FLORENT XAVIER

SYNTHESE VOCALE

INTEGRATION DU FRANÇAIS AU SYSTEME MARY TEXT-TO-SPEECH

DIRIGÉ PAR

Prof. Catherine Pelachaud *(DR CNRS –LTCI Telecom Paristech)*
Dr. Gérard Chollet *(DR CNRS-LTCI Telecom Paristech)*

EN COLLABORATION AVEC

Dr. Sotiris Manitsaris *(ESPCI Paristech)*
Thodoris Mironidis *(Université de Macédoine, Grèce)*

M2 ATIAM
2010 – 2011

REMERCIEMENTS

Mes remerciements vont tout d'abord à Catherine Pelachaud, pour son accueil, sa confiance et son dévouement. Je remercie également très chaleureusement Gérard Chollet, pour son expertise, ses conseils et son encadrement.

Un grand merci à Sotiris Manitsaris, Thodoris Mironidis et Guillaume Galou pour leur aide très précieuse et leur implication dans mon stage. Merci aussi à Frédéric Bechet, concepteur de LIA_phon, pour sa contribution, ainsi qu'à Jun Cai, Pierre Lanchantin et Xavier Rodet pour leur collaboration.

Enfin, merci à toute l'équipe du DFKI et les développeurs de Mary Text-To-Speech, Sathish Pammi, Ingmar Steiner, Nickolay Schmyrev, Marcela Charfuelan et Hugues Sasse. Un merci tout particulier à Marc Schröder sans qui ce stage n'aurait pas eu lieu d'être, et dont la disponibilité, l'amabilité et les compétences furent pour moi une aide inestimable.

TABLE DES MATIERES

INTRODUCTION

La parole est un atout que seul nous, êtres humains, possédons dans tout le monde animal. La générer naturellement résulte d'une combinaison complexe de phénomènes physiques et d'interprétations psychoacoustiques. Cependant, de tous temps, l'homme a toujours voulu comprendre, expliquer ses phénomènes, mais également les générer artificiellement. D'après la légende, si d'une légende il s'agit bien, le pape Sylvestre II (~950 – 1003) aurait lui même conçu une « tête de bronze parlante » répondant par oui ou par non à ses questions [1][2]. Cette machine aurait été similaire à un automate binaire et *aurait été façonnée « sous une certaine conjonction des étoiles qui se place exactement au moment où toutes les planètes sont en train de commencer leur course»* [2]. Les premiers travaux significatifs sur la génération de la parole remontent en 1791, avant la de plus venue avec la reconnaissance vocale est écrié ont offert aussi démarrer découverte du signal électrique, avec Wolfgang von Kempelen (1734 – 1804) et sa machine de synthèse vocale mécanique [1][3]. Celle-ci était basée sur des modèles de la langue et des lèvres et était capable de produire des voyelles et consonnes [1]. Mais c'est surtout en 1939 et l'invention du VODER par Homer Dudley qui fera franchir à cette discipline un bond formidable en avant [1][4]. Désormais, il est plus juste de parler de synthèse vocale, faite à partir de modèles acoustiques du signal de parole, aussi appelée synthèse formantique. De nombreuses autres techniques de synthèse plus perfectionnées verront le jour, améliorant l'intelligibilité du discours synthétisé.

La synthèse vocale à partir du texte (Text-To-Speech synthesis) permet de convertir un texte donné en un signal audio de parole. Ainsi peut-on imaginer de multiples usages pour cette technologies, tous plus utiles les uns que les autres. Il ne s'agit pas ici de remplacer l'homme, mais de le décharger de tâches ingrates et contraignantes. Prenons par exemple un annuaire téléphonique comprenant des milliers – voir des millions – d'entrées. Il n'est pas envisageable pour un locuteur d'enregistrer autant de numéros. De même peut-on prendre comme exemple l'intégration de tel système aux agents conversationnels, comme c'est le cas pour le projet GRETA [5], dirigé par Catherine Pelachaud, avec le système TTS *Acapela*. Mais une telle technologie peut également accompagner l'homme dans son quotidien, le compléter. Et c'est en ce sens que la synthèse vocale prend toutes ces lettres de noblesse. Peu d'inventions peuvent en effet se vanter d'avoir le pouvoir de rendre la parole à l'homme muet. Une des premières personnalités à se servir de système TTS pour communiquer est sans doute le scientifique Stephen Hawking [6]. Le projet REVOIX dirigé par Bruce Denby poursuit dans cette voix en proposant un système de synthèse vocale se basant en premier lieu sur les mouvements linguaux et labiaux pour générer du texte et à partir de cela de la parole [7].

La synthèse vocale a très peu de progrès à faire en termes d'intelligibilité, la plupart des types de synthèses étant très fidèles aux situations réelles. En revanche, il est toujours possible de distinguer le vrai du faux et ce principalement à cause des modélisations des émotions traduites dans la parole (informations prosodiques) qui produisent des voix beaucoup trop monotones, et biaisées par le corpus initial. Cependant, ce stage souligne l'importance des systèmes TTS, et surtout de leurs applications et contributions au développement du genre humain, les projets GRETA et REVOIX à l'appui. Pour ce faire, une démocratisation des systèmes TTS doit être envisagée, ce qui n'est pas tout à fait le cas au regard du peu d'applications existantes. Aujourd'hui, il n'existe aucun système TTS libre et opensource perfectionné (synthèse basé sur des Modèles de Markov Cachés et par sélection d'unités) permettant de synthétiser du français. Les chercheurs doivent se rabattre sur des solutions commerciales, certes très performantes, mais onéreuses et dont le code n'est pas libre. Le but de ce stage est donc de fournir à la communauté scientifique, et en particulier aux projets GRETA et REVOIX, une voix de synthèse MMC (Modèles de Markov Cachés) libre de qualité et participer ainsi à l'essor de projets aussi ambitieux que nécessaires, que ce soit dans le domaine de la synthèse vocale ou non. Cela se fera par l'intégration du français au système Mary Text-To-Speech du DFKI, qui consiste principalement à la conception/intégration des modules de traitement du langage naturel. Pour valider le travail, nous créerons la voix française à l'aide d'un corpus de trois mille phrases phonétiquement équilibrée et des outils mis à disposition par le système.

Ce mémoire se déroulera comme s'en suit. Nous ferons l'état de l'art des différentes techniques de synthèse vocal avant de nous focaliser sur la synthèse MMC. Puis nous listerons et évaluerons quelques unes des solutions existantes les plus utilisées. Le système Mary sera décrit de manière détaillée puis la conception et l'intégration de la voix française au système seront exposées. Nous finirons par l'évaluation des résultats obtenus et la définition de nouveaux objectifs pour de futurs travaux sur le sujet.

Chapitre I

PRINCIPES DE LA SYNTHESE VOCALE ET ETAT DE L'ART

I.1 LA SYNTHESE TTS

I.1.1 PRINCIPES

La synthèse Text-To-Speech permet de créer un signal de parole à partir d'un texte donné. Quelque soit le type de synthèse (ce que nous étudierons par la suite), un système TTS comprend deux modules principaux : 1) un module de traitement de texte composé de fonctions qui permettront de décomposer le texte en un ensemble d'unités phonétique distinctes (phonèmes, diphones, position contextuelles du mot dans la phrase…). Certains systèmes perfectionnés intègrent un traitement prosodique, c'est-à-dire, l'intonation dans la voix (décrite dans la partie I.2). On appelle cet ensemble les modules TALN (Traitement Automatique du Langage Naturel) [8]. 2) un synthétiseur vocal, prenant comme entrée la sortie du module précédent, et en déduisant le signal audio de parole.

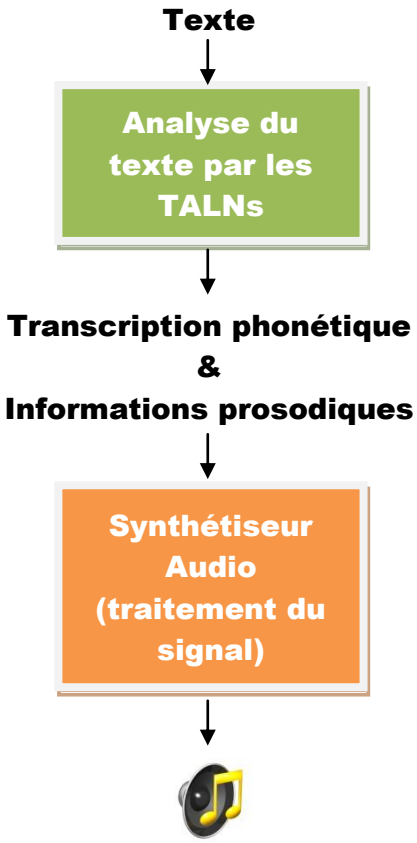


Fig I.1. *Vue d'ensemble d'un système TTS* [9]

I.1.1.1 NATURAL LANGUAGE PROCESSING COMPONENTS (NLP)

Les modules NLP (Natural Language Processing), ou TALN en français, ont pour objectifs de transcrire le texte donné en informations compréhensible par le synthétiseur vocal en vue de la génération du signal de parole. Ces dites informations, déterminés en plusieurs étapes, sont :

- La nature ou fonction du mot dans le texte, POS ou Part-Of-Speech (nom, verbe, complément etc.)

ex :
```
Je      suis        ton              père.
[Sujet][Verbe][Adjectif Possessif][Nom Masculin]
```

- la transcription phonétique de chaque mot

ex : transcription phonétique au format SAMPA [10] *(voir en Annexe An.1., le tableau des phonèmes)*
`Je suis ton père -> /Z@ sHi to~ pER/`

- Les informations prosodiques, l'intonation dans la voix (intensité, durée, hauteur)

I.1.1.1.1 Tokenizer

La première étape consiste à segmenter le texte. Il est primordial dans un premier temps de segmenter le texte donné en phrases, paragraphes, titres etc. En effet, le simple fait d'avoir des ponctuations à chaque fin de phrases n'est pas suffisant, les titres, par exemple, en étant dépourvue. Le résultat de la phonétisation change grandement en fonction de la qualité de la segmentation [11]. Ces entités sont ensuite découpées en unités distinctes, à savoir mots, voire mots composés. L'exemple du dessus est trivial.

`Je suis ton père.` donnera
```
Je
suis
ton
père
.
```

En revanche, pour ce qui est des cas plus atypiques, comme la phrase :

`Dans la coll. Dupond, l'ouvrage "Pourquoi moi ?" à pour code : IV.12.14` [11]

il n'est pas suffisant de segmenter selon les points et points d'exclamation. Ainsi, un ensemble de règles codant des heuristiques, couplé à un lexique de référence [11] doit pouvoir prendre en considération les abréviations, les citations, bref tout type d'expressions qui ne peuvent être déduites directement du texte en l'état.

I.1.1.1.2 Preprocessing

La phase de *preprocessing*, ou nettoyage du texte, permet de lever toutes ambiguïtés concernant les mots segmentés et d'en déduire une forme prononçable en vue de la phonétisation. Pour le français, plusieurs traitements sont appliqués pour obtenir un texte *phonétisable* : 1) une ré-accentuation des termes en majuscule [11] (ex. `PERE -> père`). Notons que certains mots en majuscule peuvent comprendre plusieurs ré-accentuations possible comme par exemple le mot `MARCHE`, qui peut à la fois être `marche` ou `marché`. 2) les abréviations (ex. `coll. -> collection`). 3) les chiffres arabes, romains, numéros de téléphone etc. (ex. `01.04.72.33.41 -> zéro un, zéro quatre, soixante-douze, trente-trois, quarante-et-un`). 4) les caractères spéciaux (ex. `# -> dièse`). 5) la date, l'heure (ex. `13h21 -> treize heures vingt-et-un`). 6) les unités de mesure, les devises monétaires etc. (ex. `34€ le Kg -> trente-quatre euros le kilogramme`). Cette phase de nettoyage est un témoin de la versatilité du système TTS, et sa faculté à s'adapter à des cas complexes, comme par exemple la phrase :

```
Dans la coll. Dupond, l'ouvrage "Pourquoi moi ?" à pour code : IV.12.14
```

Qui donnerait donc

```
Dans la collection Dupond l'ouvrage pourquoi moi à pour code quatre douze quatorze
```

I.1.1.1.3 Part-Of-Speech tagger

L'étiquetage morphosyntaxique (*part-of-speech tagging*) est une phase dans laquelle chaque segment nettoyé se voit attribué une position contextuelle dans la phrase associée. Cette étape est nécessaire à la phonétisation car elle relève certaines ambiguïtés liées à cette position.
Voici un exemple significatif dans lequel la phonétisation est une conséquence directe de l'étiquetage :

```
Mon fils dénude les fils de cuivre.
```

Il n'y a aucune différence entre `fils` (l'enfant) et `fils` (de cuivre) du point de vue syntaxique alors que les transcriptions phonétiques diffèrent. On parle alors d'homographes hétérophones (autre exemple, le mot `est`, désignant à la fois le verbe être à la troisième personne du singulier, ou le point cardinal). Cependant, l'un est un nom masculin singulier, l'autre un nom masculin pluriel. Des règles peuvent être établies afin de distinguer de tels cas, par exemple en testant le mot voisin (analyse contextuelle). Ainsi `fils` s'il est suivie par `laine, coton, fer, cuivre etc.` sera perçu comme le nom masculin pluriel. L'étiquetage ne concerne pas simplement la position du mot dans la phrase, mais bien la position contextuelle de celui-ci. Un autre exemple significatif est :

```
Le président et le ministre président l'assemblée.
```

Aucune différence syntaxique entre `président` (sujet) et `président` (verbe), pourtant, c'est bien leurs positions respectifs qui modifient leur prononciations. Afin de distinguer le sujet du verbe, une analyse morphologique [12] peut être envisagée, c'est-à-dire une décomposition du mot en racine+préfixe+suffixe [12]. L'un des procédés les plus courants pour venir à bout de ces problèmes est d'établir un dictionnaire de racines morphologiques et des règles de décomposition [12]. Certains mots, en plus d'être homographes hétérophones, possèdent plusieurs transcriptions phonétiques selon que le mot suivant débute ou non par une voyelle. C'est le cas du mot `plus` *(voir en annexe, les différentes transcriptions phonétique de* `plus` *selon le contexte, An.3.)*. C'est également le cas de certains chiffres.

ex : `cinq` -> `/se~k/`
`cinq mille` -> `/se~k mil/` ou **`/se~ mil/`**
Une caractéristique particulière du français est la gestion des liaisons. Frédéric Béchet écrit:

> "La liaison est la survivance de quelques enchaînements de consonnes finales en ancien français : autrefois, toutes les consonnes finales étaient prononcées ; en français moderne, un certain nombre de ces consonnes sont muettes dans les mots pris isolément mais on les prononce lorsque le lien entre un mot (à finale consonantique) et le mot suivant (à l'initiale vocalique) est assez fort pour que se conserve l'enchaînement ancien. Toute la difficulté du traitement des liaisons résulte précisément de l'appréciation de la force de ce lien. " [11]

Ainsi pour exemple, dans la phrase :

```
Je suis.
```

Le « s » de `suis` n'est pas prononcé. Idem pour la phrase suivante :

```
Je suis ton père.
```

En revanche, pour celle-ci :

`Je suis un pirate.`

Le « s » est prononcé (`Je suis un pirate -> /Z@ **sHi z9~** piRat/`). De ce fait, la transcription phonétique seul des mots `suis` et `un` n'est plus suffisante. Il suffit dans la grande majorité des cas, de phonétiser la dernière lettre selon que le mot suivant débute par une voyelle ou non, mais de l'ajouter à la transcription phonétique du mot suivant (/z9~/). Certaines liaisons sont obligatoires (Etats-Unis -> `/eta zyni/`, et non pas `/eta yni/`), d'autres en revanche sont facultatives (prend un thé). Il existe également des liaisons interdites (`et eux`, le t de `et` ne doit jamais se prononcer) [13]. Certains phénomènes découlent directement de la liaison comme la dénasalisation (`bon ami` -> `/bO nami/` et non `/bo~ nami/`) et l'ouverture de certaines voyelles finales (`léger accroc` -> `/leZE rakRo/` et non `/leZe rakRo/`) [14].

Une seconde caractéristique propre au français est le e muet, e caduc, ou encore schwa. Ce phonème est un problème dans le sens où il peut disparaître du discours sans pour autant modifier le sens du mot [15]. Nous sommes capables de reconnaître de dernier, que l'on prononce ou non le schwa, c'est-à-dire sous toutes les formes acoustiques dudit mot. Mais une gestion de schwa est nécessaire au réalisme de la voix de synthèse. Effectivement, hormis dans le cas d'un discours très soutenue ou encore dans un contexte musical, nombreux sont les e à ne pas être prononcés dans le langage de tous les jours.

ex : `Je ne sais pas.`
`/Z@ nse pa/` plutôt que `/Z@ n@ se pa/`

On remarque aussi une différence de prononciation selon les individus (`/@/` ou `/3/`) mais également une différence dans le maintien du phonème selon les zones géographiques et donc les accents (par exemple dans le sud de la France)[15].

I.1.1.1.4 Phonemiser

Le module de phonétisation prend en entrée les segments nettoyés et étiquetés et en déduit la transcription phonétique. Le plus souvent, les entités (généralement des mots, chiffres, sigles etc.) ainsi que leur transcription sont contenus dans un lexique. Pour chaque segment, on regarde les entrées de ce lexique, et on lit simplement la transcription. Remarquons que pour la langue française, la phonétisation comprend des difficultés qui rendent la seule lecture de lexique insuffisante en amont de la segmentation. Par conséquent la fidélité du texte phonétisé est en très grande partie due à l'étiquetage syntaxique qui relève les problèmes de liaisons, d'élision du schwa et des homographes hétérophones.

Si le mot donné n'est pas présent dans le lexique, on applique des règles de transcription de lettres (letter-to-sound rules). Ces règles sont pour la plupart du temps définies selon un lexique de mots et leurs transcriptions respectives. Selon ce corpus, pour chaque lettre, un nombre fini de phonèmes lui est associé. Cette lettre et les phonèmes de cet ensemble forment des pairs dont la probabilité est calculée [16]. Puis, selon le texte à phonétiser, un arbre de décision [17] (algorithme CART[17] [18]) est construit afin de prédire quel phonème correspond le mieux, en prenant le contexte dans lequel il se situe[16]. Dans certains langages, l'usage de règles peut être suffisant et avantageux par rapport au coût de calcul peut représenter un lexique de transcriptions. C'est par exemple le cas pour l'espagnol, ne comptant que 50 règles, contre 300 pour le français [12].

En sortie des modules TALN nous avons donc une suite de segment phonétisé. Par exemple ayant pour entrée le texte suivant, voici la chaîne de traitement et ces sorties :

Dans la coll. Dupond, l'ouvrage "Pourquoi moi ?" à pour code : IV.12.14

Segmentation	Nettoyage	Etiquetage	Phonétisation
Dans	dans	dans [Préposition]	da~ [Préposition]
la	la	la [Déterminant Féminin Singulier]	la [Déterminant Féminin Singulier]
coll.	collection	collection [Nom Féminin Singulier]	k0lEksjo~ [Nom Féminin Singulier]
Dupond	Dupond	dupond [Nom Propre]	dypo~ [Nom Propre]
l'	l'	l [Déterminant Masculin Singulier]	l [Déterminant Masculin Singulier]
ouvrage	ouvrage	ouvrage [Nom Masculin Singulier]	uvRaZ [Nom Masculin Singulier]
Pourquoi	pourquoi	pourquoi [Adverbe]	puRkwa [Adverbe]
Moi	moi	moi [Pronom Personnel]	mwa [Pronom Personnel]
?	?	? [Point d'interrogation]	_ [Silence]
à	à	a [Préposition]	a [Préposition]
pour	pour	pour [Préposition]	puR [Préposition]
code	code	code [Nom Masculin Singulier]	kOd [Nom Masculin Singulier]
IV	quatre	quatre [Chiffre]	katR@ [Chiffre]
12	douze	douze [Chiffre]	duz [Chiffre]
14	quatorze	quatorze [Chiffre]	katORz@ [Chiffre]

Fig I.2. Sorties d'un module de TALN

Bien évidemment, les formats de sortie varient selon les systèmes et leurs architectures. Un standard reconnu (souvent indépendants desdits systèmes) est un langage à balise appelé SSML (Speech Synthesis Markup Language) [19] ou encore une de ses extensions le W3C SSML [20] (voir en Annexe, un exemple d'une sortie d'un texte phonétisé au format W3C SSML, An.2.). De plus, nous avons présenté ici le format de transcription phonétique le plus reconnu en synthèse TTS, à savoir le format SAMPA [10]. Il existe bien sur de nombreux autres formats (XSAMPA [21], API (Alphabet Phonétique International) [22] etc.).
Nous omettons volontairement le traitement prosodique, faisant pourtant partie des modules TALN, afin de le développer de manière détaillé dans la section I.2.

I.1.1.2 DIGITAL SPEECH PROCESSING (DSP)

Les informations phonétiques et prosodiques composent les données d'entrée du synthétiseur vocal. A partir de ceux-ci sont déduits des paramètres acoustiques selon des méthodes qui varient grandement de la nature même du type de synthèse. Nous allons voir dans la section suivante ces principaux types.

I.1.2 LES DIFFERENTS TYPES DE SYNTHESES

Il serait plus juste de nommer cette section « stratégies de synthèse ». En effet il ne s'agit pas de synthèse de signaux à proprement parlé, mais plutôt de la manière dont les paramètres acoustiques vont être déduits des transcriptions phonétiques et informations prosodiques. Les signaux sont ensuite synthétisés selon différents procédés, le plus souvent indépendamment du type de stratégie définie (synthèse par prédiction linéaire, famille PSOLA etc.). Les recherches se sont tout d'abord penchées sur les phénomènes acoustiques de la voix et leurs modélisations possibles (ex. synthèse formantique). En effet, les mémoires de l'époque relativement faible ne pouvant contenir une importante base de données, ce genre de procédé était plus adapté aux moyens d'alors. Avec l'expansion exponentielle des capacités mémorielles, le choix s'est vite tourné vers des synthèses basées sur des bases de données audio, ou corpus de paroles, aux tailles conséquentes (de quelques centaines de Mo à plusieurs Go). Ces corpus de paroles sont des enregistrements d'un grand nombre de phrases dites phonétiquement équilibrés (afin de couvrir tous les phénomènes de coarticulations [23]). Les synthèses basées sur ce principe d'échantillons (synthèse MMC et synthèse concaténative) diffèrent dans leurs utilisations et méthodes de sélection de ces enregistrements. Il existe toutefois certaines règles qui ne dépendent pas du type de synthèse. Par exemple, on remarque que la concaténation de simples phonèmes ne marche pas [12] (ex. `/k/ + /i/ = /pi/` au lieu de `/ki/`). Il est préférable dans un premier temps de privilégier les transitions entre phonèmes (diphones, triphones, polysons etc.).

I.1.2.1 SYNTHESE PAR REGLES

La synthèse par règles repose sur des modélisations acoustiques du signal de parole, autrement dit sur la génération de formants. Elle part du principe que si tout expert en phonétique est capable de lire le spectrogramme d'un discours, des règles pour le générer peuvent donc en être déduite [24]. Celles-ci, décrivant l'évolution temporelle des formants et la coarticulation des phonèmes, permettent de créer artificiellement un spectre de signal vocal. Par exemple, la Figure I.3 montre une règle décrivant l'évolution des deux premiers formants pour une suite de quatre phonèmes voisés (pas nécessairement des voyelles)[1]. Le signal électronique est ensuite généré par un synthétiseur à formants (DECTalk, InfoVox) [12]. La parole résulte de l'excitation des cavités supra glottiques par le flux d'air provenant des poumons [25]. Ce phénomène peut s'expliquer par une modélisation source filtre, où la source représenterait le flux d'air modulé par les cordes vocales. Ce « signal » est ensuite filtré par toute la partie articulatoire (mâchoire, langue, lèvres) [25]. Il s'agit donc d'une synthèse qui ne fait intervenir que du traitement de signal, sans aucun échantillon de voix. Cette synthèse a pour principal atout d'être très léger en terme de puissance de traitement requise et de mémoire. Elle est donc principalement adaptée aux systèmes embarqués [1]. En termes d'intelligibilité, la plupart des systèmes s'en sortent, même à débits rapides [1]. En revanche, la distinction entre la voix humaine et une voix synthétisée par cette méthode est instantanée, cette dernière ayant un timbre très robotique.

[1] Plus d'informations sur le principe de règles ici [26].

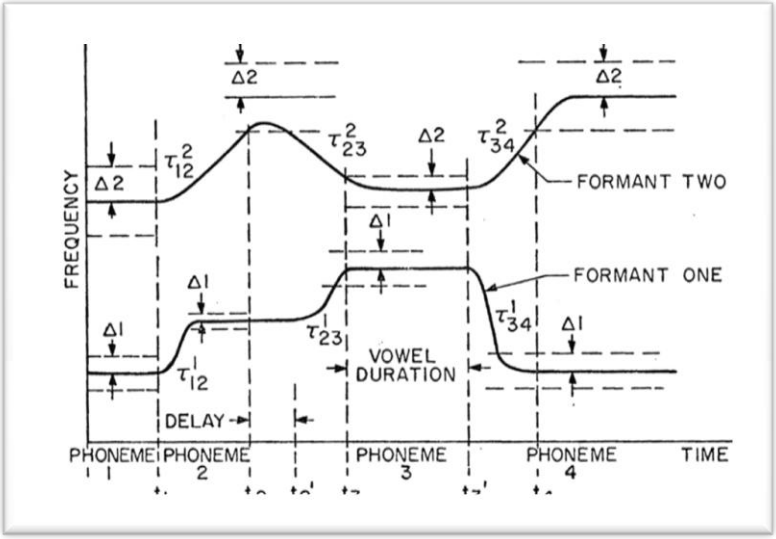


Fig I.3. *Transition des deux premiers formants calculés à partir des valeurs cibles et des constantes de temps* (26)

Par exemple, le synthétiseur de Klatt (1980) est un synthétiseur à formants réputé et modélise le signal par le principe source (signal voisé, fricatives, sifflements) filtre (système de filtrage parallèle et en cascade). La Figure I.5 ci-dessous montre comment est modélisé le conduit vocal. Les formants sont générées à partir de filtres tout pôles du second ordre.

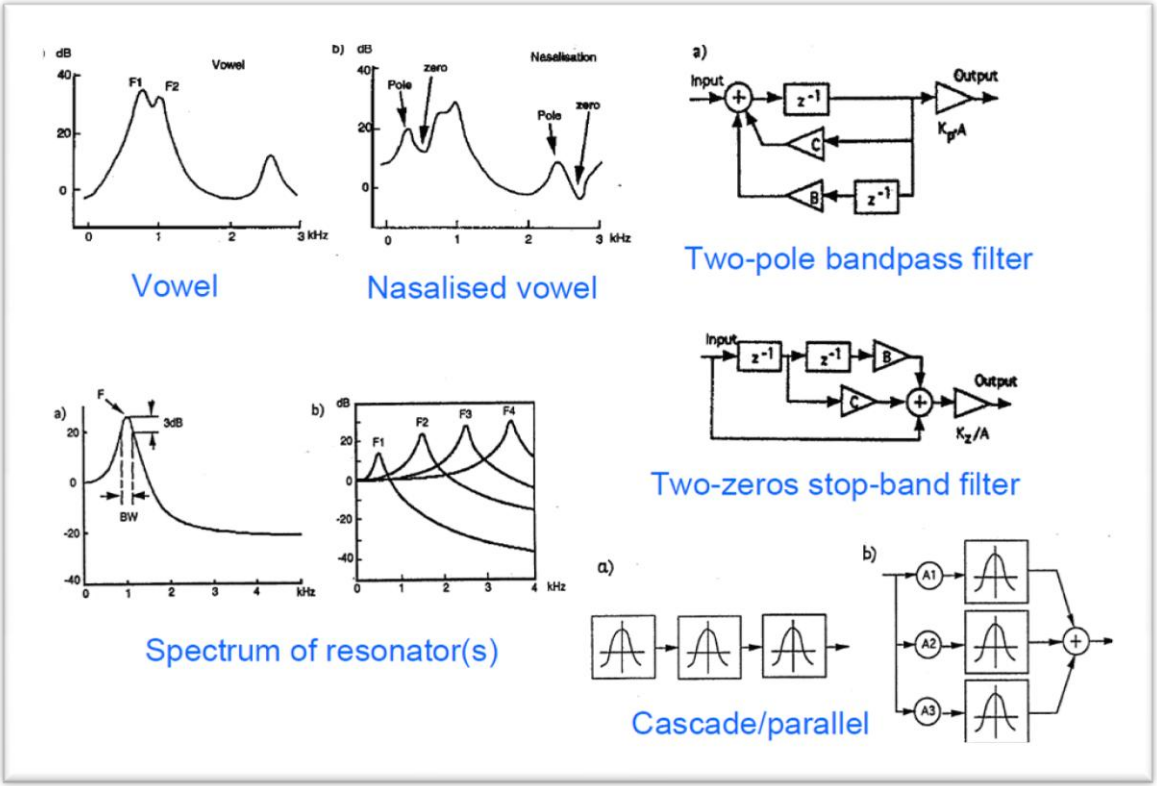


Fig I.5. *Modélisation des formants, synthétiseur de Klatt* (27)

I.1.2.1 SYNTHESE PAR CONCATENATION D'UNITES

Avec les capacités de stockage de données devenues conséquentes, le choix se tourne d'avantage sur la synthèse concaténative que sur celle par règles, limitée par son manque cruel de naturel. Elle ne consiste plus en une modélisation à proprement parlé du phénomène acoustique de la voix, mais d'une mise bout à bout de segments de paroles enregistrés. Ces segments ont une durée définie par les technologies, mais également par le cahier des charges. Ainsi, plus les segments seront long (mots, phrases etc.), plus la qualité du résultat final sera naturel et intelligible, au prix d'un coût important en terme de stockage mais aussi de développement. Historiquement, la première synthèse de ce genre est sans doute la synthèse mot-à-mot, que l'on peut toujours retrouver, par exemple dans les annonces ferroviaires. La simplicité du système n'a d'égal que son manque de naturel et de flexibilité, se contentant de lire des séquences d'échantillons sonores sans aucun traitement phonétique ou prosodique. On se tourne très vite vers d'autres solutions bien plus avantageuses. La Figure I.6 montre le schéma d'ensemble d'un système de synthèse par sélection d'unités quelconque : extraction de celles-ci dans un corpus de parole (transcriptions phonétiques et échantillons sonores), génération de la prosodie, concaténation et génération de la forme d'onde.

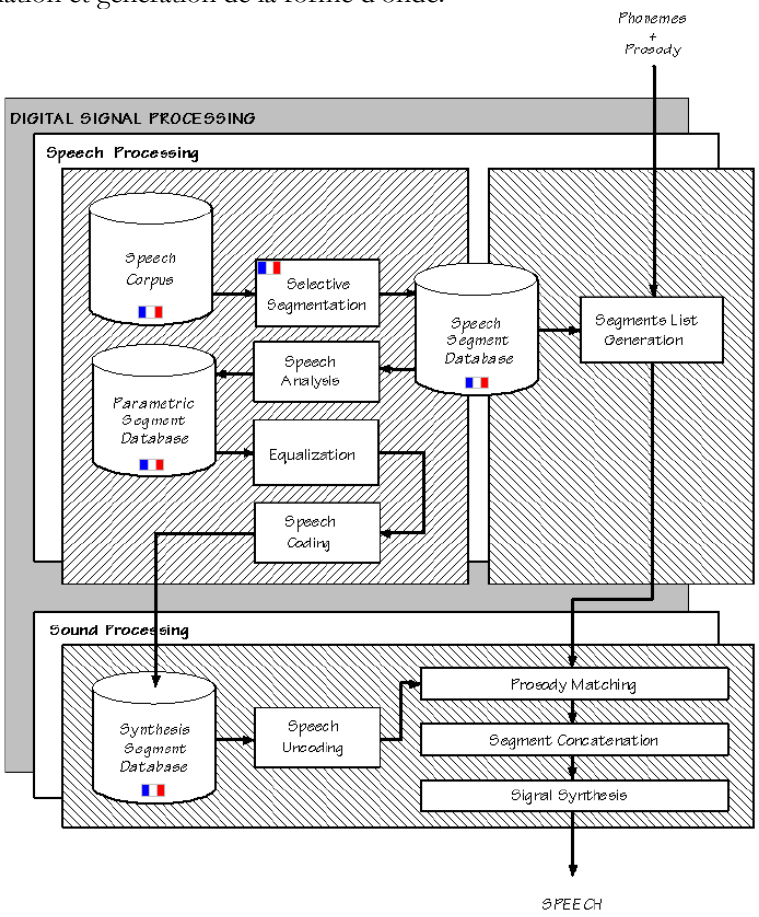
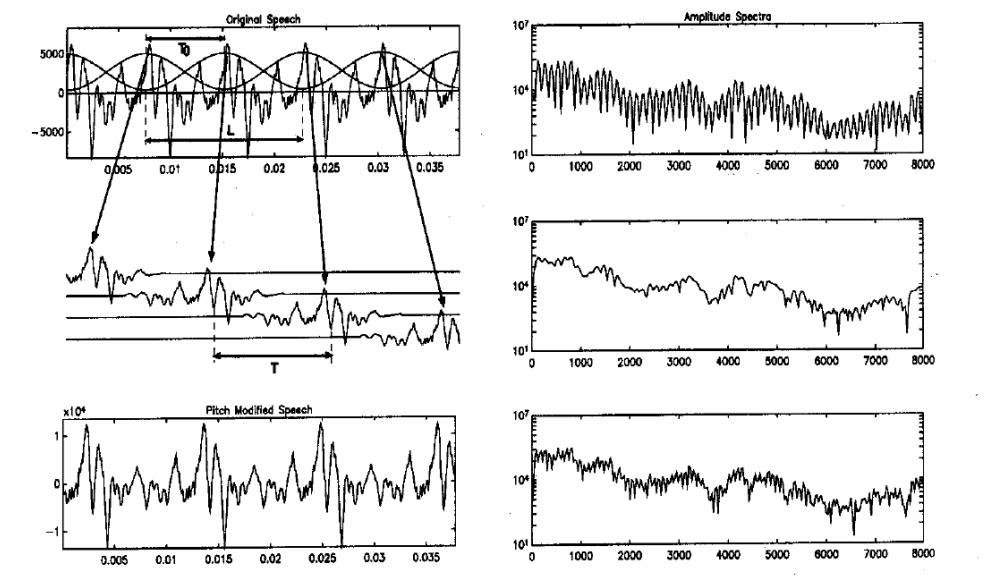


Fig I.6. *Synthèse par sélection d'unités (diphones ou plus)* [27]

Si les phénomènes de coarticulations de phonèmes nous apprennent qu'il est impossible de concaténer des simples phonèmes (la parole étant un processus continu) il a fallut trouver dans un premiers temps la plus petite unité capable de décrire les transitions entres sons. C'est ainsi que l'on définit le diphone comme étant *la transition entre deux phonème successifs, c'est-à-dire la portion du signal de parole comprises entre les noyaux stables de deux phonèmes consécutifs* [25]. La synthèse par diphones fut donc la première synthèse concaténative car elle ne demandait pas un stockage trop important (entre 1500 et 2000 diphones enregistrés, soit environ 2 à 6 minutes de paroles [25]). Le résultat est une synthèse plus naturelle que la synthèse par formant mais très perfectible et peu flexible [28]. On associe souvent un synthétiseur de type TD-PSOLA [27], comme le synthétiseur

MBROLA [29], à la synthèse par diphones. Il consiste à décomposer le signal de la parole en fenêtres overlapées et synchronisées sur la période du fondamental de la voix. Cela a pour principal intérêt de modifier facilement la prosodie (voir Figure I.7 ci-dessous).



Le processus de réharmonisation spectrale de TD-PSOLA. A gauche, les signaux, à droite les spectres correspondants. Le signal modifié (en bas) a bien la même enveloppe spectrale que le signal de départ (en haut); mais pas la même fréquence fondamentale.

Fig I.7. *TD-PSOLA* [27]

Du fait de la qualité relativement basse de la synthèse par diphones, le choix d'unités plus grandes et de tailles variables s'est imposé (syllabes, demi-syllabes, mots, phrases, logatomes). Chaque unité est trouvée en plusieurs exemplaires afin de couvrir un maximum de contextes. On nomme ainsi ce nouveau type de synthèse concaténative la synthèse par sélection d'unités dans une large base de données (constituée de plusieurs dizaines d'heures d'élocution). Ces enregistrements sont alignés automatiquement (souvent en utilisant des modèles de Markov cachés) [30] avec les transcriptions phonétiques correspondantes afin d'en extraire les unités acoustiques lors de la création de la B.D.D. (pitch, durée, position contextuelle etc.). Lors de la synthèse TTS, le texte en entrée phonétisé est convertit en un ensemble de trajectoires. Les unités sont ensuite sélectionnées en fonction de leur similarité (phonétique et prosodique) de telle sorte à minimiser une fonction de coût *C*. Celle-ci (à compléter) est donnée comme la somme des fonctions C^t [2] et C^c [3] [28] :

$$C^t(t_i, u_i) = \sum_{j=1}^{p} w_j^t \, C_j^t(t_i, u_i) \quad (1)$$

$$C^c(u_{i-1}, u_i) = \sum_{k=1}^{q} w_k^c \, C_k^c(u_{i-1}, u_i) \quad (2)$$

[2] Fonction de coût mesurant la distorsion spectrale entre une unité cible u et une unité de sortie t

[3] Fonction de coût de concaténation mesurant la discontinuité spectrale entre deux unités segmentales successives [82]

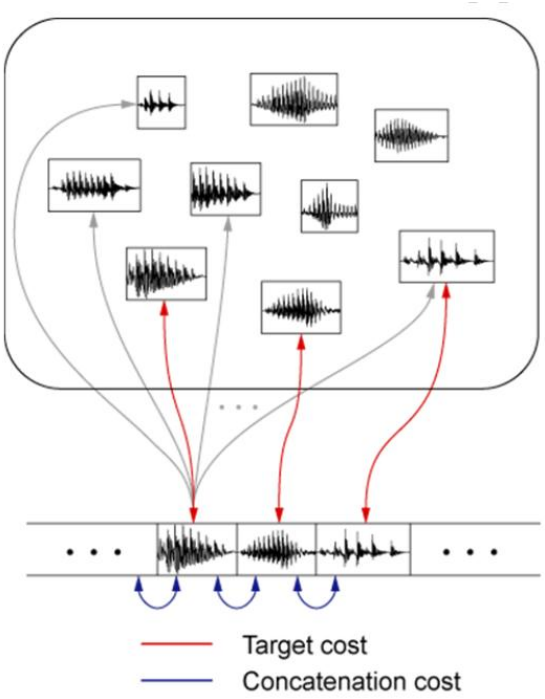


Fig I.8. *Illustration des fonctions de coûts* [27]

Au final, la synthèse par sélection d'unités a le mérite d'être très proche, en termes de qualité et d'intelligibilité de la voix humaine, ceci s'expliquant en partie par la faible part qu'occupe le traitement du signal. Elle s'est largement démocratisée et constituent la solution la plus utilisée dans la plupart des systèmes commerciaux (Acapela, Loquendo etc.). En revanche, elle demeure difficile à mettre en œuvre, du fait de son énorme base de données à transcrire phonétiquement (par alignement automatique et correction manuelle) et reste très couteuse par rapport à ses homologues.

Entre les gigantesques bases de données de la synthèse par sélection d'unités et le traitement du signal omniprésent dans la synthèse par règles, un compromis a été trouvé dans la synthèse basé sur les Modèles de Markov Cachés [4] (synthèse MMC). Tout comme ses homologues, elle nécessite un corpus de parole (moins conséquent que pour le type de synthèse précédent mais obligatoirement phonétiquement équilibré) qu'il est nécessaire d'aligner avec la transcription phonétique correspondante, en général, toujours avec l'aide des MMCs [31]. Les paramètres acoustiques sont ensuite modélisés par des MMCs construites à partir dudit corpus à l'aide d'un algorithme d'apprentissage automatique [32]. Celui-ci est basé sur le critère de maximum de vraisemblance. Cela consiste à déterminer à partir d'un MMC λ donné le vecteur de paramètres acoustiques [5] (ou vecteur d'observations) O de façon à maximiser la probabilité $P(O|\lambda)$ [32].

[4] Les MMCs sont des machines d'états finis générant une séquence d'observations temporelles discrètes [32] (distribution Gaussienne dans chaque état [33]). Ils sont très utilisés en reconnaissance de la parole.

[5] Contient les informations relatives à l'enveloppe spectrale, le pitch et les amplitudes du bruit multi-bandes [33]

$$P(O|\lambda) = \sum_{\forall Q} P(O, Q|\lambda)$$

Avec Q, la séquence d'états tels que

$$Q = \{(q_1, i_1), (q_2, i_2), \dots, (q_T, i_T)\} \quad [6]$$

En général, construire des MMCs pour la synthèse nécessite 5 états pour chaque phonème, dans tous les contextes possibles, c'est-à-dire diphones, triphones, mots etc. (à l'inverse de la reconnaissance vocale plus restrictive sur le choix d'unités segmentales) [33]. La Figure I.9 ci-dessous illustre le fonctionnement global de HTS[7].

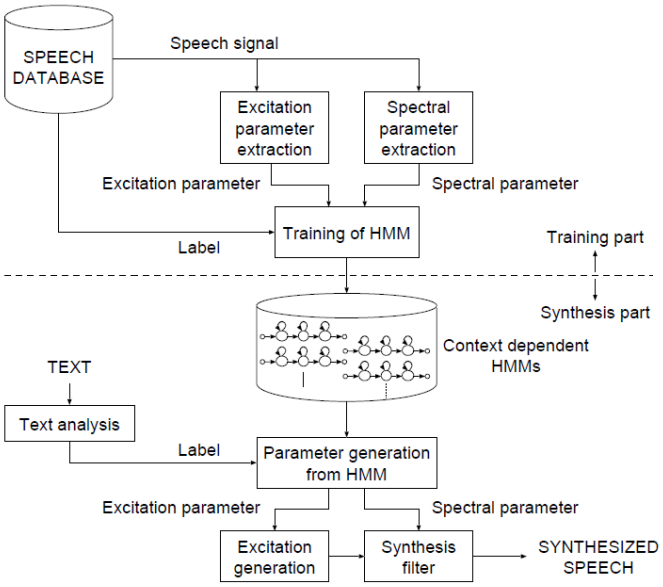


Fig I.9. *Vue d'ensemble de la synthèse HMM* [27]

Une conséquence directe d'un tel système implique que pour toute création d'une nouvelle voix (c'est-à-dire à partir d'un nouveau corpus), il faut recréer de nouveaux MMCs, adaptés à partir de modèles préexistants. En revanche, cela implique aussi que le système, entraîné, permet de modéliser les paramètres qui caractérisent la voix du locuteur. Ainsi on peut mesurer un taux de ressemblance entre la voix synthétique et la voix original de près de 88% [31] ce qui rend ce système parfait pour les applications d'impostures.[8]

Au cours de notre stage, et dans le cadre du projet Openmary, c'est sur la synthèse MMC que nous allons nous pencher, et notamment le synthétiseur HTS.

[6] Avec (q, i) indiquant le i[ème] modèle de mélange de l'état q.
[7] Voir la section suivante pour plus de détail.
[8] Plus d'informations sur la synthèse MMC [31], *Chapter 2, page 27*.

I.1.3 LA SYNTHESE MMC AVEC HTS

HTS est un système de synthèse MMC open source crée en 2002 par le Nagoya Institute [34] (voir la vue d'ensemble du système, figure I.9, page 13). Il est proposé sous forme de patch du système de reconnaissance vocale MMCs HTK.
Comme tout système de synthèse concaténative, il distingue deux phases :

- La phase d'entraînement
- La phase de synthèse à proprement parlé

I.1.3.1 LA PHASE D'ENTRAINEMENT

Dans le premier cas, le processus extrait du corpus de paroles à la fois les informations phonétiques à l'aide de HTK [35] (alignement avec la transcription phonétique, durées et positions des phonèmes), mais également les paramètres d'excitations ($\log(F_0)$) [9] grâce à snack [36] et les paramètres spectraux (coefficients MFCCs, MGC[10], delta et delta-delta) [34], [37], [38] grâce à SPTK (Speech Signal Processing Toolkit) [39]. De plus, chaque MMC, dépendant du contexte, dispose d'une fonction de densité de probabilité de la durée de l'état (state duration PDF) afin de modéliser l'évolution temporelle de la voix [34], [37] [11]. Les paramètres d'excitations et de durées sont alors unifiés dans un même système. Chacun de ces paramètres (F0, spectre et durée) sont influencés par des facteurs indépendants et doivent être entrainés selon tous les contextes possibles. Seulement, la base de donnée audio étant relativement faible (environ 1000 phrases), il est difficile de retrouver tous ces contextes. Ainsi, un algorithme de regroupement contextuel basé sur des arbres décisionnels est mis en place, indépendamment pour chaque paramètres (du fait des facteurs influents qui leurs sont propres) [37] comme l'illustre la Figure I.10.

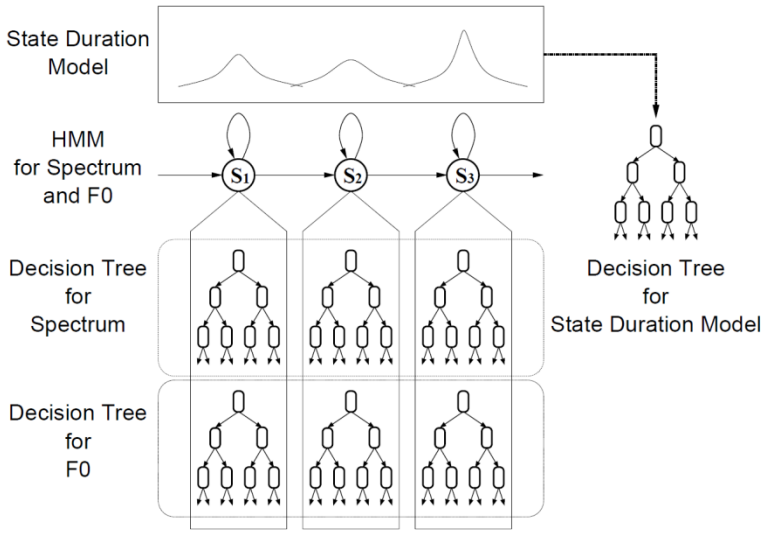


Fig I.10. *Arbre décisionnel pour le regroupement contextuel* [34]

[9] La séquence d'observation des F_0 est composée d'une valeur monodimensionnelle continue et d'un symbole discret (pour les signaux non voisés). De ce fait, utiliser les MMCs continues et discrets n'est pas possible [34], [79]. Pour les modéliser, HTS a recours à des MSD-HMMs (Multi Space Distribution HMMs) [79] tels que la probabilité de chaque état est définie par une distribution de probabilité multidimensionnelle [79].
[10] Les coefficients MGC, Mel-Generalized Cepstrum, à partir duquel la représentation spectrale a une résolution plus proche de l'oreille humaine [80]
[11] Les paramètres peuvent également être extraits par STRAIGHT ou HNM (voir section suivante)

I.1.3.1 LA PHASE DE SYNTHESE

La synthèse se déroule comme explicité dans la section I.2.1.1 (synthèse MMC). Les paramètres du fondamental, du spectre et de la durée [12] sont déterminés de manière à maximiser la probabilité de sortie de leurs états respectifs en utilisant l'algorithme de génération de paramètres acoustique [32]. La forme d'onde est ensuite générée à partir des MFCCs et du F_0 à l'aide d'un filtre MLSA [13] (Mel Log Spectrum Approximation) (40), [41]. Ceci peut se faire, soit couplé à un simple train d'impulsions comme suit :

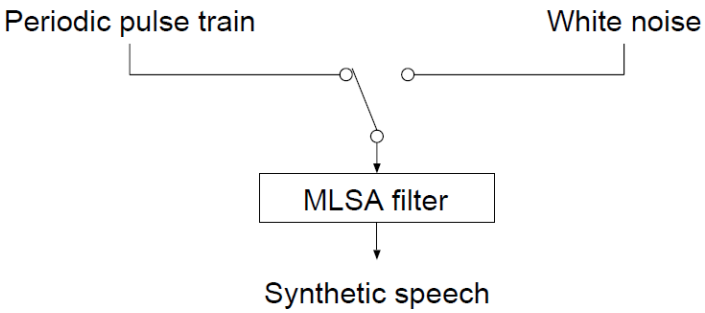


Fig I.11. *Modèle d'excitation simple* [42]

Ou bien selon un model d'excitation mixte [42], comme le synthétiseur STRAIGHT [43] représenté ci-dessous.

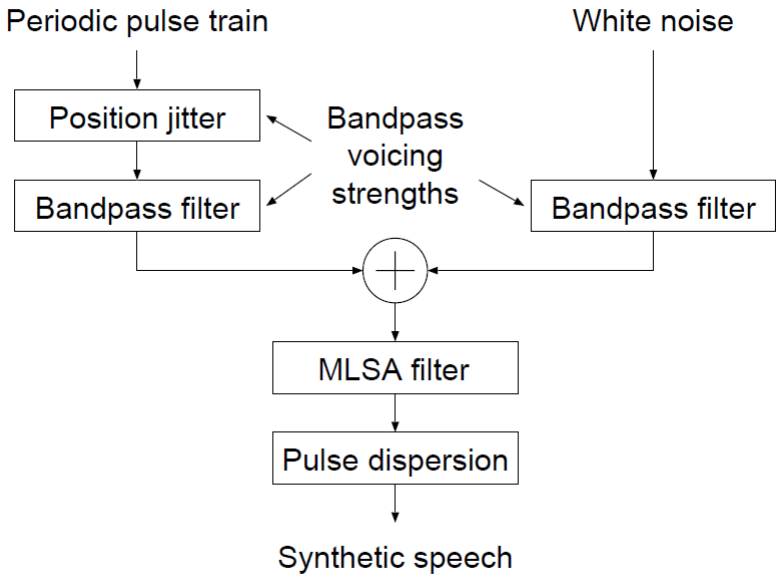


Fig I.12. *Modèle d'excitation mixte* [42]

[12] Tous les paramètres sont dynamiques, les coefficients des vecteurs étant dynamiques [34].
[13] Le filtre MLSA est un simple et robuste synthétiseur vocal dont les coefficients du filtre sont obtenus à l'aide d'une transformée linéaire du Mel cepstre (transformée en cosinus discrète appliquée au log du spectre de puissance du signal représenté sur une échelle non linéaire de Mel). Plus d'informations ici (40).

D'autres types de synthèses peuvent se coupler à HTS comme HNM (Harmonic plus Noise Model) [44]. Ce procédé se base sur le principe de décomposer le signal de voix en une partie déterministe (les harmoniques résultantes des zones voisées) et une partie stochastique (le bruit résultant des zones non voisées), voir Figure I.13. HNM est réputé pour sa qualité de synthèse (son non métallique), provenant pour la plus grande partie du lissage effectué lors de la concaténation entre les unités segmentales, ainsi que de la conservation des énergies BF et HF. Son principal défaut, en outre, est la grande capacité de calcul requise [44].

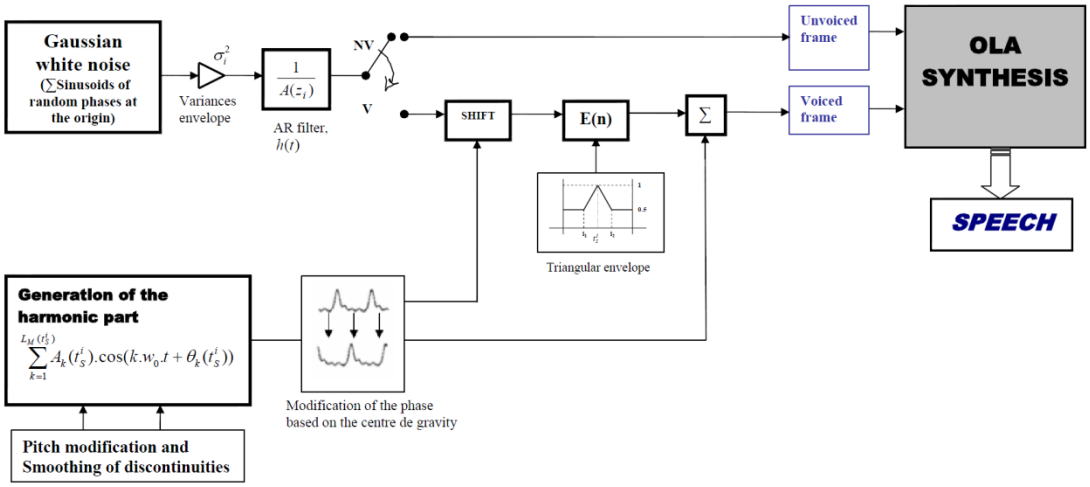


Fig I.13. *Vue d'ensemble de la synthèse HNM* [44]

Notons que HNM, tout comme STRAIGHT, permet également l'extraction de paramètres acoustiques en amont de la phase d'entraînement par HTS (Figure I.14).

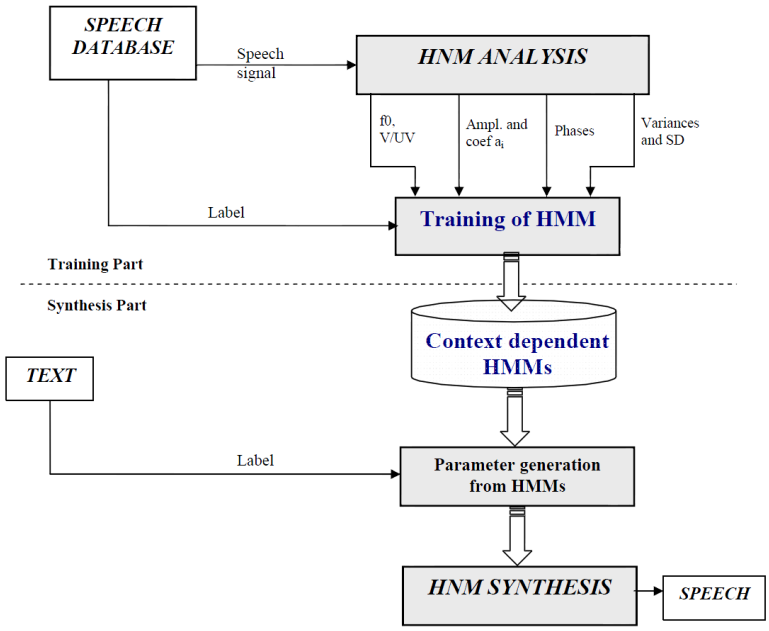


Fig I.14. *Intégration à HTS (entraînement + synthèse)* [44]

La prosodie fait partie intégrante du Traitement Automatique du Langage Naturel. Nous avons toutefois décidé de lui consacrer une section entière car elle représente sans doute l'avenir en matière de synthèse vocale.

Tous les systèmes de synthèse à partir du texte sont plus ou moins intelligibles, qu'elle soit par règles ou par sélection d'unités. Même s'il est important que la phrase soit compréhensible, que l'on puisse reconnaître le locuteur comme c'est le cas pour la synthèse MMC, il n'existe aujourd'hui qu'un seul facteur qui permette de distinguer une voix réelle d'une voix artificielle : il s'agit bien évidemment de la prosodie. Albert Di Cristo écrit :

> *"La prosodie (ou la prosodologie) est une branche de la linguistique consacrée à la description (aspect phonétique) et à la représentation formelle (aspect phonologique) des éléments de l'expression orale tels que les accents, les tons, l'intonation, et la quantité, dont la manifestation concrète, dans la production de la parole est associée aux variations de la fréquence fondamentale (F_0), de la durée et de l'intensité (paramètres prosodiques physique), ces variations étant perçues par l'auditeur comme des changements de hauteur (ou de mélodie), de longueur et de sonie (paramètres prosodiques subjectifs). Les signaux véhiculés par ces paramètres (…) transmettent (…) des informations linguistiques déterminantes pour la compréhension des énoncés et leur interprétation pragmatique dans le flux du discours."* (45)

Ainsi la prosodie est telle une mélodie rythmée par les contextes phonétiques, parsemée de pauses et d'intonations. Elle se caractérise principalement par des variations de durées, d'intensité et de fréquences fondamentales (46). Pour les systèmes ne nécessitant qu'une faible quantité de mots synthétisés (ex. les annonceurs de gares), la modification de la prosodie peut ne pas être nécessaire. Néanmoins, dès que l'écoute d'une voix artificielle prolongée est requise, son absence est très rapidement fatigante, voire insupportable. En effet lorsque nous parlons, les phonèmes (ou syllabes) n'ont pas une durée constante, sans quoi la voix serait horriblement monotone (rythme). Idem, une fréquence fondamentale constante impliquerait une *voix de robot*, très monocorde et agaçante (mélodie).

ex : J…e…n…e…p…a…r…l…e…p…as…c…o…mm…ç…a

On peut donc définir la prosodie comme la musique du langage. La Figure I.15 montre un découpage prosodique en fonction de différents groupes (de souffles, de sens, syllabes) de la phrase « Le gentil petit chat boit du lait».

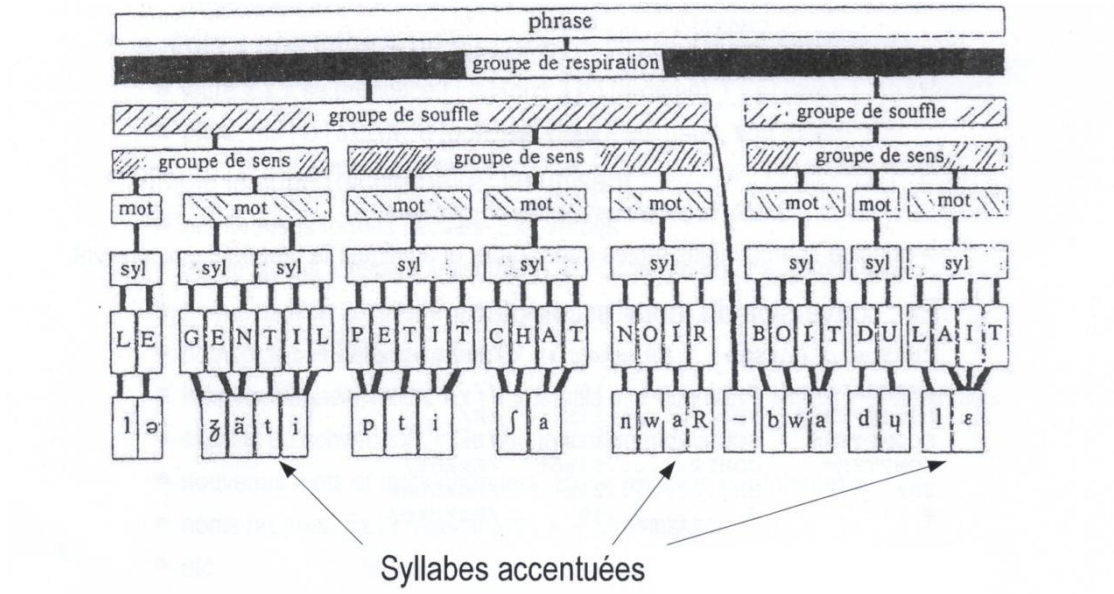


Fig I.15. *Exemple de découpage hiérarchique et de position des accents* (12)

On dénombre quatre fonctions principales liées à la prosodie. La première, la fonction distinctive, permet de distinguer à partir de la syntaxe, et souvent l'absence de marqueurs syntaxique, le type d'énoncé, i.e. déclarative, interrogative ou encore impérative (46).

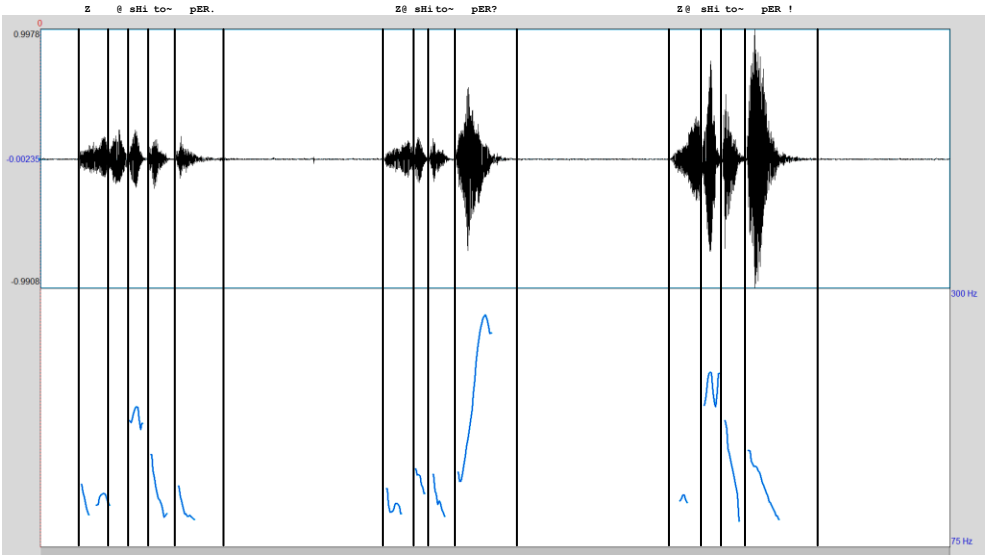


Fig I.16. *« Je suis ton père » : déclarative, interrogative et impérative (enveloppe + pitch), réalisé avec Praat*

La seconde fonction se nomme fonction démarcative. Elle permet, à l'aide de la segmentation, de retrouver l'organisation syntaxique et sémantique de l'énoncé et ainsi lever les ambiguïtés (46). Par exemple, `le père missionnaire` et `le permissionnaire` ont une transcription phonétique identique. C'est bien la prosodie, et l'accentuation du /mi/ de `père` **`mi`**`ssionnaire` qui permet de distinguer l'un de l'autre. Une accentuation peut se caractériser par une fréquence fondamentale, une intensité et une durée plus élevée que les unités segmentales avoisinantes. La troisième fonction, la fonction focalisatrice, est définie par l'emphase de certains mots dans une phrase, l'accentuation, la mise en valeur (46). Dans le texte pur [14] tout comme dans la transcription phonétique, il n'est pas possible de définir ce qui est d'une emphase ou non. Dans ce cas de figure aussi, cette dernière peut être caractérisée par une accentuation dans l'intonation. Enfin la dernière fonction, sans doute la plus difficile à formaliser, est la fonction expressive (46) caractérisant les émotions perceptibles dans la voix. A ce jour, aucun système n'est capable de synthétiser correctement une voix émotive à l'aide de paramètres physiques. Il est encore moins possible de déduire à partir des informations syntaxiques et phonétiques des signes d'émotivité. La plupart des systèmes TTS, comme Loquendo (47), ne se contente que d'une base de données gigantesque de certains sons [15] axées sur des émotions comme la joie, la colère, l'indécision et la tristesse. Ainsi, le texte en entrée doit être balisé par l'utilisateur comme suit :

```
Hello?! Anybody there?! I'm Susan. \item=Kiss \item=Laugh Do you have a
minute?! I'm feeling a bit lonely here. \item=Cry-big I think it might
help if i talked to someone. So listen, my voice already sounds pretty
fantastic, right?! Guess what?! You can make it sound even better. You
can add various effects, \item=Oooh or, with a little punctuation,
certain phrases become more expressive: that's impossible, or rather,
that's impossible! Well then?! What would you like me to say? I'm all
ears! (48)
```

[14] Dénué de police sophistiquée.
[15] Synthèse par sélection d'unités dans une large base de données

Bien sur, ces ajouts ne sont que des tentatives, souvent infructueuses, de rendre le discours plus vivant en le parsemant d'onomatopées et de bruits. Cela ressemble plus à une solution de secours qu'à une réelle réponse au problème. Bien souvent, le résultat est plus que troublant, car l'on passe d'une voix relativement neutre, à une voix très émotive, ce qui tend largement à la schizophrénie ! Il y a donc des problèmes de discontinuité entre émotions qui sont un élément majeure dans la distinction entre une voix artificielle et une voix naturelle. De plus, la fonction expressive ne doit pas être confondue avec la fonction distinctive, bien que ces deux soient naturellement liées. Si bien souvent la colère peut se manifester dans une phrase impérative, donc par une variation du pitch, de la durée et de l'intensité, il demeure que les émotions sont dans la majeure partie des cas plus difficilement caractérisables. Elles se traduisent beaucoup par un changement de timbre, ce qui n'est pas une composante essentielle à la distinction de phrases déclaratives, interrogatives ou impératives. En revanche, c'est bien le timbre de la voix joyeuse ou encore triste qui prédomine. Bien évidemment, la hauteur, la durée et l'intensité ont un rôle à jouer, mais ne sont pas suffisants pour être caractéristique d'une émotion. D'un point de vue très général, dans le monde du traitement du signal, il n'a pas été trouvé de descripteurs physiques traduisant l'émotion. L'intégralité des systèmes (ex. modèles arousal/valence [49]) utilisent l'apprentissage automatique. Il est donc envisageable d'entraîner une certains nombre de phrases décrivant une émotion distincte, l'étiqueter en tant que tel à la fois dans le corpus et dans le texte à synthétiser comme le cas des balises de loquendo.

La prosodie étant définie par une hauteur, une durée et une intensité, elle peut être synthétisée à l'aide de la méthode TD-PSOLA comme décrite dans la section I.1.2.1 ou à l'aide de MMC [50].

I.3 LES SOLUTIONS EXISTANTES

De nombreuses solutions, commerciales, opensource et gratuites existent. Nous pouvons distinguer les systèmes commerciaux tels qu'Acapela [51] ou Loquendo [47] basés sur la synthèse par sélection d'unités dans une large base de données. A ce jour, il s'agit probablement de la synthèse la plus intelligible avec un résultat acoustique très naturel. En revanche, comme énoncé dans la section précédente, aucun résultat significatif en matière d'émotions perceptible dans la voix n'a vu le jour. Pour ce qui est des synthèses opensource, les solutions comme Festival [52], Free TTS [53] proposent plusieurs types de synthèses (diphones, MMC) et les résultats sont plus qu'encourageants.

Une bonne mesure de la qualité d'un système réside dans le WER (Word Error Rate) [54] qui indique le taux de mots incorrectement reconnus par un logiciel de reconnaissance vocal dans un signal synthétisé par rapport à un texte donné. Plus le taux est bas, plus le système est intelligible. Il se définit par [54] :

$$WER = \frac{S + D + I}{N}$$

Avec :
N : le nombre de mots dans le texte donné
S : le nombre de mots mal reconnus
D : le nombre de mots omis
I : le nombre de mots ajoutés

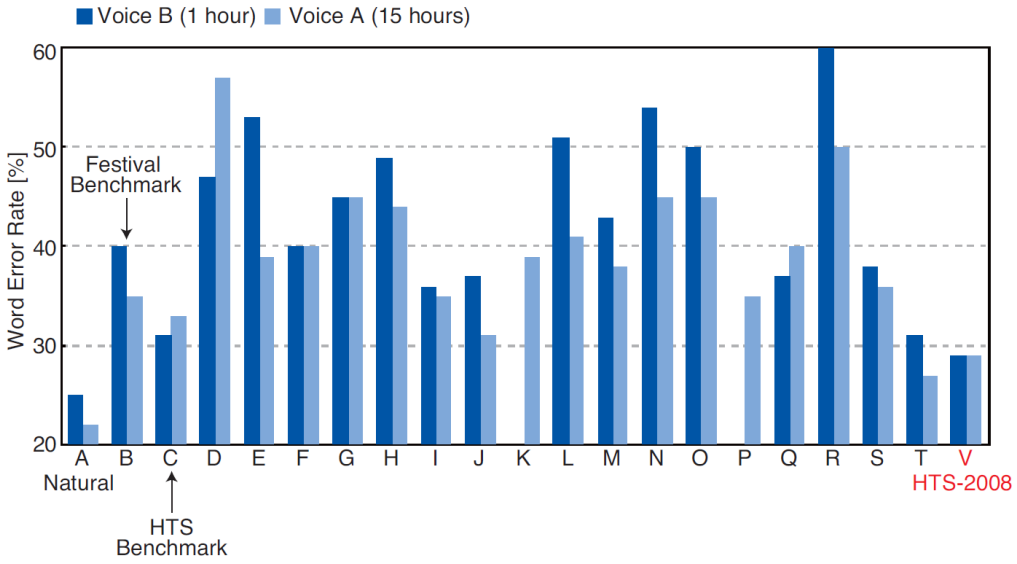


Fig I.17. *WER de plusieurs systèmes de synthèses dont Festival et HTS* [55]

Il n'est pas envisageable de fournir un test complet de tous les TTS existants. La Figure I.17 ci-dessus montre le WER de différents systèmes dont Festival et HTS. Notons simplement le très bon score de ce dernier avec un WER proche de celui de la voix naturelle. On remarquera également par expérience que les logiciels commerciaux ont un léger avantage face à leurs homologues opensource (bien qu'on ignore dans le graphique quels sont les systèmes représentés par des lettres). Enfin, de nombreux tests de solutions TTS pour la langue allemande (commerciaux et gratuites) peuvent être trouvés ici [56].

• • •

L'état de l'art se termine sur cette vue d'ensemble des systèmes de synthèse vocal existants. Il est à présent temps de présenter celui qui est l'objet même de ce stage, le système Mary Text-To-Speech, ou encore Openmary. Comment est-il conçut ? Quel est son architecture ? Est-il stable, robuste, performant ? Quels sont ces principaux atouts, mais également ses faiblesses ? Nous allons tenter d'éclaircir ses points dans le chapitre suivant.

Chapitre II

PRESENTATION DU SYSTEME MARY TTS

II.1 PRESENTATION

Concevoir un système TTS fonctionnel demande de la patience et de l'attention. Il est d'avantage compliqué pour un utilisateur lambda de créer une voix de synthèse à partir de rien. C'est dans l'optique de simplifier la tâche, à la fois pour les développeurs et pour les utilisateurs, qu'est né au début des années 2000 le projet Openmary. Conçu en très grande partie par Marc Schroëder du DFKI (das Deutsche Forschungszentrum für Künstliche Intelligenz), Mary TTS (Modular Architecture for Research on speech sYnthesis), ou encore Openmary, est un environnement complet de synthèse à partir du texte entièrement codé en Java. Il est de ce fait fonctionnel pour tout système d'exploitation capable d'accueillir une machine virtuelle Java (dont bien sur Windows, Mac OS et toute distribution Linux). Le système peut se décomposer en modules bien distincts, ce qui facilite l'ajout d'une nouvelle langue. Il suffira de créer les modules propres à cette dernière [16] et les intégrer au système entier, procédé similaire à l'ajout de plug-in. Le traitement s'effectuant pas à pas, chaque composant communique, en entrée et en sortie, à l'aide d'une représentation des données basée sur du XML, le MaryXML. Ces représentations restent visibles pour l'utilisateur, qui peut donc contrôler le bon déroulement du traitement. Des modules de traitement génériques existent et peuvent servir de point de départ à l'intégration d'une nouvelle langue au système. L'enchevêtrement de ces modules peut être représenté dans le diagramme par bloc suivant :

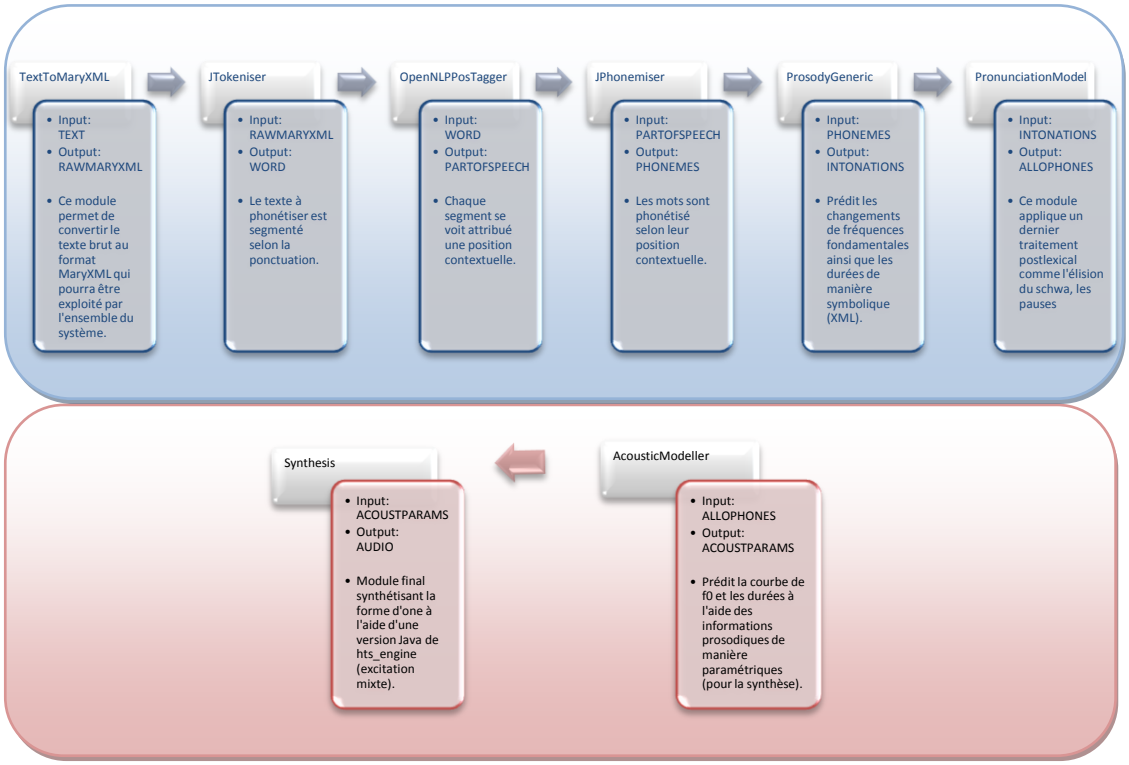


Fig II.1. *Vue d'ensemble du traitement TTS modulaire de Mary, en bleu le TALN, en rose la synthèse.*

[16] Dans la plupart des cas, la création seule de modules de TALN suffit.

Mary permet la synthèse concaténative : concaténation de diphones avec MBROLA, synthèse par sélection d'unités dans une large base de données et enfin synthèse MMC avec HTS. Indépendamment du type de synthèse, Openmary fournit des scripts permettant de créer à partir de zéro une voix dans un nouveau langage. Cela inclut la création d'un grand corpus, à partir de la base de données du Wikipédia de la langue, et un logiciel pour l'enregistrement de ce dernier (Redstart [57]). Sont également fournit des scripts permettant de créer des composants de TALN (pour la phonétisation) à partir des modules génériques et d'un lexique déduit du corpus. La procédure complète peut être trouvée ici [58], (voire annexe An.4, le schéma d'ensemble de cette procédure).

Une fois les modules de traitement crées, on dispose d'une interface graphique pour l'importation de la nouvelle voix. Cela consiste en l'alignement par MMC des fichiers audio avec les transcriptions phonétiques des fichiers textes et l'extraction de vecteurs acoustiques. Dans le cas d'une synthèse MMC, cet utilitaire permet également l'entraînement des triphones du corpus par HTS. Notons toutefois que les procédures de développement sont uniquement valables sous environnement UNIX[17]. L'outil d'importation de voix est décrit point par point ici [59] et la procédure complète peut être trouvée ici [60].

D'un point de vue utilisateur, Mary est basé sur une architecture client/serveur. A chaque utilisation, le serveur doit être lancé et le client échange avec ce dernier les requêtes permettant le traitement. Cette approche permet une plus grande flexibilité, mais également une portabilité accrue. Des serveurs à la capacité de calcul et de stockage supérieurs pourraient être accessibles via des systèmes plus modestes (ex. systèmes embarqués). On pourrait alors depuis ceux-ci utiliser les voix installées dans lesdits serveurs sans avoir à les importer [60] mais simplement en recevant le résultat par l'intermédiaire d'échanges client/serveur. Notons que le code est entièrement multithread, notamment car il n'utilise aucune variable globale.

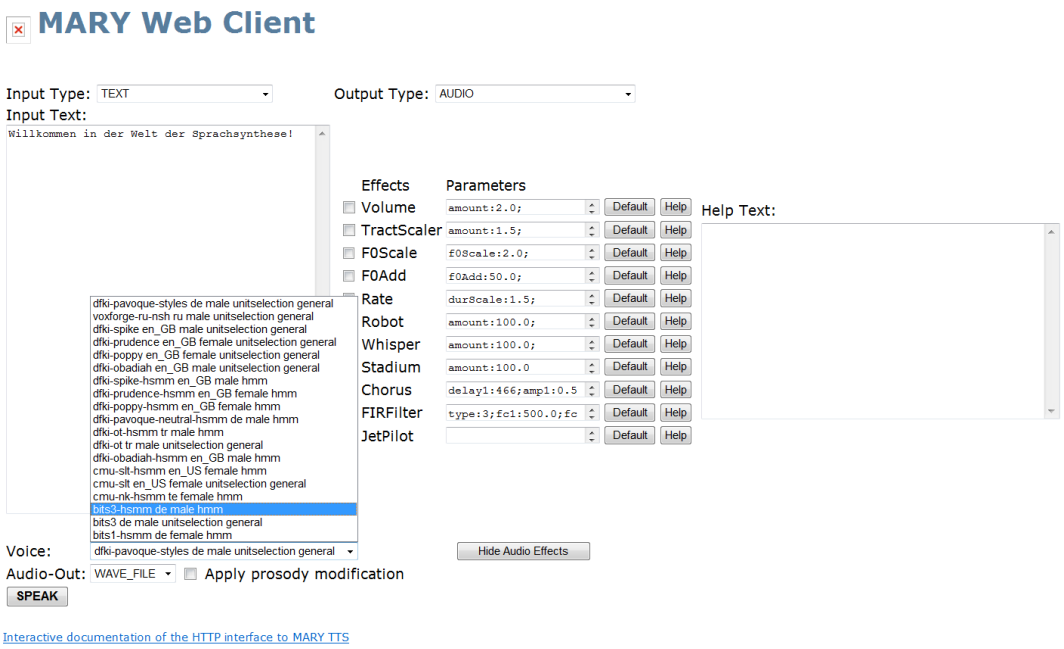


Fig II.2. *Mary Client*

[17] Les scripts sont codés en shell et perl, et certains logiciels dont l'outil fait appel tels festvox pour l'alignement MMC ou HTS sont uniquement valable pour UNIX.

L'utilisateur définit le type d'informations en entrée et en sortie. L'aspect modulaire, comme décrit précédemment, lui permet de visualiser à n'importe quelle étape du traitement `TEXT -> AUDIO` les informations représentées en MaryXML (nous décrirons dans la section suivante les différents types de format MaryXML). Il est par exemple possible de donner en entrée, à la place de texte brut, du MaryXML sous quelque forme que ce soit (`PHONEMES`, `INTONATIONS` etc.) ou même du SSML [61], [20]. On remarque dans cette figure qu'il est possible de choisir la voix dans le menu déroulant selon qu'il s'agisse d'une voix masculine ou féminine, selon le type de synthèse (MMC, diphones, sélection d'unités) et la langue. Des effets audionumériques basiques (murmures, voix robotique etc.) peuvent être obtenus en cochant les cases associées. Dans la version 4.3.0 de Mary [18] sont disponibles les voix anglaise, américaine, allemande, télougou, turc et russe. Chacune de ces langues dispose de modules de TALN adaptés. L'objectif présent est d'ajouter le français à cette liste.

[18] Dernière version en date de la rédaction de ce mémoire.

II.2.1 MARYXML

Comme énoncé précédemment, Mary utilise un langage basé sur du XML comme représentation bas niveau des données, le MaryXML. S'inspirant en grande partie des langages SABLE et SSML [19], il permet à l'utilisateur non expert de s'assurer du bon fonctionnement du traitement inter-modulaire, en rendant accessible les données propres à la synthèse vocale, à l'aide du processus de sérialisation XML (passage de données binaire en XML). A l'inverse, l'utilisateur peut donner en entrée du système du MaryXML exploitable ainsi à l'aide d'une desérialisation (processus inverse de la sérialisation). Il peut donc très facilement modifier certains paramètres (phonétiques et prosodiques par exemple) directement en éditant le MaryXML. Il existe plusieurs types de représentations du MaryXML, entrée comme sortie (voir en annexe An.5. la liste de tous les types d'entrées possibles). Chaque format partage la même syntaxe, mais apporte plus d'informations selon sa place dans la chaîne de traitement. Par exemple, si l'on prend comme `TEXT`[19] : `Je suis, hélas, ton père.` , le format `INTONATIONS` en sortie du module `ProsodyGeneric` (génération de la prosodie) qui comprend la transcription phonétique et les informations phonétiques donne [20] :

```
<?xml version="1.0" encoding="UTF-8"?>
<maryxml xmlns="http://mary.dfki.de/2002/MaryXML"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="0.5"
xml:lang="fr">
<p>
<s>
<prosody pitch="+5%" range="+20%">
<phrase>
<t ph="Z@" pos="[PPER1S]">
je
</t>
<t ph="sHi" pos="[VE1S]">
suis
</t>
<t ph="_" pos=",">
,
</t>
<boundary breakindex="4" tone="H-L%"/>
</phrase>
</prosody>
<prosody pitch="+0%" range="+0%">
<phrase>
<t ph="elas" pos="[ADV]">
hélas
</t>
<t ph="_" pos=",">
,
</t>
<boundary breakindex="4" tone="H-L%"/>
</phrase>
</prosody>
<prosody pitch="-5%" range="-20%">
<phrase>
<t ph="to~" pos="[DETMS]">
```

[19] Seule l'information TEXT n'est pas au format MaryXML (texte brut codé en UTF-8).

[20] Bien que le travail portant sur l'intégration du français à Mary n'a pas encore été présenté, il est décidé toutefois d'expliciter le MaryXML avec les sorties des modules français de TALN par soucis de cohérence.

```
ton
</t>
<t ph="pER" pos="[NMS]">
père
</t>
<t ph="_" pos=".">
.
</t>
<boundary breakindex="5" tone="L-L%"/>
</phrase>
</prosody>
</s>
</p>
</maryxml>
```

Chaque unité segmentale est entouré des balises `<t> </t>`.

```
<t ph="pER" pos="[NMS]">
père
</t>
```

`ph` représente la transcription phonétique de l'unité, ici `pER` et `pos` sa position contextuelle dans la phrase. Le texte a été phonétisé par LIA_phon[21] (11), la position `[NMS]` signifie Nom Masculin Singulier (Voir en annexe An.6. le tableau des étiquettes de positions contextuelle utilisées par LIA_phon).

Pour les informations prosodiques, les groupes de souffles (suites de syllabes précédées et suivies de pauses) sont entourés par les balises `<phrase> </phrase>`. Ceux-ci sont encapsulé dans une balise `<prosody> </prosody>` définissant le pitch et l'intensité.

```
<prosody pitch="+5%" range="+20%">…</prosody>
```

Les ponctuations sont annotées comme suit :

```
<t ph="_" pos=".">
.
</t>
<boundary breakindex="5" tone="L-L%"/>
```

La transcription phonétique de chaque ponctuation est _. Leur étiquette, s'il s'agit d'une ponctuation en milieu (, : ;) ou en fin de phrase (. ! ? …) est respectivement `"."` et `","`. La balise

```
<boundary breakindex="5" tone="L-L%"/>
```

indique le changement d'intonation, pitch et pause, lié à la présence de ponctuation. La variable `tone` correspond à la transcription ToBI (62), c'est-à-dire une représentation des accents et intonations (l'ensemble des symboles ToBI et leur signification sont disponibles ici (63)).

Il n'est pas nécessaire de décrire tous les formats du MaryXML. Voir en annexe An.7. l'ensemble des formats et leur description pour la chaîne de traitement :
`TEXT -> RAWMARYXML -> PHONEMES -> INTONATIONS -> ALLOPHONES -> ACOUSTPARAMS`

[21] LIA_phon est un système graphème-phonème complet. Voir le chapitre 3 pour plus de détail.

II.2.2 MODULE GENERIQUE

Openmary dispose de modules génériques applicables à toutes les langues. Ces modules sont des classes codées en Java. Elles héritent toutes de la classe `InternalModule`. Ainsi chacune d'elles doivent avoir un constructeur comme suit, ainsi qu'une méthode `process` :

```
import org.w3c.dom.Document;

import marytts.datatypes.MaryData;
import marytts.modules.InternalModule;

/**
 * Le squelette de tout module interne.
 * @author Florent
 *
 */

public class MyModule extends InternalModule
{
    public MyModule ()
    {
        super ("MyModule",
                MaryDataType.INPUT
                MaryDataType.OUTPUT);
    }


    /**
     * La méthode process. Prend en entrée n'importe quelle donnée
     * c'est-à-dire, TEXT, WORD, PHONEMES etc. réalise le traitement,
     * crée un document MaryXML qu'elle retourne en sortie.
     *
     * @param d
     */
    public MaryData process (MaryData d) throws Exception
    {
        Document doc = d.getDocument();
        doc = DoSomething();
        MaryData result = new MaryData(outputType(), d.getLocale());
        return result;
    }


    /**
     * Une méthode lambda traitant les données.
     *
     */
    public Document DoSomething ()
    {
        //…faire quelque chose !
        Document D = null;
        return D;
    }

}
```

Le constructeur définit l'entrée et la sortie MaryXML du module (ici `INPUT` et `OUTPUT`). La méthode `process` [22] prend en entrée `INPUT` et sort `OUPUT`. Chaque module partageant ces mêmes

[22] Initialement, la méthode `process` provenant de la superclasse `InternalModule` est évidemment redéfinit (Override) à l'instanciation du nouveau module.

caractéristiques, le système entier peut être représenté dans un diagramme de bloc, comme énoncé précédemment (figure II.1, page 25).

Hormis les modules propres au français détaillés dans le chapitre 3, la langue française, comme toutes les langues, utilise certains modules génériques : `Text2MaryXML`, `ProsodyGeneric`, `AcousticModeller` et `Synthesis`. Le diagramme en bloc (Figure II.1, page 25) donne une présentation brève de chaque module générique. Hormis pour le premier d'entre eux qui ne présente aucun intérêt à proprement parlé (le module `Text2MaryXML` convertissant le texte brut en `RAWMARYXML`), nous allons détailler le fonctionnement de ces classes, ainsi que la théorie en amont.

II.2.2.1 PROSODYGENERIC

- Entrée: `PHONEMES`
- Sortie: `INTONATIONS`

Ce module détermine de manière symbolique la prosodie. Il consiste à étiqueter le texte d'accentuations (modification du pitch en fonction de locations cibles) et d'intonations en fin de phrase selon les paramètres syntaxiques de celle-ci. Ainsi ce module se distingue du suivant décrit en II.2.2.3 qui lui transcrit cet étiquetage en une courbe concrète de f_0 et de durée, c'est-à-dire, passer du mode de représentation symbolique (le MaryXML) au paramétrique (paramètres acoustiques). Un ensemble de règles s'occupent d'étiqueter le texte de symboles ToBI [62], [63]. Celles-ci sont déduites de l'analyse automatique d'un corpus labellisé selon cette norme et sont majoritairement dépendantes de la ponctuation et des positions contextuelles des unités segmentales [19]. Le format ToBI est surtout valable pour l'anglais. Bien que de nombreuses règles d'intonations soient communes à la majorité des langues, comme le fait d'accentuer les noms propres et les verbes, il peut s'avérer nécessaire d'adapter le module pour un rendu plus approprié au langage développé. Une fois les paramètres prosodiques déterminés, l'intonation globale de la phrase est générée en fonction du type de celle-ci (interrogative, déclarative, exclamative). Voir la section II.2.1 pour une analyse détaillée du format de sortie du module de génération de prosodie.

II.2.2.2 PRONUNCIATIONMODEL

- Entrée: `INTONATIONS`
- Sortie: `ALLOPHONES`

Ce module apporte les derniers traitements de TALN afin de fournir une sortie MaryXML riche en informations. Lorsque les transcriptions phonétiques des syllabes de chaque unité segmentales sont déterminés par les modules précédents, de même que les intonations en fin de phrase et les accentuations des syllabes, la classe `PronunciationModel` procède à une restructuration du MaryXML par un nombre de règles post lexicales [19]. Directement à partir des transcriptions phonétiques sont déduites des règles de prononciations plus fines, incluant par exemple les phénomènes d'élision du schwa et les enchaînements entre mots (liaisons ou pauses). De plus, chaque syllabe phonétisée est décomposée en les phonèmes qui la composent. Ces phonèmes sont classifiés dans un fichier propre au langage, le fichier `allophones.**.xml`, où `**` représente la langue («`fr`» pour le français). Sont renseignés dans ce dernier le type de chaque phonème, selon qu'il soit une consonne (fricative, occlusive etc.) ou une voyelle (antérieure, postérieure etc.). Voir en annexe An.8. le fichier `allophones.fr.xml` définit pour le français, en se basant sur la liste de phonèmes de LIA_phon (Annexe An.1.).

II.2.2.3 ACOUSTICMODELLER

- Entrée: `ALLOPHONES`
- Sortie: `ACOUSTPARAMS`

Prenant comme entrée un MaryXML riche en informations phonétiques et prosodiques, le module `AcousticModeller` a pour fonction de prédire une courbe de F_0 et de durées, le procédé dépendant du type de synthèse employé. Contrairement aux modules `ProsodyGeneric` et `PronunciationModel` qui s'occupe de déterminer une prosodie de manière symbolique, ici la sortie de ce module est une liste composée des segments phonétiques (phonèmes ou syllabes) et trajectoires de leurs durées et pitch (conversion des symboles ToBi en fréquences). Selon le type de synthèse, le module instancie les classes appropriées qui correspondent à différents algorithmes.

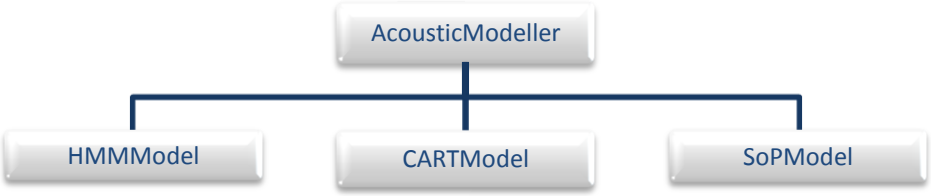


Fig II.3. *Schéma de la classe AcousticModeller et ses algorithmes*

S'il s'agit de synthèse MMC, l'algorithme basé sur les MMC (`HMMModel`) sera choisi. Idem s'il s'agit de synthèse par sélection d'unités, l'algorithme CART (`CARTModel`) sera choisi pour déterminer les trajectoires de F_0 et de durées. Les modèles SoP (Sums of Products Model [64]) ont été implémenté comme des modèles de régression linéaire selon ce papier [65]. Cependant ceci reste la configuration par défaut, le développeur pouvant très bien définir par exemple un algorithme CART pour la fréquence fondamentale et un algorithme MMC pour les durées. Il n'existe pas encore de test quant au meilleur choix d'algorithme selon le type de synthèse. Simplement, les modèles SoP de la classe `SoPModel` sont implémentés de manière rudimentaire comparé aux autres modèles, d'après son auteur. Toujours d'après lui, il apparaîtrait que la prédiction de F0 par CART ne fonctionne pas dans le cas d'une voix MMC.

II.2.2.4 SYNTHESIS

- Entrée: `ACOUSTPARAMS`
- Sortie: `AUDIO`

Le module de synthèse varie dans son fonctionnement selon le type de synthèse souhaité (ici la synthèse MMC). Comme énoncé précédemment, la synthèse MMC se décompose en deux parties : l'entraînement à partir du corpus, et la synthèse elle-même. La procédure de création de voix utilisant le *VoiceImportTools* se sert des scripts HTS (uniquement disponible sous environnement UNIX) pour la phase d'entraînement (voire section I.1.3). Il est impossible de se servir de ce dernier pour la phase de synthèse, sinon le principe de portabilité multiplateforme de Java ne serait pas respecté. Ainsi Mary dispose d'un propre portage de HTS en Java. Pour la génération de la forme d'onde, elle utilise également un portage Java de hts_engine[23], à quoi y est ajouté un modèle d'excitation mixte. Ici la sortie du module n'est plus du MaryXML mais un signal audio (mono, 16 bits, 16 kHz).

[23] hts_engine est un générateur de forme d'onde à partir des données en sortie de HTS, basé sur le modèle d'excitation simple [81].

II.3.1 QUALITES DES VOIX DISPONIBLES

Il n'existe actuellement aucun test évaluant l'intelligibilité des voix de Mary. Un test mesurant le WER ne peut être mis en place, il faudrait pour cela disposer à la fois de la voix naturelle et synthétique du même interlocuteur. Une discussion subjective sur l'intelligibilité est proposée pour les voix MMC [24] anglaises. Le texte à synthétiser est le suivant :

> "There is no escape. Don't you make me destroy you? You do not yet realize your importance. You have only begun to discover your power! Join me and I will complete your training. With our combined strength, we can end this destructive conflict and bring order to the galaxy." (66)

Ce texte relativement long comprend notamment une phrase interrogative et une impérative. Celles-ci doivent normalement se traduire par un changement prosodique. D'une manière générale, toutes les voix présentent les mêmes caractéristiques. On note une très bonne intelligibilité. A l'oreille humaine, aucun mot n'a été omis ou synthétisé de manière à ne pas pouvoir le reconnaître. Cependant, dans la version actuelle, la prosodie générée ne distingue pas la différence entre une phrase déclarative, interrogative et impérative. Ainsi, « Don't you make me destroy you ? » et « You have only begun to discover your power !" n'ont pas l'intonation adaptée. On remarque aussi quelques problèmes de lissage entre certaines unités segmentales, ainsi que des problèmes de macro-prosodie (67) (phénomène de discontinuité prosodique à l'échelle du phonème qui se traduit par un trémolo dans la voix synthétique). Enfin l'écoute du texte synthétisé est assez pénible à écouter dans son intégralité, ceci s'expliquant sans doute par une courbe des durées manquante de réalisme (segments souvent trop long dans leurs contextes).

II.3.2 VITESSE DE TRAITEMENT

La durée du traitement `TEXT -> AUDIO` est un élément clé à l'évaluation du système. Idéalement, on doit pouvoir considérer ce dernier comme « temps réel », à l'image de la parole naturelle. Pour les systèmes sonores, on considère un processus temps réel lorsque son temps d'exécution est inférieur à une période d'échantillonnage (68). Dans le cas de la voix de synthèse échantillonnée à 16 kHz, le processus de calcul doit être inférieur à 16 ms. Toujours dans ce scénario idéal, le traitement devrait s'effectuer segments par segments pour les textes trop longs, donc traiter les tâches d'analyses syntaxiques et de synthèses en parallèles.

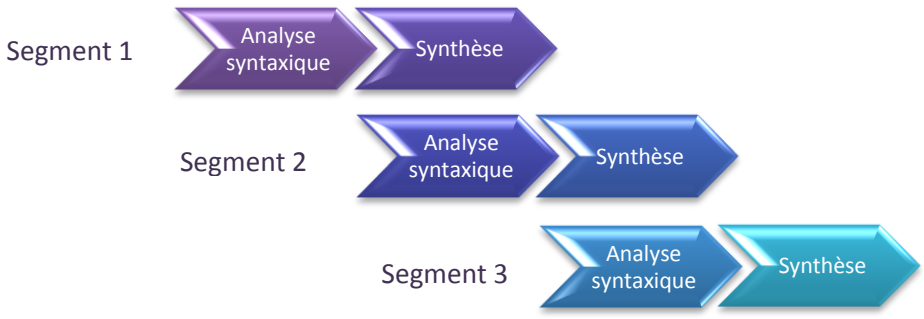


Fig II.4. *Schéma d'un processus idéal*

[24] De nombreuses voix de synthèse par sélection d'unités sont disponibles, mais par contraintes de temps et parce que ce type de synthèse ne rentre pas dans le cadre du stage, elles ne seront pas testées.

Ceci reste compliqué dans l'état actuel des choses, chaque segment dépendant de son contexte environnant pour être synthétisé. En réalité, le texte entier est traité en bloc uniforme comme une seule et même entrée, ce qui rend le temps de traitement très variable selon plusieurs facteurs hormis le nombre de caractères. Une phrase comportant des abréviations, dates, heures etc. mettra plus de temps à être synthétisé compte tenu du nettoyage du texte. Par exemple, voici le détail (fichier log du Mary server) du traitement d'une phrase courte en allemand comme `ich liebe dich`.

```
Request processed in 199 ms.

TextToMaryXML took 0 ms
JTokeniser took 1 ms
Preprocess took 1 ms
OpenNLPPosTagger took 2 ms
JPhonemiser_de took 2 ms
Prosody took 4 ms
PronunciationModel took 2 ms

AcousticModeller took 33 ms
Synthesis took 153 ms

Request handled successfully.
```

Les modules de TALN prennent 12 ms et la synthèse 186 ms, soit un total arrondie de 199 ms. Cela reste beaucoup trop long pour une utilisation temps réel, mais raisonnable toutefois dans le cas contraire.

II.3.3 INCONVENIENTS DUES A L'INSTALLATION

L'installation de l'utilitaire d'importation de voix sous UNIX, le *VoiceImportTools*, est laborieuse et la création d'une nouvelle voix, même dans un langage préexistant est une tâche non sans embûches. Chaque composante de celui-ci requiert des applications externes, des librairies etc. qui ne sont pas signalées dans le tutoriel. Il est donc difficile quand les problèmes surviennent d'en trouver la cause. Des scripts sont proposés pour l'installation automatique des applications externes. Certaines commandes procèdent à des téléchargements de versions obsolètes, voire plus du tout disponibles (comme c'est le cas pour HTK/HTS) obligeant ainsi l'utilisateur à les installer manuellement. Une mise à jour du tutoriel est proposée en Annexe An.9. où l'on retrouve toutes les autres causes qui font du *VoiceImportTools* un outils, bien que très intéressant pour l'utilisateur, perfectible et en manque de documentation. Il est fort probable, d'après ces concepteurs, que tous ces problèmes seront résolus dans la prochaine version de Mary.

• • •

Après avoir fait l'état de l'art sur la synthèse vocale et la présentation du système Mary Text-To-Speech, le travail effectué à l'occasion de ce stage, l'intégration du français au système, sera présenté en détail dans le chapitre suivant. Nous commenterons les résultats obtenus puis nous ferons le point sur ce qui a été réalisé, ce qui ne l'a pas été et ce qui reste à faire en aval de ce projet.

Chapitre III

MODELISATION DE LA LANGUE FRANÇAISE DANS MARY TTS

III.1 POSTULAT DE DEPART

III.1.1 CONTRAINTES

Le système Mary étant modulaire et disposant de classes génériques comme énoncé dans le chapitre 2, l'intégration du français au système ne nécessitait exclusivement que le codage des modules de TALN, c'est-à-dire un système graphème-phonème complet (`RAWMARYXML -> PHONEMES`) puis la création de la voix avec le *VoiceImportTool*. Le *NewLanguageSupport* propose une méthode pour la création de modules à partir des classes `JPhonemiser` et `JTokeniser` (voir en Annexe, An.4.). La création du phonétiseur comprend un dictionnaire basé sur le corpus du Wikipédia de la langue. L'étiquetage syntaxique et les règles lettres-à-sons sont établit par apprentissage automatique, à l'aide d'un outil de transcription semi-automatique[25] (69). Au final, tout ceci se résume dans la plupart des cas à parcourir un lexique de mots selon le texte d'entrée, sinon à appliquer des règles lettres-à-sons trop génériques, puis à concaténer les transcriptions phonétiques correspondantes. De plus, le corpus tiré de Wikipédia est, par définition, neutre. L'interlocuteur pourra donc difficilement y transcrire des émotions. Si dans une bonne majorité des langages ce procédé est suffisant, ou du moins reste une bonne base, il n'en est pas de même pour le français, réputé pour ces subtilités syntaxiques, comme l'élision des schwas, les phénomènes de liaison et les homographes hétérophones à outrance. Parce que la conception d'un système graphème-phonème complet et fonctionnel dépasse le cadre d'un stage de fin d'études, la solution la plus approprié est de réutiliser un système opensource déjà existant. LIA_phon (présenté dans la section III.2), système de phonétisation de texte pour le français, a donc été intégré à Mary. Il a été choisi pour sa robustesse, sa performance et parce qu'il est libre.

III.1.2 CORPUS DE PAROLE

Afin d'attester de l'intégration d'une langue à Mary, il est nécessaire d'en créer la première voix de synthèse. Nous avons opté pour une voix MMC pour sa flexibilité et sa performance dans la modélisation des paramètres de l'interlocuteur. La synthèse par sélection d'unités a été écartée car elle ne satisfaisait pas ces conditions, en plus d'être très coûteuse à réaliser (corpus massif). Nous disposons de deux corpus de parole. Le premier, de 516 phrases phonétiquement équilibrées, fut conçu par Lise Buchman et Cédric Gendrot [26], et enregistré par Pierre Roussel [27]. Le second corpus provient de la base de données PolyVar (70) de 3000 phrases enregistrées par Florent Xavier à l'ESPCI. Les statistiques[28] montrent l'effectif de chaque phonème dans ces deux corpus en Annexe (An.10.). On constate une différence peu significative de leurs fonctions de répartitions des phonèmes, l'écart type de la différence des deux fonctions étant de $\sigma = 0{,}0158619$ ce qui indique qu'ils sont relativement équilibrés de la même manière. En revanche, le corpus du LPP comporte 5168 triphones (suite de trois phonèmes successifs), alors

[25] L'utilisateur doit corriger la transcription phonétique d'un ensemble de mots donnés déduite par apprentissage automatique, selon l'ensemble de phonèmes que contient le fichier `allophones.**.xml`.

[26] Du LPP (Laboratoire de Phonétique de Paris)

[27] Du laboratoire SIGMA de l'ESPCI

[28] Statistiques réalisés par Jun Cai de l'ESPCI.

que le PolyVar en comporte 10637. Ce dernier apparaît par conséquent comme le plus riche en contexte, ce qui constitue un argument de poids dans le choix de corpus pour voix MMC (compte tenu du « faible » nombre de phrases). Les enregistrements sont aussi très différents les uns des autres. Pour le premier corpus, l'interlocuteur a pris une voix très neutre et monotone, en articulant un maximum pour faciliter la reconnaissance vocale. En résulte très peu de schwa non prononcés et une modélisation de la durée peu réaliste. A l'inverse, partant du principe que les algorithmes MMC extraient les caractéristiques acoustiques directement du corpus, le second a été enregistré de manière très naturelle, afin de couvrir un maximum de contextes conversationnels, en prenant des intonations très différentes selon le sens des phrases[29].

III.2 LIA_PHON, UN SYSTEME GRAPHEME-PHONEME COMPLET

III.2.1 PRESENTATION

LIA_phon est un système de phonétisation pour le français conçut par Frédéric Béchet au LIA (Laboratoire Informatique d'Avignon) au début des années 2000. Il répond d'abord au besoin des développeurs de disposer d'un système ouvert, paramétrable et libre[30] pour la construction de systèmes de reconnaissance de la parole. Il permet également la validation de certaines théories de TALN, *notamment concernant le traitement des mots inconnus* (11). Uniquement disponible sous environnements UNIX (distributions Linux, Mac OS), il se présente sous la forme de scripts en shell appelant des exécutables. Ceux-ci s'appuient sur des sources codés en C (.c et .h) et des fichiers de données (règles de phonétisation, lexique de mots, d'abréviations etc.), auxquels sont assignés des commandes lançables à partir du terminal (voir en Annexe An.11. le tableau détaillée des scripts, programmes et fichiers de données). Les règles représentent sont des conséquences directes de la qualité de transcription. Elles se présentent comme suit :

```
REGLE(NUM,<C_GAUCHE,GRAPH,C_DROIT>,PHO,EX)-> CONT ;
```

`REGLE` représente le nom de celle-ci (`LIAISON`, `CONTEXT` etc.), `NUM` son numéro, `C_GAUCHE` le contexte à gauche de la graphie[31], `GRAPH` la graphie à transcrire, `C_DROIT` le contexte à droite, `PHO` la transcription phonétique de la graphie, `EX` un exemple et `CONT` une contrainte grammaticale. Pour coder les contextes, on utilise le symbole `S` pour la gestion des pluriels, _ un espace, `C` une consonne et `V` une voyelle. Le script réalisant la transcription graphème-phonème, `lia_text2phon` se présente comme suit :

```
(…)
$LIA_PHON_REP/script/lia_taggreac -reacc | \
$LIA_PHON_REP/script/lia_phon -syllabe -liaison
```

A l'image de tout composant de TALN, et d'après le code, LIA_phon effectue en premier lieu la segmentation en mots (`lia_nett`), puis l'étiquetage morphosyntaxique et la ré-accentuation (`lia_taggreac -reacc`) et enfin la transcription graphème-phonème (`lia_phon -syllabe -liaison`). D'une façon plus détaillée, le fonctionnement interne de LIA_phon peut se résumer en six phases : (1) le formatage, (2) l'étiquetage morphosyntaxique, (3) l'étiquetage sémantique, (4) la gestion des liaisons, (5) les règles de phonétisation, et enfin (6) le post-traitement.

[29] Intonations se rapprochant plus d'un journal télévisé que d'une pièce de théâtre.
[30] License GNU (General Public License)
[31] La graphie peut être un mot entier, une syllabe, un suffixe, toute suite de lettres significatives.

III.2.1.1 LE FORMATAGE

LIA_phon réalise le formatage en deux parties. La segmentation en phrases, puis en mots (*tokenisation*) et le nettoyage du texte en caractères alphabétiques exclusivement (chiffres romains et arabes, caractères spéciaux, syntaxte HTML etc.). Bien que ce module ne nécessite pas un traitement très compliqué, il demeure un processus quasi chaotique quant au résultat de la synthèse finale (11). `lia_nett` est le script en charge du formatage.

```
$LIA_PHON_REP/bin/lia_tokenize $LIA_PHON_REP/data/$LIA_PHON_LEX.tab | \
$LIA_PHON_REP/bin/lia_sentence $LIA_PHON_REP/data/list_chif_virgule.tab | \
$LIA_PHON_REP/bin/lia_nett_capital $LIA_PHON_REP/data/$LIA_PHON_LEX.tab | \
$LIA_PHON_REP/bin/lia_nomb2alpha $LIA_PHON_REP/data/list_chif_virgule.tab | \
$LIA_PHON_REP/bin/lia_unmotparligne
```

Le texte en entrée est tout d'abord segmenté en unités (paragraphe, phrase et mot[32]) (`lia_tokenize`). Plus qu'un simple découpage selon les espaces ou les ponctuations, un ensemble de règles codant des heuristiques ainsi qu'un lexique de référence permet une *tokenisation* pertinente pour le français. Ceci est justifié notamment pour gérer les problèmes de tiret (mots composés) comme « `c'est-à-dire` » qui forme un seul et même mot. Bien que l'apostrophe, en général, au contraire du tiret, distingue les mots comme « `qu'un` », il existe quelques exceptions comme « `l'un` » qui ne forme qu'un mot (71). Ainsi ces exceptions sont représentés dans le lexique de référence de LIA_phon `lex10k` (ou `lex80k`[33]) comme ceci :

```
l'un PINDMS 38093 l'un [34]
```

Les règles de *tokenisation* permettent de traiter la majeure partie des cas, incluant les abréviations (se terminant par un point, comme M.). Par la suite `lia_sentence` découpe le texte en phrases. Ici s'achève la phase de segmentation. Le nettoyage du texte commence par gérer les problèmes de capitalisation. Ceux-ci peuvent avoir des conséquences directes sur l'étiquetage et donc la phonétisation (ex. les sigles : DOS et dos, est-ce un sigle ou un nom masculin ? Doit-on l'épeler ou le lire ?). LIA_phon dispose des règles suivantes : (1) si la phrase entière est en majuscule, elle est convertie en minuscule à l'aide du ré accentueur `lia_taggreac`. (2) si dans une phrase en minuscule, un mot commence ou est entièrement en majuscule, il reste inchangé. (3) la majuscule du mot de début de phrase est convertit en minuscule puis ré accentué si le mot est présent dans le lexique de référence (ex. `Etait -> etait -> était`). L'étape d'après consiste à transcrire les chiffres romains et arabes en lettres (`lia_nomb2alpha`). Enfin le script `lia_unmotparligne`, comme son nom l'indique, affiche un mot par ligne.

III.2.1.2 L'ETIQUETTAGE MORPHOSYNTAXIQUE

Ce module consiste à attribuer à chaque mot une étiquette morphosyntaxique en vue de la phonétisation. Disposant d'un jeu de 105 étiquettes, l'étiqueteur statistique a été entraîné sur un corpus labélisés par un expert phonéticien. Il est basé sur un model triclasse (72). Ce dernier est basé sur les principes suivants : (1) la probabilité d'un couple (mot, POS), i.e. la probabilité que le mot soit associé à ce POS, est dépendant des deux mots précédents. (2) la probabilité d'apparition du mot dans un contexte ne dépend que de son POS. (3) la probabilité d'apparition du POS dépend uniquement des POS précédents (72). L'algorithme de Viterbi permet de déterminé la séquence d'étiquettes ayant la probabilité la plus grande. L'étiqueteur comporte en outre deux fonctions distinctes : la ré-accentuation des mots en majuscule (`MARCHE` = `marche` ou

---

[32] La segmentation en paragraphe permet de garder une structure logique, mais également de distinguer ce qui est du corps du texte ou du titre. La segmentation en phrase est essentielle à la génération de prosodie (11).

[33] Il est en effet possible de choisir le lexique lex80k de 80 000 mots plutôt que celui de 10 000, simplement en lançant les commandes :

```
make lex80k
setenv LIA_PHON_LEX lex80k
```

[34] `[mot] [POS] [nombre du couple mot/POS dans le corpus d'entraînement] [mot]`

marché ?) et la gestion des homographes hétérophones. Le module est traité par le biais du script lia_taggreac.

```
$LIA_PHON_REP/bin/lia_quicktagg \
   -lextag $LIA_PHON_REP/data/lm3classe.arpa.sirlex \
   -morpho $LIA_PHON_REP/data/model_morpho \
   -lexgraf $LIA_PHON_REP/data/$LEXFILE.sirlex \
   -pmc $LIA_PHON_REP/data/$LEXFILE \
   -ml $LIA_PHON_REP/data/lm3classe.arpa | \
$LIA_PHON_REP/bin/lia_rajoute_lemme_ecg $LIA_PHON_REP/data/$LEXFILE.sirlex \
   $LIA_PHON_REP/data/lm3classe.arpa.sirlex      $LIA_PHON_REP/data/$LEXFILE
$OPT_PMC
```

Les exécutables lia_quicktagg et lia_rajoute_lemme_ecg utilisent les règles d'étiquetage comme décrites en Annexe, An.11. Pour la ré-accentuation, l'étiqueteur syntaxique, entraîné sur un corpus de 260 000 mots tirés du journal «Le Monde » est suffisant pour lever les ambiguïtés. Chaque mot possédant un accent est associé dans le lexique à ses formes sans accents. Ainsi selon l'étiquette apposé à la forme non accentuée, il suffit de la remplacer par le couple (mot accentué/POS) correspondant (ex. LE MARCHE -> [le, DET] [marche, NMS] -> [le, DET] [marché, NMS]). De même, la majeure partie des homographes hétérophones sont traités par l'étiquetage ([couvent, verbe] ≠ [couvent, NMS]). Pour les mots inconnus dont les suffixes sont –ent (prononcé dans le cas d'adverbe, omis dans le cas du verbe conjugué à la 3ème personne du pluriel) peuvent poser problème. Une fonction est alors rajoutée à l'étiqueteur, permettant si le mot n'est pas présent dans le lexique, d'analyser son suffixe et sa position contextuelle pour l'étiqueter convenablement (ce qui aura pour but de traiter l'éventuel homographe hétérophone). Pour ceux qui appartienent à la même classe syntaxique, comme fils de fer et fils du père, (NMP) des règles sur le contexte permettent de lever l'ambiguïté. La règle suivante, veut dire que si le mot fils est suivie de « de » et de « laine ⊕ fer ⊕ soie ⊕ coton ⊕ cuivre ⊕ métal ⊕ nylon » alors il est étiqueter comme FILS_FIL.

```
CONTEXT(32,<M,<"fils",c2>,<"de",c3>,<g4,c4>>,"FILS_FIL","fils de fer") ->
ou_bien(g4,["laine","fer","soie","coton","cuivre","métal","nylon"]) ;
```

III.2.1.3 L'ETIQUETAGE SEMANTIQUE

D'après la documentation de LIA_phon (11), l'étiquetage sémantique permet de nettoyer le texte de toutes ambiguïtés. Il est dit qu'il traite le cas des expressions chiffrées (numéros de téléphone, dates etc.). Dans la pratique, il fait moins que cela. Il est en effet capable de traiter les chiffres séparés par des virgules, s'il s'agit de chiffres décimaux ou non (ex. 12,8 -> douze virgule huit mais 2, 3 et 4 -> deux trois et quatre). Dans le premier cas de figure la ponctuation est oralisée tandis que dans le second elle ne l'est pas. Le contexte dans lequel se situe de l'expression chiffrée permet de déterminer la forme exacte, s'appuyant sur le lexique list_chif_virgul qui contient les mots pouvant être précédés d'un chiffre décimal (ex. millions, degré etc.). Il permet aussi de traiter convertir les chiffres romains. Néanmoins, contrairement à ce qui est écrit dans la documentation, il ne transcrit pas encore les abréviations, mais gère les sigles. Ceux-ci sont repérés grâce au module précédent de capitalisation. Si le mot en question est maintenu en lettres capitales, il est donc un sigle. Les règles de décision contenues dans le fichier desigle.pron détermine si le sigle doit être lu (ONG), prononcé (IRCAM) ou bien les deux (CD ROM). Les fichiers epeler_sig.pron et lire_sig.pron contiennent les règles de prononciations. L'étiquetage sémantique permet en outre de repérer les noms propres et d'y appliquer une prononciation dépendante du langage d'origine. A cette fin-ci l'étiqueteur a été entraîné sur un corpus de 10 000 noms propres d'origines différentes tirés du journal « Le Monde ». Le type du nom propre (prénom, nom etc.) est déterminé par son contexte. Cette information est responsable de la prononciation associée. Par exemple, s'il s'agit de deux noms propres, la probabilité d'être étiqueté en tant que <PRENOM><NOM> est supérieur. Ainsi l'origine linguistique de cet ensemble dépend de l'origine de chacun de ses composants (par ex. Steffi

Graphe -> Italien + Allemand -> Allemand). Bien sur au final, il ne s'agit pas d'appliquer aux noms une prononciation propre au langage d'origine, mais telle qu'elle aurait été si un locuteur français, sachant les noms étrangers, le prononcerait.

III.2.1.4 LA GESTION DES LIAISONS

Ce module récupère le texte nettoyé et étiqueter. Il constitue la première partie de la phonétisation. Par soucis de simplification, LIA_phon ne considère que des liaisons obligatoires ou interdites. A l'image des traitements précédents, la détection d'une liaison se fait à l'aide de considérations syntaxiques, comme la catégorie, et contextuelles (les catégories des mots l'entourant). Les règles sont contenues dans le fichier `regles_l.pro3`. Elles permettent également de gérer les problèmes de dénasalisation, la gestion des apostrophes et de quelques cas particulier (plus, Jésus-Christ, Moyen-âge, la prononciation ou non de la fin de certains chiffres comme dix). Par exemple, la règle suivante décrit une liaison obligatoire :

```
liaison(49,<<"il",c1>,<g2,c2>,m3,m4>,"","," T ","il est , il était")->
   ou_bien(g2,["est","était"]);
```

Elle signifie : lorsque le contexte de gauche est « `il` », les mots « `est` » ou « `était` », s'ils sont suivis par deux autres mots, on oralise le `T` de fin.

III.2.1.5 LES REGLES DE PHONETISATION

Jusqu'à présent, le texte n'est toujours pas phonétisé. Les règles d'étiquetage des modules précédents permettent de renseigner sur la nature des mots, et sur les règles de prononciation à appliquer. Ainsi la phonétisation s'appuie sur onze bases de données : (1) un lexique de plus de 1 000 règles de phonétisation contenus dans le fichier `french01.pron`. (2) les huit bases `propername_[1-8].pron` qui contiennent les règles de prononciation selon l'origine linguistique. (3) la base epeler_sig.pron pour la phonétisation des sigles qu'on épelle et (4) la base lire_sig.pron pour ceux qu'on lit.

`french01.pron`	
Ex. et signific.	`regle(1145,<l,"y",V>,"yy","","paye") -> ;`
	lorsque la graphie « `y` » est précédé de n'importe quelle lettre (la variable `l`) et suivie d'une voyelle (`V`), elle se prononce `/yy/` (`/j/` en SAMPA) comme dans `paye`
`propername_[1-8].pron`	
Ex. et signific.	`regle(19,<"  ","hyu",l>,"yyyou","","Hyundai")->;`
	lorsque le mot commence par « `hyu` » et est suivie par n'importe quelle lettre, on prononce cette graphie `/yyou/` (`/ju/` en SAMPA) comme `Hyundai`
`epeler_sig.pron`	
Ex. et signific.	`regle(14,<l,"f",l1>,"aiff","","ffi") -> ;`
	pour les sigles à épeler, prononcer la lettre `f` `/aiff/` (`/Ef/` en SAMPA)
`lire_sig.pron`	
Ex. et signific.	`regle(98,<l,"f",l1>,"ff","","ffi") -> ;`
	pour les sigles à lire, la lettre `f` se prononce `/ff/` (`/f/` en SAMPA)

Fig III.1. *Exemples de règles de phonétisation dans LIA_phon*

D'une manière générale, dans chaque base, les règles sont triées selon un score calculé à l'initialisation du programme, en fonction de la graphie, des contextes droite et gauche et éventuellement des contraintes grammaticales. Plus le score d'une règle est élevé, plus elle doit être traitée en priorité. De ce fait, les exceptions sont placées avant les règles communes afin

d'être traité en priorité. La graphie est tout d'abord segmentée en syllabes. On recherche d'une manière dichotomique la règle la plus approprié au niveau du mot entier Puis on teste si le mot est un sigle, un nom propre (étranger ou non). Sinon on recherche la règle appropriée aux ensembles de syllabes.

III.2.1.6 LE POST-TRAITEMENT

Le nettoyage (gestion des liaisons etc.) et les règles de phonétisation vus précédemment découlent de l'interaction entre les mots, en prenant en compte des éléments syntaxiques. Le post-traitement se soucie toujours de l'interaction entre les mots, mais cette fois au niveau des transcriptions phonétiques, ceci pour améliorer le rendu vocal final. Dans ce module sont traités les phénomènes d'élision du schwa et d'assimilation. Par exemples, la transcription phonétique de `quelques autres` est `kElk zotR`. Cependant, le `kz` est difficilement prononçable, le `k` étant une occlusive sourde non voisée et le `z` une fricative voisée. Ainsi le module de post-traitement re-phonetisera la transcription en `kElg zotR`. Les règles à la base du traitement sont contenues dans le fichier `rule_phon.pro` .Par exemple la règle

```
#regle(6,<l,"iioe",l>,"yyoe","","ingénieur")
```

signifie que tout diphone `/iioe/` (`/i9/` en SAMPA) indépendamment de son contexte doit être re-phonétisé `/yyoe/` (`/j9/` en SAMPA) comme dans ingénieur. Notons que la graphie n'est plus une syllabe ou une suite de syllabe, mais bien une transcription phonétique. Les schwas sont également traités avec la même base de règles (insertion de schwas) :

```
#regle(12,<L,"||",B>,"ee","","entre liquide et occl sonore")
```

Ici, lorsque une consonne liquide et suivie d'une occlusive sonore, comme `l` et `b`, on rajoute un `e` entre les deux pour donner `leb` (ex. `sol brillant` -> `sOl bRia~` -> `sOl3 bRia~`).
Au final, la sortie du script `lia_text2phon` pour la phonétisation d'une phrase comme Je suis ton père est. [35] :

```
<s>       ##        [ZTRM->EXCEPTION]
je        jjee      [PPER1S]
suis      ssuuii    [VE1S]
ton       tton      [DETMS]
père      ppairr    [NMS]
.         ##        [YPFOR]
</s>      ##        [ZTRM->EXCEPTION]
<FIN>     ????      []
```

III.2.2 EVALUATION

LIA_phon est un outil déterministe remarquablement fiable. Sa première version testée montrait déjà une précision[36] de 99%. Parmi les erreurs, les noms propres étrangers en représentent près de 25,6% alors qu'ils ne constituent que 5,8% du corpus. De même, l'étiquetage morphosyntaxique, pour la ré-accentuation du texte, affiche d'excellentes performances[37] : taux d'erreur de 0,52%, soit 3,39% des mots accentués dont seulement 1,78% provoqueront des erreurs de prononciation. L'expérience personnelle démontre un outil en évolution, peu de mots étrangers ont eu une transcription phonétique inappropriée. En revanche, on peut pointer le

[35] Les transcriptions phonétiques sont au format liaphon qui se caractérise par deux lettres pour chaque phonème. Bien que très adapté au français, elle doit être convertie en SAMPA pour Mary.
[36] LIA_phon fut testé en 1998 sur un corpus de 26 000 mots par l'AUF (Agence Universitaire de la Francophonie)
[37] L'étiqueteur a été testé sur un corpus de 150 000 mots.

fameux problème du `plus`, le `s` de `non plus` étant oralisé alors qu'il ne devrait pas. A l'inverse, quelques problèmes de liaison notamment pour le mot `Etats-Unis`, dont la liaison, obligatoire pourtant, n'est pas réalisé, malgré la règle

```
context(17,<m1,<"Etats",c2>,<"Unis",c3>,m4>,"","Z  ","liaison entre Etat et
Unis") -> ;
```

Un autre défaut est la gestion de l'élision des schwas. En pratique, aucun texte phonétisé à ce jour ne comportait de e muets : ils ont tous été phonétisés. Or, la synthèse vocale, pour sembler naturelle, se doit d'omettre certains –e. L'insertion de schwa elle marche très bien. En revanche le formatage n'est pas aussi complet que ce qui est décrit dans la documentation. LIA_phon ne gère malheureusement pas à l'heure actuelle le nettoyage d'abréviations, d'expressions chiffrées complexes, hormis pour la conversion « chiffres à lettres », arabes (décimaux ou non) comme romains. Nous avons tenté de phonétiser la phrase citée en exemple dans la documentation de Frédéric Béchet :

```
Dans la coll. Dupond, l'ouvrage "Pourquoi moi ?" à pour code : IV.12.14
```

Voici ce que nous avons récupérer (sortie convertie en SAMPA et mise sur une ligne) :

```
da~ la kol dypo~ luvRaZ puRkwa mwa a puR kOd katR duz katoRz
```

Un autre exemple :

```
il est 20h30     donne     il e ????
il est 20 h 30   donne     il e ve~ aS tRa~t
```

En conclusion, LIA_phon, malgré ses quelques problèmes de prétraitement, d'élision de schwa et de phonétisation d'exceptions, demeure un excellent outil, rapide et très fiable affichant des taux d'erreurs plus que convenables pour un système graphème-phonème. La facilité d'ajout des règles de phonétisation ou d'étiquetage le rendent facilement évolutif, donc facile à améliorer. Hormis les qualités et défauts intrinsèques au programme, c'est surtout sa portabilité (uniquement valable sous environnement UNIX) qui fait défaut au projet d'intégration du français à Mary. Cependant, le moteur de phonétisation étant codé en langage C, il n'est pas impossible de porter LIA_phon à Windows, pour peu que l'on recode les scripts de communication entre module en langage compréhensible par le système d'exploitation, ou que l'on porte le système entier en Java (comme HTS).

III.3 CONCEPTION DES MODULES DE TALN

Comme énoncé précédemment, les composants de TALN sont les seuls nécessaires à la création d'un nouveau langage pour Mary. C'est-à-dire, créer le module qui prend en entrée du `RAWMARYXML` et sort du `PHONEMES`. Celui-ci doit ensuite être intégré au système Mary selon la procédure que nous verrons dans la section III.4. Dans un tout premier temps, il était envisagé de recréer un système graphème-phonème complet de zéro, sans suivre la procédure du *NewLanguageSupport* (c'est-à-dire paramétrer les modules génériques `JTokeniser` et `JPhonemiser`). Des modules de prétraitement pour nettoyer le texte des motifs de dates, heures, numéros de téléphone, unités de mesures, devises et abréviations, inspirés de leurs équivalents pour la langue allemande, ont été crées et tester avec succès pour la majeure partie d'entres eux. Cependant, par contraintes techniques et par manque de temps, il n'a pas été possible de continuer le développement de cette façon. Compte tenu du fait que nous disposions de LIA_phon pour la phonétisation complète, et avec l'aimable autorisation de son auteur (license GNU), il suffit simplement d'utiliser ce dernier à partir de Java à l'aide d'appels systèmes. Très concrètement, notre phonétiseur est composé de trois classes : (1) la première, que l'on peut considérer comme un `main`, se nomme `PhonemiserFR`. C'est elle qui récupère le `RAWMARYXML`

fournit par le module `TextToMaryXML`, appelle LIA_phon et crée le MaryXML de sortie, `PHONEMES`. (2) la seconde, `getPOSorPhonemFR`, transforme la sortie de LIA_phon afin d'être exploité par Mary, en récupérant la transcription phonétique (convertit en SAMPA), les mots et les étiquettes morphosyntaxique. (3) enfin la classe `preprocessFR` effectue le prétraitement complémentaire à LIA_phon en amont de la phonétisation. Même si ces classes ne font pas appel à des théories de programmations complexes, elles sont cruciales au bon fonctionnement du système en général. Il suffit d'une erreur très simple, comme par exemple inclure par erreur un silence `/_/` en début de la transcription phonétique pour littéralement fausser tout le reste du processus d'importation de voix (problèmes d'alignement entre la transcription et les échantillons sonores). L'architecture client/serveur de Mary doit aussi été respectée. Chaque classe est *thread-safe*, ce qui signifie qu'elle peut être instanciée autant de fois que possible (c'est-à-dire exécutée par plusieurs processus) sans problèmes. Pour cela, aucune variable globale n'a été utilisée. En effet, les variables utilisées par les programmes vivent à l'intérieur de chaque processus. Lorsqu'une variable globale est crée, elle est utilisée par tous les processus. Le risque de conflit devient alors important surtout si celle-ci est accessible en écriture (modification par l'un, problème d'accès par l'autre, deadlock). Le fonctionnement interne du module peut être présenté de manière simplifiée par le schéma suivant.

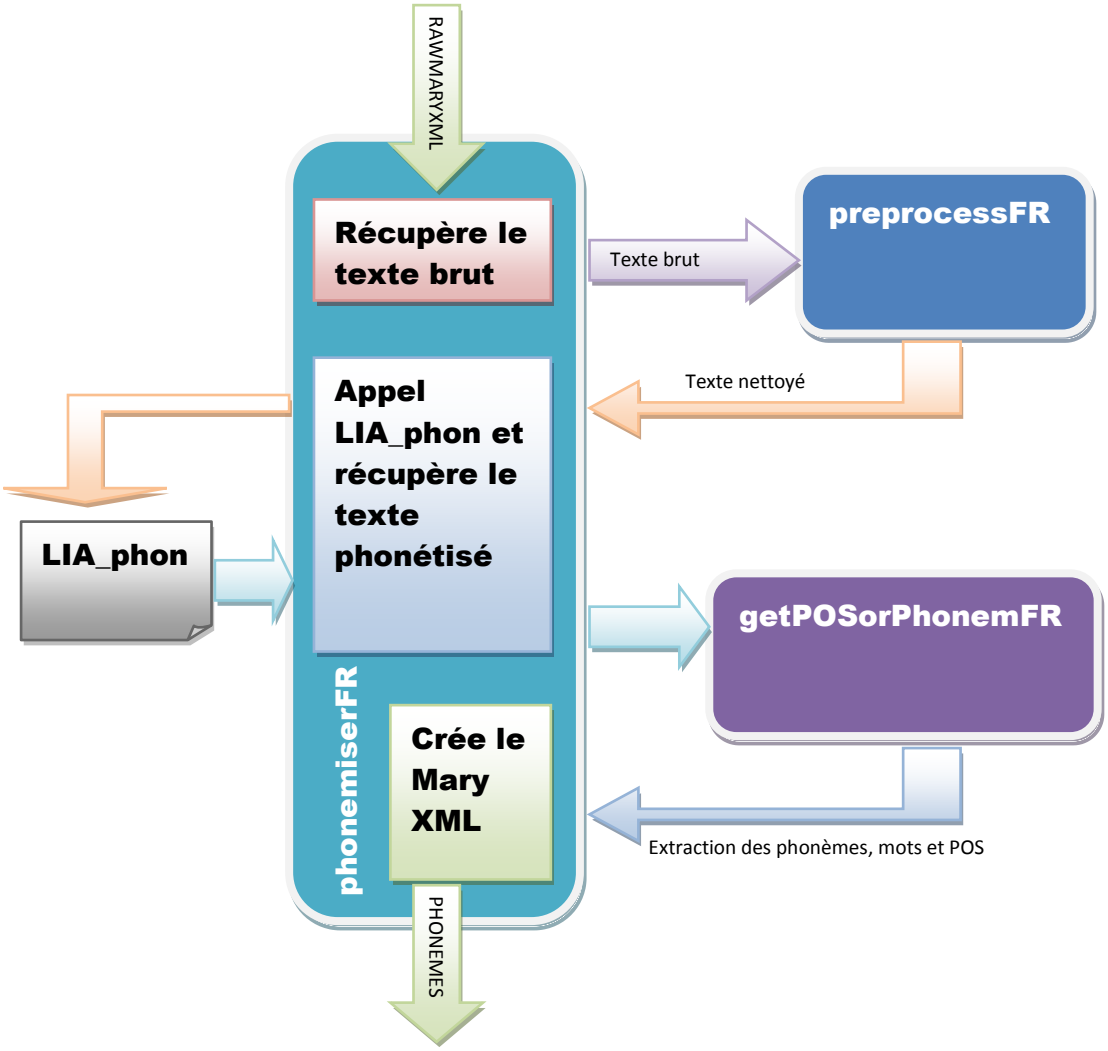


Fig III.1. *Vue d'ensemble des composants de TALN pour le français*

III.3.1 PHONEMISERFR

La classe `PhonemiserFR`[38] est le module de phonétisation de notre système. Elle hérite de la superclasse `InternalModule` et possède donc un format d'entrée (`RAWMARYXML`), un format de sortie (`PHONEMES`) et une valeur de `locale`[39] (`fr`). Sa méthode `process` (voir en Annexe, An.12.) est la méthode principale et indique le nombre de tâches effectués dans l'ordre. La première d'entre elle correspond à analyser le document MaryXML d'entrée. Par choix d'architecture, et parce qu'il est préférable de n'appeler qu'une seule et unique fois LIA_phon pour phonétiser le texte en entier, notre module ne travaille jamais avec du MaryXML, mais avec du texte brut (pour le nettoyage et la phonétisation). Il est donc primordial d'extraire ce dernier du `RAWMARYXML`. La méthode `readFromXML` prend en entrée l'argument de la méthode `process`, à savoir une `MaryData`[40] `d` et retourne le texte (`textInput`). Celui-ci est ensuite nettoyé à l'aide de la méthode preprocess. Contrairement à LIA_phon qui effectue le nettoyage après la segmentation, il n'y a pas d'autre choix ici que de le nettoyer avant. Il s'agit pour ce faire d'instancier la classe preprocessFR (que nous étudierons dans la section III.3.2) et lui passer comme argument `textInput` pour récupérer le texte nettoyé, `textClean`. Il est maintenant prêt à être phonétisé par LIA_phon. Afin de maintenir l'aspect « universel » du programme, nous prévoyons plusieurs versions de LIA_phon, valable pour UNIX, Mac et Windows. `process` fait appel à la méthode `whichOS` qui selon le système d'exploitation, renvoie la commande adéquate pour lancer le programme externe associé. La machine virtuelle Java renseigne directement sur le système sur lequel elle est installée en retournant un `String`, à l'aide de la ligne de code suivante.

```
String os = System.getProperty("os.name").toLowerCase();
```

En testant la valeur de `os`, le programme sait quelle commande lancer pour l'exécution de LIA_phon. Par exemple :

```
else if ((os.indexOf("nix")>-1) || (os.indexOf("nux"))>-1)
{
    //Unix command
command1 = path+"/script/lia_text2phon";

}
```

Dans l'état actuel des choses, il n'existe pas encore de version de LIA_phon pour Windows. Mac OS étant un système d'exploitation basé sur UNIX, il est fort probable que les appels systèmes fonctionnent dans ce cas (non testé). Notons ici la variable `path` (en argument de la méthode whichOS) qui représente le chemin absolu jusqu'au répertoire `/lia_phon`. Elle est définie en début de `process` sous le nom de `LIA_PHON_REP` et passer en argument à `whichOS` comme suit :

```
//LIA_Phon path
String LIA_PHON_REP = new String(MaryProperties.maryBase() +
"/lib/modules/fr/lia_phon");
(…)
//Which OS?
String Command = whichOS(LIA_PHON_REP);
```

[38] La Javadoc de cette classe est disponible en Annexe, An.13. Toutes les méthodes, leurs résumés, arguments et valeurs de retour y sont indiqués

[39] `locale` est essential pour indiquer au système entier la langue qu'a choisi de traiter l'utilisateur Elle conditionne ainsi la chaîne de traitement, i.e. les modules adéquats à utiliser.

[40] Une `MaryData` correspond à une forme quelconque de donnée exploitable par le système Mary (voir en Annexe, An.5. la liste des différents `MaryData`). Ici la `MaryData` correspond au MaryXML `RAWMARYXML`.

Par défaut il est décidé de conserver tous les programmes externes nécessaires au français dans le répertoire `$MARY_BASE/lib/modules/fr` où `MARY_BASE` est le chemin absolue jusqu'au répertoire de l'installation de Mary (obtenus avec `MaryProperties.maryBase()`). Une fois la commande définie, l'appel système de LIA_phon peut être effectué par la méthode `callLIA`. Celle-ci a besoin de trois arguments : (1) la commande elle-même (`Command`), (2) le texte à phonétiser (`textClean`) et (3) le chemin d'accès `LIA_PHON_REP`. Ce dernier étant déjà inclus dans la commande, il est normal de penser qu'il est inutile de s'en servir à nouveau. Cependant, pour l'exécution de ses scripts et même de ses programmes sources, LIA_phon a besoin d'une variable d'environnement `LIA_PHON_REP`. Comme énoncé précédemment, l'usage de variables globales est à proscrire dans la plupart des cas pour garantir que le programme soit *thread-safe*. C'est pourquoi elle n'est pas implémentée en tant que variable globale, bien qu'il est possible que cela ne cause aucun problème, n'étant que disponible en lecture. Concrètement, voici comment fonctionne la méthode `callLIA`. Elle crée l'instance unique de la classe `Runtime` nécessaire au lancement de toute application externe à Java. Un processus natif `p1` est renvoyé par la méthode statique `exec()`. Il permet le lancement de la commande déterminée précédemment, avec pour argument la variable d'environnement `LIA_PHON_PATH`.

```
//Create the runtime to execute lia_phon for Windows or for UNIX
Runtime runtime = Runtime.getRuntime();

//Create the process to execute lia_phon
Process p1 = runtime.exec(Command, env);
```

Il est possible de communiquer avec le processus p1 directement depuis Java à l'aide de trois flux : (1) Le programme envoie le texte à phonétiser au format `ISO8859-1` [41] à l'entrée standard de `p1` (`stdin`) à travers un flux de sortie `OutputStream`, récupérable par la méthode `getOutputStream()`.

```
//Input of the process p1, ie the text to phonetize
PrintWriter toP1 = new PrintWriter(new
OutputStreamWriter(p1.getOutputStream(), "ISO-8859-1"), true);
```

(2) Le texte phonétisé est ensuite récupéré depuis la sortie standard (`stdout`) du processus à travers un flux d'entrée `InputStream` que récupère la méthode `getInputStream()`. L'information dudit flux est ensuite recomposée en `String` et retourné par la méthode `callLIA`.

```
//Output of the process p1, ie the text phonetized
BufferedReader fromP1 = new BufferedReader(new
          InputStreamReader(p1.getInputStream(), "ISO-8859-1"));
```

(3) Il existe également un flux d'entrée renseignant sur les messages d'erreurs provenant de l'application externe ou du système d'exploitation (chemin inexistant, commande non valide) et récupérable par la méthode `getErrorStream()`. Si dans la phase de développement il fut plus que nécessaire (surtout pour déterminer la bonne commande à écrire, bien différente dans la syntaxe que celle à entrer dans le terminal), il n'est pas forcément justifier de la garder dans le code finalisé. Une fois que le texte phonétisé au format liaphon est retourné à Java, il doit être encore formaté pour la création du document XML grâce à la méthode `convert2MaryXML()`. A l'intérieur de celle-ci est instanciée la classe `getPOSorPhonemFR` (détaillée,dans la section III.3.3) qui analyse la transcription phonétique en sortie de LIA_phon pour extraire ligne par ligne ce qui est du mot, de la transcription phonétique pure (convertie en SAMPA) et de l'étiquette morphosyntaxique. Chacune de ces valeurs sont stockées dans des tables respectives (`WOR`, `PHON` et `POS`). Elles servent à créer le XML, de sorte que chaque *token* i du document se présente comme suit (voir en Annexe, An.7. le MaryXML `PHONEMES` complet de la transcription phonétique de « `Je suis ton père` ») :

[41] Seul format d'encodage bien compris par LIA_phon, nécessaire pour les accents.

```
<t ph=PHON[i] pos=POS[i]>
WOR[i]
</t>
```

convert2MaryXML retourne le document XML marydoc à la méthode process. Enfin, celle-ci le transforme en MaryXML PHONEMES pour le français. On crée un objet MaryData ayant pour argument MaryDataType.PHONEMES et locale.FRENCH. puis on lui applique la méthode setDocument(doc) pour compléter le MaryXML avec le document XML marydoc. Finalement, conformément à toute méthode process de la superclasse InternalModule, PHONEMES est retourné puis récupéré par le module générique suivant (génération symbolique de prosodie, ProsodyGeneric).

III.3.2 PREPROCESSFR

LIA_phon n'effectuant pas le nettoyage complet du texte, deux choix se présentent. Ajouter des règles de phonétisation et compléter les lexiques directement dans celui-ci ou coder à partir de zéro des modules en Java destiné à cette tâche. Parce qu'il est préférable de tout coder en Java par soucis de portabilité, et dans l'éventuel portage de LIA_phon en Java, la deuxième solution fut choisie, le nettoyage en amont de la phonétisation. La classe preprocessFR ne se charge en réalité que d'instancier des classes associée chacune à un traitement bien spécifique. Au stade actuel de développement, la classe appelle, dans l'ordre : (1) la classe sms2alpha chargée de nettoyer le langage SMS. De nos jours, bien que la pratique soit quelque peu controversée, il n'est pas rare de voir des textes en langages SMS ailleurs que dans le contexte du téléphone portable (ex. sur Internet, forums etc.). Pourtant, aucune solution, commerciale ou non, ne dispose d'un tel traitement, malgré la « nécessité » d'une technologie appropriée. On dénombre parmi les potentielles applications susceptibles d'en profiter, la synthèse vocale à partir de texto dont mettent à disposition les répondeurs téléphoniques (lors de l'envoi d'un message à une ligne fixe), ou encore pour les logiciels de messagerie instantanée (Windows Live Messenger, Facebook, Gmail etc.) où les utilisateurs communiquent presque exclusivement par langage SMS. Concrètement, ce dernier est en majeur partie constitué d'abréviations caractérisé par un faible nombre de mots et la quasi-absence de voyelles (ex. bjr -> bonjour), de langage « phonétique » (ex. c facil -> c'est facile), d'insertion de lettres en majuscules et de chiffres à prononcer (ex. 1Tré100 -> intéressant). (2) la classe abrev2alpha qui s'occupe de réécrire les abréviations en mots complets. (3) devis2alpha pour convertir les symboles monétaires et (4) mesu2alpha pour les unités de mesures. Ces modules de prétraitement sont triviaux et consiste à l'heure actuelle à parcourir un lexique [42] de mots et transcriptions correspondantes. Le texte d'entrée, d'abord segmenté de manière rudimentaire en fonction des espaces, est plusieurs fois soumis aux même méthodes, c'est-à-dire parcourir le lexique, chercher les entrées et les remplacer par les transcriptions. Il aurait été plus judicieux, si le développement en restait là, de n'avoir qu'une seule et unique base de données à ne parcourir qu'une seule fois. Mais les modules demandant plus de traitement (notamment pour le module de nettoyage de langage SMS), nous avons séparé les modules, en vue d'une amélioration future. D'autres modules de prétraitement (dates et heures) utilisant des expressions régulières pour repérer des motifs d'expressions chiffrés sont en cours de développement et ne sont pas encore fonctionnels. Quoiqu'il en soit, les lexiques suffisent à traiter la majeure partie des cas.

III.3.3 GETPOSORPHONEMFR

La transcription en sortie de LIA_phon n'étant pas directement exploitable par le système, cette classe s'occupe du formatage pour la création du MaryXML grâce à sa méthode

[42] Les lexiques sont représentés dans des HashMap, tables de données utilisées en Java pour le remplacement de mots.

`phonemSeek()`. Elle est instanciée par la classe `PhonemiserFR`, depuis la méthode `convert2MaryXML()`. Dans un premier temps, le `String` en sortie de LIA_phon est analysé ligne par ligne. Chacune d'entre elles est divisée en trois parties en fonction des espaces (une ligne de la transcription de LIA_phon étant mise sous la forme `[mot] [phonèmes] [POS]`) et stockées dans un tableau de `String tab` de taille 3 (index de 0 à 2). Ainsi par exemple, pour une ligne `i`, la transcription phonétique est stockée dans le String `tab[2]`. Pour déterminer depuis `PhonemiserFR` quelle information doit être extraite, on change la valeur de l'un des attributs de `getPOSorPhonemFR`, un entier appelé `whichPart`. Cette valeur représente l'index du tableau `tab`. Par conséquent, si l'on veut extraire la transcription phonétique ou l'étiquette morphosyntaxique, il suffira depuis la méthode `conver2MaryXML` de mettre respectivement `whichPart` à 1 ou à 2. Puis dans la méthode `phonemSeek()` selon cette valeur, on récupère l'information `tab[whichPart]` et on lui applique le traitement en conséquence : (1) la transcription phonétique est transformé au format SAMPA à l'aide d'une `HashMap` qu'on nomme `LIA_PHON_TO_SAMPA` représentant les phonèmes au format liaphon et leur équivalent au format SAMPA (voir Annexe, An.1. les phonèmes dans ces deux différents formats). Etant donné que les phonèmes de liaphon sont tous composés de deux caractères, il suffit d'analyser chaque mot phonétisé toutes les deux caractères, puis remplacer par l'entrée correspondante dans la `HashMap` grâce à la méthode `get()`.

```
String sampaPhoneme = LIA_PHONE_TO_SAMPA.get(liaPhonPhoneme);
```

(2) Les étiquettes morphosyntaxiques n'ont pas besoin d'être transcrit dans un standard, qui pour l'instant n'existe pas dans Mary. De plus elles ne sont utilisées qu'en vue d'obtenir la transcription phonétique. Comme nous l'avons déjà avec LIA_phon, elles ne servent pas à grand-chose, si ce n'est à remplir le MaryXML. Par contre, les étiquettes de ponctuation sont des exceptions, car elles sont strictement nécessaires à la prédiction de fins de phrases et d'intonations par le module de génération de prosodie. Pour ce cas précis, et seulement celui-ci, les étiquettes morphosyntaxiques doivent être convertit. Pour le point, « `[YPFOR]` » devient « . » et pour toute autre symbole de ponctuation, « `[YPFAI]`» devient « , », toujours grâce aux principes de substitution très largement utilisé précédemment. Enfin, on construit un `String` en regroupant les `tab[whichPart]` de chaque ligne, que l'on retourne ensuite à la classe `PhonemiserFR`.

III.4 INTEGRATION A OPENMARY

Le développement des programmes de TALN en Java terminé, il faut encore créer les fichiers systèmes (configurations et données) et effectuer les tâches qui les intégreront en tant que module au système Mary. Il suffit pour cela de suivre le *NewLanguageSupport*. Pour rappel, comme le système de création du module de phonétisation n'a pas été respecté, cette procédure n'est pas à considérer dans son intégralité. Il est proposé en Annexe, An.15. une version adaptée au cas présent. Enfin lorsque le programme est opérationnel et intégré au système, la voix de synthèse peut être crée à l'aide de l'utilitaire d'importation de voix[43], en suivant le *HMMVoiceCreation*. Toutefois, comme énoncé dans la section II.3.3, ce dernier n'est pas complet et comporte de nombreuses subtilités. Bien conscient que tous les problèmes potentiels n'ont pas été rencontrés, une mise à jour est proposée en Annexe An.9. apportant des solutions à ceux-ci. Pour crée la voix de synthèse, la première étape consiste à préparer le corpus. Chacune de ses lignes doit être contenue dans un seul fichier texte avec le fichier audio correspondant (ex. `utt_1.txt` et `utt_1.wav` etc.). Puis le composant `AllophoneExtractor` extrait toutes les transcriptions

[43] Interface graphique appelant des composants aux fonctions spécifiques pour la création d'une voix. Elle est conçue, en théorie, de manière à ce qu'un utilisateur non-expert puisse s'en servir facilement.

phonétiques du corpus au format MaryXML, `ALLOPHONES` (informations phonétiques, découpage de ceux-ci en syllabes et phonèmes et informations prosodiques). Ceci est réalisé grâce aux composants de TALN précédemment développés lancés à travers le serveur Mary. Les transcriptions sont ensuite alignées avec les fichiers audio du corpus grâce à l'ehmm labeller, programme externe du Festvox toolkit. Il se sert de MMC à densité de probabilité continue (73) dont la séquence cachée des états, déterminée par l'algorithme de Baum-Welsh (74), représente la segmentation des phonèmes. En sortie de ce traitement, les fichiers .lab sont crées et représentent chaque phonème de la transcription avec son temps de fin `t` (en seconde) dans le fichier audio. Voici par exemple le mot confiance dans le fichier lab correspondant. Le phonème `/o~/` est ici repéré entre 3,525s et 3,58s dans l'enregistrement.

```
3.525000 38 k
3.585000 39 o~
3.655000 40 f
3.755000 41 j
3.885000 42 a~
4.110000 43 s
```

On définit dans le fichier `features.txt` l'ensemble des paramètres acoustiques que l'on veut extraire à partir du MaryXML `ALLOPHONES` (paramètres ToBI…). Ils sont déterminés d'abord sous forme de MaryXML `ACOUSTPARAMS` en utilisant le module de génération de paramètres acoustiques `AcousticModeller`, puis sont ensuite stockés sous forme de liste dans un fichier .pfeat (`phone features vector`). Pour chaque phrase, les fichiers .pfeat et .lab correspondants sont comparés par le composant `PhoneLabelFeatureAligner`. L'idéal serait que tous soient « alignés » ensemble. Le cas échéant, le programme tente un alignement automatique, soit en supprimant les pauses du fichier .lab non présents dans le fichier .pfeat et en étirant les durées des phonèmes avoisinant, soit, s'il en manque, en en rajoutant de durée égale à zéro. Dans l'éventualité où certains fichiers ne sont pas alignés, il est possible de corriger manuellement le fichier `RAXMARYXML` de la phrase, ou directement son fichier .lab. Une fois cette phase achevée, l'outil appelle HTS pour entraîner le corpus. Le composant `HMMVoiceMakeData` commence, dans un premier temps, à extraire les paramètres acoustiques des fichiers audio : coefficients MGC[44] et log de la fréquence fondamentale `lf0`. Mary rajoute une extension pour déterminer aussi l'intensité (`str`), les coefficients de Fourrier[45] (`mag`), et les variances globales (`gv`). Ce processus prend plusieurs heures et peut aboutir sur un erreur. Si cela devait arriver, il est possible de relancer l'opération, en indiquant au système de ne pas recalculer les paramètres qui l'ont déjà été à l'aide d'un booléen pour chacun d'entres eux (ex. `HMMVoiceMakeData.makeSTR = 0`). Enfin, on peut lancer la phase d'entraînement grâce au composant `HMMVoiceMakeVoice`. A l'instar du précédent composant, le traitement prend beaucoup de temps et il est possible de répéter les étapes désirées avec le même système de booléen. Finalement, la voix crée, on l'intègre au système Mary avec le `HMMVoicePackManager`. Celui-ci crée un fichier `mary-myvoice-fr-4.3.0-component.zip` dans le répertoire `$MARY_BASE/download`. Pour l'installation sur n'importe quel machine/système d'exploitation, dans le cas où la voix n'est pas encore disponible en téléchargement, il suffira de copier/coller (1) le .config présent dans cet archive dans le répertoire `$MARY_BASE/conf/`, (2) le dossier `lib/voices/myvoice` dans le dossier `$MARY_BASE/lib/voices` et (3) le fichier `$MARY_TTS/download/mary-myvoice-fr-4.3.0-component.xml` dans le répertoire `$MARY_BASE/installed`. Enfin on recompile (`ant`) et on redémarre l'ordinateur (voir en Annexe, An.15 une capture du client Mary avec la voix française).

[44] Les coefficients MGC sont préférés aux coefficients MFCC car il est démontré qu'une analyse MGC est plus performante qu'une analyse MFCC et LPC pour la reconnaissance vocale (80).
[45] Pour la synthèse à excitation mixte.

III.5 EVALUATION

La voix de synthèse crée, nous pouvons procéder à l'évaluation du système entier. Celle-ci se déroule en deux phases : la justesse et la performance du traitement syntaxique, puis la qualité, évaluée selon des critères subjectives, des voix MMC françaises synthétisées à partir des corpus LPP et PolyVar.

III.5.1 LE TRAITEMENT NLP

Hormis les défauts et qualités relatifs à LIA_phon, dont principalement le fait d'être uniquement disponible pour environnement UNIX, le module de phonétisation `PhonemiserFR` présente des caractéristiques, certes perfectibles, mais non négligeable. La phase de nettoyage du texte, en dehors du fait que toutes les classes ne sont pas encore fonctionnelles, se résume pour le moment à des lectures et substitutions dans des dictionnaires de mots et de transcriptions. Il serait plus judicieux : (1) pour le module de nettoyage de langage SMS d'implémenter des règles de phonétisation adaptées accompagnées de lexique d'exception pour les mots inconnus (ex. lorsque le contexte gauche de la consonne `t` est une consonne quelconque et que son contexte droit est nul, alors prononcer la consonne plus `ent`, comme `pdt -> pendant`. Une exception serait `slt -> salut` et non `salent`). On pourrait très bien envisager d'entraîner le programme sur un corpus de SMS retranscrit, ce qui, bien évidemment, n'existe pas. (2) pour les abréviations, en revanche, il est difficile d'imaginer quelconques règles pour les traiter, malgré leur relativement grand nombre. (3) De même pour le traitement des devises et unités de mesures. Cependant pour ces deux derniers, le nombre étant relativement restreint, le dictionnaire de transcriptions reste la plus simple et la meilleure solution. Toutefois, bien que rudimentaire, cela suffit dans la grande majorité des cas. La principale limite de cette technique est due au fait que chaque mot est d'abord segmenté en fonction des espaces, puis recherché dans le lexique. Par conséquent, si `34 dB` sera transcrits en `34 Décibel` [46] grâce à la segmentation qui distingue `34` de `dB`, il n'en est pas de même pour `34dB` qui sera alors reconnu comme étant un seul et unique mot. Tout ceci demeure néanmoins facilement corrigeable en distinguant les chiffres des lettres, comme le fait LIA_phon. Mais alors on alourdit le système, en faisant une première segmentation selon les espaces, puis en regardant chaque caractère qui compose le mot, sachant que le programme externe fera le même traitement en aval. Et c'est là que le bât blesse. Ces redondances ralentissent le système à outrance, alors que la phase de nettoyage implémentée pourrait être réalisée en même temps que celle proposée par LIA_phon. De plus, l'appel système par la classe `Runtime` demande elle aussi un temps de traitement trop long. Malheureusement, à la date d'écriture de ce rapport (`5 Août 2011`), il subsiste un bug difficilement explicable dans la classse `getPOSorPhonemFR` pour extraire les informations de la transcription phonétique au format liaphon. Chaque ligne de celle-ci (revoir si nécessaire sa forme en section III.2.1.6) est segmenté par la méthode `split()` selon le caractère souhaité (ici l'espace) et stocke les groupes dans une table de taille 3. Pour les récupérer, il suffit donc tout simplement d'accéder aux entrées de ce dernier : l'index 0 pour le mot, l'index 1 pour l'étiquette morphosyntaxique et l'index 2 pour les mots phonétisés Si la récupération des deux derniers se fait sans aucuns soucis, accéder à la dernière renvoie une exception. Ce problème, certes très primaire, oblige par conséquent à lancer un deuxième appel système au script `lia_nett` de LIA_phon pour ne récupérer cette fois que le texte segmenté en mots. Ajouter à cela toutes les recherches/substitutions, pourtant implémentées de la manière la plus optimale, pour le nettoyage du texte, de la transcription phonétique et pour la conversion au format SAMPA et toutes les maladresses due au style même de programmation, font que le module de phonétisation prend en moyenne 130 ms pour phonétiser (`TEXT -> RAWMARYXML`) une phrase d'environ 70 caractères, alors que la moyenne dans les autres langues est de 50 ms. En prenant comme moyenne pour le temps de synthèse 150 ms, le traitement Text-To-Speech durerait presque 300 ms, soit près d'un tiers d'une seconde, ce qui

[46] Le -é de Décibel ne constitue pas une faute d'orthographe volontaire, mais aide à la phonétisation pour ne pas être phonétisé `/d2sibEl/`.

est inacceptable. Les solutions pour raccourcir ce temps serait de limiter à un l'appel système à LIA_phon et peut être même de bannir le nettoyage en amont, le système Mary n'étant pas destiné à être commerciale, mais un outil pour l'aide à l'éducation et la recherche. Un autre défaut perfectible, mais beaucoup plus complexe à solutionner, est la génération de prosodie. Le fait d'avoir utilisé un module générique se servant du standard ToBI adapté à l'anglais n'est pas la meilleure solution. Enfin, la transcription phonétique n'est pas segmentée en syllabes comme elle devrait l'être (voir en Annexe, An.7.). Chaque mot est représenté comme composé d'une seule syllabe, à savoir lui-même. Cela entraîne inévitablement une prononciation et une prosodie légèrement faussées. Mais de manière globale, les MaryXML en sortie sont tout à fait dans les normes, que ce soit au niveau du contenu phonétique et prosodique ou au niveau de la forme même, ce qui était l'objectif premier. D'autre part, même si le programme ne peut être que disponible sous UNIX, conséquence direct lié à LIA_phon, l'architecture client-serveur de Mary permet l'installation du système dans n'importe quel serveur Linux ou Mac, puis d'être accessible à partir de n'importe quelle machine Windows ou autre, en ouvrant un client Mary et en échangeant des requêtes http avec le serveur.

III.5.2 LA QUALITE DES VOIX

Nous avons tenté de créer deux voix, l'une basée sur le corpus LPP, l'autre sur le PolyVar. Pour rappel, le premier était constitué en théorie de 516 phrases phonétiquement équilibrées. Dans la pratique, son nombre a été rabaissé à 510 (fichiers sons vides) et l'alignement avec la transcription phonétique ne s'est fait que pour 389 fichiers (fichiers audio ne correspondant pas avec la transcription). De ce fait, lors de la phase d'entraînement, HTS renvoi l'erreur suivante :

```
ERROR [+2121]  HInit: Too Few Observation Sequences [0]
```

Cela signifie que le corpus est pauvre en contexte, et qu'il existe probablement des allophones à entraîner qui ne sont pas dans la base de données. Par conséquent, le corpus LPP a été abandonné pour nous concentrer exclusivement sur le corpus PolyVar. Dans un premier temps, nous avons crée une voix test, avec seulement 50 phrases afin de gagner du temps et résoudre les éventuels problèmes qui pourraient se manifester après des heures de traitement. Très étrangement, l'erreur précédente n'a pas été rencontrée, ce qui sous-entend que ces 50 phrases sont plus riches en contexte que les 389 du premier corpus. Comme prévu, la qualité de cette voix test était très réduite et ne présentait aucun intérêt sauf celui d'ouvrir la voie à la véritable voix basée sur le corpus complet. En tout et pour tout, le traitement pour sa génération dépassait les 26 heures (treize pour l'alignement, trois pour l'extraction des paramètres acoustiques et dix pour l'entraînement). Pour son évaluation, on dispose d'une population de dix auditeurs[47] qui ont attribué une note de 0 à 5 pour chacun des trois critères suivants : (1) l'intelligibilité, c'est-à-dire la compréhension globale des mots. Moins on fait d'effort pour comprendre le sens de la phrase, plus la voix est intelligible. (2) le naturel de la voix, si elle se rapproche ou non de la voix humaine, si elle est agréable à écouter. Enfin (3) la similarité entre la voix synthétisée et celle du locuteur du corpus. La synthèse MMC étant réputé pour modéliser les paramètres acoustiques de ce dernier, cette composante est essentielle à l'évaluation de la voix.

Bien que nous distinguions l'intelligibilité du naturel, il n'est pas sans dire que les deux sont intimement liés de manière évidente. Ces deux critères ont été jugés avec le texte suivant :

> *"Voici la première voix de synthèse française créée pour le système Mary. La modélisation des pauses n'est pas parfaite et les problèmes de syllabification peuvent amoindrir la qualité. Cependant elle constitue un bon début avant la phase d'optimisation."*

Les auditeurs ont noté l'intelligibilité avec une moyenne égale à **4,15/5** (écart type $\sigma = 0{,}502$) et le naturel avec une moyenne égale à **2,75/5** (écart type $\sigma = 0{,}25$).

[47] De tout âge, spécialiste ou non de la synthèse vocale.

La similarité a été évaluée en comparant l'enregistrement réel et la synthèse de la phrase :

"Il savait que le peuple le soutenait, ce qui lui a donné confiance."

Les résultats sont une moyenne égale à **3,9/5** (écart type σ = 0,374).

Enfin un test du WER a été mis en place pour le texte :

"Il savait que le peuple le soutenait, ce qui lui a donné confiance. On peut enregistrer de la musique sur une bande magnétique. Une école normale prépare les jeunes gens au métier d'instituteur"

Nous avons utilisé le moteur de reconnaissance vocale de Windows[48]. Voici les résultats en sortie :

- voix de synthèse:

Il savait que le peuple soudanais ce qui lui a donné confiance on peut enregistrer de la musique sur une bande magnétique une école normale prépare les jeunes gens au métier d'instituteur

$$\text{Soit } WER = \frac{1+1+0}{34} = 1/17 \approx 0{,}06$$

- voix réelle [49] :

Il savait que le peuple le soutenait ce qu'il a donné confiance ont enregistré de la musique sur une bande magnétique une école normale pris par les jeunes gens de métier d'instituteur

$$\text{Soit } WER = \frac{4+1+1}{34} = 6/17 \approx 0{,}36$$

Les conclusions sont assez troublantes, la voix de synthèse présentant un WER six fois plus bas que la voix naturelle. Sans démontrer qu'elle l'est parfaitement, ce test met en tout cas en évidence le fait que la voix de synthèse est très intelligible, ce qui confirme les observations faites par les auditeurs. Elle manque toutefois de naturel. Ceci s'explique sans doute à cause des modélisations de durées et des pauses, ces dernières étant souvent beaucoup trop longues. Une autre raison à cela est le fait que la transcription phonétique n'a pas été segmentée en syllabes comme elle aurait du l'être. De ce fait, la prosodie n'est pas correcte, et pour beaucoup d'exemples, adopte un modèle anglais (en raison du module générique `ProsodyGeneric`). Enfin certains artéfacts (bruits de souffles, de salive, plosives) apparaissent lors des pauses. Ce problème est inhérent aux enregistrements du corpus, qui sont pourvus de pauses initiales et finales relativement longues (~ 1sec) et parfois parsemées de ces bruits. Cependant le résultat est satisfaisant et la voix de synthèse demeure très similaire à celle du locuteur. Il faut pourtant garder à l'esprit toutes les optimisations, traitement TALN compris, qu'on peut apporter pour obtenir un résultat encore plus naturel. Le plan d'action pour le prochain mois consistera à inclure les fonctions de syllabification des transcriptions phonétiques et à améliorer le style de codage pour augmenter les performances de ce dernier.

~ ✷ ~

[48] En réalité, le logiciel utilisé et sa performance ne compte pas, puisque l'intérêt d'un tel test est de mettre en évidence la différence entre le taux d'erreurs de la voix synthétisée et celui de la voix réelle.

[49] A partir du moment où « enregistré » et « enregistrer » ont exactement la même transcription phonétique, donc le même signal associé, on ne considère pas ce cas comme un mot mal reconnus.

CONCLUSIONS ET PERSPECTIVES

Au cours de ce stage de fin d'études, il fut donné la possibilité d'explorer le domaine, il faut l'avouer, quasi-inconnue pour ma part, de la synthèse vocale à partir du texte. L'objectif était d'apporter à la communauté scientifique, mais aussi à l'utilisateur non-expert, la première voix MMC française libre et prête à l'emploi. Mais au terme de ses six mois de développement, qu'avons-nous réellement apporté ? Un système perfectible mais fonctionnel, aux perspectives d'améliorations gigantesques. La première d'entre elles sera de porter LIA_phon intégralement en Java, comme il a été fait avec le moteur de synthèse de HTS par le DFKI, pour enfin assurer la polyvalence inhérente au langage de programmation. Le portage éradiquerait également tout traitement redondant lié au nettoyage du texte ou à la création du MaryXML. On ne disposerait alors que d'un seul module unifié, beaucoup plus robuste et performant. L'expérience a en effet démontré que la multiplication de programmes externes, en particulier pour le cas du *VoiceImportTool*, multipliait proportionnellement le nombre d'erreurs potentielles. Les mises à jour des tutoriels ont été rédigés dans l'espoir de ne plus avoir à rencontré des difficultés uniquement d'ordre techniques, mais qui pourtant ont fait perdre beaucoup de temps. Un temps qui sans doute aurait pu être consacré à l'étude et la conception d'un module de génération prosodique adapté et perfectionné.

C'est à n'en pas douter là l'enjeu de la synthèse vocale pour le futur, presque aussi intelligible que la voix humaine, mais encore si peu naturelle. L'émotion dans le signal, définie par des descripteurs physiques, utopie ou réalité ? L'analyse avec Praat d'échantillons sonores décrivant des contextes émotifs différents mais subtils, en collaboration avec le psychologue Sylwia Hyniewska, n'ont aboutit à aucunes conclusions assez significative pour en étayer le contenu. Le fait est que même à l'heure actuelle, aucune solution ne semble apporter une réponse satisfaisante à cette problématique. Mary tente bien, sans réel succès, de synthétiser des émotions à l'aide du programme *Emospeak* [75], mais la route est longue pour que nous puissions un jour nous tromper dans la distinction entre une voix de synthèse et la voix naturelle. Notons toutefois la puissance des MMC, qui à partir d'un corpus a priori de taille modeste, arrive à extraire les informations acoustiques lui permettant de modéliser les caractéristiques du locuteur initial. Il s'agit plus que d'une simple alternative à la synthèse par sélection d'unités dans de larges bases de données, cette dernière se révélant être trop rigide et difficilement paramétrable. Aujourd'hui les dispositifs de stockages sont largement en mesure de contenir un corpus de 3 000 fichiers sonores de cinq secondes, quantifiés/échantillonnés à 16bits/16kHz (~409 Mo), ceci restant valable même pour les systèmes embarqués.

La génération de la forme d'onde, responsable direct de la qualité du signal et donc du confort d'écoute, peut être sensiblement amélioré en utilisant la synthèse HNM plutôt que le modèle d'excitation mixte implémenté par défaut dans Mary. Les études prouvent en effet que cette technologie est supérieure à ces homologues, en particulier pour le lissage entre les unités concaténées. Il reste à ce sujet encore quelques progrès à faire, des phénomènes de discontinuité et de macro-prosodie étant encore observés. De plus, bien que Shannon garantisse un signal vocale échantillonné à 16kHz sans recouvrement spectrale, dans la pratique cet échantillonnage est encore trop faible, même pour les solutions Text-To-Speech commerciales, pour offrir un confort d'écoute satisfaisant, et naturel. Il est en effet dommage de mettre en œuvre des systèmes de générations d'onde complexes et performant lorsqu'au final la qualité du son est amoindrie par une fréquence d'échantillonnage faible. Bien conscient qu'il ne s'agit ici que de considérations purement commerciales, il est cependant assez difficile de faire autrement au risque d'avoir des temps de calcul, déjà non négligeable, trop élevé.

Car en effet, la synthèse vocale n'aura de finalité que dans un traitement temps réel et une prosodie réellement vivante, qui ne serait plus générée dans l'optique de rendre le discours supportable, mais qui serait vecteur d'informations cruciales, non syntaxiques, ni phonétiques. C'est alors que le véritable potentiel de cette technologie pourra enfin se révéler dans des projets de recherche ambitieux et aidant au développement humain. Comme la « restitution » du don de parole aux personnes muettes, ou encore une interface homme-machine révolutionnaire et plus haut niveau que jamais matérialisée dans les agents conversationnels. Enfin, terminons sur cette phrase du mathématicien Euler, qui écrivit en 1761, bien avant la découverte de l'électricité :

> « Ce serait certes une invention considérable, que celle d'une machine capable de reproduire nos paroles, avec leurs sons et leurs articulations. Je crois que la chose n'est pas impossible. »

A notre tour de croire que la réalisation de telles technologies ne sera qu'une question de temps.

BIBLIOGRAPHIE

1. Speech Synthesis. *Wikipédia.* [En ligne]
http://en.wikipedia.org/wiki/Speech_synthesis#Microsoft_Windows.

2. Sylvestre II. *Wikipédia.* [En ligne]
http://fr.wikipedia.org/wiki/Sylvestre_II#L.C3.A9gende_ou_r.C3.A9alit.C3.A9_.3F.

3. Wolfgang von Kempelen. *Wikipédia.* [En ligne]
http://fr.wikipedia.org/wiki/Wolfgang_von_Kempelen.

4. Synthèse Vocale. *Wikipédia.* [En ligne] http://fr.wikipedia.org/wiki/Synth%C3%A8se_vocale.

5. PELACHAUD, Catherine. GRETA: Embodied Conversational Agent. [En ligne]
http://perso.telecom-paristech.fr/~pelachau/Greta/.

6. Stephen Hawking. *Wikipédia.* [En ligne] http://fr.wikipedia.org/wiki/Stephen_Hawking.

7. Le projet REVOIX. *Laboratoire SIGMA.* [En ligne] http://www.neurones.espci.fr/Revoix/.

8. SCHROEDER, Marc. Mary TTS : overview. *Mary TTS.* [En ligne]
http://mary.dfki.de/documentation/overview.

9. —. Mary Presentation. [En ligne] 14 Juillet 2010.
http://www.dfki.de/~schroed/teaching/2010-enterface/mary-presentation.pdf.

10. Symboles SAMPA. *Wikipédia.* [En ligne] http://fr.wikipedia.org/wiki/Symboles_SAMPA.

11. BECHET, Frédéric. *LIA_phon, un système complet de phonétisation de textes.* 2001.

12. DOVAL, Boris. *Système de synthèse de la parole.* 2010.

13. LOVE, Nathan. La liaison. *Western Kentucky University.* [En ligne]
http://edtech.wku.edu/~nlove/Multi-handouts/liaison.htm#interdite.

14. GOLDMAN, Jean-Philippe, LAENZLINGER, Christopher et WEHRLI, Eric. La phonétisation de « plus », « tous » et de certains nombre : une analyse phono-syntaxique. [En ligne] http://www.atala.org/doc/actes_taln/AC_0009.pdf.

15. RACINE, Isabelle. Les effets de l'effacement du schwa sur la production et la perception de la parole en français. [En ligne] 2008. http://archive-ouverte.unige.ch/downloader/vital/pdf/tmp/r0d3i7h0bi0j924ngnuso587s7/out.pdf. no. L. 652.

16. Building letter-to-sound rules automatically. *Festvox.* [En ligne]
http://festvox.org/bsv/x1469.html.

17. Decision tree learning. *Wikipédia.* [En ligne]
http://en.wikipedia.org/wiki/Decision_tree_learning.

18. CART Algorithm. *SPSS Support.* [En ligne]
http://support.spss.com/ProductsExt/SPSS/Documentation/Statistics/algorithms/14.0/TREE-CART.pdf.

19. SCHROEDER, Marc et TROUVAIN, Jürgen. The german text-to-speech system Mary, a tool for Research, Development, and Teaching. *Mary TTS*. [En ligne] 2003. http://mary.dfki.de/documentation/publications/schroeder_trouvain2003.pdf.

20. BURNETT, Daniel C., WALKER, Mark R. et HUNT, Andrew. Speech Synthesis Markup Language (SSML) version 1.0. *w3*. [En ligne] 7 Septembre 2004. http://www.w3.org/TR/speech-synthesis/.

21. X-SAMPA. *Wikipédia*. [En ligne] http://fr.wikipedia.org/wiki/X-SAMPA.

22. The International Phonetic Association. *University College of London*. [En ligne] http://www.langsci.ucl.ac.uk/ipa/ipachart.html.

23. Coarticulations. *Wikipédia*. [En ligne] http://en.wikipedia.org/wiki/Coarticulation.

24. Synthèse de la parole, état de l'art. *Agence Wallonne des Télécommunications*. [En ligne] 14 Juin 2006. http://www.awt.be/web/can/index.aspx?page=can,fr,voc,200,010.

25. MOULINES, Eric et CAPPE, Olivier. Synthèse de la parole à partir du texte. [En ligne] http://ingemeca.org/docs/Genie_Informatique/Documents%20num%E9riques/Technologies%20d'acquisition/Synth%E8se%20de%20la%20parole%20%E0%20partir%20du%20texte.PDF.

26. RABINER, Lawrence Richard. Speech synthesis by rule: an acoustic domain approach. [En ligne] 12 Mai 1967. http://dspace.mit.edu/bitstream/handle/1721.1/29182/Rabiner_Lawrence_PhD_1967.pdf;jsessionid=7EE301049C7414A78BF04454656A758E?sequence=1.

27. DRYGALJO, Andrzej. Speech Processing. *SCGWWW*. [En ligne] http://scgwww.epfl.ch/courses/Traitement_de_la_parole-2010-2011-pdf/11-Speech%20Processing%202011-05-20.pdf. ELD 230.

28. BERRY, Jeff. Scottish Gaelic Text-To-Speech Synthesis. *The University of Arizona*. [En ligne] 7 Mars 2008. http://www.u.arizona.edu/~jjberry/sgttsPresentation.pdf.

29. MBROLA. *Wikipédia*. [En ligne] http://en.wikipedia.org/wiki/MBROLA.

30. Forced alignment, overview. *Institute for Signal and Information Processing*. [En ligne] http://www.isip.piconepress.com/projects/speech/software/tutorials/production/fundamentals/v1.0/section_04/s04_04_p01.html.

31. MASUKO, Takashi. HMM-Based Speech Synthesis and Its Applications. [En ligne] Novembre 2002. http://www.kbys.ip.titech.ac.jp/masuko/masuko-doctor.pdf.

32. TOKUDA, Keiichi, et al. Speech parameter generation algorithms for HMM-based speech synthesis. [En ligne] Juin 2000. http://hts.sp.nitech.ac.jp/?plugin=attach&refer=Publications&openfile=tokuda_icassp2000.pdf. 10.1109/ICASSP.2000.861820.

33. YAMGISHI, Junichi, RICHMOND, Korin et KING, Simon. Hidden Markov Model-based speech synthesis. [En ligne] http://homepages.inf.ed.ac.uk/ckiw/rpml/HMM_speech_synthesis.pdf.

34. ZEN, Heiga, et al. The HMM-based Speech Synthesis System (HTS) Version 2.0. [En ligne] http://www.sp.nitech.ac.jp/~zen/english/index.php?plugin=attach&refer=Publications%2FInternational%20conferences&openfile=zen-ssw6.pdf.

35. RESCH, Barbara. Automatic Speech Recognition with HTK. [En ligne] http://www.spsc.tugraz.at/system/files/asr.pdf.

36. The snack sound toolkit. *Royal Institute of Technology.* [En ligne] 2004. http://www.speech.kth.se/snack/.

37. TOKUDA, Keiichi, ZEN, Heiga et BLACK, Alan W. An HMM-based speech synthesis system applied to English. [En ligne] 2002. http://www.cs.cmu.edu/~awb/papers/IEEE2002/hmmenglish.pdf. IEEE SSW.

38. LIN, Chi-Yueh. Mandarin TTS using HTS Toolkit. [En ligne] 25 Janvier 2010. http://diana.ee.nthu.edu.tw/NGASR/workshop/20100125-05-Mandarin%20TTS%20using%20HTS%20toolkit.pdf. NGASR Workshop.

39. Reference manual for Speech Signal Processing Toolkit Ver. 3.4.1. [En ligne] 28 Avril 2011. http://ovh.dl.sourceforge.net/project/sp-tk/SPTK/SPTK-3.4.1/SPTKref-3.4.1.pdf.

40. LLAGUNO CARUNCHO, Cecilia. Cepstral analysis synthesis on the Mel frequency scale, and an adaptative algorithm for it. [En ligne] Avril 2008. http://www2.spsc.tugraz.at/www-archive/AdvancedSignalProcessing/SS08-SpeechSynthesis/cepstral_analysis.pdf.

41. FUKUDA, Toshiaki, et al. An adaptive algorithm for Mel-Cepstral analysis of speech. [En ligne] 1992. http://www.sp.nitech.ac.jp/~tokuda/selected_pub/pdf/conference/fukada_icassp1992.pdf. 10.1109/ICASSP.1992.225953 .

42. YOSHIMURA, Takayoshi, et al. Mixed excitation for HMM-based speech synthesis. [En ligne] Septembre 2001. http://mirlab.org/conference_papers/International_Conference/Eurospeech%202001/papers/page2263.pdf. Proc. of Eurospeech.

43. ZEN, Heiga et TODA, Tomoki. An Overview of Nitech HMM-based Speech Synthesis System for Blizzard Challenge 2005. [En ligne] Septembre 2005. http://festvox.org/blizzard/bc2005/IS052192.PDF. Proc. of Interspeech2005 (Eurospeech).

44. VANDROMME, David. Harmonic plus Noise Model for concatenative speech synthesis. [En ligne] 2005. http://publications.idiap.ch/downloads/reports/2005/vandromme_2005.pdf.

45. DI CRISTO, Albert. Interpréter la prosodie. [En ligne] 2000. Actes des 23e JEP : 13-38..

46. MILLOTTE, Séverine. Le rôle de la prosodie dans le traitement syntaxique adulte et l'acquisition de la syntaxe. [En ligne] 10 Octobre 2005. http://www.ehess.fr/lscp/persons/anne/papiersPDF/Thesis-Severine-Millotte.pdf.

47. Démo en français. *Loquendo.* [En ligne] http://www.loquendo.com/fr/demo-centre/demo-tts/francais/.

48. Démo TTS Interactive. *Loquendo.* [En ligne] http://www.loquendo.com/fr/demo-centre/demo-tts-interactive/.

49. Emotion and Memory. *Wikipédia.* [En ligne] http://en.wikipedia.org/wiki/Emotion_and_memory#Emotional_arousal_and_valence.

50. OBIN, Nicolas, RODET, Xavier et LACHERET, Anne. HMM-based Prosodic Structure Model Using Rich Linguistic Context. [En ligne] 2010. http://articles.ircam.fr/textes/Obin10c/index.pdf. INTERSPEECH 2010: 1133-1136.

51. Text-To-Speech demo. *Acapela.* [En ligne] http://www.acapela-group.com/text-to-speech-interactive-demo.html.

52. Festival Speech Synthesis System. *The Center for Speech Technology Research.* [En ligne] http://www.cstr.ed.ac.uk/projects/festival/.

53. Free TTS. *Free TTS.* [En ligne] http://freetts.sourceforge.net/docs/index.php.

54. Word Error Rate. *Wikipédia.* [En ligne] http://en.wikipedia.org/wiki/Word_error_rate.

55. YAMAGISHI, Junichi, et al. The HTS-2008 System: Yet Another Evaluation of the Speaker-Adaptive HMM-based Speech Synthesis System in The 2008 Blizzard Challenge. [En ligne] http://www.cstr.ed.ac.uk/downloads/publications/2008/HTS2008.pdf.

56. German Text-To-Speech. *ttssample.* [En ligne] 28 Juin 2011. http://ttssamples.syntheticspeech.de/.

57. Redstart. *Mary TTS.* [En ligne] http://mary.opendfki.de/wiki/RedStart.

58. NewLanguageSupport. *Mary TTS.* [En ligne] http://mary.opendfki.de/wiki/NewLanguageSupport.

59. VoiceImportComponents. *Mary TTS.* [En ligne] http://mary.opendfki.de/wiki/VoiceImportComponents.

60. VoiceImportToolsTutorial. *Mary TTS.* [En ligne] http://mary.opendfki.de/wiki/VoiceImportToolsTutorial.

61. SCHRÖDER, Marc et TROUVAIN, Jürgen. rien. [En ligne] 2003. http://mary.dfki.de/documentation/publications/schroeder_trouvain2003.pdf.

62. ToBI. *The Ohio State University Department of Linguistics.* [En ligne] 1999. http://www.ling.ohio-state.edu/~tobi/.

63. PORT, Robert. ToBI Intonation Transcription Summary. [En ligne] 1999. http://www.cs.indiana.edu/~port/teach/306/tobi.summary.html.

64. Lecture 9. *festvox.* [En ligne] http://festvox.org/11752/slides/lecture9.pdf. 11-752, LTI, Carnegie Mellon.

65. GOUBANOVA, Olga et KING, Simon. Bayesian networks for phone duration prediction. [En ligne] 29 Octobre 2007. http://peer.ccsd.cnrs.fr/docs/00/49/91/98/PDF/PEER_stage2_10.1016%252Fj.specom.2007.10.002.pdf. 10.1016/j.specom.2007.10.002 .

66. KASDAN, Lawrence et BRACKETT, Leigh. Star Wars: The Empire Strikes Back script. *IMSDb.* [En ligne] 1980. http://www.imsdb.com/scripts/Star-Wars-The-Empire-Strikes-Back.html.

67. BANNERT, Robert. Variations in the perceptual modelling of macroprosodic organization of spoken Swedish: prominence and chunking. [En ligne] 1995. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.47.8688&rep=rep1&type=pdf. PHONUM 3, 1995, 31–52.

68. ROAD, Curtis. *L'audionumérique, musique et informatique.* s.l. : Dunod, 2007. ISBN 978-2-10-051249-2.

69. TranscriptionTool. *Mary TTS*. [En ligne] http://mary.opendfki.de/wiki/TranscriptionTool.

70. PolyVar. *ELDA*. [En ligne] 2002. http://www.elda.org/catalogue/en/speech/S0046.html.

71. MAMPEY, Roger. LLIA_phon : guide pour développeur. [En ligne] 3 Avril 2004. http://download.gna.org/lliaphon/devguide.pdf.

72. MERIALDO, Bernard. Modèles probabilistes et étiquetage automatique. [En ligne] 1995. http://www.eurecom.fr/util/publidownload.en.htm?id=88. Volume 36, N°1-2, 1995 , pp 7-22 .

73. MISSAOUI, Oualid et FRIGUI, Hichem. Optimal feature weighting for the Continuous HMM. [En ligne] 11 Décembre 2008. http://figment.cse.usf.edu/~sfefilat/data/papers/MoBT9.36.pdf. 10.1109/ICPR.2008.4761076 .

74. HUGGINS-DAINES, David et RUDNICKY, Alexander I. A Constrained Baum-Welch Algorithm for Improved Phoneme Segmentation and Efficient Training. [En ligne] Septembre 2006. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.132.8804&rep=rep1&type=pdf. Proceedings of Interspeech 2006.

75. Emospeak. *Mary TTS*. [En ligne] http://mary.dfki.de/team/non-working-demos/emospeak.

76. BECHET, Frédéric. *LIA_phon*. [README] 2001.

77. HMMVoiceCreation. *Mary TTS*. [En ligne] http://mary.opendfki.de/wiki/HMMVoiceCreation.

78. N-gramme. *Wikipédia*. [En ligne] http://fr.wikipedia.org/wiki/N-gramme.

79. TOKUDA, Keiichi, et al. Multi-Space Probability Distribution HMM. [En ligne] Mars 2002. http://www.wushido.com/hmm/docs/msp_hmm.pdf. IEICE Trans. Inf. & Syst..

80. TOKUDA, Keiichi, et al. Mel-Generalized Cepstral Analysis — a unified approach to speech spectral estimation. [En ligne] Septembre 1994.

81. hts_engine. [En ligne] 7 Juillet 2011. http://hts-engine.sourceforge.net/.

82. ZHAO, Yong, et al. Measuring target cost in unit selection with KL-divergence between context-dependent HMMs. [En ligne] http://research.microsoft.com/en-us/groups/speech/0100725.pdf.

ANNEXES

AN. 1. TABLEAU DES PHONEMES AU FORMAT LIA ET SAMPA [76]

liaphon	SAMPA	Exemples
ii	i	idiot, ami
ei	e	ému, été
ee	3	ce, je, ne
ai	E	perdu, maison
aa	a	alarme, patte
oo	O	obstacle, corps
au	o	auditeur, beau
ou	u	coupable, loup
uu	y	punir, élu
EU	2	creuser, deux
oe	9	malheureux, peur
eu	@	petite, fortement
in	e~	peinture, matin
an	a~	vantardise, temps
on	o~	rondeur, bon
un	9~	lundi, brun
yy	j	piétiner, choyer
ww	w	quoi, fouine
pp	p	patte, repas, cap
tt	t	tête, net
kk	k	carte, écaille, bec
bb	b	bête, habile, robe
dd	d	dire, rondeur, chaud
gg	g	gauche, égal, bague
ff	f	feu, affiche, chef
ss	s	sœur, assez, passe
ch	S	chanter, machine, poche
vv	v	vent, inventer, rêve
zz	z	zéro, raisonner, rose
jj	Z	jardin, manger, piège
ll	l	long, élire, bal
rr	R	rond, charriot, sentir
mm	m	madame, aimer, pomme
nn	n	nous, punir, bonne
gn	J	oignon, charlemagne, agneau
uy	H	huit, lui, poursuivre
##	_	silence

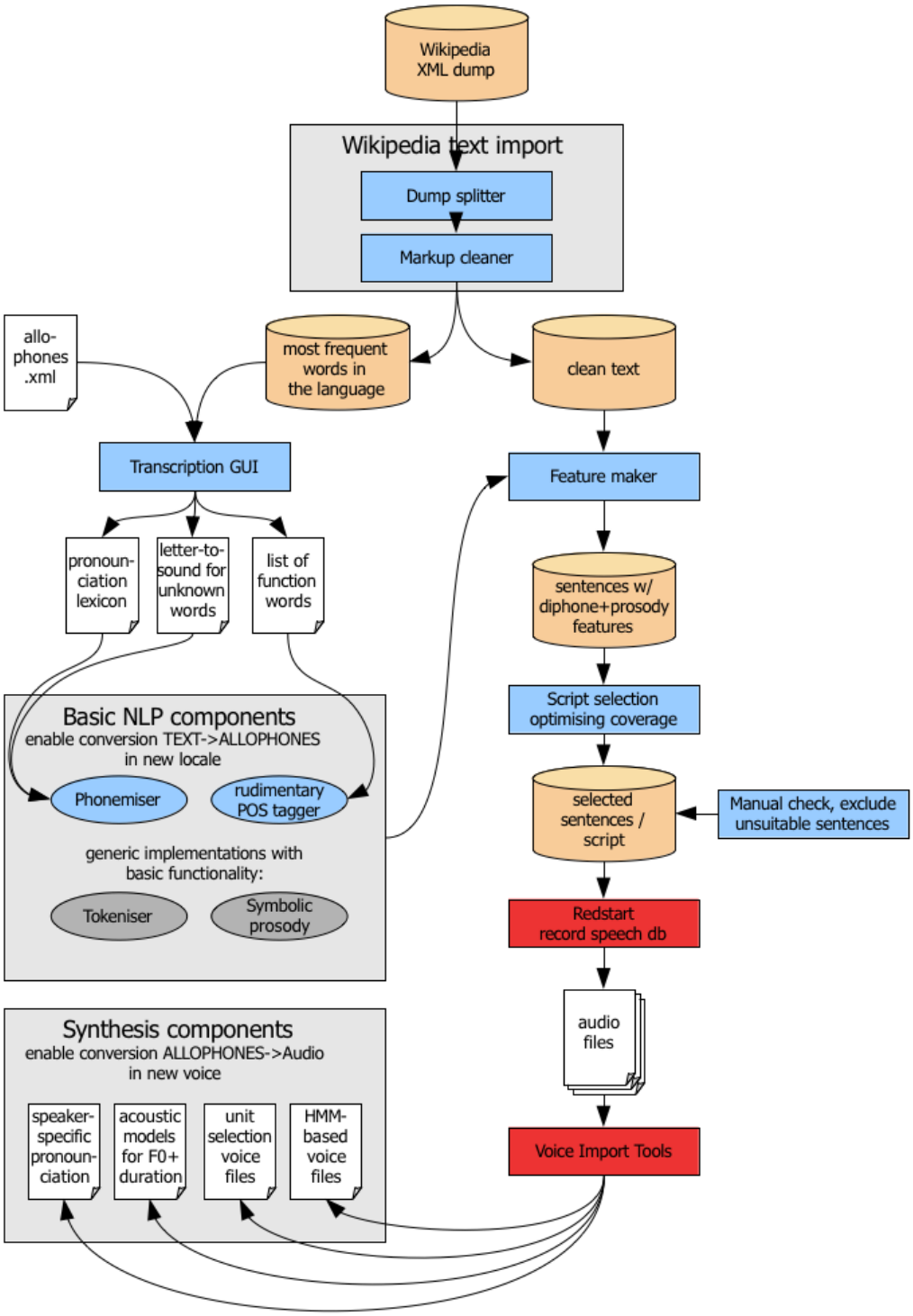
AN. 2. SORTIE AU FORMAT W3C SSML D'UN TEXTE PHONETISE (SANS INFORMATIONS PROSODIQUES) [20]

```
<?xml version="1.0"?>
<speak version="1.0" xmlns="http://www.w3.org/2001/10/synthesis"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://www.w3.org/2001/10/synthesis
                   http://www.w3.org/TR/speech-synthesis/synthesis.xsd"
         xml:lang="en-US">
  <phoneme alphabet="ipa" ph="t&#x259;mei&#x325;&#x27E;ou&#x325;"> tomato
</phoneme>
  <!-- This is an example of IPA using character entities -->
  <!-- Because many platform/browser/text editor combinations do not
       correctly cut and paste Unicode text, this example uses the entity
       escape versions of the IPA characters.  Normally, one would directly
       use the UTF-8 representation of these symbols: "təmei˽ɾou˽". -->
</speak>
```

AN. 3. DIFFERENTES TRANSCRIPTIONS DU MOT PLUS [14]

	Plus -	**Plus +**
En final	/s/ jamais prononcé *Il n'en veut plus*	/s/ facultatif *Un peu plus* -> /ply/ ou /plys/
Le mot suivant commence par une consonne	/s/ jamais prononcé *Plus du tout*	/s/ jamais prononcé *Plus beau*
Le mot suivant commence par une voyelle ou un h muet	Selon le registre, /z/ prononcé ou non *Il n'y en a plus assez* /ply zase/ ou /ply ase/	/z/ prononcé *Plus avancé* /ply zava~se/

AN. 4. SCHEMA D'ENSEMBLE DE L'INTEGRATION D'UN NOUVEAU LANGAGE
(58)



AN. 5. LISTE DES MARYDATA (MARYXML OU AUTRE)

Document	Document XML basique du package org.w3c.dom.Document	
Nom	Entrée	Sortie
APML	✓	✗
SSML	✓	✗
SABLE	✓	✗
RAWMARYXML	✓	✓
TOKENS	✓	✓
WORDS	✓	✓
PARTSOFSPEECH	✓	✓
PHONEMES	✓	✓
INTONATION	✓	✓
ALLOPHONES	✓	✓
ACOUSTPARAMS	✓	✓
REALISED_ACOUSTPARAMS	✓	✓
String	Texte brut	
Nom	Entrée	Sortie
TEXT	✓	✗
SIMPLEPHONEMES	✓	✗
AudioInputStream	Flux audio en entrée du système	
Nom	Entrée	Sortie
AUDIO	✗	✓
List	Liste de données	
Nom	Entrée	Sortie
PRAAT_TEXTGRID	✓	✓
REALISED_DURATIONS	✓	✓
HALFPHONE_TARGETFEATURES	✗	✓
TARGETFEATURES	✗	✓

AN. 6. TABLEAU (NON COMPLET) DES ETIQUETTES DE POSITIONS CONTEXTUELLES DANS LIA_PHON

ADV	Adverbe	DETMP	Déterminant Masculin Pluriel
ADVNE	Ne	DETMS	Déterminant Masculin Singulier
ADVPAS	Pas	NMS	Nom Masculin Singulier
AFP	Adjectif Féminin	NMP	Nom Masculin

	Pluriel		Pluriel
AFS	Adjectif Féminin Singulier	NFS	Nom Féminin Singulier
AINDFP	Adjectif Indéfini Féminin Pluriel	NFP	Nom Féminin Pluriel
AINDFS	Adjectif Indéfini Féminin Singulier	PDEMFP	Pronom Démonstratif Féminin Pluriel
AINDMP	Adjectif Indéfini Masculin Pluriel	PDEMFS	Pronom Démonstratif Féminin Singulier
AINDMS	Adjectif Indéfini Masculin Singulier	PDEMMP	Pronom Démonstratif Masculin Pluriel
AMP	Adjectif Masculin Pluriel	PDEMMS	Pronom Démonstratif Singulier Pluriel
AMS	Adjectif Masculin Singulier	PREP	Préposition
CHIF	Chiffre ou nombre	PREPADE	A de
COCO	Conjonction de Coordination	PREPAU	Au
COSUB	Conjonction de Subordination	PREPAUX	Aux
DETFP	Déterminant Féminin Pluriel	PREPDES	Des
DETFS	Déterminant Féminin Singulier	PREPDU	du
V1S	Verbe 1ère personne du Singulier	VINF	Verbe à l'infinitif
V2S	Verbe 2ème personne du Singulier	…	…
V3S	Verbe 3ème personne du Singulier	YPFOR	Point
V1P	Verbe 1ère personne du Pluriel	YPFAI	Toute autre ponctuation
V2P	Verbe 2ème personne du Pluriel		
V3P	Verbe 3ème personne du Pluriel		
VA1S	Auxiliaire Avoir 1ère personne du singulier		
…	…		
VE1S	Auxiliaire Être 1ère personne du Singulier		
…	…		

AN. 7. DESCRIPTION DES FORMATS MARYXML UTILISES POUR L'INTEGRATION DU FRANÇAIS A OPENMARY

Dans le cas du français, la chaîne de traitement se fait comme suit :

`TEXT -> RAWMARYXML -> PHONEMES -> INTONATIONS -> ALLOPHONES -> ACOUSTPARAMS`

Si l'on prend comme entrée du système `TEXT` : `Je suis, hélas, ton père.`

- On obtient en `RAWMARYXML` :

```
<?xml version="1.0" encoding="UTF-8"?>
<maryxml xmlns="http://mary.dfki.de/2002/MaryXML"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="0.5"
xml:lang="fr">
<p>
Je suis, hélas, ton père.
</p>
</maryxml>
```

Le `RAWMARYXML` apporte très peu d'informations mais est nécessaire pour la suite du traitement. `xml:lang` précise la langue dans laquelle le texte sera traité, informations cruciales pour définir quelles seront les modules de TALN à utiliser. Ceci représente la variable `locale` qui est ici égale à `"fr"`.

- Le format `PHONEMES` :

```
<?xml version="1.0" encoding="UTF-8"?>
<maryxml xmlns="http://mary.dfki.de/2002/MaryXML"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="0.5"
xml:lang="fr">
<p>
<s>
<t ph="Z@" pos="[PPER1S]">
je
</t>
<t ph="sHi" pos="[VE1S]">
suis
</t>
<t ph="_" pos=",">
,
</t>
<t ph="elas" pos="[ADV]">
hélas
</t>
<t ph="_" pos=",">
,
</t>
<t ph="to~" pos="[DETMS]">
ton
</t>
<t ph="pER" pos="[NMS]">
père
</t>
<t ph="_" pos=".">
.
</t>
</s>
</p>
```

```
</maryxml>
```

C'est une représentation plus riche, indiquant la transcription phonétique des unités segmentales du texte à synthétiser.

- Le format INTONATIONS :

```
<?xml version="1.0" encoding="UTF-8"?>
<maryxml xmlns="http://mary.dfki.de/2002/MaryXML"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="0.5"
xml:lang="fr">
<p>
<s>
<prosody pitch="+5%" range="+20%">
<phrase>
<t ph="Z@" pos="[PPER1S]">
je
</t>
<t ph="sHi" pos="[VE1S]">
suis
</t>
<t ph="_" pos=",">
,
</t>
<boundary breakindex="4" tone="H-L%"/>
</phrase>
</prosody>
<prosody pitch="+0%" range="+0%">
<phrase>
<t ph="elas" pos="[ADV]">
hélas
</t>
<t ph="_" pos=",">
,
</t>
<boundary breakindex="4" tone="H-L%"/>
</phrase>
</prosody>
<prosody pitch="-5%" range="-20%">
<phrase>
<t ph="to~" pos="[DETMS]">
ton
</t>
<t ph="pER" pos="[NMS]">
père
</t>
<t ph="_" pos=".">
.
</t>
<boundary breakindex="5" tone="L-L%"/>
</phrase>
</prosody>
</s>
</p>
</maryxml>
```

Voir la section II.2 pour une explication détaillée du MaryXML INTONATIONS.

❖ Le format ALLOPHONES :

```
<?xml version="1.0" encoding="UTF-8"?>
<maryxml xmlns="http://mary.dfki.de/2002/MaryXML"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="0.5"
xml:lang="fr">
<p>
<s>
<prosody pitch="+5%" range="+20%">
<phrase>
<syllable ph="_">
<ph p="_"/>
</syllable>
</t>
<t ph="Z@" pos="[PPER1S]">
je
<syllable ph="Z @">
<ph p="Z"/>
<ph p="@"/>
</syllable>
</t>
<t ph="sHi" pos="[VE1S]">
suis
<syllable ph="s H i">
<ph p="s"/>
<ph p="H"/>
<ph p="i"/>
</syllable>
</t>
<t ph="_" pos=",">
,
<syllable ph="_">
<ph p="_"/>
</syllable>
</t>
<boundary breakindex="4" tone="H-L%"/>
</phrase>
</prosody>
<prosody pitch="+0%" range="+0%">
<phrase>
<t ph="elas" pos="[ADV]">
hélas
<syllable ph="e l a s">
<ph p="e"/>
<ph p="l"/>
<ph p="a"/>
<ph p="s"/>
</syllable>
</t>
<t ph="_" pos=",">
,
<syllable ph="_">
<ph p="_"/>
</syllable>
</t>
<boundary breakindex="4" tone="H-L%"/>
</phrase>
</prosody>
<prosody pitch="-5%" range="-20%">
<phrase>
<t ph="to~" pos="[DETMS]">
ton
<syllable ph="t o~">
<ph p="t"/>
<ph p="o~"/>
</syllable>
```

```
</t>
<t ph="pER" pos="[NMS]">
père
<syllable ph="p E R">
<ph p="p"/>
<ph p="E"/>
<ph p="R"/>
</syllable>
</t>
<t ph="_" pos=".">
.
<syllable ph="_">
<ph p="_"/>
</syllable>
</t>
<boundary breakindex="5" tone="L-L%"/>
</phrase>
</prosody>
</s>
</p>
</maryxml>
```

Ce format représente la sortie finale des modules de TALN. En plus de toutes les informations phonétiques et prosodiques présentés dans le MaryXML `INTONATIONS` en section II.2, on retrouve les balises `<syllable>` `</syllable>` qui entoure les syllabes des unités segmentales. On remarque toutefois que dans le cas de la langue française, les syllabes ne sont pas bien identifiées. Par exemple, la bonne sortie pour le mot `hélas` devrait être

```
<t ph="elas" pos="[ADV]">
hélas
<syllable ph="e l">
<ph p="e"/>
<ph p="l"/>
</syllable>
<syllable ph="a s">
<ph p="a"/>
<ph p="s"/>
</syllable>
</t>
```

Chacune des syllabes phonétisées est décomposé selon les phonèmes qui les composent à l'aide d'un fichier XML propre au langage synthétisé, le fichier `allophones.**.xml` (les `**` représentent la valeur de `local`). Celui-ci contient tous les phonèmes utilisés, et les classifie (voyelles, consonnes, fricative, liquide etc.).

Le fichier XML a été réalisé à l'aide de l'article Wikipédia sur les symboles SAMPA [10] où sont décrits tous les phonèmes (tableaux des voyelles et des consonnes).

```
<allophones name="sampa" xml:lang="fr"
                features="vlng vheight vfront vrnd ctype cplace cvox">
        <silence ph="_"/>

        <vowel ph="i" vlng="l" vheight="1" vfront="1" vrnd="-"/>
        <vowel ph="e" vlng="l" vheight="2" vfront="1" vrnd="-"/>
        <vowel ph="3" vlng="l" vheight="1" vfront="1" vrnd="-"/>
        <vowel ph="E" vlng="l" vheight="2" vfront="1" vrnd="-"/>
        <vowel ph="a" vlng="l" vheight="3" vfront="2" vrnd="-"/>
        <vowel ph="0" vlng="s" vheight="2" vfront="3" vrnd="+"/>
        <vowel ph="o" vlng="l" vheight="2" vfront="3" vrnd="+"/>
        <vowel ph="u" vlng="l" vheight="1" vfront="3" vrnd="+"/>
        <vowel ph="y" vlng="l" vheight="1" vfront="2" vrnd="+"/>
        <vowel ph="2" vlng="l" vheight="2" vfront="2" vrnd="+"/>
        <vowel ph="9" vlng="s" vheight="2" vfront="2" vrnd="+"/>
        <vowel ph="@" vlng="a" vheight="2" vfront="2" vrnd="-"/>
        <vowel ph="e~" vlng="l" vheight="2" vfront="1" vrnd="-" ctype="n"/>
        <vowel ph="a~" vlng="l" vheight="3" vfront="2" vrnd="-" ctype="n"/>
        <vowel ph="o~" vlng="l" vheight="2" vfront="3" vrnd="+" ctype="n"/>
        <vowel ph="9~" vlng="l" vheight="2" vfront="2" vrnd="+" ctype="n"/>

        <consonant ph="p" ctype="s" cplace="l" cvox="-"/>
        <consonant ph="t" ctype="s" cplace="a" cvox="-"/>
        <consonant ph="k" ctype="s" cplace="v" cvox="-"/>
        <consonant ph="b" ctype="s" cplace="l" cvox="+"/>
        <consonant ph="d" ctype="s" cplace="a" cvox="+"/>
        <consonant ph="g" ctype="s" cplace="v" cvox="+"/>
        <consonant ph="f" ctype="f" cplace="b" cvox="-"/>
        <consonant ph="s" ctype="f" cplace="a" cvox="-"/>
        <consonant ph="S" ctype="f" cplace="p" cvox="-"/>
        <consonant ph="v" ctype="f" cplace="b" cvox="+"/>
        <consonant ph="z" ctype="f" cplace="a" cvox="+"/>
        <consonant ph="Z" ctype="f" cplace="p" cvox="-"/>
        <consonant ph="l" ctype="l" cplace="a" cvox="+"/>
        <consonant ph="R" ctype="f" cplace="u" cvox="-"/>
        <consonant ph="m" ctype="n" cplace="l" cvox="+"/>
        <consonant ph="n" ctype="n" cplace="a" cvox="+"/>
        <consonant ph="J" ctype="n" cplace="p" cvox="+"/>
        <consonant ph="H" ctype="r" cplace="l" cvox="+"/>
        <consonant ph="w" ctype="r" cplace="l" cvox="+"/>
        <consonant ph="j" ctype="r" cplace="p" cvox="+"/>

</allophones>
```

CPLACE
Consonant cplace: l = labial, a = alveolar, p = palatal, b = labio_dental, d = dental, v = velar

CTYPE
Consonant type: s = stop, f = fricative, a = affricative, n = nasal, l = liquid.

CVOX
Consonant voicing: + = on, - = off

VC
Vowel or consonant: + = vowel, - = consonant.

VFRONT
Vowel frontness: 1 = front, 2 = mid, 3 = back.

VHEIGHT
Vowel height: 1 = high, 2 = mid, 3 = low.

VLNG
Vowel length: s = short, l = long, d = dipthong, a = schwa.

VRND
Lip rounding: + = on, - = off.

AN. 9. MISE A JOUR DU VOICEIMPORTTOOL TUTORIAL POUR LA CREATION D'UNE VOIX MMC POUR MARY 4.3.0 (77)

- Download openmary-standalone-install-4.3.0

- Run on the UNIX terminal

```
java -jar openmary-standalone-install-4.3.0.jar
```

- While installing, rename `MARY TTS MARYTTS` without space

- Check the box « `voiceImportTools` ».

- Run on the UNIX terminal

```
MARY_BASE=/your/path/to/MARYTTS
export MARY_BASE
```

- If you want to keep the `MARY_BASE` variable, edit your `/.bashrc` file by doing in root mode

```
gedit ~/.bashrc
```

- And then copy/paste into it and save

```
MARY_BASE=/your/path/to/MARYTTS
export MARY_BASE
```

- Verify by running

```
echo $MARY_BASE
```

- Install the `libx11` library by launching

```
sudo apt-get install libx11-dev
```

- Install ActiveTCL

```
http://www.activestate.com/activetcl/downloads
```

- Run on the UNIX terminal to install all the external programs

```
$MARY_BASE/lib/external/check_install_external_programs.sh -install
```

- When everything is installed (you might install some external manually), run

```
$MARY_BASE/lib/external/check_install_external_programs.sh -check
```

Nothing should be missing. Note that Mary 4.3.0 works only with HTK 3.4, HDecode 3.4, HTS 2.1 for HTK 3.4 and hts-engine 1.01. Pay attention, these software are deprecated. If you don't have them, download and install them as written in their respective `INSTALL` file.

- In your `myvoice` directory, create a sh file named import.sh and write in it

```
export MARY_BASE="/your/path/to/MARY TTS/"
java -Xmx1024m -jar $MARY_BASE/java/voiceimport.jar
```

- Run on the UNIX terminal

```
bash /my/path/to/myvoice/import.sh
```

- Enter the path to `myvoice` in the textBox and click ok. Pay attention on the fact that your corpus must be larger than 10 sentences.

- Run the `AllophoneExtractor`. You will need to run the `mary server`. So run in the console

```
bash $MARY_BASE/bin/maryserver.sh
```

and check if the directory `/prompt_allophones` is full of xml files corresponding to the `ALLOPHONES` transcription of the corpus

- Before running the `EHHMLabeler`, go into the settings (the "wrench" button at the left of the text box) and change the `EHMMLabeler.dir` by **removing the /bin (if there is some)**

- Run it. It can take several hours/days. Make sure not to restart your machine if you happen to think the process stucks.

- Check the `/ehmm/lab` directory (normally full of .lab files)

- Before running the `LabelPauseDeletor`, go into the settings and edit

```
LabelPauseDeleter.threshold  =  10
```

- Check if the /lab directory is full of lab files

- Before running the `TranscriptionAligner` you have to convert the « , » into « . » in the .lab file of the directory `/lab`. To do this, into the directory, create a sh file, name it « `bar.sh` » (or whatever you want) write this script, then run it

```
for i in /my/path/to/myvoice/lab/*.lab
do
        echo ${i}
        sed -i 's/,/./g' /my/path/to/myvoice/lab/*.lab
done
```

- Run the `TranscriptionAligner` and check if the `/allophones` directory is full of xml `ALLOPHONES files`.

- Run the `PhoneUnitLabelComputer` and check if the `/phonelab` directory is full of lab files. For the French voice, at the moment (05 August 11), **you must not run this component!** Just copy/paste all the lab files in `/lab` into `/phonelab`. If you don't do this, you will have almost all your files misaligned with the features files (so impossible to synthesize the voice).

- Run the `FeatureSelectionComponent`. Check if all the features suits you and then click "`save`". Check if the text file in `/mary/features.txt` exists and is not empty.

- Run the `PhoneUnitFeatureComputer`. Make sure the mary server is open, or do it as previously explained. Check the `/phonefeatures` directory.

- Before running the `PhoneLabelFeatureAligner` you have to convert the « , » into « . » in the .lab file of the directory `/phonelab`. To do this, into the directory, create a sh file, name it « `bar2.sh` », write this script and run it

```
for i in /my/path/to/myvoice/phonelab/*.lab
do
        echo ${i}
        sed -i 's/,/./g' /my/path/to/myvoice/phonelab/*.lab
done
```

- Run it. If some of your lab files are not aligned with your features files, try to edit the `RAWMARYXML`, or the label files. If you don't want to do this, skip or remove. Be careful, the HMM components won't work with misalignment files! The process would take hours and fail because of this. If there is a problem you cannot correct, please remove.

- Before running the `HMMVoiceDataPreparation` check properly if the `hts` directory exists in `$MARY_BASE/lib/external`. Please note that this directory does not depend on the installation of `hts/htk`, but of Mary itself. If you don't have it, you can find the directory there:
  http://mary.opendfki.de/browser/branches/fr-branch/lib/external?order=name#hts

  Pay attention, you will need all the external programs properly installed! Then run it.

- Before running the `HMMVoiceConfigure`, go into the settings and edit the following field with your own parameters.

```
HMMVoiceConfigure.dataSet      =  french_set_name
HMMVoiceConfigure.speaker      =  speaker_name
HMMVoiceConfigure.lowerF0      =  40   (male=40,  female=80)
HMMVoiceConfigure.upperF0      =  280  (male=280, female=350)
```

  You can modify any parameters you want life if you were doing `make + parameters` in the original HTS. Run it.

- Run the `HMMVoiceFeatureSelection`, check the features that interests you then click save.

- Run the `HMMVoiceMakeData`. The process takes several hours and crash if there is a misalignment problem in some lab and features file. If it happens, you don't need to recalculate all the features (lf0, fourier magnitudes, etc.). Generally if it crashing because of the misalignment, before running it again, go into the settings and edit the field like following:

```
HMMVoiceMakeData.makeSTR       =  0
HMMVocieMakeData.makeCMPMARY   =  1
```

- Run the `HMMVoiceMakeVoice`. Process takes several hours/day. Pay attention your computer doesn't crash.

- When the making of the voice is over, you can add this to Mary. Before running the `HMMVoicePackager` go into the settings, and edit the fields you want. For example:

```
HMMVoicePackager.useMixExc   =  true if using mixed excitation
HMMVoicePackager.useGV       =  true if using global variance in parameter
generation.
HMMVoicePackager.useAcousticModels  =  true  to  allow  prosody  modification
specified in MARYXML
```

- When running is complete, check in `$MARY_BASE/download` if you have a `mary-myvoice-local.zip` file containing:

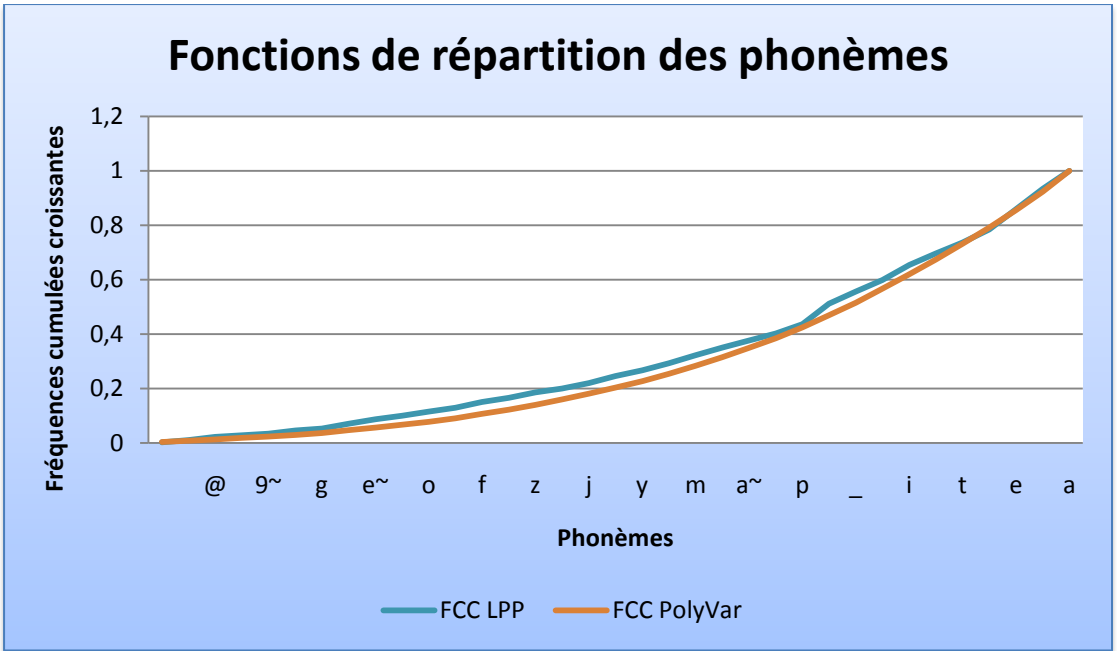
A mary config file: `french-hsmm-voice.config`
HMM files corresponding to this voice:
one example of phonefeatures for testing the synthesiser:
`data/phonefeatures/my_voice_my_name_xxxx.pfeats`
the HTS trees: `voices/qst001/ver1/*.inf`
the HTS PDF models: `voices/qst001/ver1/*.pdf`
global variance models (if useGV is set to true): `data/gv/gv-*-littend.pdf`
filter taps for mixed excitation: `data/filters/mix_excitation_filters.txt`
`trickyPhones.txt` file, if one was created during training

- Finally in the zip file, copy/paste:
  - the `.config` file into `$MARY_BASE/conf/`
  - the `/lib/voices/myvoice` into `$MARY_BASE/lib/voices/myvoice`

  And copy/paste the `$MARY_BASE/download/mary-myvoice-local.xml` into `$MARY_BASE/installed`.
  Then run `ant`, restart your machine and here you go!

AN. 10. STATISTIQUES DES CORPUS LPP ET POLYVAR

Phonèmes	Corpus LPP	PolyVar	Phonèmes	Corpus LPP	PolyVar
H	31	511	y	339	2955
@	119	533	O	290	3100
S	156	706	m	349	3585
9~	68	728	n	382	4007
9	82	729	a~	388	4158
g	166	769	k	338	4617
w	94	911	p	349	4691
e~	228	1317	E	453	5265
b	218	1342	_	1020	6000
o	185	1366	d	588	6059
Z	202	1375	i	564	6907
f	182	1832	3	747	7013
u	285	2092	t	560	7169
z	206	2122	s	549	7758
v	273	2363	e	632	7939
j	181	2535	l	992	8492
o~	274	2896	a	1012	9138
			R	888	10144



AN. 11. TABLEAU DESCRIPTIF DES SOURCES DE LIA_PHON (PROGRAMMES, SCRIPTS ET FICHIER DE DONNEES) (76)

Data	Répertoire contenant les fichiers de données. Ceux-ci contiennent les règles de phonétisation et différents lexiques de mots, d'exceptions etc.
Règles de transcription	Les différentes règles de transcriptions, dont les principales stockées dans le fichier french01.pron
desigle.pron	Règles de décisions pour épeler/lire les acronymes
epeler_sig.pron	Règles de prononciations pour épeler les acronymes
lire_sig.pron	Règles de prononciation pour lire les acronymes
french01.pron	Règles de prononciation pour le français standard
propername_[1-8].pron	Règles de prononciations pour différents ensembles de noms propres
Règles d'étiquetage et de post-traitement	Règles pour entre autre la gestion de liaison, d'élision de schwas
model_[un,bi,tri]	Modèle à une, deux ou trois lettres pou l'identification de nom propre
model_morpho.[un,bi,tri]	Modèle à une, deux ou trois lettres pou l'identification de mots inconnus dans la phase d'étiquetage
regles_l.pro3	Règles gérant les liaisons entres mots
regles_retik	Règles d'étiquetage syntaxique pour le post-traitement
rule_phon.pro	Règles de post-traitement pour les chaînes phonétisées
rule_variante.pro	Règles permettant la génération de prononciations alternatives

Dictionnaire	Dictionnaires utilisés par LIA_phon à l'exception de ceux utilisés par l'étiqueteur syntaxique
h_aspi.sirlex	Liste d'exceptions de mots commençant par « h »
list_chif_virgul	Liste de mots pouvant être suivie par un chiffre
list_exep	Liste d'exceptions de prononciations contenues dans french01.pron
Informations pour l'Étiqueteur syntaxique	Dictionnaire directement utilisés par l'étiqueteur pour la phonétisation
lex10k	Lexique de 10 000 mots utilisé par l'étiqueteur syntaxique
lex80k	Lexique de 80 000 mots utilisé par l'étiqueteur syntaxique
lm3classe.arpa	N-gramme [78] des POS obtenues par entraînement sur un corpus basé sur « Le Monde »
Script	Répertoire contenant les scripts qui sont directement lancés par l'utilisateur. Les scripts font appel à des exécutables produits à partir des programmes sources en C
lia_make_lexique_reacc	Compile les données du lexique pour l'étiqueteur syntaxique
lia_nett	Segmente en mots et phrases, nettoie le texte des mots.
lia_delete_lexicon	Supprime tous les fichiers compilés d'un lexique donné
lia_taggreac	Etiquetage syntaxique et ré accentuation du texte (si omissions d'accents)
lia_phon	transcription graphème-phonème (appel d'exécutables)
lia_text2phon	transcription phonétique complète d'un texte (appel de script)
lia_text2mbrola	transcription graphème-phonème et prosodie pour le synthétiseur MBROLA
lia_lex2phon	Phonétisation d'un lexique avec pour chaque mot toutes ces étiquettes syntaxique possibles.
lia_genere_phrase_mbrola	Génère un fichier MBROLA par phrase
Répertoires des sources	Chaque répertoire est assigné à une fonction particulière et contient les sources .c et .h correspondantes
aphon	Transcription graphème-phonème
format	Segmentation
liaison	Gestion des liaisons
libgram	N-gramme
post_process	Post-traitement de chaînes de caractères
prosodie	Génération de paramètres prosodiques
retik	Étiquetage spécifique à la transcription graphème-phonème
tagg	Étiquetage morphosyntaxique et ré-accentuation
variante	Génération de prononciation alternative

```
    /**
     * The main. It should launch the proper version of lia_phon
     *depending on which OS you have.
     * Should be "thread safe", as we don't use any static or global
     *variables (except for the
     * preprocessing classes).
     *
     * @param args
     * @throws Exception
     */
    public MaryData process(MaryData d) throws Exception
    {
        //LIA_Phon path
        String LIA_PHON_REP = new String(MaryProperties.maryBase() +
"/lia_phon");

        //Text to phonetize in ISO 8859-1 format
        String textInput = new String();

        //Text clean
        String textClean = new String();

        //Text phonetized (the lia_phon output)
        String textPhone = new String();

        //Read from the XML
        textInput = readFromXML(d);

        //clean the text
        textClean = preprocess(textInput);

        //Which OS?
        String wOS = new String("");
        Object Command[] = whichOS(LIA_PHON_REP);

        //os command to launch
        String osCommand1 = (String) Command[0];

         //Call LIA_phon and phonetize the text
         textPhone = callLIA(osCommand1, textClean, LIA_PHON_REP);

         //Convert to MaryXML
         Document marydoc;
         marydoc = convert2MaryXML(textPhone, textClean,
LIA_PHON_REP);
         MaryData phonemesData = new MaryData(MaryDataType.PHONEMES,
                 Locale.FRENCH, false);
                 phonemesData.setDocument(marydoc);
         return phonemesData;
    }
```

MARYTTS.LANGUAGE.FR
CLASS PHONEMISERFR

```
java.lang.Object
  └marytts.modules.InternalModule
      └marytts.language.fr.PhonemiserFR
```

All Implemented Interfaces:
marytts.modules.MaryModule

```
public class PhonemiserFR
extends marytts.modules.InternalModule
```

French phonemizer and POStagger. This class feeds the MaryXML with - words - phonemes - parts of speech We call lia_phon and then apply preprocessing, and conversion to SAMPA format. LIA_phon developped by FrÃ©dÃ©ric Bechet, LIA.

Author:
Florent Xavier

Field Summary

Fields inherited from interface marytts.modules.MaryModule
MODULE_OFFLINE, MODULE_RUNNING

Constructor Summary

PhonemiserFR()

Method Summary

static java.lang.String	**callLIA**(java.lang.String Command, java.lang.String text, java.lang.String path) Method that call LIA_phon.
org.w3c.dom.Document	**convert2MaryXML**(java.lang.String textPhon, java.lang.String textClean, java.lang.String path) Method that reads from the OuptutTEST.txt in ISO 8859-1 in order to convert it to MarXML and wite it in MaryXMLfr.txt
static java.lang.String	**preprocess**(java.lang.String str) Method that calls the preprocessFR class in order

	to clean the text.
marytts.datatypes.MaryData	**process**(marytts.datatypes.MaryData d) The main.
static java.lang.String	**readFromXML**(marytts.datatypes.MaryData d) Method that read from the XML.
static java.lang.String[]	**string2Array**(java.lang.String str) Very simple method that convert a string into array by splitting the \n
static java.lang.String[]	**whichOS**(java.lang.String path) A simple method that check the OS and then defines which command to execute.

Methods inherited from class marytts.modules.InternalModule
getLocale, getState, inputType, name, outputType, powerOnSelfTest, shutdown, startup

Methods inherited from class java.lang.Object
equals, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Constructor Detail

phonemiserFR

```
public PhonemiserFR()
```

Method Detail

string2Array

```
public static java.lang.String[]
string2Array(java.lang.String str)
```

Very simple method that convert a string into array by splitting the \n

Parameters:

str -

Returns:

---

whichOS

```
public static java.lang.String[] whichOS(java.lang.String path)
```

A simple method that check the OS and then defines which command to execute.

Returns:

---

convert2MaryXML

```
public org.w3c.dom.Document
convert2MaryXML(java.lang.String textPhon,

java.lang.String textClean,

java.lang.String path)
```

```
                                                    throws java.lang.Exception
```
Method that reads from the OuptutTEST.txt in ISO 8859-1 in order to convert it to MarXML and wite it in MaryXMLfr.txt

Parameters:

`textPhon` -

`textClean` -

Throws:

`java.lang.Exception`

---

callLIA

```
public static java.lang.String callLIA(java.lang.String Command,
                                        java.lang.String text,
                                        java.lang.String path)
                                 throws java.lang.Exception
```
Method that call LIA_phon. The UNIX command is launched, and we give as input for the process the text to phonetize directly. We don't need to give a text file as argument.

Parameters:

`Command` -

`text` -

Throws:

`java.lang.Exception`

---

preprocess

```
public static java.lang.String preprocess(java.lang.String str)
```
Method that calls the preprocessFR class in order to clean the text.

Parameters:

`str` -

Returns:

---

readFromXML

```
public static java.lang.String
readFromXML(marytts.datatypes.MaryData d)
```
Method that read from the XML. Basically, we try to get the plain text as our whole code works with strings as input.

Parameters:

`d` -

Returns:

---

process

```
public marytts.datatypes.MaryData
process(marytts.datatypes.MaryData d)
                                    throws java.lang.Exception
```
The main. It should launch the proper version of lia_phon depending on which OS you have. Should be "thread safe", as we don't use any static or global variables (except for the preprocessing classes).

Specified by:

`process` in interface `marytts.modules.MaryModule`

Overrides:

`process` in class `marytts.modules.InternalModule`

Parameters:

args -
Returns:
A MaryData object of type `outputType()` encapsulating the processing result.

This method just returns its input. Subclasses should override this.

Throws:
`java.lang.Exception`

Package **Class** **Use** **Tree** **Deprecated** **Index** **Help**

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

AN. 14. NEWLANGUAGESUPPORT 1.1

- Assuming the NLP modules are developed, export in `$MARY_BASE/java` all your classes in a `.jar` file named `mary-**.jar`, where `**` stands for your `locale`. For example using eclipse, `File -> Export -> General -> Archive File`, then select all the classes needed. Browse to the directory, enter the name then `OK` and `Finish`.

- Create a `**.config` like for exemple the `fr.config`

```
##############################################################################
# MARY TTS configuration file.
##############################################################################

name = fr
fr.version = 4.3.0

# Declare "group names" as component that other components can require.
#  These correspond to abstract "groups" of which this component is an
instance.
provides = a-language

requires = \
    marybase \

##############################################################################
############################## The Modules  ##################################
##############################################################################

modules.classes.list = \
   marytts.language.fr.PhonemiserFR \


#############################################################################
######################## Module settings  ###################################
#############################################################################

fr.allophoneset = MARY_BASE/lib/modules/fr/lexicon/allophones.fr.xml

featuremanager.classes.list = \
   marytts.features.FeatureProcessorManager(fr)
```

Basically, replace "fr" by your locale. "The modules" contains the path to your NLP components. You may have several, and the order of processing does not matter. "Module settings" contains the path to your allophone XML file, and other files needed by your system (lexicon, trees, ToBI parameters etc.). The featuremanager.classes.list is absolutely crucial to inform the system of the availability of the new language. If you forget this line, you will not be able to run the HMMVoiceMakeData, i.e. extract the acoustic parameters from the audio files of your corpus.

- Edit the $MARY_BASE/build.xml file and write the same information than the other language, adapting them with your own parameters.

- Copy/paste the directory "**" in your workspace into $MARY_BASE/java/marytts/language/**.

- Then open a terminal, cd to MARY_BASE then run ant to compile the system.

- It is now time to test whether your NLP components have the right output or not, regarding of their input. In eclipse, start the Mary server (class marytts.server.Mary.java) in debug mode (click right on the class, Debug as -> Java Application). This will open the Mary server in the localhost, port 59 125. Then open your internet browser and enter this URL http://localhost:59125/documentation.html to open a Mary client. Scroll down to the POST example, then fill the text area:

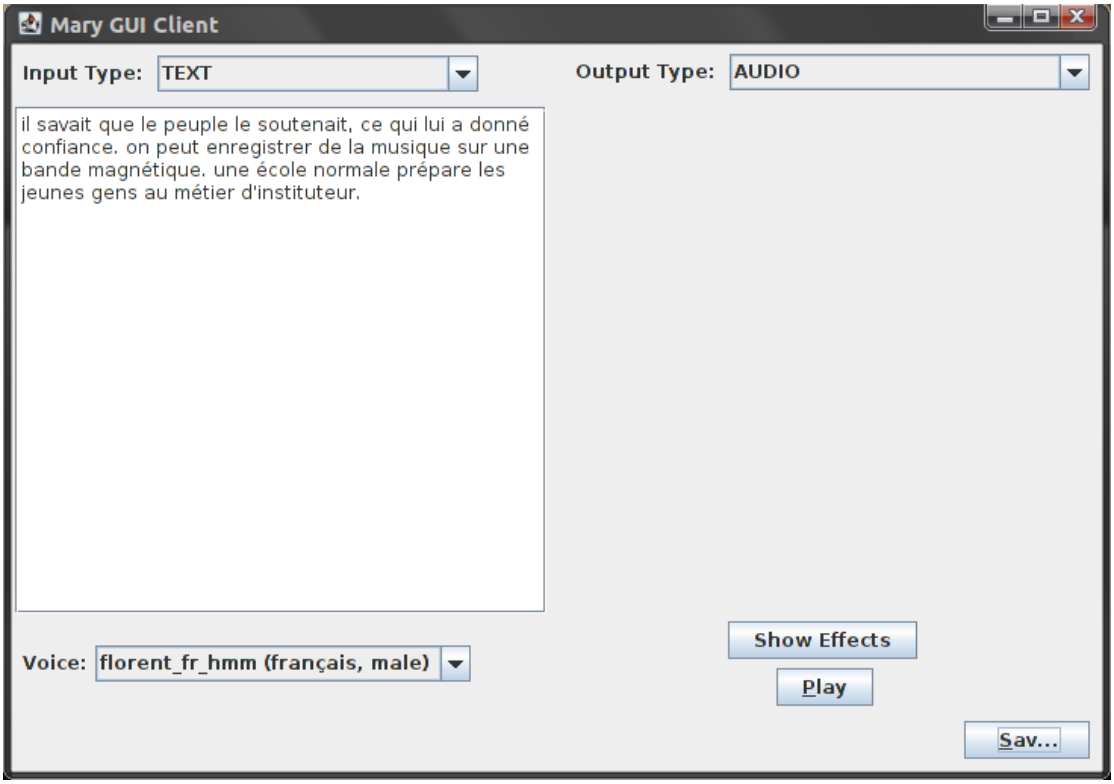
```
INPUT_TEXT : your text to process
INPUT_TYPE: the MaryDataType of the input
OUTPUT_TYPE: the MaryData_Type of the output
LOCALE: the language you want to process
AUDIO: the format of the audio file (wave, AU etc.)
```

Try to test the whole NLP processing by submitting something like this:

```
INPUT_TEXT : Yeah! My text in my language.
INPUT_TYPE: TEXT
OUTPUT_TYPE: ALLOPHONES
LOCALE: **
AUDIO: (let it empty as you don't have the voice yet)
```

If you want to test a particular module, adapt the input and output type (ex. to test the prosodic module, input is PHONEMES, output is INTONATION). If the processing fail or the output is not the one expected, get back to your code, tweak it, then re-do all the previous points.

- NLP components only go from TEXT to ALLOPHONES. However, if you coded or re-adapted other modules for the acoustic parameters generations, the procedure above doesn't change.


Mary GUI Client
Input Type:
TEXT
Output Type:
AUDIO
il savait que le peuple le soutenait, ce qui lui a donné confiance. on peut enregistrer de la musique sur une bande magnétique. une école normale prépare les jeunes gens au métier d'instituteur.
Voice:
florent_fr_hmm (français, male)
Show Effects
Play
Sav...