ peut être calculé récursivement, mais cette fois-ci dans un sens inverse

$$\beta_j(t) = \sum_k a_{jk} b_k(y_{t+1}) \beta_k(t+1)$$

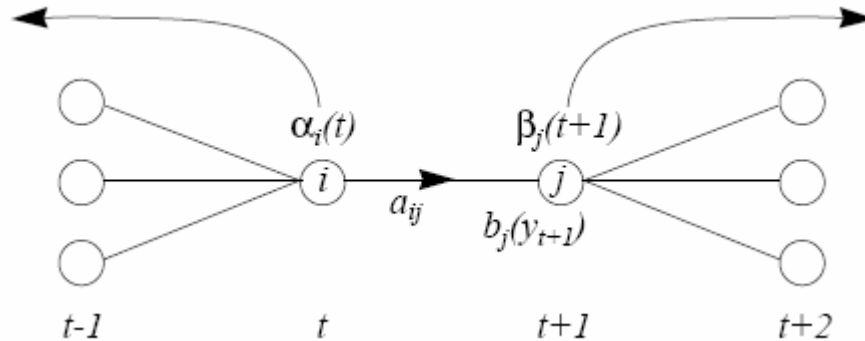


Cette récursivité est initialisée à l'instant T en fixant $\beta_j(t)$ à 1.0 pour l'état final, et 0.0 pour tous les autres états.

Ensuite, nous allons définir $\gamma_{ij}(t)$ comme étant la probabilité de passer de l'état i vers l'état j à l'instant t , en sachant que la suite d'observation entière y_1^T a été généré par le HMM :

$$\gamma_{ij}(t) = P(i_t \rightarrow j | y_1^T) = \frac{P(i_t \rightarrow j, y_1^T)}{P(y_1^T)} = \frac{\alpha_i(t) a_{ij} b_j(y_{t+1}) \beta_j(t+1)}{\sum_k \alpha_k(t)}$$

Le numérateur dans la dernière égalité peut être compris en regardant la figure qui suit. Le dénominateur reflète le fait que la probabilité de générer y_1^T est égale à la probabilité de générer y_1^T lorsqu'on s'arrête sur n'importe quel état final k .



Soit $N(i \rightarrow j)$ le nombre attendu de fois où la transition de l'état i vers l'état j est utilisée, de l'instant 1 à l'instant T :

$$N(i \rightarrow j) = \sum_t \gamma_{ij}(t)$$

En faisant la somme de tous les états de destination j , nous obtenons $N(i \rightarrow *)$, ou $N(i)$ qui représente le nombre attendu de fois où l'état i est visité, de l'instant 1 à l'instant T :

$$N(i) = N(i \rightarrow *) = \sum_j \sum_t \gamma_{ij}(t)$$

En sélectionnant seulement ceux qui sont occasionnés quand l'état i génère la sortie u , on obtient $N(i, u)$:

$$N(i, u) = \sum_{t: (y_t=u)} \sum_j \gamma_{ij}(t)$$

Enfin, nous pouvons ré-estimer les paramètres \mathbf{a} et \mathbf{b} , en générant \bar{a} et \bar{b} en sélectionnant un simple coefficient parmi ces termes :

$$\bar{a}_{ij} = P(i \rightarrow j) = \frac{N(i \rightarrow j)}{N(i \rightarrow *)} = \frac{\sum_t \gamma_{ij}(t)}{\sum_j \sum_t \gamma_{ij}(t)}$$

$$\bar{b}_i(u) = P(i, u) = \frac{N(i, u)}{N(i)} = \frac{\sum_{t: (y_t=u)} \sum_j \gamma_{ij}(t)}{\sum_t \sum_j \gamma_{ij}(t)}$$

On peut prouver qu'en substituant $\{\bar{a}, \bar{b}\}$ par $\{a, b\}$ cela fera toujours croître $P(y_1^T)$, jusqu'à un maximum local. Ainsi, en répétant cette procédure pour un nombre d'itération, les paramètres du HMM seront optimisés pour les données d'apprentissage.

Création d'un modèle acoustique avec SphinxTrain

Introduction

SphinxTrain est une application qui permet de créer des modèles acoustiques pour les différentes versions de Sphinx.

Créé à la base pour Sphinx2 et les différentes déclinaisons de Sphinx3, il est également utilisé pour la version Java à savoir Sphinx4. En effet, ce dernier utilise les mêmes modèles acoustiques que Sphinx3, cependant il faudra à la fin de la création procéder à quelques opérations supplémentaires, puisque Sphinx4 utilise des fichiers jar pour la lecture de ces modèles (auxquels il faut fournir quelques informations).

En débutant avec Sphinx et plus généralement avec le domaine de la reconnaissance vocale, j'ai regretté le fait qu'il y ait peu d'informations concernant la création d'un modèle acoustique.

C'est pourquoi, après avoir bien « pataugé » je propose ce petit tutorial expliquant par étape comment utiliser SphinxTrain pour créer un modèle acoustique permettant la reconnaissance des chiffres français allant de zéro à neuf.

Dans un souci de clarté et de gain de temps, à chaque étape j'ai mis en exergue les points susceptibles de provoquer une erreur et sur laquelle on pourrait perdre beaucoup de temps à chercher la solution.

Installation

Bien sûr, il est nécessaire de télécharger SphinxTrain. On peut le récupérer à l'adresse suivante <http://cmusphinx.sourceforge.net/html/download.php>

L'installation se fait de manière classique avec les commandes suivantes (exécuter la dernière commande en tant que root)

```
tar xvzf SphinxTrain.tar.gz
cd SphinxTrain
./configure
make
make install
```

Sur le site de Sphinx se trouve un lien sur la documentation qui cependant est peu complète et surtout peu claire du point de vue pratique de l'utilisateur de l'application.

Il est préférable se référer aux indications présentes dans le fichier nommé « tinydoc.txt » contenu dans le répertoire SphinxTrain/doc.

Cependant, comme l'a plusieurs fois souligné Arthur Chan (un des créateurs de Sphinx et SphinxTrain) les informations contenues dans ce fichier sont pour la plupart désuètes et risquent de ne pas fonctionner.

Configuration de SphinxTrain

La phase d'apprentissage d'un nouveau modèle est composée de différentes étapes. Il faut avant tout donner un nom au modèle en créant le répertoire correspondant qui en portera le nom. Dans notre cas, nous l'appellerons chiffres.

```
mkdir chiffres
cd chiffres
```

Il faut à présent créer la structure du répertoire que l'on vient de créer. Cela se fait automatiquement grâce au script Perl fournit par SphinxTrain.

```
SPHINXTRAIN = /home/mon_nom/mon_rep/SphinxTrain/tutorial/SphinxTrain
$SPHINXTRAIN/scripts_pl/setup_SphinxTrain.pl -task chiffres
```

Il est impératif à ce niveau, de créer deux fichiers

```
touch etc/chiffres.fileids
touch etc/chiffres.transcription
```

Le premier fichier (chiffres.fileids) contient la liste des fichiers audio (sans leurs extensions) à utiliser pendant la phase d'apprentissage du modèle. Fichiers qui devront par la suite être copiés dans le répertoire */wav* créé automatiquement précédemment.

Le second fichier (chiffres.transcription) contient les transcriptions des phrases prononcées dans leur fichier audio respectifs (fichiers audio sans leurs extensions).

Voici un exemple des deux fichiers

etc/chiffres.fileids	etc/chiffres.transcription
audio1	<s> un deux trois </s> (audio1)
audio2	<s> deux huit neuf neuf </s> (audio2)
audio3	<s> quatre cinq six cinq sept </s> (audio3)
...	...

Quelques informations complémentaires doivent être apportées

- ce qui a été dit précédemment suppose donc qu'on a préalablement créé les fichiers audio. Ceux-ci doivent respecter le format suivant
 - **16 bits** (une précision de 8 bits serait insuffisante)
 - **16 KHz ou 8Khz** (mieux vaut préférer 16 KHz, 8KHz étant réservé pour la téléphonie. De plus, Sphinx semble mieux fonctionner avec du 16KHz)
 - **mono** (un seul canal, le stéréo n'étant pas supporté)

- bien que la documentation dit qu'il faut créer des fichiers au format wav (d'ailleurs, on notera la présence du répertoire /wav que l'on a souligné précédemment), SphinxTrain n'accepte pas les headers (informations d'en-têtes contenues par exemple avec le format wav). C'est pourquoi, il est nécessaire de convertir les fichiers créés en format raw (comprendre par là en format brut, sans headers).
Pour cela, il existe sous Linux un très bon logiciel libre qui s'appelle sox, et à partir duquel on pourra effectuer la conversion de la manière suivante :

```
sox in.wav -t raw -w -r 16000 -c1 out.raw
```

où l'option

- -w indique une précision de 16bits
- -c1 indique l'utilisation d'un seul canal
- -r 16000 indique un taux d'échantillonnage de 16KHz

- Il est recommandé de créer un petit script qui permettra de convertir en raw, tous les fichiers wav contenus dans un répertoire donné !

Il faut ensuite créer le fichier de remplissage « filler » qui contient tous les sons contenus dans les fichiers audio mais qui ne représentent pas de la parole à proprement parler. Par exemple on pourra y recenser les passages audio où y figurent les rires, les sonneries de téléphones ou tout simplement les « euh » « hum » « ah ».

```
touch etc/chiffres.filler
```

Notons que même si l'on ne souhaite pas reconnaître ce genre de sons, il faut malgré tout le créer. De plus, ce fichier possède au minimum le contenu suivant :

etc/chiffres.filler	
<s>	SIL
</s>	SIL
<sil>	SIL

Ne pas oublier d'inclure à la fin du fichier un retour à la ligne. En effet, il est impératif et fait partie des erreurs presque invisibles.

Il faut maintenant convertir les fichiers audio du format raw, vers un format adapté à SphinxTrain et que l'on appellera fichiers de caractéristiques (features files).

Nous utilisons pour cela un script.

```
cd chiffres
bin/make_feats -ctl etc/chiffres.fileids
```

Attention cependant ! Il existe dans ce script une erreur non corrigée qui provient de la mauvaise utilisation du format de fichiers.

C'est pourquoi il faut éditer le fichier « bin/make_feats » pour y rechercher la ligne comportant

```
"bin/wave2feat -verbose yes -c \"$ctl\" -nist yes "
```

et la remplacer par la ligne suivante

```
"bin/wave2feat -verbose yes -c \"$ctl\" -raw yes "
```

Tout comme on a créé le fichier filler, créons le dictionnaire et la liste des phonèmes.

Rappelons qu'un phonème est la plus petite unité discrète permettant de distinguer des mots les uns des autres. C'est une entité abstraite qui peut correspondre à plusieurs sons.

```
touch chiffres.dic
touch chiffres.phone
```

Quelques précisions sur ces deux fichiers

- chiffres.dic va contenir la liste de tous les mots contenus dans le fichier de transcription (chiffres.transcription) avec pour chacun leur décomposition en phonèmes. Un mot peut être prononcé de différentes manières, c'est pourquoi certains mots sont suivis d'un chiffre entre parenthèse indiquant une variante de la prononciation.
- chiffres.phone va quant à lui contenir la liste de tous les phonèmes contenus dans le fichier chiffres.dic, en prenant soin de ne mettre qu'un phonème par ligne.

Voici le contenu des fichiers que nous utiliserons pour notre modèle.

chiffres.dic	chiffres.phone
cinq s in	a
cinq(2) s in k	ai
deux d eu	d
huit y i	e
huit(2) y i t	eu
neuf n eu f	f
quatre k a t r	i
sept s e t	in
six s i	k
six(2) s i s	n
trois t r w a	o
un un	r
zero z ai r o	s
	SIL
	t
	un
	w
	y
	z

Tout comme dans le filler, penser à mettre une ligne vide à la fin de chaque fichier.

Enfin, il ne nous reste plus qu'à lancer le script de vérification qui nous permet de valider la configuration et de pouvoir procéder à la création du modèle acoustique.

```
./scripts_pl/00.verify/verify_all.pl
```

Malgré le soin pris dans la configuration de SphinxTrain, quelques erreurs lors de cette étape peuvent apparaître. Dans la plupart des cas cela résulte de la version de SphinxTrain utilisée et s'avère être bénigne.

Par exemple, il suffira simplement de renommer les fichiers chiffres.fileids et chiffres.transcription respectivement par chiffres_train.fileids et chiffres_train.transcription.

Aussi, si vous utilisez une ancienne version de SphinxTrain, il se peut que le fichier de configuration « etc/sphinxtrain.cfg » contienne la ligne suivante

```
$CFG_FEATFILE_EXTENSION = 'feat'
```

Dans ce cas il est préférable de le remplacer par

```
$CFG_FEATFILE_EXTENSION = 'mfc'
```

Enfin, pour tout autre erreur, veuillez vous reporter à la petite FAQ présente à la fin de ce tutorial.

On remarquera que, suite à l'exécution du script de vérification, un fichier HTML est créé. Celui-ci s'avérera fort utile pour le débogage et l'évaluation de l'efficacité du modèle. En effet, durant toutes les étapes qui vont suivre, ce fichier sera mis à jour et contiendra toutes les erreurs et tous les avertissements (warnings), ainsi qu'une trace de l'exécution.

Création du modèle acoustique

Fort heureusement, SphinxTrain contient une suite de scripts à exécuter. Ils possèdent un numéro qui correspond à l'ordre d'exécution.

Néanmoins, il ne suffit pas tout simplement de les exécuter un par un (bien que cela fonctionnerait également). En effet, certains scripts ne sont utiles que pour Sphinx2, d'autres inutiles dans le cas de la création d'un modèle acoustique Context-Indépendant (CI).

On entend par là, un modèle acoustique où les phonèmes ne sont pas modélisés de sorte à tenir compte du contexte dans lesquels ils apparaissent. En effet, un son ne va pas posséder les mêmes caractéristiques selon le son qui le précède et le son qui le suit. Dans le cas d'un CI on ne tient pas compte de cette information, ce qui a pour conséquence de réduire le temps de création du modèle et de réduire en même temps la qualité de la reconnaissance (dans le cas d'un grand dictionnaire).

Les scripts de numéros 1, 8 et 9 sont exclusivement utilisés pour Sphinx2.

Pour ceux qui seraient intéressés par la création d'un modèle pour cette version de Sphinx, prenez note qu'il existe une petite correction à apporter au script.

En effet, l'exécution de la commande

```
./scripts_pl/01.vector_quantize/slave.VQ.pl
```

fait à son tour appel à deux autres scripts, dont « kmeans.pl » qu'il faudra éditer pour remplacer la ligne

```
$logfile = "$logfile/${CFG_EXPTNAME}.kmeans.log";
```

par

```
$logfile = "$logdir/${CFG_EXPTNAME}.kmeans.log";
```

On peut à présent utiliser le reste des scripts.

```
./scripts_pl/02.ci_schmm/slave_convvg.pl
./scripts_pl/03.makeuntiedmdef/make_untied_mdef.pl
./scripts_pl/04.cd_schmm_untied/slave_convvg.pl
./scripts_pl/05.builtrees/make_questions.pl
./scripts_pl/05.builtrees/slave.treebuilder.pl
./scripts_pl/06.prunetree/slave.state-tie-er.pl
./scripts_pl/07.cd-schmm/slave_convvg.pl
```

Ne sont pas représentés ci-dessus les scripts 8 et 9 qui comme nous l'avons dit, ne sont utiles que pour Sphinx2.

Si l'on souhaite créer un CI, il suffit de s'arrêter après l'exécution du script numéro 2.

Par ailleurs, à chaque étape, il est tout à fait possible d'utiliser le modèle intermédiaire créé.

A la fin de l'exécution, regarder dans le répertoire chiffres/model_parameters pour avoir le modèle acoustique.

Pour utiliser le modèle créé sous Sphinx4, il suffit de suivre les instructions données à l'adresse suivante :

<http://cmusphinx.sourceforge.net/sphinx4/doc/UsingSphinxTrainModels.html>

FAQ

1) *Des erreurs apparaissent lors de l'exécution du script de vérification. Que faire ?*

Penser avant tout à vérifier si SphinxTrain a bien été configuré et si certains phonèmes présents dans le dictionnaire, apparaissent bien dans le fichier portant l'extension .phone.

Parfois quand on crée le fichier de transcription (ou autre) sous un système d'exploitation tel que Windows, les retours à la ligne sont modélisés par un simple « \n » or pour que SphinxTrain marche il faut que les retours à la ligne se fassent par « \n\r ».

Par conséquent, la solution est de refaire (automatiquement au travers d'un script) tous les retours à la ligne.

Une erreur simple mais que l'on voit difficilement provient dans le fichier des transcriptions. Celui-ci réclame un espace entre le délimiteur <s> et le mot qui le suit mais également un autre espace entre </s> et le mot qui précède. Par conséquent une ligne comme <s>un deux</s> devra être remplacée par <s> un deux </s>

2) *Comment puis-je obtenir des statistiques sur les phonèmes utilisés dans le modèle ?*

SphinxTrain possède un petit outil (mk_mdef_gen) que l'on peut utiliser de la manière suivante

bin/mk_mdef_gen	-dictfn	etc/chiffres.dic
	-fdictfn	etc/chiffres.filler
	-lsnfn	etc/chiffres_train.transcription
	-phnlstfn	etc/chiffres.phone
	-ocountfn	stat.txt

Cela créera un fichier stat.txt qui contiendra diverses informations, en particulier la fréquence d'apparition des phonèmes.

Il est recommandé en effet que la fréquence d'apparition obéisse à une distribution aussi équitable que possible et ce, pour améliorer la reconnaissance.

3) *Comment créer un modèle acoustique à 8KHz ?*

Pour créer un modèle acoustique à 8KHz (correspondant à une bande étroite, comme celle du téléphone), il existe deux solutions.

- Soit refaire des enregistrements audio en PCM, 8KHz, 16 bit, signé.
- Soit utiliser la technique du downsampling qui consiste à utiliser un logiciel tel que sox, pour re-échantillonner à 8KHz des fichiers qui à l'origine sont en 16KHz

Par ailleurs, il faut modifier le fichier bin/make_feats. Au niveau de l'appel de wave2feats, apporter/modifier les options suivantes

-lowerf	200
-upperf	3500
-nfft	256
-srate	8000
-nfilt	31

4) *Combien d'enregistrements audio effectuer ?*

Dans le cas d'un petit modèle tel que celui créé dans le tutorial, j'ai créé 100 fichiers audio. Cela correspond à un minimum. En effet, plus grande est la quantité de données, meilleur sera le modèle.

Il faut savoir que l'on ne compte pas les données en terme de nombre de fichiers mais en heures d'enregistrements. Ainsi, 10 heures de données audio correspondent à un petit modèle acoustique tout à fait raisonnable.

Néanmoins, de gros modèle acoustique représentant de larges vocabulaires utilisent environ 80 heures de corpus audio.

5) *Pourquoi je n'arrive pas à utiliser mon modèle acoustique avec les démos fournies dans Sphinx4 ?*

Si une exception de type `NullPointerException` est levée, cela provient forcément du classpath. Dans ce cas, il faut simplement modifier par exemple le fichier manifest contenu dans le répertoire qui permettra la création du fichier jar de la démo.

6) *J'ai vu que l'on utilise les Hidden Markov Models (HMM ou Modèles de Markov Cachés) pour modéliser les phonèmes. Mais combien d'états par phonèmes ?*

Il existe différentes techniques de traitement de la parole (Dynamic Time Warping, réseaux de neurones) dont les Modèles de Markov Cachés.

Déterminer le nombre d'état relève directement d'un compromis entre performance et coût. C'est en faisant plusieurs tests qu'on arrive à trouver le nombre optimal d'état.

La plupart du temps on utilise des HMM à 3 états, et parfois même à 5 états.

La raison provient d'une question de topologie. Dans un modèle à 3 états on utilise typiquement une topologie gauche-droite (c'est-à-dire que les transitions se font uniquement d'un état vers lui-même ou d'un état vers son successeur). Par conséquent, les sauts d'états ne sont pas possibles, ce qui présente l'avantage d'être facilement implantable (et qui plus est, fonctionne assez bien).

Néanmoins, un état correspond à une frame acoustique. Et c'est là que peut résider le problème. En effet, si l'on choisit un modèle à 6 états par exemple, et que le phonème n'est composé que de 3 frames, il va falloir introduire des sauts d'états. Cela va immédiatement affecter le nombre de chemins que l'algorithme de Viterbi doit prendre en considération et augmenter ainsi le temps de calcul.

De plus, et cela est non moins négligeable, chaque état possède une distribution de probabilité pour les observations. Ces distributions doivent être estimées durant la phase d'apprentissage

du modèle acoustique. Ainsi, il y a bien plus de paramètres à estimer (au travers de l'algorithme de Baum-Welch) et si l'on n'a pas assez de données pour les estimer, cela va considérablement affecter les performances.

7) *Quelles sont les différences entre Sphinx3 et Sphinx4 ?*

Une des premières différences est bien évidemment que Sphinx3 est écrit en C/C++ tandis que Sphinx4 est entièrement écrit en Java.

D'ailleurs pour ceux que cela intéresse, il existe une comparaison des performances des deux logiciels à l'adresse http://cmusphinx.sourceforge.net/sphinx4/#speed_and_accuracy

La différence fondamentale est au niveau de l'algorithme utilisée dans le décodage de grands vocabulaires. Il existe en effet de nombreux chemins potentiels, et le décodage de ces derniers est primordial. Sphinx3 utilise un ensemble d'arbres lexicaux et traite les chemins actifs, directement sur ces arbres.

Sphinx4 pour sa part, utilise un seul arbre lexical sans traiter les chemins actifs directement sur celui-ci.

On peut citer également les différences suivantes

- Sphinx4 permet l'utilisation de HMMs avec un nombre d'états arbitraire, et sans restrictions quant à la topologie (on peut avoir des sauts d'états au niveau des transitions). Sphinx3 ne permet d'avoir des HMMs que de 3 ou 5 états et une topologie gauche-droite (aucun saut d'état). Notons cependant que Sphinx4 ne peut tirer profit de sa flexibilité puisqu'il utilise des modèles acoustiques créés pour Sphinx3.
- Sphinx4 ne met aucune contrainte quant à la profondeur des grammaires, ce qui lui permet d'utiliser des n-grams. A l'inverse, Sphinx3 ne permet l'utilisation que de modèles 3-grams.
- Sphinx4 est beaucoup plus modulaire et configurable que Sphinx3

8) *Lors de la création d'un modèle acoustique, de quelles longueurs doivent être les enregistrements ?*

En réalité, il n'y a pas de règles précises. Ainsi certaines personnes utilisent des enregistrements de presque 60 secondes (au-delà la qualité finale se dégrade rapidement). Plusieurs conversations, soulignent cependant qu'au vu de diverses expériences, le fait que des enregistrements ne dépassant pas les 10 secondes sont préférables car fournissant de bien meilleurs résultats.

9) *Quelle différence y a-t-il entre un modèle de langage (ML) et une grammaire ?*

Le ML est basé sur les probabilités et non pas sur des règles (par exemple : probabilité d'avoir tel mot en fonction des 2 mots précédents)

Une grammaire est plus une liste de règles qui contraignent le moteur de reconnaissance.

Le ML prend en considération tous les mots possibles alors qu'une grammaire n'est pas obligée d'être exhaustive.

Avec des grammaires JSGF, on peut ajouter des éléments probabilistes; ce qui est fortement recommandé pour accroître la précision.

10) Quelles sont les avantages/inconvénients d'un modèle de langage (ML) et d'une grammaire ?

Approche basée sur la grammaire

- recommandée pour les applications de type « command and control » (i.e. robots)
- écrit à la main où l'on peut inclure tous les mots que l'on veut reconnaître
- même en utilisant des grammaire de type JSGF où on peut y définir des probabilités d'apparition, le problème est de bien définir ces probabilités

Approche ML

- facile à mettre en place à condition d'avoir un bon corpus
- fournit des probabilités; on a donc une meilleure précision
- cependant il faut choisir le bon corpus pour l'apprentissage (selon le contexte)

11) Quelle est la différence entre un modèle acoustique (MA) et un modèle de langage (ML) ?

Un MA va fournir des probabilités sur les différents phonèmes. Alors que le ML fournit au moteur de la reconnaissance (Recognizer) un moyen de combiner ces phonèmes dans des séquences de mots les plus probables.

12) J'ai une erreur lors de la normalisation de la moyenne et de la variance avec la commande « norm » à l'étape du CD untie, que faire ?

Cela provient de l'estimation de la variance. Il suffit de mettre la variable « -2passvar » à la valeur « yes ».

13) Qu'es-ce que le « force-alignment » ?

Ce procédé consiste à prendre une transcription et trouver parmi les différentes prononciations que peut avoir un mot (apparaissant dans la transcription) celui qui convient le mieux.

14) Quel est le rôle de <s> et </s> ?

Ce sont des délimiteurs où tout ce qui est en dehors n'est pas décodé. Ils fournissent également un contexte au modèle de langage et marquent une différence avec les anciennes versions de Sphinx, où pour délimiter chaque énoncé, il était nécessaire de marquer un silence.

Écriture d'un module pour Asterisk 1.2

Ce tutorial (en cours d'écriture) a pour objectif d'expliquer l'écriture de modules pour Asterisk 1.2.

En effet, la documentation étant beaucoup trop pauvre, je souhaite grâce à ce document apporter ma petite contribution au monde du logiciel libre en espérant désambiguïser certaines notions propres à la programmation d'applications pour le PABX Asterisk.

Ce tutorial s'adresse aux personnes qui ont déjà une connaissance non seulement d'Asterisk et de son jargon mais également de l'environnement Linux et de la programmation en C.

Après la mise en place de l'environnement nécessaire à l'écriture de modules Asterisk, nous procéderons à l'écriture d'une petite application qui consistera simplement à afficher le traditionnel « HelloWorld ».

Nous apporterons ensuite quelques petites astuces pour améliorer l'utilisation et l'écriture de tels modules.

Nous soulignons qu'un prochain tutorial s'attachera à l'explication d'une application un peu plus complexe qui interagit entre autre avec les flux vocaux, mais qui détaille aussi la façon d'écrire un autre type de module, à savoir les fonctions, qui sont utilisables directement dans les dialplans.

Récupération des sources Asterisk

Afin de pouvoir apporter des modifications à Asterisk, il est nécessaire d'avoir son code source. Si vous ne les possédez pas, téléchargez les sources zippées de la dernière version stable sur le site suivant <http://www.asterisk.org/>. Ensuite, extrayez les sources dans le répertoire `/usr/src` de votre système d'exploitation Linux. Si vous avez déjà installé Asterisk en utilisant un package tel que RPM, veuillez désinstaller cette version avant de continuer (notez que si vous voulez utiliser les drivers Zaptel et LibPRI, il faudra d'abord les installer avant de procéder à l'installation d'Asterisk).

Je recommande vivement de créer un lien symbolique nommé « asterisk » dans votre répertoire `/usr/src` vers les sources Asterisk que vous utiliserez. Cela facilite la navigation et réduit d'éventuelles complications avec de nouvelles versions.

```
[root@localhost src]# pwd
/usr/src
[root@localhost src]# ln -s asterisk-1.2.4 asterisk
[root@localhost src]# ls -l asterisk
lrwxrwxrwx 1 root root 14 Jun  3 12:39 asterisk -> asterisk-1.2.4
```

Pour compiler et installer Asterisk, tapez les commandes suivantes en tant que root. Notez qu'il vaut mieux au préalable vérifier que le répertoire `/usr/lib/asterisk/modules` est vide avant d'installer Asterisk afin d'éviter tout problème.

```
[root@localhost asterisk]# pwd
/usr/src/asterisk
[root@localhost asterisk]# yes | rm -rf /usr/lib/asterisk/modules/*
[root@localhost asterisk]# make && make install
```

Si vous jetez un oeil dans le fichier Makefile, vous verrez qu'il est possible de lancer des make avec certaines options, ce qui aura pour effet d'installer certains outils intéressants.

- **make upgrade** : Revient à faire un « make && make install » en sautant quelques étapes cependant, tels que l'installation des fichiers sons pour améliorer la rapidité.
- **make mpg123** : Installera une version de mpg123 sur votre système qui a été testé et déclaré compatible avec des applications Asterisk telle que MP3Player().
- **make tags** : utilise le programme ctags pour générer un fichier tags permettant à emacs de naviguer parmi les fichiers sources.
- **make samples** : copie tout ce qui est contenu dans *configs/* vers */etc/asterisk*. La plupart du temps les tutoriaux relatifs à Asterisk, invitent à taper cette commande après l'installation pour avoir déjà des fichiers de configurations SIP et des fichiers d'extensions (qu'il reste ensuite à modifier).
- **make rpm** : crée un fichier RPM.

Les répertoires suivants sont assez importants et méritent que l'on s'y attarde.

- **apps/** répertoire où sont stockés tous les modules Asterisk
- **channels/** répertoire contenant les fichiers sources définissant les protocoles utilisés par les channels (canaux de communication) tels que SIP ou IAX2.
- **codecs/** c'est à cet endroit que se trouvent les sources de codecs tels que μ -law et GSM. Les codecs sont les algorithmes de d'encodage et de compression utilisés par les protocoles du channel (que le jargon Asterisk appel « switch »).
- **formats/** les formats sont très similaires aux codecs sauf qu'ils sont utilisés pour lire des fichiers audio.
- **funcs/** répertoire où sont placées les fonctions utilisables par les dialplans.
- **res/** utilisé pour contenir les modules fournissant des fonctionnalités (exemple : music on hold).
- **pbx/** idem que res/ (exemple : configuration du dialplan).

Ecrire son propre module

Utiliser les fonctions du noyau d'Asterisk ouvre de nouvelles perspectives et permet d'adapter Asterisk à ses propres besoins. A travers un module on peut ainsi par exemple lire un fichier de configuration contenu dans le répertoire */etc/asterisk* lorsque le module est chargé, mettre en cache les données que l'on juge être fréquemment utilisées, ou même faire du traitement sur les flux vocaux d'une certaine conversation téléphonique.

Nous noterons qu'il existe différents type de modules (applications, fonctions, codecs, etc.) qui suivent néanmoins le même format. Les différences se font au niveau des Makefiles et du code que l'on écrit. Il est ainsi tout a fait possible (et idiot) d'écrire un module de type application dans le répertoire *funcs/*.

La suite de ce tutorial se focalisera sur l'écriture d'une application.

a) Explications sur les différentes fonctions de base

Les modules sont compilés en des fichiers portant l'extension *.so* (shared object). Ils peuvent être placés dans le répertoire */usr/lib/asterisk/modules* et être activés ou désactivés lors du chargement en éditant le fichier */etc/asterisk/modules.conf*.

Tous les modules doivent inclure l'entête *asterisk/modules.h* mais également définir tout un panel de fonctions. Les fonctions qui suivent retournent généralement une valeur différente de 0 en cas d'erreur, ou 0 si tout se passe bien.

Il ne faut en aucun cas définir ces fonctions en « static ».

int load_module(void)

Cette fonction est appelée lorsque le module est chargé par le noyau Asterisk. Dans cette fonction, vous devez appeler des fonctions telles que *ast_custom_function_register()* pour permettre à Asterisk de savoir exactement les services proposés par votre module.

Retourne 0 si tout s'est bien exécuté, et une valeur différente de 0 en cas d'échec.

int unload_module(void)

Cette fonction sera appelée quand Asterisk sera prêt à « décharger » votre module. Dans cette fonction il est préférable de libérer tous les services utilisés et envoyer un signal de raccrochage à tous les channels utilisant votre module.

Retourne 0 si tout s'est bien exécuté, et une valeur différente de 0 en cas d'échec.

Un échec indiquera à Asterisk qu'il n'est pas possible de décharger votre module pour le moment.

int usecount(void)

Retourne le nombre de threads, channels, etc. qui sont entrain d'utiliser votre module. Ceci peut être vu à partir de l'interpréteur de commandes d'Asterisk (CLI) en tapant par exemple *show modules*. Il est de votre responsabilité de gérer ce nombre et de raccrocher tout ce qui peut utiliser votre module jusque ce qu'il soit déchargé. Nous noterons qu'Asterisk fournit des macros facilitant la tâche.

Dans certains cas, comme lorsque le module fournit un service de courte durée, vous n'avez pas à vous préoccuper du nombre d'utilisateurs de votre module. Par exemple, *pbx_functions.c* retourne toujours 0 quand *usecount()* est appelée.

char *description(void)

Retourne une courte description de ce que fait le module. Cette description sera visible en tapant dans le CLI la commande *show modules*.

int reload(void)

Cette fonction est optionnelle et sera appelée que si l'on tape dans une console (CLI) une commande comme *reload mon_module.so* ou *reload*. Cela vous permettra de recharger votre module et remettre en mémoire cache des données externes (comme un fichier de configuration).

Cette fonction constitue une bonne alternative au chargement/déchargement d'un module, car on n'a pas besoin de raccrocher tous les channels qui utilisent le module.

Retourne 0 si tout s'est bien exécuté, et une valeur différente de 0 en cas d'échec.

char *key(void)

Fonction qui est censée retourner la valeur de `ASTERISK_GPL_KEY` qui vous oblige à confirmer le fait que votre extension est sous licence GNU. Ainsi, si vous utilisez une édition gratuite d'Asterisk et que votre module ne retourne pas la valeur de `ASTERISK_GPL_KEY`, alors il ne sera pas chargé.

Cela permet de prévenir l'utilisation dans la version gratuite d'Asterisk, de modules écrits uniquement pour la version « Business Edition ».

b) Ecriture de l'application HelloWorld

Les applications doivent être appelées « *app_NOM.c* » où *NOM* sera le nom de votre application.

Il faut ensuite placer le fichier créé dans le répertoire *apps/* avec les autres sources des applications Asterisk.

Enfin, il faut également « *app_NOM.so* » à la liste d'applications contenue dans *apps/Makefile*.

Dans l'exemple qui suit nous appellerons notre application « *app_hello* ».

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "asterisk.h"
#include "asterisk/module.h"
#include "asterisk/pbx.h"
#include "asterisk/channel.h"
#include "asterisk/logger.h"
#include "asterisk/options.h"

static char *tdesc = "Petit exemple d'application Hello World";

static char *app = "Hello";

static char *synopsis = "Ecrit Hello World à l'écran";

static char *descrip =
" Hello(): Ne prend aucun argument, écrit seulement\n"
"         Hello World à l'écran.\n";

/* macros utiles contenues dans module.h pour compter le nombre
 * d'utilisateurs du module */
STANDARD_LOCAL_USER;
LOCAL_USER_DECL;

static int hello_exec(struct ast_channel *chan, void *data)
{
    struct localuser *u;

    /* Ajoute ce channel à la liste chaînée des utilisateurs.
     * Par conséquent, si nous avons soudainement besoin de
     * décharger notre module, nous savons qui raccrocher
     * proprement. Le contenu de cette macro référence aussi bien
     * la variable chan que les variables définies par
     * LOCAL_USER_DECL */
    LOCAL_USER_ADD(u);

    /* il faut que vous ayez lancé Asterisk avec au moins 2 'v'
     * pour voir ce message. Vous pouvez cependant toujours
     * ajuster la verbosité en tapant la commande
     * 'set verbosity x' */
    if (option_verbose > 2)
        ast_verbose(VERBOSE_PREFIX_3 "Hello World!\n");

    /* Nous n'allons plus avoir besoin de gérer ce channel */
    LOCAL_USER_REMOVE(u);

    return 0;
}

int load_module(void)
{
    int res;
    res = ast_register_application(app, hello_exec, synopsis, descrip);

    return res;
}

```

```

int unload_module(void)
{
    int res;
    res = ast_unregister_application(app);

    /* S'il y a toujours des channels utilisant ce module, leur
     * envoyer un signal de raccrochage logiciel*/
    STANDARD_HANGUP_LOCALUSERS;

    return res;
}

char *description(void)
{
    return tdesc;
}

int usecount(void)
{
    int res;
    STANDARD_USECOUNT(res);

    return res;
}

char *key(void)
{
    return ASTERISK_GPL_KEY;
}

```

Editer le fichier d'extensions et apporter à l'utilisateur de votre choix une extension du même style que l'exemple qui suit.

/etc/asterisk/extensions.conf

```

exten => 888,1,Answer
exten => 888,n,Hello()
exten => 888,n,Hangup

```

Lorsque l'utilisateur appellera le numéro 888, vous pourrez voir dans le CLI quelque chose qui ressemble à cela :

```

-- Executing Answer("SIP/105-5b14", "") in new stack
-- Executing Hello("SIP/105-5b14", "") in new stack
-- Hello World!
-- Executing Hangup("SIP/105-5b14", "") in new stack

```

De plus, toujours à partir du CLI, mais en tapant les commandes qui suivent, vous obtiendrez les informations suivantes :

```
*CLI> show application Hello

  -= Info about application 'Hello' -=

[Synopsis]
Ecrit Hello World à l'écran

[Description]
Hello(): Ne prend aucun argument, écrit seulement
         Hello World à l'écran.
```

Bien que cet exemple d'application ne constitue pas un module réellement utilisable, il permet de poser les bases par rapport à la compréhension de l'écriture d'applications en utilisant les fonctions du noyau Asterisk.

Dans la section qui suit, nous apporterons quelques informations supplémentaires assez utiles qui faciliteront l'écriture et l'utilisation de vos modules.

Accélérer le développement

Jusqu'à maintenant, vous avez probablement arrêté Asterisk, exécuté la commande *make upgrade* puis relancé Asterisk, à chaque fois que vous avez apporté une modification à votre module et que vous avez voulu tester les effets.

Cela peut s'avérer être une tâche assez fastidieuse...Heureusement ce n'est pas l'unique façon de faire.

Car les modules Asterisk sont des bibliothèques partagées, vous n'avez pas forcément besoin d'arrêter entièrement le serveur Asterisk pour recharger vos modifications. C'est un aspect assez important de l'architecture d'Asterisk car il y a de fortes chances pour que vous ne vouliez en aucun cas arrêter (aussi peu que ce soit) votre serveur téléphonique.

La console Asterisk (CLI) fournit des commandes pour charger dynamiquement des modules.

```
*CLI> load app_hello.so
Loaded /usr/lib/asterisk/modules/app_hello.so => (Petit exemple
d'application Hello World)
  -= Registered application 'Hello'
*CLI> unload app_hello.so
  -= Unregistered application 'Hello'

[root@localhost asterisk]# asterisk -r -x "load app_hello.so"
Loaded /usr/lib/asterisk/modules/app_hello.so => (Petit exemple
d'application Hello World)
  -= Registered application 'Hello'
[root@localhost asterisk]# asterisk -r -x "unload app_hello.so"
  -= Unregistered application 'Hello'
```

Ainsi une alternative à la fastidieuse étape de compilation susmentionnée, serait de décharger le module, taper la commande *make upgrade* pour charger à nouveau le module. Le seul problème avec ce procédé, c'est que les scripts de compilation d'Asterisk sont assez longs,

prennent un certain temps et sont toujours re-exécutés (donc même si aucun changement n'a été apporté, tout est malgré tout recompilé).

Fort heureusement, il existe une méthode bien plus rapide qui consiste à compiler uniquement votre module, le décharge, l'installe, et le recharge, le tout d'une seule traite.

Il existe ainsi un script écrit en Perl que l'on peut trouver dans le répertoire *contrib/scripts* et qui s'appelle « *astxs* ».

Copiez ce script dans */usr/bin* et donnez lui les droits d'exécutions avec la commande *chmod*.

Astxs peut seulement être exécuté à partir de la racine du répertoire contenant les sources Asterisk (à moins que vous ne vouliez définir un ensemble de variable manuellement).

Lancé à partir du répertoire d'Asterisk, ce script peut auto-détecter la plupart de vos configurations et compiler votre module correctement.

```
[root@localhost asterisk]# ps | grep asterisk
25729 pts/5      00:00:00 safe_asterisk
25750 pts/5      00:00:13 asterisk

[root@localhost asterisk]# pwd
/usr/src/asterisk

[root@localhost asterisk]# astxs -install -autoload apps/app_hello.c
gcc -I/usr/src/asterisk -I/usr/src/asterisk/include -pipe -Wall -Wstrict-
prototypes -Wmissing-prototypes -Wmissing-declarations -g3 -Iinclude -
I../include -D_REENTRANT -D_GNU_SOURCE -O6 -march=i686 -
DZAPTEL_OPTIMIZATIONS -fomit-frame-pointer -fPIC -
DUSE_ODBC_STORAGE -DEXTENDED_ODBC_STORAGE -c apps/app_hello.c -o
apps/app_hello.o
gcc -shared -Xlinker -x -o apps/app_hello.so  apps/app_hello.o
/usr/sbin/asterisk -rx 'unload app_hello.so'
  == Unregistered application 'Hello'
/bin/cp -p apps/app_hello.so /usr/lib/asterisk/modules
/usr/sbin/asterisk -rx 'load app_hello.so'
  Loaded /usr/lib/asterisk/modules/app_hello.so => (Petit exemple
d'application Hello World)
  == Registered application 'Hello'
```

Dans l'exemple du shell ci-dessus, Asterisk est en cours d'exécution et nous nous trouvons dans le répertoire du code source. En lançant *astxs*, l'option *-install* demande à *astxs* de compiler notre module. L'option *-autoload* va faire en sorte qu'*astxs* charge notre nouvelle version du module dans Asterisk.

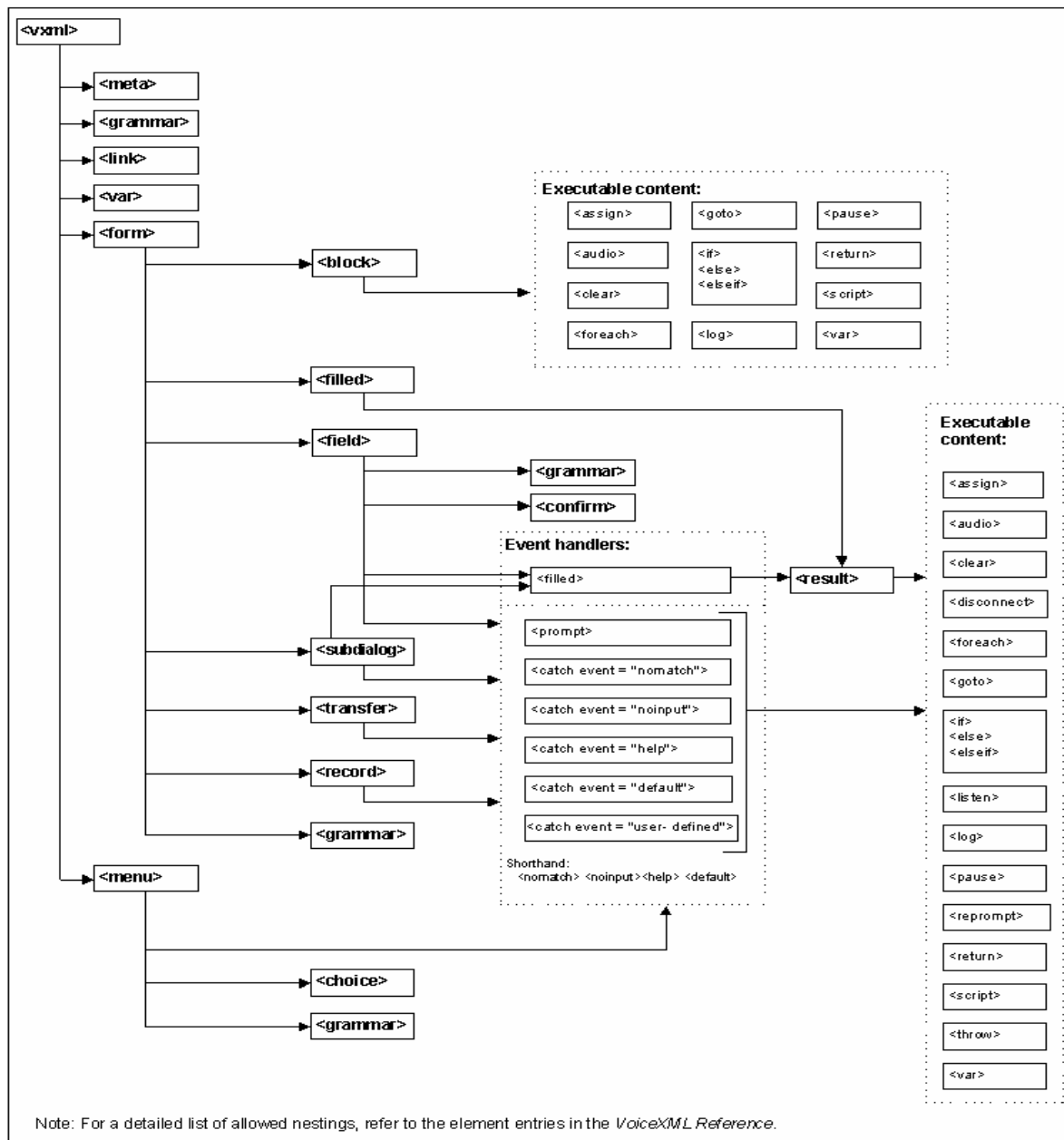
Ici s'achève la première partie de ce tutorial qui je l'espère a su d'une part rassembler les notions de base concernant l'écriture de modules, et d'autre part vous a donné l'envie de contribuer à l'amélioration de ce fabuleux PABX qu'est Asterisk ;-)

Eléments du langage VoiceXML

Élément	Objectif
<i>assign</i>	Assigne une valeur à une variable
<i>audio</i>	Lit un fichier son au sein d'un élément <i>prompt</i>
<i>block</i>	Un conteneur pour un code exécutable (non interactif)
<i>catch</i>	Capture un événement
<i>choice</i>	Définit un élément de menu
<i>clear</i>	Efface une ou plusieurs variables d'élément de formulaire
<i>disconnect</i>	Déconnecte une session
<i>else</i>	Employé dans les éléments <i>if</i>
<i>elseif</i>	Employé dans les éléments <i>if</i>
<i>enumerate</i>	Raccourci pour l'énumération des choix dans un menu
<i>error</i>	Capture un événement erreur
<i>exit</i>	Sort d'une session
<i>field</i>	Déclare un champ de saisie dans un formulaire
<i>filled</i>	Une action exécutée quand les champs sont remplis
<i>form</i>	Un dialogue pour la présentation d'informations et la collecte de données
<i>goto</i>	Aller à un autre dialogue dans le même document ou un document différent
<i>grammar</i>	Indique une grammaire de reconnaissance vocale ou une grammaire DTMF
<i>help</i>	Capture un événement aide
<i>if</i>	Logique conditionnelle simple
<i>initial</i>	Déclare une logique initiale sur une entrée dans un formulaire (à initiative mixte)
<i>link</i>	Définit une transition commune à tous les dialogues dans la portée du lien
<i>log</i>	Génère un message de débogage
<i>menu</i>	Un dialogue pour choisir entre plusieurs destinations
<i>meta</i>	Définit un élément de métadonnée en tant que couple nom/valeur
<i>metadata</i>	Définit une métainformation en utilisant un schéma de métadonnée
<i>noinput</i>	Capture un événement non-entrée
<i>nomatch</i>	Capture un événement non-correspondance
<i>object</i>	Interagit avec une extension personnalisée
<i>option</i>	Indique une option dans un élément <i>field</i>
<i>param</i>	Paramètre dans un élément <i>object</i> ou <i>subdialog</i>
<i>prompt</i>	Place en file d'attente la synthèse vocale et la sortie audio vers l'utilisateur
<i>property</i>	Contrôle les paramètres de la plateforme d'implémentation.
<i>record</i>	Enregistre un échantillon audio
<i>reprompt</i>	Joue la file d'attente sur un champ lorsque celui-ci est revisité après un événement
<i>return</i>	Retour d'un sous-dialogue.

Élément	Objectif
<i>script</i>	Définit un bloc de logique de script ECMAScript côté client
<i>subdialog</i>	Invoque un dialogue en tant que sous-dialogue du dialogue courant
<i>submit</i>	Soumet des valeurs à un serveur de documents
<i>throw</i>	Suscite un événement.
<i>transfer</i>	Transfère l'appelant vers une autre destination
<i>value</i>	Insère la valeur d'une expression dans une <i>invite</i>
<i>var</i>	Déclare une variable
<i>record</i>	L'élément de niveau supérieur dans chaque document VoiceXML

Hierarchie d'un fichier VoiceXML



Bibliographie / Références

Titre : A tutorial on hidden Markov models and selected applications in speech recognition
Auteur : Rabiner
ISSN : 0018-9219

Titre : Asterisk
Auteurs : Jared Smith, Jim Van Meggelen, Leif Madsen
ISBN : 2841773949

Spécification du langage VoiceXML
<http://www.w3.org/TR/voicexml20/>

Cours du MIT sur la reconnaissance vocale
<http://ocw.mit.edu/OcwWeb/Electrical-Engineering-and-Computer-Science/6-345Automatic-Speech-RecognitionSpring2003/LectureNotes/index.htm>

Modèles de Markov Cachés
<http://www.essi.fr/~leroux/parolehmm/parolehmm.html>

Glossaire

Coarticulation

Selon les phonèmes qui l'entourent dans une phrase, un phonème n'est pas prononcé de la même manière. Ce phénomène n'est pas audible, ni le locuteur ni l'auditeur ne s'en rendent compte, néanmoins la coarticulation est primordiale à l'intelligibilité de la phrase

Corpus

Terme utilisé en linguistique. Un corpus est un ensemble d'informations (textes) qui est utilisé comme première approximation d'une description linguistique ou pour vérifier des hypothèses à propos du langage.

CTI

Ensemble des techniques permettant la mise en oeuvre d'applications reposant sur un interfonctionnement d'applicatifs informatiques et d'applicatifs téléphoniques

Diphone

C'est la liaison de deux phonèmes voisins.

Voici par exemple, les 5 diphones du mot 'salut' : _s + sa + al + lu + u_ (où _ est le silence). Le nombre de diphones possibles avec 38 phonèmes est 38^2 .

DTMF

Dual Tone Multi Frequency. Technique consistant à utiliser deux séries de tons, une pour les rangées, une pour les colonnes, pour identifier les touches d'un clavier de téléphone. Quand on presse une touche, deux tons sont donc joués en même temps.

FFT

La transformation de Fourier consiste à décomposer un signal périodique quelconque en une somme de signaux sinusoïdaux de différentes amplitudes et déphasages. Cette transformation permet de réaliser un spectre en amplitude du signal et de procéder à des filtrages de ce signal. La transformation rapide de Fourier (FFT) est un procédé mathématique simplifié qui permet dans certaines conditions de faire cette transformation rapidement (comme son nom l'indique).

Formant (F1, F2, F3, F4)

Les cordes vocales fabriquent une énergie sonore. Celle-ci est résonnée et filtrée par des cavités de résonance: les cavités nasale, buccale, labiale et pharyngale. Chaque cavité à une fréquence de résonance propre. La phonation, permet d'amplifier certaines fréquences. Ces fréquences amplifiées s'appellent les formants. Les formants permettent de distinguer des voyelles ayant la même fréquence fondamentale, la même amplitude et la même durée. Les formants (abréviation F1, F2, F3, F4) permettent d'identifier le timbre des sons.

Fondamentale (ou F0)

La fréquence fondamentale (abréviation "Fo" ou "F0") est la fréquence (composante spectrale) la plus grave (basse) d'un son. Cette fréquence fixe la hauteur des sons. Un son grave se caractérise par une fréquence basse. Un son aigu se caractérise par une fréquence élevée. La fréquence fondamentale est générée par les vibrations laryngées. La fréquence fondamentale correspond à l'harmonique zéro, qui est l'harmonique le plus grave.

H323

Le protocole H.323 regroupe un ensemble de protocoles de communication de la voix, de l'image et de données sur IP. C'est un protocole développé par l'UIT-T. Il est dérivé du protocole H.320 utilisé sur RNIS.

IVR

C'est un système de réponse automatique personnalisable proposant à l'appelant une liste de services. L'IVR (Interactive Voice Response) est une technologie permettant une interaction entre un téléphone et une base de données afin d'obtenir des informations ou de générer des actions en pressant des touches sur le téléphone.

Ce système peut être utilisé par exemple pour saisir un code PIN ou le numéro de téléphone d'un correspondant.

Phone

Unité servant à comparer la puissance des sons de fréquences différentes du point de vue de l'impression ressentie.

Phonème

Un phonème est un son.

Exemples: le **en** de **vendredi** ou le **t** de **trottoir**.

Prosodie

Partie de la phonétique qui étudie l'intonation, l'accentuation, les tons, le rythme, les pauses, la durée des phonèmes.

SIP

C'est un protocole utilisé en voix sur IP permettant de transférer de la voix, de la vidéo ou des données à travers un réseau. SIP est le sigle de Session Initiation Protocol. C'est un protocole IETF décrit par le document rfc3261. SIP est un remplacement de H323.

Le protocole SIP se charge de l'authentification et la localisation des multiples participants, et aussi de la négociation sur les types de média utilisables par les différents participants.

Softphone

C'est un logiciel que l'on utilise pour faire de la téléphonie sur Internet depuis son ordinateur. Il existe un très grand nombre de softphones dont l'un des plus célèbres est X-Lite proposé par X-ten.

RTP

Real-time Transport Protocol. Protocole de transport en temps réel, que certains voudraient voir utiliser sur l'Internet. Il permet par exemple d'utiliser le multicast, mais ce n'est pas un protocole fiable.