Rapport de première Soutenance

SD\n – ALIAS

*<**A** **L**exeme **I**s **A** **S**ound>*

Julie THÉODOSE-SOREL Mathieu BASCLE
Nicolas TANDÉ Yohan BAINIER

25 février 2005

Table des matières

1 Introduction

Section Disparu à la ligne (SD\n) au rapport pour cette première soutenance. Nous allons vous présenter tout au long de ce document les résultats du développement du projet depuis son élaboration lors du cahier des charges. Tout d'abord, présentons les membres de la cellule : Yohan BAINIER, Mathieu BASCLE, Nicolas TANDÉ et, la plus charmante de nos membres, Julie THÉODOSE-SOREL.

Depuis la création du cahier des charges, le développement d'**ALIAS**, notre *Text To Speech*[1], s'est déroulé sans trop d'incidents et c'est ainsi que nous allons vous présenter ici le fruit de notre travail, à savoir la réalisation de l'*Interface* graphique par Julie, du *Parser* par Yohan et Mathieu, qui a demandé beaucoup de documentation (merci la langue française d'être remplie d'exceptions, de diverses liaisons et raffinements de ce genre) et enfin du *Speaker* par Nicolas, car après avoir isolé les phonèmes, c'est encore mieux de pouvoir les lire (sinon ce ne serait pas un TTS) !

[1] TTS

2 Interface

2.1 Qu'est ce que *Qt* ?

Qt est un utilitaire complet, permettant le développement d'applications graphiques en *C++*. Cet utilitaire a été développé par TrollTech[2].

2.2 Pourquoi *Qt* ?

Bien que nous ayons choisi de programmer en langage C, nous avons choisi un utilitaire qui utilise le *C++*. Et ceci pour plusieurs raisons.

Tout d'abord, *Qt* est un outil d'interface multiplateformes. En effet, il est inutile de rappeler que nous avons prévu d'accomplir notre destiné et donc , de développer une application sous Unix et sous Mac. Il était donc préférable de choisir *Qt* à son concurrent *GTK+*. De plus, *Qt* est orienté objet, ce qui n'est pas négligeable.

2.3 Démarrer avec *Qt*

L'installation de *Qt* est simple. Nous ne nous attarderons donc pas sur la chose. Nous allons plutôt parler de l'utilisation fonctionnelle de *Qt*. En effet, il est possible de taper directement son code *Qt* dans un éditeur, puis de compiler ce code. Cependant, nous avons préféré faire l'usage d'un éditeur de code, qui est lui même livré avec *Qt*. Il s'agit de *Qt* designer.

2.4 *Qt* Designer

2.4.1 Présentation

Qt Designer est un éditeur de code, facile à appréhender Il suffit de sélectionner puis déposer le *Widget* voulu. Sur la gauche, nous avons une présentation des différents *Widgets* et les éléments à la droite permettent une meilleure navigation au sein de notre projet.

Pour débuter, il suffit d'un clic sur *File*, *new*, *new project*. Ensuite, on ajoute une *Qdialog* à notre projet. La *Qdialog* sera la fenêtre principale de notre projet, celle sur laquelle sera déposée tous les *Widgets*.

[2]http ://www.trolltech.com/

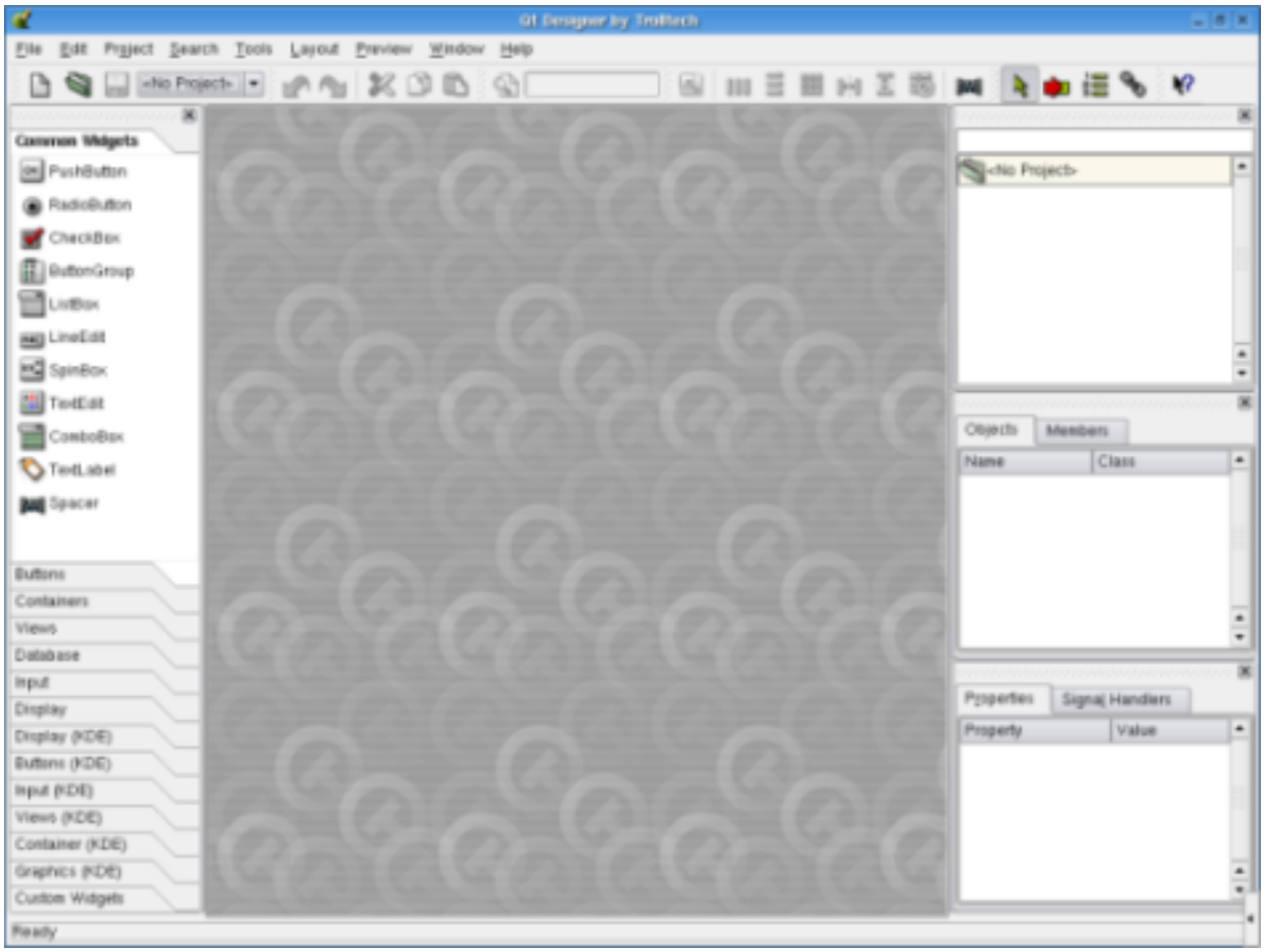


FIG. 1 – *Qt* Designer

Une fois notre interface réalisée, il ne reste plus qu'à compiler notre projet. L'utilitaire *qmake* nous permet d'éditer un *Makefile*, et une fois celui-ci créé, un simple *make* dans le répertoire et nous sommes en présence de notre projet.

2.4.2 Définition des objets utilisés

- *Widget*
  L'interface graphique est composée de *Widgets*, qui ne sont qu'une façon de nommer les fenêtres, boutons, champs de saisie, et autres combobox.
- *Signal* et *Slot*
  Les *signaux* et *slots* permettent l'interaction entre objets. L'utilisation

d'un *Widget*, tel que le clic sur un bouton, produit un événement qui émet un *signal*. Ce *signal* appelle alors un slot, c'est à dire une fonction qui est la réponse correspondante au *signal*. Dans notre cas, le *signal* pourra être un clic sur un bouton, et le slot, la fonction qui sera alors effectuée, par exemple l'effacement, la fermeture.

2.5 L'interface

Nous avons fait une interface plutôt simple pour cette première soutenance. De fait, celle-ci est composée de trois *TextEdit*, qui permettent l'écriture du texte, et de quatre boutons.

Dans le plus grand des *TextEdit*, nous écrivons la phrase que nous souhaitons. Dans le deuxième *TextEdit*, apparaît, à l'heure actuelle, le résultat du parsage réalisé (Le principe du parsage sera défini un peu plus tard, dans ce rapport). Quant au troisième *TextEdit*, il affiche différents renseignements relatifs au son.

On constate qu'il y a quatre boutons (*Pushbutton*). Le premier permet le parsage de la phrase contenue dans le premier *TextEdit*, selon le principe des "*signal / slots*" expliqué précédemment.
On appelle la fonction de parsage sur ce texte, puis le résultat est affiché dans le second *TextEdit*. Par ailleurs, le bouton lecture permet selon un même procédé de traiter le texte. Le bouton "effacer" efface les entrées contenues dans le *TextEdit*, et le bouton "quitter" permet de quitter l'application.

Cette interface n'est pas définitive, et bénéficiera, d'une dernière retouche pour la seconde soutenance.

3 Module de Traitement du Langage Naturel

Un *Text To Speech* comporte une phase essentielle appelée le **Traitement du Langage Naturel**[3] qui permet de générer la suite de *phonèmes* correspondant au texte passé au module TLN. Celui-ci est composé d'un *analyseur syntaxique* et d'un *générateur de phonèmes*. Mais avant de continuer, il faut connaître la phonétique[4].

3.1 Phonétique

En français, il a **26 lettres**, **37 *phonèmes*** et plus de **130 graphèmes** différents. Un *phonème* correspond à une unité de son et un graphème est une occurrence graphique de ce son.

	Mot-clé	Autres graphèmes
[i]	lit	stylo, île, maïs, meeting
[é]	télé	Parler, nez, pied, messieurs, poignée, volontiers
[è]	règle	chienne, merci, jouet, mais, maître, payer, treize, être, Noël, volley
[a]	sac	à, femme
[u]	lune	sûr, eu (avoir au passé composé)
[e]	je	
[E]	feu	nœud, jeûne
[F]	fleur	cœur, club

[3] TLN
[4] http ://phonetique.free.fr/

	Mot-clé	Autres graphèmes
[U]	poule	où, goûter, football, août
[o]	vélo	landau, bateau, drôle,
[O]	pomme	album, alcool, capharnaüm
[A]	pâte	
[D]	un	parfum
[C]	lapin	chien, pain, peinture, daim, imparfait, syndicat, sympa
[B]	gant	dent, jambe, empereur, paon, Caen
[~]	ballon	ombre, punch
[J]	pied	crayon, soleil, paille, hyène, païen
[V]	huit	sueur, suave, ennuyeux
[W]	doigt	ouate, wallon, équateur, moelle, poêle, croît, asseoir (+ nasale : loin)

Bilabiales	[p]	pile	appartement
	[b]	bol	abbaye
	[m]	mur	flamme
Dentales	[t]	table	datte
	[d]	dé	addition
	[n]	noeud	anniversaire
	[G]	ligne	manière
Vélaires	[k]	cadeau	qualité, képi, accord, orchestre, ticket, coq
	[g]	gâteau	bague, aggraver, second, ghetto
	[N]	parking	

		Mot-clé	Autres graphèmes
Labio-dentales	[f]	flûte	phare, affaire
	[v]	valise	wagon
Dentales	[s]	soleil	poisson, citron, garçon, démocratie, science, asthme, six
	[z]	maison	zoo, deuxième, blizzard
Palatales	[ʃ]	chat	short, schéma, fasciste
	[ʒ]	jupe	girafe
[l]		lampe	elle
[ʁ]		roue	beurre, rhume

3.2 Analyseur syntaxique

Dans un premier temps, le TLN doit valider le texte ; pour cette première soutenance, la vérification se fait de la manière suivante :

- mettre la chaîne de caractères en minuscules.
- déterminer si les symboles spéciaux de la langue (ponctuation, guillemets) ne se répètent pas de manière consécutive.
- vérifier qu'il n'y a pas de symboles "baroques" (© Olivier RODOT) comme '=', etc... Nous ne gérons pas non plus les parenthèses.

Prenons par exemple la chaîne "Bonjour,, tout, le Monde !". Une fois passée à la fonction `lowerify` nous obtenons la chaîne "bonjour,, tout, le monde !". Puis la fonction `is_valid` appliquée à cette dernière chaîne déclenchera une erreur indiquant que le symbole ',' n'est pas valide puisqu'il n'est pas autorisé à être présent deux fois consécutives.

Il reste tout de même une partie importante à réaliser : la gestion de la nature des mots. En effet, la chaîne "les poules du couvent couvent" ne pourra être décomposée dans la bonne série de phonèmes car rien ne permet encore – à l'analyseur syntaxique – de faire la distinction entre un verbe et un nom commun.

3.3 Générateur de phonèmes

Cette partie du module TLN a été codée en deux temps : de manière intuitive puis logiquement. Mais quelque soit la manière de coder le générateur, nous avons dû implémenter une *file* pour stocker les *phonèmes* dans leur ordre de création.

3.3.1 Étude intuitive (ou naïve)

Cette étude se résume à un simple parcours de chaîne pendant lequel un *phonème* est associé (puis enfilé) à une séquence de deux symboles. Voici un exemple de génération de *phonèmes* avec la chaîne "Bonjour" :

- la chaîne est "lowerifiée" et devient "bonjour".
- 'b' n'est pas suivi d'un autre 'b' donc on avance de 1 et on enfile [b].
- 'o' est suivi de 'n' donc on avance de 2 (car 'n' vient de servir) et on enfile [~].
- on avance de 1 et on enfile [j].
- 'o' est suivi de 'u' donc on avance de 2 (car 'u' vient de servir) et on enfile [U].
- enfin, pour 'r' qui est seul dans son coin, on avance de 1 et on enfile [R].

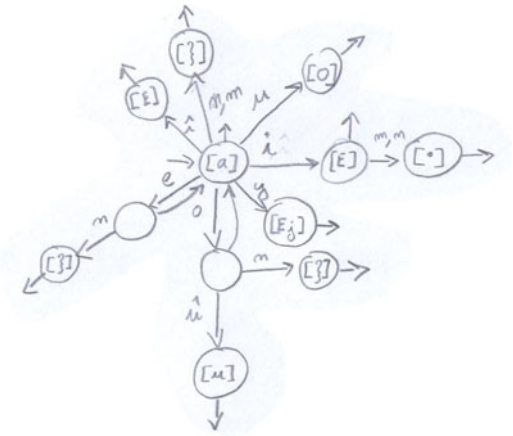
Finalement, la séquence obtenue est [b ~ j U R]. C'est donc, à priori, une méthode efficace. Mais si on considère cette nouvelle chaîne, qui, une fois "lowerifiée", devient "bonne année", on se rend compte que la génération des *phonèmes* n'est pas correcte :

- 'b' n'est pas suivi d'un autre 'b' donc on avance de 1 et on enfile [b].
- 'o' est suivi de 'n' donc on avance de 2 (car 'n' vient de servir) et on enfile [~].
- 'n' est suivi de 'e', on avance de 2 est on enfile [n e].
- 'a' est suivi de 'n', on avance de 2, on enfile [B].
- 'n' est suivi d'un symbole non présent dans l'alphabet, on avance de 1, on enfile [n].
- 'é' est un symbole spécial, on avance de 1, on enfile [é].
- enfin, on avance de 1, on enfile [e].

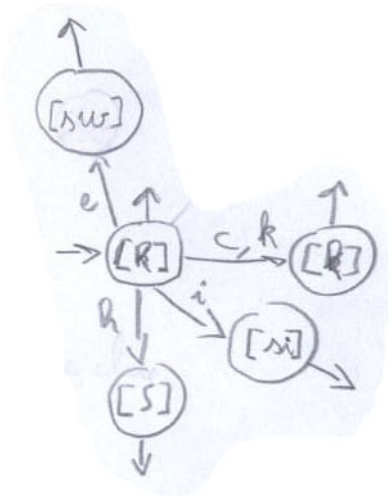
La séquence ainsi générée est [b ~ n e B n é e] ce qui serait prononcé "bon ne an né e". C'est très loin d'être [b O n a n é].

3.3.2 Étude logique

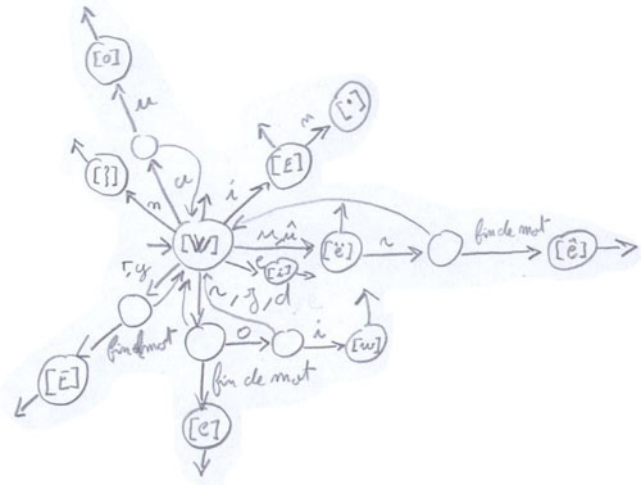
Nous venons de voir une méthode très limitée, celle-ci utilise un outil informatique : les automates. Nous avons dû construire un automate (pas nécessairement déterministe) pour chaque lettre. Voici les principaux automates :



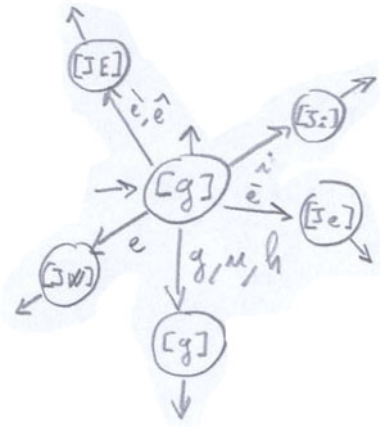
Automate du 'a'



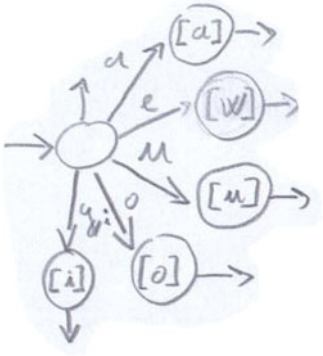
Automate du 'c'



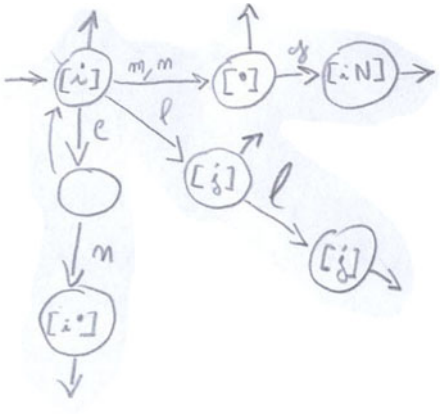
Automate du 'e'



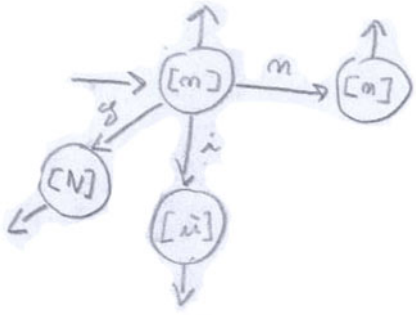
Automate du 'g'



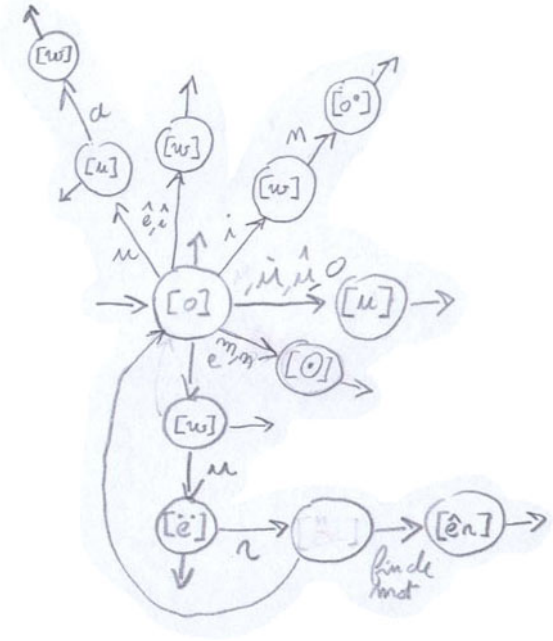
Automate du ′h′



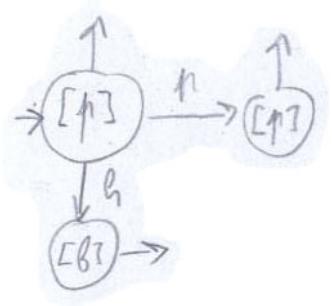
Automate du ′i′



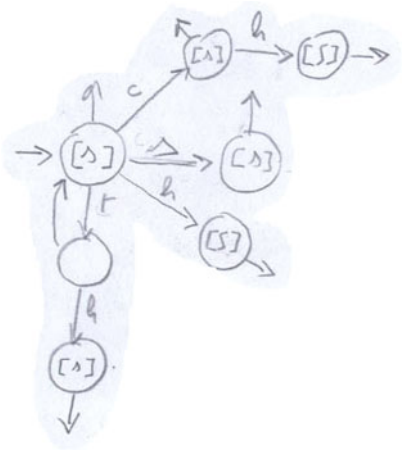
Automate du ′n′



Automate du ′o′



Automate du 'p'



Automate du 's'

Reprenons l'exemple précédent ("bonne année") pour illustrer la capacité d'un automate à gérer toutes les situations qui ne dépendent pas de la nature des mots :

- 'b' n'est pas suivi d'un autre 'b' donc on avance de 1 et on enfile [b].
- 'o' est suivi de 'n' **mais** 'n' est suivi d'un autre 'n' donc on avance de 1 et on enfile [O] ;
- 'n' est suivi de 'n', on avance de 2 est on enfile [n].
- 'e' est suivi du caractère spécial espace, il n'est donc pas prononçable, on avance de 1.
- 'a' est suivi de 'n' **mais** 'n' est suivi d'un autre 'n' : on avance de 1, on enfile [a].
- 'n' est suivi de 'n', on avance de 2 est on enfile [n].
- 'é' est un symbole spécial, on avance de 1, on enfile [é].
- enfin, 'e' est suivi du caractère spécial '\0', on avance de 1.

On obtient au final la séquences de phonèmes [b O n a n é]. Cette méthode est très efficace, nous resterons donc sur cette voie pour la suite. Quelques exceptions ne sont pas encore gérées comme la distinction entre le **eu** de f**eu** [E] et celui de fl**eur** [F].

Le module de *Traitement du Langage Naturel* est avancé au-delà de nos espérances (qui étaient restées à une simple étude naïve du texte). Nous allons donc pour la prochaine soutenance implémenter la gestion d'un dictionnaire contenant les exceptions du français et leurs séquences de *phonèmes* associées.

4 *Speaker*

4.1 Choix du format audio

4.1.1 PCM

Les cartes sons utilisent le format PCM (*Pulse Code Modulation*) en interne, mais plusieurs variantes existent. Les données peuvent être alignées en *Little Endian*, ou en *Big Endian*, et avec une fréquence d'échantillonage et un nombre de canaux variables. Ce format a l'avantage d'être celui de la carte, mais il est très gourmant en espace disque. Nous avons donc dû choisir un format de compression.
Nos échantillons seront très courts, contenant juste des *phonèmes*, les différences de tailles ne seront donc pas significatives d'un format à l'autre.

4.1.2 MP3

Le premier format de compression qui nous est venu à l'esprit est le très répandu MP3 (*MPEG-1/2 Audio Layer 3*). C'est un format de compression avec pertes réduisant la taille d'un son en moyenne de 12 fois. Il supprime les fréquences inaudibles pour l'oreille humaine, ainsi que les sons très faibles, tout en maintenant une qualité d'écoute très bonne. Ce format a été étudié pour un débit binaire (*bitrate*) fixe, cependant, il peut-être utilisé avec un débit binaire variable. Toutefois, ce format est soumis aux brevets logiciels dans certains pays, et pour un projet qui pourrait être commercialisé, cela peut-être génant.

4.1.3 Ogg Vorbis

Tout comme le MP3, le Ogg Vorbis est un format de compression avec perte. Cependant, il a été conçu pour un débit binaire variable, permettant une qualité sonore constante, et non un débit constant. De plus, ce format utilise des API libres (licence BSD), et très bien documentées. Il semble donc corriger les légers défauts du MP3.

4.1.4 Speex

Ce format est diffusé par Xiph.org, la même fondation qui fournit l'Ogg Vorbis. Il est conçu pour compresser de la voix, cependant il s'agit d'un algorithme de compression avec perte, et n'a sa place que dans des contextes de diffusion par réseau (*Streaming*). Il ne semble donc pas nous convenir.

4.1.5 Notre Choix

Nous nous sommes donc naturellement tournés vers le format Ogg Vorbis.

4.2 Principe

4.2.1 Différents Threads

Nous utilisons une Thread différente pour la lecture audio. Ceci permet de ne pas bloquer le reste du programme pendant la lecture, qui peut être très longue (la durée totale de tous les *phonèmes* mis bout-à-bout).

4.2.2 Contrôle de la Thread audio et passage de sons

Une fois la Thread audio lancée avec en paramètre le type de sortie désiré, il faut pouvoir lui communiquer les sons à lire, ainsi que pouvoir l'arrêter, ou même changer de matériel audio. Ceci a été réalisé à l'aide des *Mutex* et des variables conditionnnelles.
La Thread attend un signal venant de la variable conditionnnelle. Puis vérifie des booléens pour savoir si le matériel doit changer, ou s'il faut quitter. Ensuite elle vérifie l'état d'une file qui doit contenir les sons à lire, et les lire si elle n'est pas vide.
Sur la Thread principale du programme, il suffit de changer la valeur des booléens, puis d'envoyer un signal pour contrôler la Thread audio, ou alors d'enfiler des sons, et d'envoyer le signal pour qu'ils soient lus.

4.3 Implémentation

4.3.1 Ogg Vorbis

Nous utilisons bien évidement l'API Ogg Vorbis venant de Xiph.org pour la partie décompression. Il s'agit d'ouvrir un fichier Ogg, et de le parcourir pour obtenir les données PCM correspondantes.

4.3.2 xBSD et Linux 2.4 : OSS

Les systèmes BSD et Linux (jusqu'au 2.4 officielement) utilisent le système OSS (*Open Sound System*) pour lire des sons. En fait, il s'agit d'ouvrir le périphérique associé (le plus souvent */dev/snd*) et d'y écrire les données PCM, après avoir pris le soin de l'initialiser et de le paramétrer avec *ioctl*.

4.3.3 Linux 2.6 : ALSA

Depuis la version 2.5 de linux, et donc la version stable 2.6, le système utilise ALSA (*Advanced Linux Sound Architecture*) pour lire un son. Malgré que la compatibilité OSS soit toujours proposée, il n'y a aucune obligation qu'elle soit présente sur une machine. Pour garantir une installation aisée, il nous est paru plus utile d'intégrer son support dans notre projet.
Pour utiliser ALSA, nous devons comme pour OSS initialiser le matériel, qui n'est plus identifié par un fichier, mais par une référence dans le système, comme par exemple *hw :0,0*. Cette fois ci, l'envoi de données se fait d'une manière asynchrone : nous intérrogeons ALSA pour savoir la taille disponible dans son buffer, et nous demandons à l'API Ogg Vorbis de nous décompresser cette taille, pour y introduire les données PCM. Cette méthode permet de décompresser et de lire à la volée, ce qui doit être lu, et pas plus.

4.4 Ce qui a été fait

Au final, pour cette soutenance, nous avons un système multi-threadé permettant la lecture, et l'enchainement d'échantillons Ogg Vorbis sur les systèmes utilisant OSS ou ALSA, comme prévu dans le cahier des charges.

5 Conclusion

Première partie de la mission terminée. C'est ainsi que l'on pourrait qualifier cette première soutenance. En effet, le travail qui avait été assigné à chacun de nous dans le plan de mission (cahier des charges) s'est vu réalisé avec succès. Même si de nombreux problèmes ont été soulevés lors de nos recherches, nous ne perdons pas espoir.

Pour la prochaine soutenance, c'est avec une *Interface* terminée et pleinement fonctionnelle, un *Parser* toujours plus efficace avec une liste d'exceptions implémentée et une lecture des sons (*Speaker*) toujours plus fluide et crédible que nous nous tiendrons devant vous.

Certes, réussir la mission demandera encore de nombreux efforts et de sacrifices, mais une chose est sûre, le texte se fera entendre à la fin, brisant ainsi des années de silence imposé.

Premier rapport terminé.