ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

IDIAP
MARTIGNY - VALAIS - SUISSE

Réalisation d'un Majordome vocal

Par Samuel Vannay
7ème semestre SSC
janvier 1998

Supervision EPFL
Prof. Giovanni Coray

Supervision IDIAP
M. Olivier Bornet

1. Table des matières

2. Introduction

Dès les années 50, des recherches ont été menées en vue de l'utilisation de la parole comme moyen de communication entre un homme et un ordinateur. Depuis 1986, avec la généralisation de l'utilisation des modèles de Markov cachés, 10 ans de recherches ont aboutit à la commercialisation de logiciels de reconnaissance de la parole de plus en plus performants. IBM et son Via Voice ou Dragon System et son NaturallySpeaking sont les exemples les plus frappants. Pour environ 200 francs, ces logiciels offrent des outils de reconnaissance de la parole en continue, tournant sur des PC standards.
L'offre créant la demande, il n'est pas surprenant de voir plus en plus d'outils faisant appel à la reconnaissance vocale. Les serveurs vocaux interactifs notamment sont des applications permettant la consultation de données ou l'obtention d'information par téléphone. Le but de ce projet est de créer une telle application, qui doit principalement reconnaître le nom d'une personne et donner son numéro de téléphone. L'implémentation d'une telle application repose, bien sûr, sur deux techniques : celles de gestion de la ligne téléphonique et celle de reconnaissance de la parole.

3. XTL

XTL est une API (application programming interface) qui permet la gestion d'une ligne téléphonique. Elle permet d'implémenter des applications téléphoniques qui font ou reçoivent des appels, les mettent en pause, les dévient ou les enregistrent. Elle permet aussi de générer et de détecter des impulsions DTMF. XTL est implémenté dans une librairie C++ et est indépendant du hardware. On peut par exemple utiliser XTL sur une ligne ISDN ou sur une ligne ATM.

- Programmation objet

Des objets représentent de manière abstraite les services ou connections téléphoniques. La station fait office d'appareil téléphonique, le prestataire de service qui fournit une connexion entre le réseau et un appareil est représenté par un objet "Provider " et l'appel par un objet "Call ".

- Programmation événementielle

Les applications réalisées avec XTL sont des machines à état finies. L'état initial correspond à l'établissement de l'appel et l'état final correspond à la coupure de la connexion. Le passage d'un état à un autre est réalisé lorsqu'une modification sur la ligne ou sur l'appareil téléphonique à lieu. Un objet appelé dispatcher fait le lien entre une ligne téléphonique et une application. Il distribue les événements se produisant sur une ligne à l'application concernée.

- Dispatcher, provider et call

Le dispatcher détecte tout changement sur la ligne ou dans l'appareil téléphonique. Lorsqu'il détecte un événement, il transmet le signal adéquat au programme concerné. Il aiguille en quelque sorte les signaux. Lorsqu'un appel est détecté, il appelle directement le constructeur du call et le lie à un provider. Le call fournit des méthodes permettant de travailler directement sur le flot de données circulant sur la ligne téléphonique. Il contient aussi l'implémentation de tous les états du système. Le provider s'occupe de la signalisation sur la ligne téléphonique. Ainsi, le dispatcher appelle des méthodes du call ou du provider, selon que les événements détectés concernent les données sur la ligne ou la signalisation de cette ligne.
Tout programme XTL repose sur un dispatcher, un provider et un call.

4. STRUT

STRUT, acronyme de " Speech Training and Recognition Unified Tool ", a été développé pour faire de la recherche en reconnaissance de la parole et pour le développement d'applications liées à ce domaine. Cet outil est composé de multiples modules indépendants permettant entre autres l'échantillonnage, l'extraction de caractéristiques, la segmentation, l'estimation de probabilité, l'entraînement de modèles et le décodage. L'IDIAP disposant du code source de ce logiciel écrit en C++, l'utilisation de ce logiciel peut se faire de deux manières différentes.

- Ligne de commande

La manière la plus immédiate d'utiliser STRUT est la ligne de commande. Toutes les différentes composantes de STRUT sont exécutables indépendamment. Cependant, elles peuvent communiquer entre elles par le biais de pipes, de socket ou de fichiers. Un programme utilisant STRUT se fait donc en écrivant un script shell qui lance séquentiellement les différents exécutables. Le passage des arguments et des options à chaque exécutable et entre exécutables est décrit dans des fichiers setup ou command.

- Code C++

Les différentes composantes de STRUT sont disponibles sous forme de librairies ou classes C++. On peut donc écrire un programme C++ qui inclue et utilise les librairies STRUT.

5. Voice Dialing

Bien que le Voice Dialing ait été implémenté lors d'un stage d'été, il est utile de mentionner ici quelques unes des ses caractéristiques. En effet, ce démonstrateur est une application intégrant STRUT à XTL De plus, ce programme n'étant pas totalement semblable au Majordome, certaines options pourront être comparées.

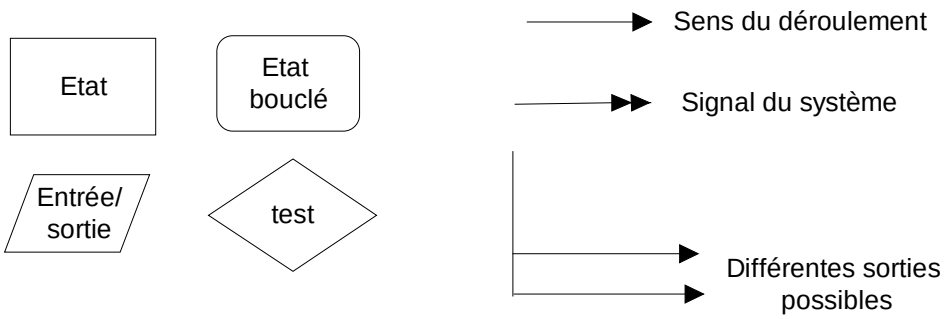
5.1 Description

Le Voice Dialing est un démonstrateur facilitant la numérotation téléphonique en donnant le numéro de téléphone d'une personne dont le nom est prononcé. Dans un premier temps l'utilisateur est prié d'enregistré deux fois le nom d'une personne puis de composer le numéro de téléphone de cette personne. En répétant cette opération, chaque utilisateur se crée un répertoire de numéros. Ensuite, il suffit de prononcer un nom et le démonstrateur donne le numéro associé.

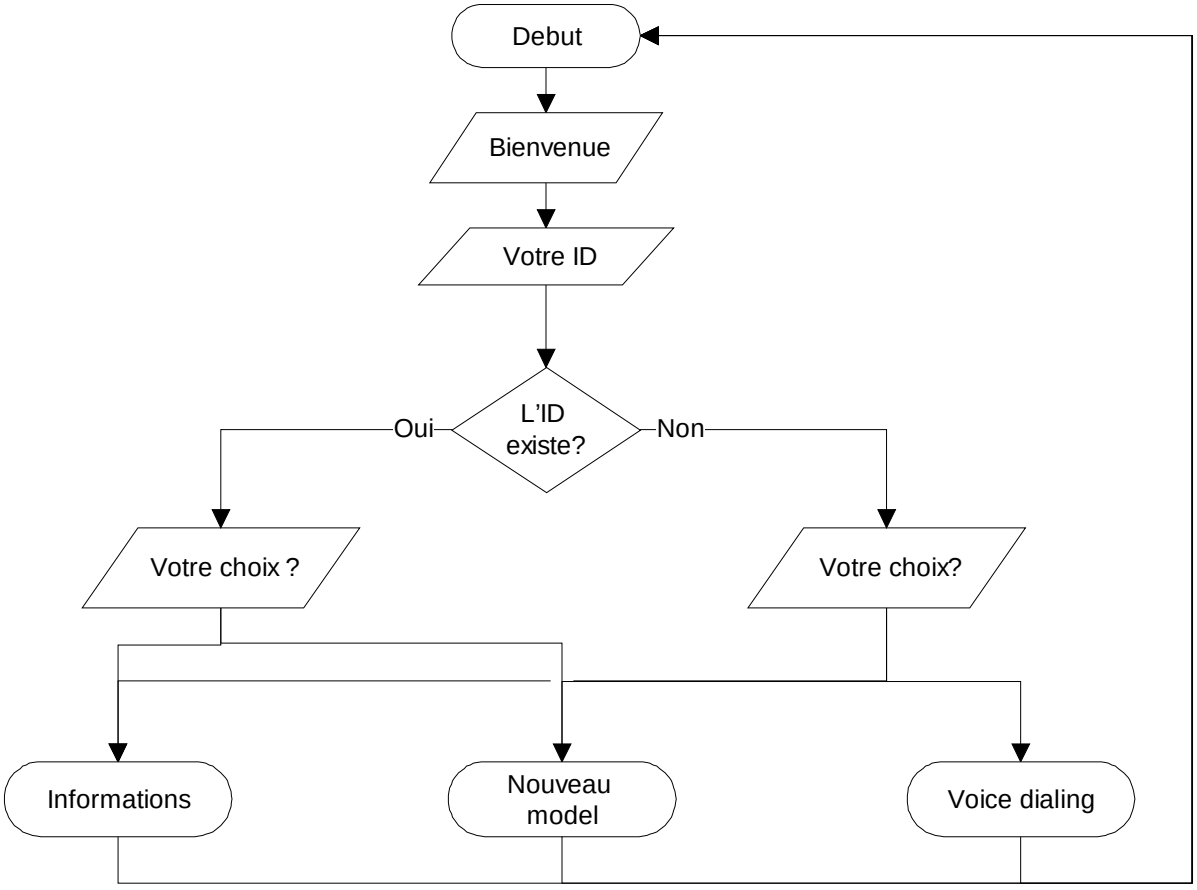
5.2 Organigramme

5.2.1 Légende

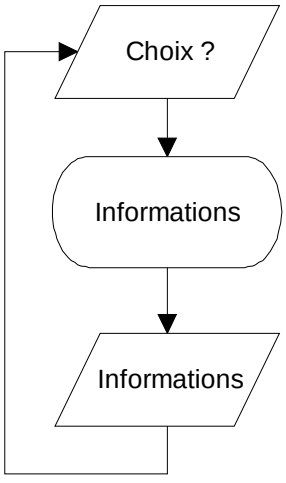
Les conventions suivantes ont été utilisées pour la représentation des différents organigrammes.



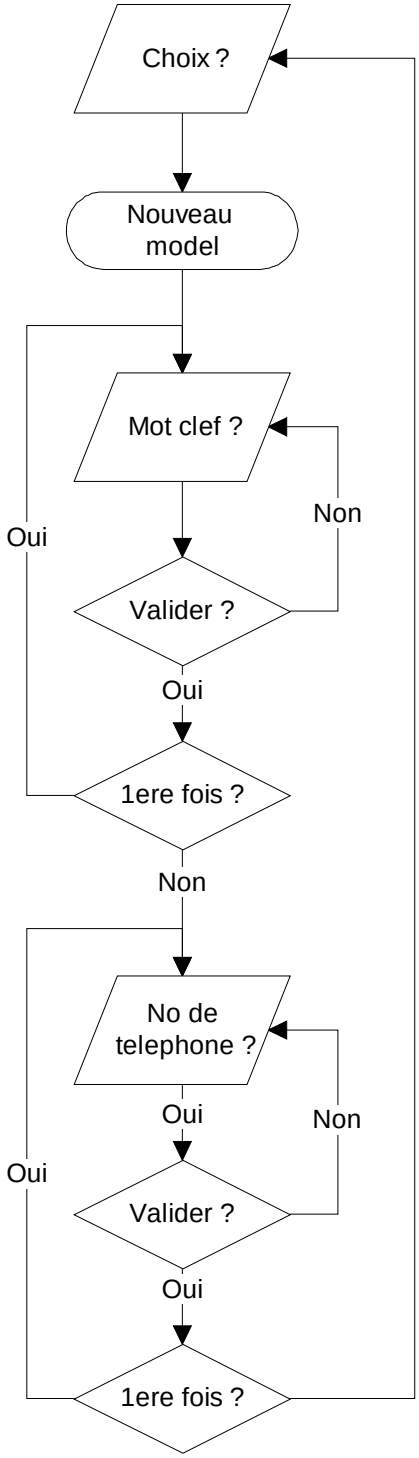
5.2.2 Début



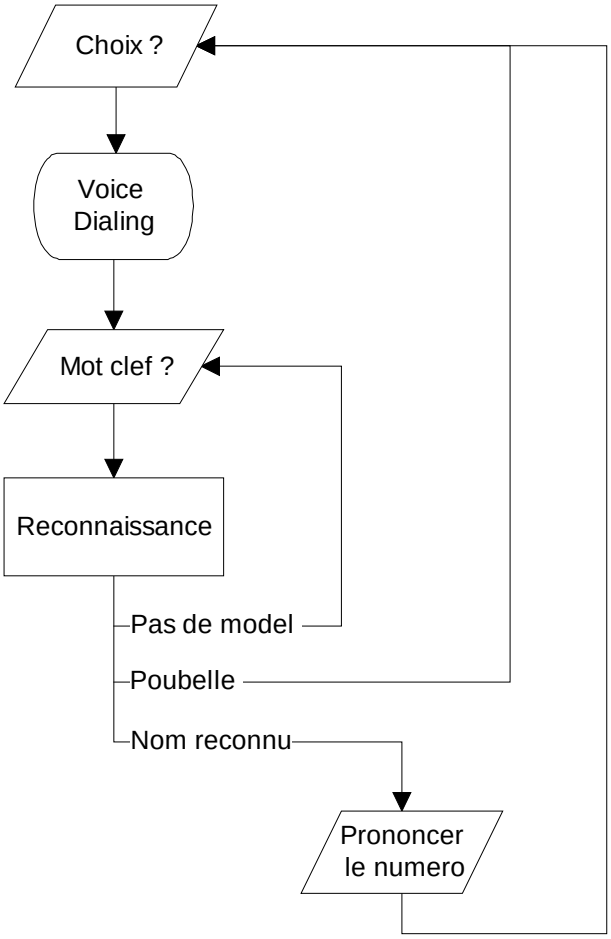
5.2.3 Informations



5.2.4 Nouveau model



5.2.5 Voice Dialing



Le programme se termine lorsque l'utilisateur appuie sur la touche "#".

5.3 Intégration de STRUT dans XTL

Deux choix se présentent pour l'implémentation du Voice Dialing : soit intégrer STRUT à XTL, soit XTL à STRUT. La seule possibilité d'utiliser ces deux logiciels simultanément est d'inclure les méthodes de STRUT dans un programme XTL. Il n'est pas possible d'inclure des méthodes XTL dans un programme STRUT. En effet, comme décrit plus haut, XTL est à programmation événementielle. Un programme utilisant XTL doit nécessairement avoir une boucle infinie sur son dispatcher dans sa méthode *main()* pour tester en continu les changements d'état de la ligne téléphonique. L'ossature du programme est donc obligatoirement celle de XTL avec une boucle sur le dispatcher dans le main(), un provider et un call dans lequel seront implémentés les différents états dans lequel le système peut se trouver. Les méthodes de STRUT étant disponibles dans des classes indépendantes, elles pourront être appelées depuis n'importe quel état du système.
Par ailleurs, STRUT nécessite que différentes variables d'environnement soient définies et que différents paramètres soient passés lors du lancement de l'application. La *fonction ParseCommandLine(int argc, char *argv[])* prend toutes les chaînes de caractères qui suivent le nom du programme tapé à la ligne de commande et les répartit dans les variables STRUT correspondantes. Un script shell est utilisé pour faciliter le lancement du programme.

5.4 Reconnaissance de la parole

La reconnaissance se fait sur les fichiers contenant les enregistrement du locuteur. Deux étapes se distinguent donc : l'enregistrement puis la reconnaissance.

5.4.1 Enregistrement

Des fonctions de XTL sont utilisée pour commander l'enregistrement de la voix du locuteur sur un fichier. La méthode *doRecord (char * toRecord, int nLength, tState nextSate)* implémentée dans le call, prend comme paramètre

- le nom du fichier dans lequel sera enregistré l'appel,
- la durée maximale d'enregistrement en secondes,
- l'état qui suit celui d'où est effectué l'appel à cette méthode.

L'enregistrement se termine soit quand la durée maximale est atteinte, soit lorsque l'utilisateur appuie sur la touche '#' ou la touche '*'. L'enregistrement est composé des valeurs de la modulation par impulsion et codage (PCM) des échantillons de l'appel. L'appel est en effet échantillonné à 8 kHz, puis quantifié et compressé sur 8 bits selon la loi A. Les classes demoCall et demoProv utilisées dans ce démonstrateur dérivent des classe XTLCall et XTLProv qui prennent déjà en compte les paramètres de la ligne téléphonique et fournissent des méthodes de bas niveau pour l'enregistrement et la diffusion de sons.

5.4.2 Reconnaissance

La technique de reconnaissance utilisée ici est la DTW (dynamic time warping). Le call inclus les fichiers de STRUT dont il a besoin, notamment le fichier Model.h qui définit la classe Model. Cette classe a les trois méthodes void *addModel(char *)*, void *recognizeModel(char *, char *)*, void *deleteModel()*. Comme leur nom l'indique clairement, ces méthodes servent à ajouter un modèle à la liste existante, à en rechercher un parmi la liste et à en supprimer un. L'implémentation de ces méthodes fait appel à la classe VoiceDialing. C'est cette classe qui contient toutes les classes et méthodes nécessaires à la reconnaissance de la parole, notamment RastaPLP pour l'extraction des caractéristiques du signal et DTW pour la création des modèles. La classe Model permet de travailler à un niveau d'abstraction relativement élevé et facilite la programmation de la classe call.

5.5 Résultats

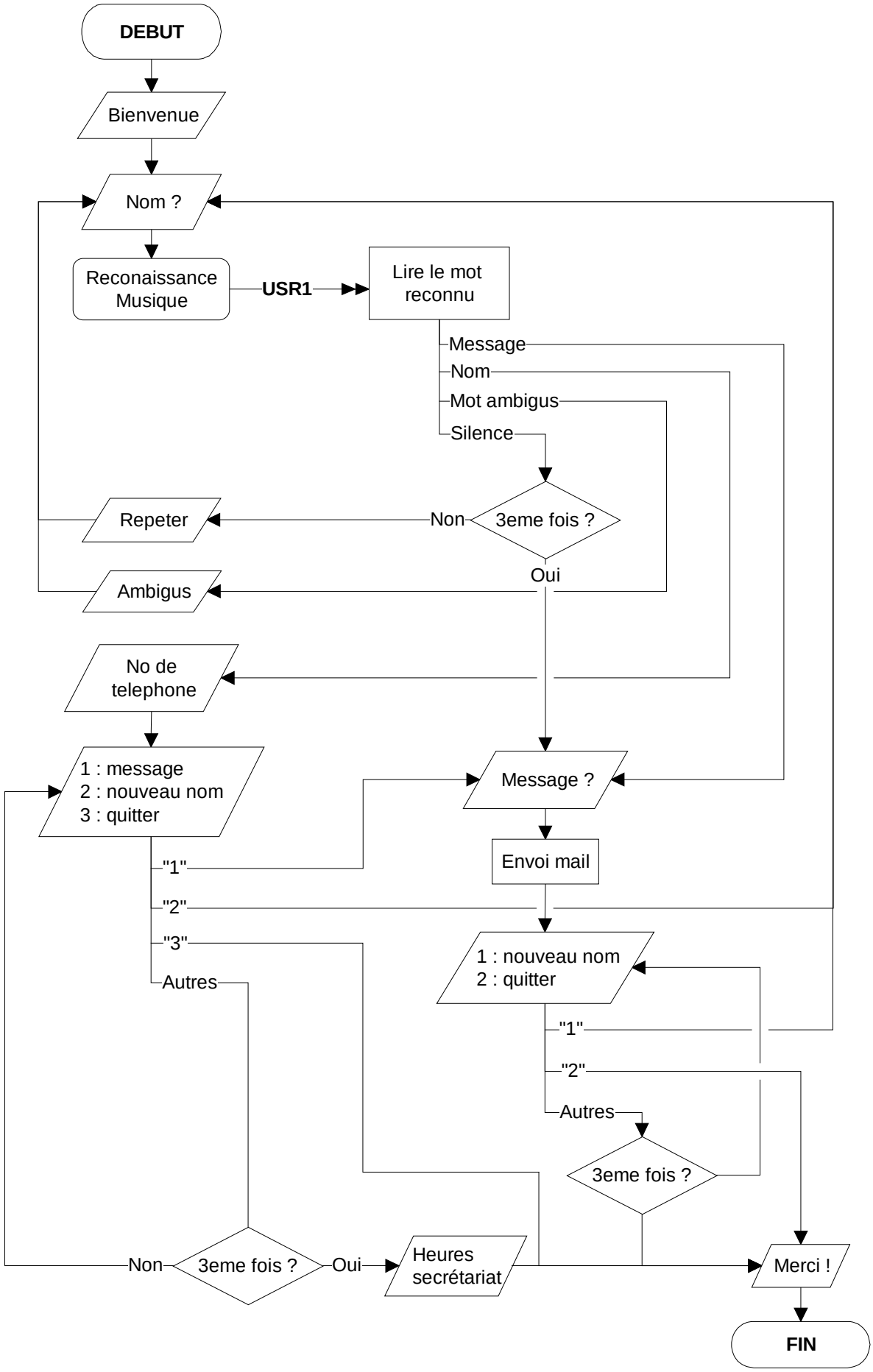
La rapidité d'exécution et de reconnaissance du Voice Dialing sont remarquables. Le déroulement du dialogue est très fluide. La communication sans fichiers des différents modules de STRUT marche ici pleinement. Le temps de réponse pourrait même être amélioré en traitant les données dès leur arrivée, sans attendre la fermeture du fichier d'enregistrement. Cette rapidité se paye par une qualité de reconnaissance limitée, notamment par le niveau du bruit de fond, et par une mise en service laborieuse. Il faut en effet passablement de temps pour se constituer un répertoire, sans compter les fausses manipulations possibles.

6. Le Majordome

6.1 Description

Le Majordome est une application qui permet de transmettre des appels téléphoniques en l'absence de secrétaire. Il demande à l'appelant le nom de son correspondant et donne le numéro interne correspondant. Ensuite il propose de laisser un message, qui sera délivré sous forme d'un enregistrement sur le mail de la personne concernée.

6.2 Organigramme


DEBUT
Bienvenue
Nom ?
Reconaissance Musique
USR1
Lire le mot reconnu
Message
Nom
Mot ambigus
Silence
3eme fois ?
Non
Repeter
Oui
Ambigus
No de telephone
1 : message
2 : nouveau nom
3 : quitter
Message ?
Envoi mail
"1"
"2"
"3"
Autres
1 : nouveau nom
2 : quitter
"1"
"2"
Autres
3eme fois ?
3eme fois ?
Non
Oui
Heures secrétariat
Merci !
FIN

6.3 Implémentation

Comme pour ce qui a été réalisé pour le Voice Dialing, la reconnaissance se fait sur des fichiers contenant les enregistrement de la parole (en PCM quantifiés sur 256 niveaux et comprimés selon la loi A). L'enregistrement proprement dit est réalisé de manière totalement identique à celle décrite pour le Voice Dialing. Cependant, la partie reconnaissance utilise une autre technique et est implémentée de manière totalement différente.

6.3.1 Technique de reconnaissance

Il est évident que le Majordome doit être indépendant du locuteur, vu qu'à priori n'importe qui peut téléphoner. Cette contrainte implique l'abandon du DTW. Par ailleurs, le personnel de l'IDIAP ayant un roulement assez important, de part ses stagiaires, chercheurs invités ou doctorants, le Majordome doit pouvoir être mis à jour simplement et rapidement. En particulier, il faut éviter d'avoir à entraîner un modèle de nom à chaque modification du personnel. La solution proposée ici est d'utiliser des HMMs modélisant les phonèmes de la langue française. Le nom d'une personnes et la succession de phonèmes correspondants sont écrits dans un dictionnaire. Ainsi, une mise à jour se résume à une mise à jour du dictionnaire. Par ailleurs, les modèles de phonèmes sont entraînés une seule fois.
Un inconvénient de ce système est que les phonèmes sont seulement ceux du français et ne décrivent pas très bien la prononciation en d'autres langues. Par ailleurs, le temps de calcul est sensiblement plus long que dans le cas du DTW.

6.3.2 Détails

Contrairement au Voice Dialing, la partie traitant de la reconnaissance de la parole ne fait pas partie intégrante du programme, mais est traité à l'extérieur de celui-ci, à l'aide de scripts shell. Ce choix a été pris pour les raisons suivantes :

- différents logiciels sont capables de gérer des modèles de phonèmes. Le logiciel utilisé est STRUT, mais il pourrait être intéressant de tester le Majordome avec HTK ou d'autres logiciels. Une trop grande intégration ne permettrait plus de tels changements.
- le temps imparti pour ce projet (soit 13 jours) est trop court pour éplucher tous les modules de STRUT afin de choisir quels fichiers doivent être inclus et quelles méthodes utilisées. Il est plus rapide d'utiliser STRUT directement par ses commandes en ligne.

Le programme complet se présente donc en deux parties, l'une consacrée à XTL et à l'interface avec l'utilisateur, l'autre consacrée à la reconnaissance de la parole. Ce choix entraîne malheureusement l'utilisation de passablement de fichiers dont l'écriture et la lecture ralentissent le programme.

6.3.2.1 Code C++

Les cinq fichiers principaux sont myMajor.cc, majorProv.h, majorProv.cc, majorCall.h, majorCall.cc.
Le fichier myMajor comprend, dans le main(), l'appel au constructeur du provider majorProv et une boucle infinie sur le dispatcher. Ce fichier comprend aussi quelques méthodes concernant les signaux qui peuvent être envoyés au programme (cf. communications entre processus).
Les fichiers majorProv.h et majorProv.cc sont quasiment vide. En effet, ils implémentent la classe majorProv qui dérive de la classe XTLProv. Celle-ci implémente un provider complet.
Le corps du programme se trouve dans majorCall.h et majorCall.cc qui implémentent le call. majorCall.h comprend notamment l'énumération de tous les noms de fichier utilisés pour les dialogues et la reconnaissance ainsi que l'énumération de tous les états. majorCall.cc contient l'implémentation de l'ensemble des états du système et des méthodes nécessaire au passage de l'un à l'autre :

- méthodes d'enregistrement et de diffusion *doRecord* et *doPlay*,
- méthodes de traitement des signaux DTMF. Les signaux DTMF (chiffres de 0 à 9, * et #) sont détectés par *keyPressed(char& c)*. L'étoile est traitée de manière à forcer le passage immédiat à l'état suivant lors d'un enregistrement ou d'une diffusion. Le dièse est interprété comme l'ordre de quitter l'application. Les chiffres sont stockés dans un tableau de caractères (dtmfCode) dont la longueur est enregistrée dans la variable dtmfIndex par la méthode *updatePin(char& c)*.
- méthodes de changement d'état. Lorsque les touches * ou # sont pressées, il faut pouvoir terminer un enregistrement ou une diffusion en cours et faire passer le système à un nouvel état. Ces cas sont traités directement dans *keyPressed(char& c)*, déjà décrit ci-dessus ,afin de garantir une réaction rapide du système. Dans ces cas le tableau dtmfCode n'est pas mis à jour.
- méthodes d'interruption. Pour les raisons décrites au paragraphe communication entre processus, le programme doit pouvoir traiter des signaux qu'il reçoit d'autre processus. La méthode interrupt aiguille le

programme vers le bon traitement d'interruption selon l'état dans lequel il se trouvait lorsque l'interruption est détectée.

- méthode décrivant les états. Le passage d'un état à un autre ne se fait qu'après un enregistrement, une diffusion ou une interruption (cas exceptionnel). Chaque état comprend donc un appel soit à *doPlay()*, soit à doRecord() en précisant quel fichier lire ou écrire et l'état dans lequel le programme doit passer. Lorsqu'un appel est détecté, le constructeur du majorCall est appelé suivit de la méthode *firstAction()*. *firstAction()* appel *doPlay()* et le programme passe ensuite à la méthode *doAction()*. Cette méthode contient l'implémentation de tous les états. Une structure en *switch()* permet d'aiguiller le programme au le bon état. Remarquons que dans certains états, il n'y a rien à enregistrer ou à diffuser. Dans ce cas, pour pouvoir quand même changer d'état, *doPlay* est appelé avec le fichier SPEAK_NOTHING (fichier vide) comme paramètre.

Le déroulement du programme est donc le suivant. Le dispatcher passe un appel au provider, crée un call et appel *firstAction()*. Lorsque *doPlay* est effectué, la diffusion du fichier correspondant est lancée, l'état du programme est mis à jour et le call sort de la méthode *firstAction*. Le call se retrouve donc momentanément oisif. Lorsque la fin de la diffusion est détectée, le dispatcher appelle la méthode *afterPlay()* qui appelle à son tour *doAction* avec le nouvel état du programme. Les opérations de l'état sont effectuées, terminées par un *doPlay* ou *doRecord*. L'état est remis à jour et le call est oisif jusqu'à ce que le dispatcher rappelle la méthode *doAction*, etc.

6.3.2.2 Scripts

Les scripts sont appelés pour utiliser STRUT à l'aide de la ligne de commande. La structure générale d'une instruction est fonction database_id/database_version setup= <fichier de setup> <options>. La base de donnée utilisée doit être impérativement enregistrée dans un répertoire contenant son nom et un sous-répertoire contenant son numéro de version. Le fichier de setup permet de regrouper toutes les options dans un fichier, plutôt que de devoir les aligner sur la ligne de commande. Le fichier de setup permet de préciser aussi le nom d'un fichier de commande contenant les noms de fichiers nécessaires à la fonction. Exemple :
rasta majordome/1.0 setup= ${base_dir}/reco/rasta.stp
où rasta.stp contient :

```
command= ${STRUT_DIR}/database/${DATABASE}/reco/rasta.cmd
frame-length= 30
frame-shift= 10
sample-rate= 8000
coefficient-count= 12
log-rasta= yes
mix-coeff= 1.0
energy= yes
```

et command.stp

```
/home/speeh08/vannay/Strut/database/majordome/1.0/alaw/name.tmp
/home/speeh08/vannay/Strut/database/majordome/1.0/name.rasta
```

Cette commande procède à l'extraction des coefficients rasta des échantillons contenus dans le fichier name.tmp et les stocke dans le fichier name.rasta.
Le script fait appel séquentiellement à plusieurs instructions, qui peuvent être liées par des pipes. On évite ainsi de trop avoir recours à des fichiers pour transmettre des résultats d'une fonction à l'autre.
Le script reco.tcsh contient la suite des appels de fonctions nécessaires à la reconnaissance d'un mot. Les coefficients rasta sont extraits par *rasta*, puis sont mis sous la forme d'un fichier d'archives par create-archive. Les probabilités sont calculées par qn-forward et le résultat et choisi dans la liste des mots du dictionnaire par isolated (cf. annexes).

6.3.2.3 Programme utilitaire

Les enregistrement sont utilisés à deux occasions. La première est évidemment pour enregistrer de la parole. La deuxième est pour la détection des signaux DTMF. En effet, le programme doit être soit en état de diffusion, soit en état d'enregistrement pour pouvoir détecter les signaux DTMF (i.e. soit pendant un *doPlay*, soit pendant un *doRecord*). Lorsque l'utilisateur est prié de presser sur une touche, par exemple, le programme passe directement de son état de diffusion de la demande à celui d'enregistrement. Un enregistrement doit donc être achevé dans trois cas :

- dans tous les cas : si le temps d'enregistrement expire,
- si une touche est attendue : dès qu'un signal DTMF est détecté,
- si de la parole est enregistrée : dès que le locuteur à fini de parler.

Le premier cas est résolu à l'aide d'un simple timer et est géré par le provider. Le deuxième cas est aussi simple : le tableau de caractère dtmfCode doit être initialisé à l'aide de la méthode *clearDtmf(int)* qui permet de définir la taille maximale de dtmfCode. Lorsque celle si est atteinte, l'enregistrement est interrompu. Le troisième cas est

moins trivial. Il faut en effet écouter en temps réel la ligne pour détecter les silences et interrompre l'enregistrement lorsqu'on en a détecté un. Le programme searchSilence avait été développé pour détecter les silence à l'aide du niveau d'énergie du signal ainsi que de la variation de se niveau. Ce programme a dû être adapté pour les échantillons compressé selon la loi A. Il est lancé parallèlement à un enregistrement et interrompt celui-ci dès qu'il détecte un silence.(cf. annexes)

6.3.2.4 Fichiers divers

Les noms et coordonnées des personnes atteignables sont stockées dans plusieurs fichiers. Le dictionnaire "dictionary" (cf. annexes) contient les noms et leur différentes transcriptions phonétiques. Le fichier "ambigus" contient les noms ou prénoms des personnes homonymes. Dans le cas actuel, Hervé Bourlard et Hervé Glottin ayant même prénom, celui-ci est stocke dans "ambigus" et permet au programme de demander des précisions si seul "Hervé" est détecté. Le fichier "Phonenumbers" contient la liste de personnes avec le numéro de téléphone et l'adresse e-mail associes. Par ailleurs plusieurs fichiers sont utilises pour passer un résultat d'un programme à un autre.

6.4 Communication entre processus

searchSilence et reco.tcsh doivent pouvoir être appelés depuis le programme principal et transmettre à celui-ci qu'ils ont terminé leur exécution. Pour ce faire, on utilise le signal système USR1. Le pid du programme principal est passé dans l'appel de searchSilence et de reco.tcsh. Ainsi, ils savent où envoyer le signal USR1 lorsque l'exécution est terminée. La méthode interrupt de majorCall décrite ci-dessus permet d'intercepter ce signal et de le traiter correctement.

Ainsi lorsque le programme demande à l'utilisateur le nom de son correspondant, searchSilence est lancé et le programme passe dans un état d'enregistrement. Puis, quand searchSilence détecte un silence, il envoie le signal USR1 au programme principal qui arrête l'enregistrement et passe dans l'état de reconnaissance de la parole. Là, reco.tcsh est appelé et le programme se place dans un état bouclé sur lui même qui diffuse une musique. Cela permet de faire patienter l'utilisateur de manière plus agréable durant un temps indéterminé (temps de calcul de la reconnaissance qui peut prendre 2 ou 3 secondes sur la station eiger lorsqu'elle n'est pas trop chargée). Lorsque la reconnaissance a eu lieu, reco.tcsh envoie le signal USR1 au programme principal qui quitte l'état bouclé sur lui-même et peut donc continuer son exécution.

Par ailleurs les signaux TERM, INT, TSTP, sont aussi interceptés afin de terminer le programme de manière propre par l'appel des méthodes adéquates.

6.5 Résultats

Comme prévu, le temps pris par la reconnaissance est nettement plus long que dans le cas du Voice Dialing. Cela vient d'une part de la technique de reconnaissance utilisée, l'utilisation des HMMs étant plus lente que celle du DTW. D'autre part, l'utilisation de fichiers pour passer les résultats d'un processus à l'autre multiplie les entrées sorties qui sont assez gourmande en temps. reco.tcsh doit, par exemple, écrire le nom reconnu dans un fichier déterminé, puis le programme principal doit aller lire ce fichier. Enfin la structure même de l'application en un programme principal, des scripts et un programme secondaires implique des pertes de temps liées aux communications entre processus. Dans Voice Dialing, l'intégration de STRUT au programme principale permettait d'utiliser au maximum les capacités de communication des processus STRUT et de passer les résultats par variables.

Une première version du majordome était commandée uniquement par la voix et ne nécessité pas l'utilisation du clavier du téléphone. Cette version était certes fonctionnelle, mais était d'une ergonomie désastreuse. Lorsqu'on voulait quitter le programme sans laisser de message, il fallait attendre 2 ou 3 secondes que le programme reconnaisse qu'on avait répondu "non" à sa question. Dans la version finale, certains choix se font à l'aide des touches du téléphone. Le confort d'utilisation gagné compense largement le fait que ce mode de communication soit nettement moins naturel que la parole. Contrairement au Voice Dialing, l'équilibre entre les commandes vocales et celles par touches est assez bon. L'utilisateur ne passe donc pas son temps à pianoter.

7. Conclusion

L'étape préalable de ce projet, réalisée lors de mon stage en août et septembre m'ont permis de me familiariser avec l'écriture de scripts shell sous Unix, avec les environnements de XTL et STRUT, les problèmes liés à la reconnaissance de la parole et les techniques utilisées. L'implémentation du Voice Dialing est une application concrète illustrant cette phase de mon travail et constitue un point de comparaison pour le Majordome. Les problèmes liés à l'intégration de STRUT à XTL ont notamment étés traités. L'implémentation du Majordome m'a confronté aux problèmes liés au temps de calcul d'une reconnaissance, aux HMMs, à la communication entre

processus, à l'intégration du mail dans une application, au transfert de fichier audio par ce biais et à l'ergonomie d'une application.
Le Majordome, sous sa forme actuelle est certes fonctionnel et ergonomique, mais différentes améliorations pourraient être encore faites. Premièrement, la commutation effective de l'appelant devrait être réalisée. Il n'est pas difficile de l'implémenter à l'aide de XTL, mais dans ce cas, la commutation se fait au niveau de la carte ISDN et occupe les deux lignes à disposition. Le Majordome pourrait traiter au plus une communication. Une autre solution serait de programmer directement le central Ascom, mais ceci nécessite la connaissance détaillée des protocoles et ne peut être porte sur un autre central. Deuxièmement, une interface graphique serait souhaitable pour la mise à jour des personnes atteignables. En effet, la modification d'une personne nécessite actuellement la mise jour du dictionnaire, du fichier de nom ou prénom ambigus et du fichier de numéros téléphoniques et d'adresses e-mail. Il serait judicieux de regrouper les entrées sur une seule interface graphique. Troisièmement, les models de phonèmes utilises devraient permettre de prendre en compte des prononciations étrangères. Dernièrement, il serait souhaitable de pouvoir accéder à un mode dans lequel on peut épeler le nom du correspondant recherche de manière à pouvoir atteindre une personne, même si son nom n'est pas reconnu de manière standard.

Lausanne, le 16 janvier 1998 Samuel Vannay

8. Bibliographie

XTL Application Programmer'sGuide, *SunSoft*, Montain View, CA, 1994
STRUT User's guide, http://tcts.fpms.ac.be/speech/strut/users-guide/users-guide.html
Traitement de la parole, Complément au Trait d'Electricité, *René Boite et Murat Kunt*, PPUR, 1987
RASTA Processing of speech, *H. Heransky, N. Morgan*, IEEE transactions on speech and audio processing, vol 2, no 4, october 1994
Systèmes de télécommunication, Traite d'électricité, vol XVIII, *Pierre-Gerard Fontolliet*, PPUR, Lausanne, avril 1996
Speaker Dependent connected Speech Recognition via Dynamic Programming and Statistical Methode, *Bourlard, Kamp, Ney, Welleckens*, in Speech an Speaker Recognition, Kreger, Basel, 1985
Théorie des communication, course notes for the 3rd year students in Communication system Engineering, *P. Thiran*, EPFL, 1993

9. Annexes

9.1 Sources C++

9.1.1 myMajor.cc

```
//======================================================================
char*    ProgramTitle = "Majordome vocal";
char*    Author       = "Samuel Vannay";
char*    AuthorEmail  = "Samuel.Vannay@idiap.ch";
char*    Version      = "0.00";
//======================================================================


#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>
#include <signal.h>

#include "majorProv.h"

boolean_t End;

// pour terminer "proprement" le programme
void SigTerm( int Code )
{

  End = B_TRUE;

}


// protection, un signal USR1 ne va ainsi pas tuer le process
void SigUsr1( int Code )
{
  signal( SIGUSR1, SigUsr1 );
}


main(int argc,char **argv)
{
  char pvname[] = "";
  majorProvider *provider;
  AttacksProvider::Exception err;
  dpDispatcher::instance(new dpSLDispatcher);
  dpDispatcher &d = dpDispatcher::instance();


  // Presentation et initialisation
```

```
  cout << argv[ 0 ] << " : Started" << endl;
  End = B_FALSE;

  // installation de la procedure de fin
  signal( SIGTERM, SigTerm );
  signal( SIGINT, SigTerm );
  signal( SIGTSTP, SigTerm );
  signal( SIGUSR1, SigUsr1 );


  //Connect au provider specifie par pvname. Si pvname est NULL XTL
  //essaie de demarrer le provider par defaut
  provider = new majorProvider(err, pvname, "Samuel" );
  if (err != AttacksProvider::EXCEPTION_SUCCESS)
    {
      cerr << argv[ 0 ] << " : main:could not connect to provider" << endl;
      return 1;
    }


  //boucle sur le dispatcher
  while( !End )
    d.dispatch();

  delete provider;

  cout << argv[ 0 ] << " : Ended" << endl;

  return 0;
}
```

9.1.2 majoProv.h

```
#ifndef MAJORPROV_H
#define MAJORROV_H

#include "AttacksProv.h"

class majorProvider : public AttacksProvider
{
 public:
  majorProvider( Xtl::Exception& err, XtlString pname , char *szId );
  virtual void createCall( XtlCallState& call );

};

#endif
```

9.1.3 majorProv.cc

```
#include <iostream.h>
#include <stdlib.h>

#include "majorProv.h"
#include "majorCall.h"


majorProvider::majorProvider(Xtl::Exception& err, XtlString pname,
                      char *szId ) : AttacksProvider(err, pname, szId)

{}


// A call has been offered for ownership and the AttacksProvider object will
// attempt to claim it by creating a new call object using the call state
// of the offered call.

void majorProvider::createCall( XtlCallState& call )
{
  Xtl::Exception e = EXCEPTION_UNKNOWN;
```

```
  inCall = new majorCall( e, call, this, inCallIndex, nCalledNumber );
}
```

9.1.4 majorCall.h

```
#ifndef MAJORCALL_H
#define MAJORCALL_H

#include <xtl/xtl.h>
#include <Dispatch/sldispatcher.h>
#include <task.h>
#include "AttacksCall.h"

//Miscellaneous constants
#define INDEX_MAX_PROC  5
#define NB_FILES_SIZE 10
#define DTMF_DEFAULT_LENGTH 20
#define DTMF_CHOICE_LENGTH 1
#define DTMF_PHONE_LENGTH 15

//Important files and pathes
#define PROMPT_PATH         "/home/speech08/vannay/majordome/prompts/"
#define RECORD_PATH
"/home/speech08/vannay/Strut/database/majordome/1.0/alaw/"
#define RECO_FILE
"/home/speech08/vannay/Strut/database/majordome/1.0/reco/reco.name"
#define AMBIGUOUS_FILE
"/home/speech08/vannay/Strut/data/majordome/1.0/ambigus"
#define ADDR_FILE
"/home/speech08/vannay/Strut/data/majordome/1.0/PhoneNumbers"
#define NAME_FILE           "name.au"
#define MSG_FILE            "msg.raw"
#define CONVERTED_FILE      "msg.au"
#define MIME_FILE           "msg.mime"
#define CHOICE_FILE         "choice.au"
#define DTMF_FILE           "dtmf.tmp"
#define AUDIO_FILE          "audio.au"
#define KEYWORD_FILE        "tmp.au"

//Introduction and end messages
#define WELCOME             "bienvenue.au"
#define SPEAK_SECR_HOURS    "horaire.au"
#define SPEAK_BYE           "merci.au"

//Reco model messages
#define ASK_NAME            "prononcer.au"
#define WAIT_MUSIK          "musique.au"
#define SPEAK_NOTHING       "nothing.au"
#define SPEAK_NAME1         "no_de_tel.au"
#define SPEAK_NAME2         "est_le.au"
#define SPEAK_REPEAT        "repeter.au"
#define SPEAK_ASK_KEY1      "choices1.au"
#define SPEAK_ASK_KEY2      "choices2.au"
#define SPEAK_GIVE_MSG      "msg.au"
#define SPEAK_ASK_MSG       "demande_msg.au"
#define SPEAK_AMBIGUOUS     "preciser.au"
#define SPEAK_YES_NO        "oui_non.au"

//Default address to mail to
#define DEFAULT_RECIPIENT   "samuel.vannay"

//States
enum _State {
MAJOR_START = 0,
MAJOR_ASK_NAME,
MAJOR_RECORD_NAME,
MAJOR_NAME_RECO,
MAJOR_WAIT_NAME,
MAJOR_NAME_SWITCH,
MAJOR_SPEAK_NAME1,
```

```
MAJOR_SPEAK_NAME2,
MAJOR_SPEAK_NAME3,
MAJOR_SPEAK_NAME4,
MAJOR_REPEAT_NAME,
MAJOR_AMBIGUOUS,
MAJOR_ASK_MSG,
MAJOR_ASK_KEY1,
MAJOR_GET_KEY1,
MAJOR_CHOICE_SWITCH1,
MAJOR_RECORD_MSG,
MAJOR_SEND_MAIL,
MAJOR_ASK_KEY2,
MAJOR_GET_KEY2,
MAJOR_CHOICE_SWITCH2,
MAJOR_BYE,
MAJOR_END
};

typedef enum _State tState;

class majorCall : public AttacksCall, public Interrupt_handler
{

public:
  majorCall( Xtl::Exception&  e,
                XtlCallState&    cs,
                AttacksProvider* p,
                short            id,
                int theCalledNumber );

  virtual void interrupt();

  virtual void firstAction();
  virtual void start();

protected:

  char dtmfCode[255];
  long dtmfAdLong;
  int  dtmfIndex;
  int  dtmfLength;
  long dtmfAsLong;

  long nMessageNumber;

  char szBaseFileNameRecord[ 200 ];
  char szTmpRecord[ 200 ];
  tState myState;
  pid_t proc[ INDEX_MAX_PROC ];
  int nCalledNumber;           // which number is called now in integer ???
  char szCalledNumber[2];      // which number is called now or in string ???
  boolean_t bCanInterrupt;     // for not interrupting messages during
dialogue
  boolean_t bMusic;            // for interrupting the music

  // for simplify the sequencement actions
  virtual void doPlay( char *toPlay, tState nextState );
  virtual void doRecord( char *toRecord, int nLength, tState nextState );

  virtual void doAction();
  virtual void afterPlay();
  virtual void afterRecord();

  virtual void recordDone() ;
  virtual void keyPressed(char& c);

  virtual void clearDtmf(int);
  virtual void updatePin(char&);

private:
```

```
  boolean_t InDTMF;
  int tryNb;
  char recoName[256];
  char recoNumber[64];
  char recipient[256];
  boolean_t secrHours;
  void endAction();
  void createAudioFile(char *);
};

#endif
```

9.1.5 majorCall.cc

```
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <math.h>
#include <unistd.h>
#include <task.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <iostream.h>
#include <fstream.h>
#include <errno.h>

#include "majorCall.h"
#include "FilePlayer.h"


majorCall::majorCall(Xtl::Exception&  e,
                  XtlCallState&    cs,
                  AttacksProvider* p,
                  short            id,
                  int theCalledNumber )
  : AttacksCall( e, cs, p, id ),Interrupt_handler( SIGUSR1 )
{
  nMessageNumber = 0;
  tryNb = 0;

  nCalledNumber = theCalledNumber;
  sprintf( szCalledNumber, "%d", nCalledNumber );
  bCanInterrupt = B_TRUE;
  bMusic = B_FALSE;
  secrHours = B_FALSE;

  clearDtmf(DTMF_DEFAULT_LENGTH);
}


// les shells lances par ce programme qui sont termines peuvent
// avertir celui-ci qu'ils ont finis via ce signal
void majorCall::interrupt()
{
  if (bMusic) {
    switch( myState ) {
    case MAJOR_WAIT_NAME:
      bMusic = B_FALSE;
      myState = MAJOR_NAME_SWITCH;
      endAction();
      break;
    }
  }
  if( bCanInterrupt ) {
    switch( myState ) {
    case MAJOR_NAME_RECO:
    case MAJOR_RECORD_NAME:
    case MAJOR_NAME_SWITCH:
```

```
    case MAJOR_CHOICE_SWITCH1:
    case MAJOR_CHOICE_SWITCH2:
    case MAJOR_SEND_MAIL:
      endAction();
      break;

    default:
      endAction();
      cerr << endl << "Interrupt default case" << endl;
      break;
    }
  }
}


//termine proprement un record ou un play
void majorCall::endAction()
{
  if( recorder )
    recordDone();
  else {
    dpDispatcher& d = dpDispatcher::instance();
    d.stopTimer((dpIOHandler*)player);
    player->timerExpired( 0, 0 );
  }
}


void majorCall::doPlay( char *toPlay, tState nextState )
// - diffuse toPlay
// - initie le pas suivant
{
  char szFileName[ 255 ];   // the real file name to play

  // compute the file name to play
  strcpy( szFileName, PROMPT_PATH );
  strcat( szFileName, toPlay );

  cerr << "Playing : " << szFileName << endl;

  playAnnc( szFileName );

  // go to the next step
  myState = nextState;
}


void majorCall::doRecord( char *toRecord, int nLength, tState nextState )
  // - enregistre toRecord
  // - initie le pas suivant
{
  char szFileName[ 255 ];   // the real file name to play

  // compute the file name to record

  sprintf(szFileName, "%s/%s", RECORD_PATH, toRecord);
  // record
  recordLenght = nLength;
  recordMsg( szFileName );

  // go to the next step
  myState = nextState;

}


// ------------------------------------------------------------------------
-
//
// First action executed by the server
```

```
//
// ------------------------------------------------------------------------
-
void majorCall::firstAction()
{
  // access to the IVS
  doPlay(WELCOME, MAJOR_START );
}


void majorCall::start()
{
  char szStartRecordTime[ 50 ];
  time_t currentTime;

  doPlay( SPEAK_NOTHING, MAJOR_ASK_NAME);

}

//compose le fichier audio a partir des chiffres
//du numero de telephone
void majorCall::createAudioFile(char *number)
{
  int i=0;
  char tmp[2];
  char command[200] = "cat";

  tmp[1] = '\0';
  while (number[i] != '\0') {
    tmp[0] = number[i];
    strcat(command, " ");
    strcat(command, PROMPT_PATH);
    strcat(command, tmp);
    strcat(command, ".au");
    i++;
  }
  strcat(command, " > ");
  strcat(command, PROMPT_PATH);
  strcat(command, AUDIO_FILE);
  system(command);
}


void majorCall::doAction()
{

  ifstream recoFile(RECO_FILE);
  if (!recoFile) {
    cerr << "File recoFile cannot be opened" << endl;
    exit(1);
  }

  ifstream ambiguousFile(AMBIGUOUS_FILE);
  if (!ambiguousFile) {
    cerr << "File ambiguousFile cannot be opened" << endl;
    exit(1);
  }

  ifstream addressFile(ADDR_FILE);
  if (!addressFile) {
    cerr << "File addressFile cannot be opened" << endl;
    exit(1);
  }

  tState  nextState;
  char temp[2048];
  char recordedFile[512];
  pid_t currentProc;
  int status;
  boolean_t ambiguous = B_FALSE;
```

```
  //To send the message
  char header[1024];
  char subject[256];
  Xtl::Exception e;
  XtlCallState & callState = call_state(e);
  sprintf(temp, "%s/%s", RECORD_PATH, MIME_FILE);
  FILE *mimeFile = fopen(temp, "w");

  //test si le fichier audio contenant le nom d'une personne
  //existe afin d'eviter une erreur fatale
  if (myState == MAJOR_SPEAK_NAME1) {
    sprintf(temp, "%s/%s.au", PROMPT_PATH, recoName);
    ifstream testFile(temp);
    if (!testFile) {
      myState = MAJOR_ASK_NAME;
    }
  }

  //Implementation des etats
  cerr << "============" << myState << "=============\n";

  switch ( myState )
    {

    case MAJOR_START:
      bCanInterrupt = B_TRUE;
      start();
      break;

    case MAJOR_ASK_NAME:
      bCanInterrupt = B_TRUE;
      doPlay(ASK_NAME, MAJOR_RECORD_NAME);
      break;

    case MAJOR_RECORD_NAME:
      bCanInterrupt = B_TRUE;
      currentProc = getpid();
      sprintf(temp, "%li", currentProc);
      sprintf(recordedFile, "%s/%s", RECORD_PATH, NAME_FILE);
      proc[0]=fork();
      if( proc[0] == 0 )
      {
        execlp( "/home/speech08/vannay/majordome/utils/searchSilence",
"searchSilence", "204", "80", recordedFile, temp, "80", 0);
        exit(1);
      }
      doRecord(NAME_FILE, 20, MAJOR_NAME_RECO);
      break;

    case MAJOR_NAME_RECO:
      bCanInterrupt = B_FALSE;
      waitpid(proc[0], &status, WNOHANG);
      currentProc = getpid();
      proc[1] = fork();
      if (proc[1] == 0) {
       sprintf(temp, "%s %li",
"/home/speech08/vannay/Strut/database/majordome/1.0/reco/reco.tcsh",
currentProc);
       system(temp);
       exit(1);
      }
      doPlay (SPEAK_NOTHING, MAJOR_WAIT_NAME);
      break;

    case MAJOR_WAIT_NAME:
      bCanInterrupt = B_FALSE;
      bMusic = B_TRUE;
      doPlay (WAIT_MUSIK, MAJOR_WAIT_NAME);
      break;
```

```
    case MAJOR_NAME_SWITCH:
      bCanInterrupt = B_FALSE;
      waitpid(proc[1], &status, WNOHANG);
      recoFile >> recoName;
      while (!ambiguousFile.eof() && !ambiguous) {
      ambiguousFile >> temp;
      if (strcmp(temp, recoName)) {
        continue;
      }
      else {
        ambiguous = B_TRUE;
        nextState = MAJOR_AMBIGUOUS;
        break;
      }
      }

      if (!ambiguous) {
      //Command and control words only
      if ( (strcmp(recoName, "LS") == 0) || (strcmp(recoName, "TS") == 0)
           || (strcmp(recoName, "") == 0)) {
        tryNb++;
        if (tryNb < 3) {
          nextState = MAJOR_REPEAT_NAME;
        }
        else {
          sprintf(recipient, "%s", DEFAULT_RECIPIENT);
          secrHours = B_TRUE;
          nextState = MAJOR_ASK_MSG;
        }
      }
      else if (strcmp(recoName, "MESSSAGE") == 0) {
        sprintf(recipient, "%s", DEFAULT_RECIPIENT);
        nextState = MAJOR_ASK_MSG;
      }
      else {
        nextState = MAJOR_SPEAK_NAME1;
      }
      }
      doPlay(SPEAK_NOTHING, nextState);
      break;

    case MAJOR_SPEAK_NAME1:
      bCanInterrupt = B_TRUE;
      while (!addressFile.eof()) {
      addressFile >> temp >> recoNumber >> recipient;
      if (strcmp(temp, recoName)) {
        continue;
      }
      else {
        cout << endl << endl << "Le num. de  " << recoName << " est le " <<
recoNumber << endl << endl << endl;
        break;
      }
      }
      doPlay(SPEAK_NAME1, MAJOR_SPEAK_NAME2);
      break;

    case MAJOR_SPEAK_NAME2:
      bCanInterrupt = B_TRUE;
      sprintf(temp, "%s.au", recoName);
      doPlay(temp, MAJOR_SPEAK_NAME3);
      break;

    case MAJOR_SPEAK_NAME3:
      bCanInterrupt = B_TRUE;
      doPlay(SPEAK_NAME2, MAJOR_SPEAK_NAME4);
      break;

    case MAJOR_SPEAK_NAME4:
```

```
      bCanInterrupt = B_TRUE;
      createAudioFile(recoNumber);
      tryNb = 0;
      doPlay(AUDIO_FILE, MAJOR_ASK_KEY1);
      break;

    case MAJOR_REPEAT_NAME:
      bCanInterrupt = B_TRUE;
      doPlay(SPEAK_REPEAT, MAJOR_RECORD_NAME);
      break;

    case MAJOR_AMBIGUOUS:
      bCanInterrupt = B_TRUE;
      doPlay(SPEAK_AMBIGUOUS, MAJOR_RECORD_NAME);
      break;

    case MAJOR_ASK_MSG:
      bCanInterrupt = B_TRUE;
      doPlay(SPEAK_GIVE_MSG, MAJOR_RECORD_MSG);
      break;

    case MAJOR_ASK_KEY1:
      bCanInterrupt = B_TRUE;
      clearDtmf(DTMF_CHOICE_LENGTH);
      doPlay(SPEAK_ASK_KEY1, MAJOR_GET_KEY1);
      break;

    case MAJOR_GET_KEY1:
      bCanInterrupt = B_TRUE;
      tryNb++;
      doRecord(DTMF_FILE, 10, MAJOR_CHOICE_SWITCH1);
      break;

    case MAJOR_CHOICE_SWITCH1:
      bCanInterrupt = B_FALSE;
      if (tryNb < 3) {
       switch (dtmfCode[0]) {
       case '1':
         nextState = MAJOR_ASK_MSG;
         break;
       case '2':
         nextState = MAJOR_ASK_NAME;
         break;
       case '3':
         nextState = MAJOR_BYE;
         break;
       default:
         nextState = MAJOR_ASK_KEY1;
         break;
       }
       }
       else {
       nextState = MAJOR_BYE;
       }
      doPlay(SPEAK_NOTHING, nextState);
      break;

    case MAJOR_RECORD_MSG:
      bCanInterrupt = B_TRUE;
      currentProc = getpid();
      sprintf(temp, "%li", currentProc);
      sprintf(recordedFile, "%s/%s", RECORD_PATH, MSG_FILE);
      proc[0]=fork();
      if( proc[0] == 0 )
      {
        execlp( "/home/speech08/vannay/majordome/utils/searchSilence",
"searchSilence", "204", "80", recordedFile, temp, "80", 0);
        exit(1);
      }
      doRecord(MSG_FILE, 20, MAJOR_SEND_MAIL);
```

```
      break;

    case MAJOR_SEND_MAIL:
      bCanInterrupt = B_FALSE;
      waitpid(proc[0], &status, WNOHANG);
      sprintf( subject, "Call from %s", callState.remote_address()() );
      sprintf(header, "From: Majordome\n\
To:  %s\n\
Subject: %s\n\
Mime-Version: 1.0\n\
Content-Type: audio/basic\n\
Content-Description: msg.au\n\
Content-Transfer-Encoding: base64\n\
                                \n", recipient, subject);
      fprintf(mimeFile, "%s", header);
      fclose(mimeFile);
      sprintf (temp, "%s %s/%s %s %s/%s", "audioconvert -f
rate=8000,channels=1,encoding=alaw,format=sun,offset=0 -i
rate=8000,channels=1,encoding=alaw,format=raw,offset=0", RECORD_PATH,
MSG_FILE, " > ", RECORD_PATH, CONVERTED_FILE );
      system(temp);
      sprintf(temp, "mimencode %s/%s >> %s/%s", RECORD_PATH, CONVERTED_FILE,
RECORD_PATH, MIME_FILE);
      system(temp);
      sprintf(temp, "mail %s < %s/%s", recipient, RECORD_PATH, MIME_FILE);
      system(temp);
      if (tryNb < 3) {
      nextState = MAJOR_ASK_KEY2;
      tryNb = 0;
      }
      else {
      nextState = MAJOR_BYE;
      }
      doPlay (SPEAK_NOTHING, nextState);
      break;

    case MAJOR_ASK_KEY2:
      bCanInterrupt = B_TRUE;
      clearDtmf(DTMF_CHOICE_LENGTH);
      doPlay(SPEAK_ASK_KEY2, MAJOR_GET_KEY2);
      break;

    case MAJOR_GET_KEY2:
      bCanInterrupt = B_TRUE;
      tryNb++;
      doRecord(DTMF_FILE, 10, MAJOR_CHOICE_SWITCH2);
      break;

    case MAJOR_CHOICE_SWITCH2:
      bCanInterrupt = B_FALSE;
      if (tryNb < 3) {
      switch (dtmfCode[0]) {
      case '1':
        nextState = MAJOR_ASK_NAME;
        break;
      case '2':
        nextState = MAJOR_BYE;
        break;
      default:
        nextState = MAJOR_ASK_KEY2;
        break;
      }
      }
      else {
      nextState = MAJOR_BYE;
      }
      doPlay(SPEAK_NOTHING, nextState);
      break;

    case MAJOR_BYE:
```

```
    bCanInterrupt = B_FALSE;
    if (secrHours == B_TRUE) {
      doPlay(SPEAK_SECR_HOURS, MAJOR_END);
      }
      else {
      doPlay(SPEAK_BYE, MAJOR_END);
      }
      break;

    case MAJOR_END:
      sprintf(temp, "rm %s/%s %s/%s %s/%s %s/%s %s/%s",
            PROMPT_PATH, AUDIO_FILE,
            RECORD_PATH, MIME_FILE,
            RECORD_PATH, MSG_FILE,
            RECORD_PATH, CONVERTED_FILE,
            RECORD_PATH, DTMF_FILE);
      system(temp);
      disconnect_req();
      break;

    default:
      //bCanInterrupt = B_TRUE;
      cerr << "default " << endl;
      doPlay(SPEAK_NOTHING, MAJOR_ASK_NAME);
      break;
    }
} //end doAction()


void majorCall::afterPlay()
{
  doAction();
}

void majorCall::afterRecord()
{
  doAction();
}

void majorCall::recordDone()
{
  // derived to do another conversion type
  if (recorder)
    {

      close(msgFd);
      msgFd = -1;

      cleanRecordMachinery();
      afterRecord();
    }
}

//initialise le veteur contenant les numeros
//detectes et sa longueur
void majorCall::clearDtmf( int length )
{
  dtmfIndex = 0;
  dtmfLength = length;
  dtmfCode[0]=0;
  InDTMF = B_FALSE;
}


//met a jour le vecteur de numeros detectes
//et sa longueur
void majorCall::updatePin(char& c)
{
  if( dtmfIndex < dtmfLength )
    {
```

```
      dtmfCode[dtmfIndex++] = c;
      dtmfCode[dtmfIndex] = '\0';
    };

  if( dtmfIndex == dtmfLength )
    {
      switch (myState) {
      case MAJOR_GET_KEY1:
      myState = MAJOR_CHOICE_SWITCH1;
      break;
      case MAJOR_GET_KEY2:
      myState = MAJOR_CHOICE_SWITCH2;
      break;
      }
      endAction();
    }
}


//detecte la pression sur une touche du telephone
void majorCall::keyPressed(char& c)
{
  recordEndedBy = KEY_PRESSED;

  if ((c != '#') && ( c != '*')) {
    updatePin(c);
  }
  if (bCanInterrupt) {
    if (recorder) {
      if(( c == '#' ) || ( c == '*' )) {
      truncateMsgEnd();
      if (c == '#')
        myState = MAJOR_BYE;
      recordDone();
      }
    }
    else {
      dpDispatcher& d = dpDispatcher::instance();
      d.stopTimer((dpIOHandler*)player);
      if (c == '#')
      myState = MAJOR_BYE;
      player->timerExpired( 0, 0 );
    }
  }
}
```

9.2 Sources scripts du Majordome

9.2.1 reco.tcsh

```
#! /usr/local/bin/tcsh -f
source ~/.initStrut.tcsh
set base_dir = "/home/speech08/vannay/Strut/database/majordome/1.0"

cp ${base_dir}/reco/header ${base_dir}/alaw/name.tmp
cat ${base_dir}/alaw/name.au >> ${base_dir}/alaw/name.tmp

rasta majordome/1.0 setup= ${base_dir}/reco/rasta.stp

create-archive setup= ${base_dir}/reco/create-archive.stp >& ${base_dir}/errfile

/sym/strut/src/strut-1.04g/qn-forward $DATABASE setup= ${base_dir}/reco/qn-forward.stp \
| isolated $DATABASE setup= ${base_dir}/reco/isolated.stp \
| line | sed -e "s/[ a-z]//g" -e "s/[0-9]//g" > ${base_dir}/reco/reco.name

kill -USR1 ${1}
```

```
rm ${base_dir}/alaw/name.au ${base_dir}/alaw/name.tmp \
  ${base_dir}/rasta/name.rasta ${base_dir}/errfile
```

9.2.2 intiStrut.stp

```
#! /usr/local/bin/tcsh -f
setenv STRUT_DIR /home/speech08/vannay/Strut
setenv STRUT_DATA_DIR $STRUT_DIR/data
setenv DATABASE_ID majordome
setenv DATABASE_VERSION 1.0
setenv DATABASE $DATABASE_ID/$DATABASE_VERSION
```

9.2.3 rasta.stp

```
command= $(STRUT_DIR)/database/$(DATABASE)/reco/rasta.cmd
frame-length= 30
frame-shift= 10
sample-rate= 8000
coefficients-count= 12
log-rasta= yes
mix-coeff= 1.0
energy= yes
```

9.2.4 rasta.cmd

```
/home/speech08/vannay/Strut/database/majordome/1.0/alaw/name.tmp
/home/speech08/vannay/Strut/database/majordome/1.0/rasta/name.rasta
```

9.2.5 create-archive.stp

```
archive= $(STRUT_DIR)/database/$(DATABASE)/archive/test.rasta
phonemes=$(STRUT_DATA_DIR)/phonemes/phonemes.3state
dir= $(STRUT_DIR)/database/$(DATABASE)/rasta
extension=.rasta
recursive=no
lost-begin=1
lost-end=1
```

9.2.6 isolated.stp

```
phonemes=$(STRUT_DATA_DIR)/phonemes/phonemes.3state
dictionary=$(STRUT_DATA_DIR)/$(DATABASE)/dictionary
ls= LS
ts= TS
probs= -
output= -
mask=11111111
```

9.2.7 qn-forward.stp

```
input=$(STRUT_DIR)/database/$(DATABASE)/archive/test.rasta
output= -
weights=$(STRUT_DATA_DIR)/$(DATABASE)/models/sentences.234_600_36.weight
divide-by-priors=yes
```

9.3 Sources de searchSilence

```
/*******************************************/
/* ATTENTION, DEFINE EITHER ALAW OR LINEAR */
/*******************************************/
#define ALAW
```

```
#include <iostream.h>
#include <stdlib.h>
#include <fstream.h>
#include <math.h>
#include <signal.h>
#include <unistd.h>
#define SIL_DETECT          "/home/speech08/vannay/majordome/utils/searchSilenceAlaw"

#ifdef ALAW
int alaw2linear(unsigned char a_val);
#endif

#define SEUIL_DEBUT 100

#define MAX_LEN 1000

#ifndef Boolean
#define Boolean short
#endif
#ifndef FALSE
#define FALSE 0
#endif
#ifndef TRUE
#define TRUE 1
#endif

Boolean bEnd;
Boolean bRestart;
char *szFileName;

#ifdef LINEAR
typedef short int SampleType;
#endif
#ifdef ALAW
typedef unsigned char SampleType;
#endif

class silenceFrame {
        private:
                float value;
                silenceFrame* before;
                virtual void addAtEnd(float valeur);
                virtual void clean(float valeur);

        public:
                silenceFrame(float my_value);
             virtual ~silenceFrame();
                virtual int add(float valeur, float seuil, int min_size);
                virtual int count();
                virtual int insilence(float valeur, float seuil);

    };


silenceFrame::silenceFrame( float my_value )
  {
  value = my_value;
  before = NULL;
  };
```

```
silenceFrame::~silenceFrame()
  {
  if (before != NULL)
            delete before;
  }

void silenceFrame::addAtEnd(float valeur)
  {
  silenceFrame* place;
  silenceFrame* ptr;
  place = this;
  while (place->before != NULL)
            place = place->before;
  ptr = new silenceFrame(valeur);
  place->before = ptr;
  }

int silenceFrame::count()
  {
  int n;
  silenceFrame* place;
  place = this;
  n = 0;
  while (place != NULL)
            {
            place = place->before;
            n++;
            }
  return n;
  }

void silenceFrame::clean(float LogEnergy)
  {
  silenceFrame* place;
  silenceFrame* new_place;
  place = this;
  new_place = this;
  while (place != NULL)
            {

            if (!((place->value > 0.8*LogEnergy) && (place->value < 1.2*LogEnergy)))
              new_place=place->before;

       place = place->before;
            }

  if (new_place != NULL)
            {
            if (new_place != this)
                        {
                        this->value = new_place->value;
                        place = this->before;
                        this->before = new_place->before;
                        new_place->before = NULL;
                        delete place;
                        }
            addAtEnd(LogEnergy);
            }
```

```
  else
            {
            this->value = LogEnergy;
            this->before = NULL;
            }
  }


int silenceFrame::insilence(float LogEnergy, float seuil)
  {
  if (LogEnergy < seuil)
            return 1;
  else
            return 0;
  }

int silenceFrame::add(float valeur, float seuil ,int min_size)
  {
    if (insilence(valeur, seuil))
            {
            clean(valeur);
            }
    else
            {
            this->value = valeur;
            this->before = NULL;
            }
  return (count() > min_size);
  }


void SigTerm(int)
/* pour terminer "proprement" le programme */
    {
    bEnd = TRUE;
    }



void SigHup(int)
/* pour redemarrer "proprement" le programme */
    {

    bRestart = TRUE;
    signal( SIGHUP, SigHup );
    fflush(stdout);

    }

int main( int argc, char **argv )
    {

    ifstream fileIn;
    SampleType *buffer;
    SampleType *buffPtr;
    int SampleTypeSize;
    int filePos;
    unsigned int nRead;
    unsigned int nReadTot;
```

```
    unsigned int numSamples;
    unsigned int numOverlap;
    double energy;
    double logEnergy;
    double logEnergyMax;
    int readOK;
    int firstTime;
    int frameEnergyMax;
    int lastFrame;
    silenceFrame* Silence;
    int resultat_silence;
    int resultat_silence_precedent;
    int longueur_actuelle;
    int taille_silence;
    int pid_number;
    float seuil_silence;
    Boolean bParoleOK;
    long nbRead;

    frameEnergyMax = 0;
    lastFrame = 0;
    SampleTypeSize = sizeof( SampleType );
    Silence = NULL;

//Kill the process after 2 minutes
int rc;
pid_t pidParent;
pidParent = getpid();
rc = fork();
if (rc == 0) {
  sleep(15);
  char temp[256];
  sprintf (temp, "kill -9 %i", pidParent);
  system(temp);
  exit(1);
}

    /* installation de la procedure de fin et de restart */
    signal( SIGTERM, SigTerm );
    signal( SIGHUP, SigHup );

    switch( argc )
        {
    case 6:
            break;
    default:
            cerr << "Usage: " << argv[0] << " numSamples numOverlap file PID lengthSil" << endl;
            return 1;
            }

    numSamples = atoi( argv[1] );
    buffer = new SampleType[ numSamples ];
    numOverlap = atoi( argv[2] );
    pid_number = atoi( argv[4] );

    /* definition dimension silence    */
    taille_silence = atoi( argv[5] );
    seuil_silence = 15;
```

```
while( !bEnd )
   {
   logEnergyMax = 0;
        buffPtr = buffer;
   firstTime = 1;
        resultat_silence_precedent = -1;

        bParoleOK = FALSE;
        nbRead = 0;

        for( unsigned int i = 0; i < numSamples; i++ )
          buffer[i] = 0;

        bRestart = FALSE;
        fileIn.open( argv[ 3 ] );
        filePos = 0;
        while ( !bEnd && !bRestart )
                  {
              if( firstTime )
                     {
                     // first time => read a full window, not only the difference
                     fileIn.read( (char *)buffPtr, numSamples * SampleTypeSize );
                     readOK = !fileIn.bad() && !fileIn.eof();
                     nRead = fileIn.gcount();
                     if (readOK)
                       firstTime = 0;
                     buffPtr += nRead / SampleTypeSize;
                     nReadTot = nRead;
                  }
                  else if( buffPtr - buffer + numOverlap <= numSamples )
                     {
                    fileIn.read( (char *)buffPtr, numOverlap * SampleTypeSize );
                     readOK = !fileIn.fail();
                     nRead = fileIn.gcount();
                     buffPtr += nRead / SampleTypeSize;
                     nReadTot = nRead;
                     }
                  else
                     {
                    fileIn.read( (char *)buffPtr, ( buffer - buffPtr + numSamples ) * SampleTypeSize );
                     readOK = !fileIn.fail();
                     nRead = fileIn.gcount();
                     nReadTot = nRead;
                     buffPtr = buffer;
                if( numOverlap * SampleTypeSize > nRead )
                              {

                              fileIn.read( (char *)buffPtr, numOverlap * SampleTypeSize - nRead );
                              if( !fileIn.fail() )
                                 {
                                 nRead = fileIn.gcount();
                                 buffPtr += nRead / SampleTypeSize;
                                 nReadTot += nRead;
                                 }
                              }
                  }
                  if( buffPtr == buffer + numSamples )
                     {
                    buffPtr = buffer;
```

```
                        }
                    energy =
          if( readOK )
                        {
nbRead++;
if( nbRead > SEUIL_DEBUT )
   bParoleOK = TRUE;

                        filePos = fileIn.tellg();

                        lastFrame++;

              for( unsigned int i = 0; i < numSamples; i++ )
                    {
#ifdef ALAW
                    energy += alaw2linear( buffer[i] ) * alaw2linear( buffer[i] );
#endif
#ifdef LINEAR
                    energy += buffer[i] * buffer[i];
#endif
                    }

              logEnergy = log( energy );
              if( logEnergy > logEnergyMax )
                    {

                    logEnergyMax = logEnergy;
                    frameEnergyMax = lastFrame;

                    }

      if (Silence == NULL)
                    Silence = new silenceFrame(logEnergy);
          else
                    {
                    resultat_silence = Silence->add(logEnergy,seuil_silence,taille_silence);
                    longueur_actuelle = Silence->count();

                    if (resultat_silence != resultat_silence_precedent)
                      {
                       if (resultat_silence == 1)
                         {
if( bParoleOK )
   {
printf( "===> " );
kill(pid_number,SIGUSR1);
return 0;
   }

cerr << longueur_actuelle << ": silence\n";
                         }
                       else
                         {
                         cerr << longueur_actuelle << ": parole\n";
                         }
                      }
                    resultat_silence_precedent = resultat_silence;
                    }
             }
```

```
            else
              {
              sleep( 1 );
              fileIn.close();
              fileIn.open( argv[ 3 ] );
              fileIn.seekg(filePos);
              }
            /* printf("variables: %i %i %i %i %i\n ",fileIn.bad(),fileIn.eof(),readOK,nRead,filePos);
fflush(stdout);*/
            }
   cout << "Log Energy max : " << logEnergyMax << endl;
   cout << "on frame       : " << frameEnergyMax << endl;
   cout << "total frames   : " << lastFrame << endl;
   fileIn.close();
            }

return 0;

   }
```

9.4 Exemple de dictionnaire

```
TS sil
LS sil sil sil
TS sil sil sil
HERVE ai rr vv ei
MESSAGE mm ai ss aa jj
CEDRIC_JABOULET ss ai dd rr ii kk jj aa bb ou ll ei
CEDRIC_JABOULET ss ai dd rr ii kk
CEDRIC_JABOULET jj aa bb ou ll ei
CEDRIC_JABOULET jj aa bb ou ll ei ss ai dd rr ii kk
DOMINIQUE_GENOUD dd au mm ii nn ii kk jj ee nn ou
DOMINIQUE_GENOUD dd au mm ii nn ii kk
DOMINIQUE_GENOUD jj ee nn ou
DOMINIQUE_GENOUD jj ee nn ou dd au mm ii nn ii kk
EDDY_MAYORAZ ai dd ii mm aa mm aa yy au rr aa
EDDY_MAYORAZ ai dd ii
EDDY_MAYORAZ mm aa yy au rr aa
EDDY_MAYORAZ mm aa yy au rr aa ai dd ii
GEORG_THIMM gg ei oo rr gg tt ii mm
GEORG_THIMM gg ei oo rr gg
GEORG_THIMM tt ii mm
GEORG_THIMM tt ii mm gg ei oo rr gg
GILBERT_MAITRE jj ii ll bb ai rr mm ai tt rr
GILBERT_MAITRE jj ii ll bb ai rr
GILBERT_MAITRE mm ai tt rr
GILBERT_MAITRE mm ai tt rr jj ii ll bb ai rr
GILLES_CALOZ jj ii ll kk aa ll au
GILLES_CALOZ jj ii ll
GILLES_CALOZ kk aa ll au
GILLES_CALOZ kk aa ll au jj ii ll
HERVE_BOURLARD ai rr vv ei bb ou rr ll aa rr
HERVE_BOURLARD bb ou rr ll aa rr
HERVE_BOURLARD bb ou rr ll aa rr ai rr vv ei
HERVE_GLOTIN ai rr vv ei gg ll oo tt in
HERVE_GLOTIN gg ll oo tt in
HERVE_GLOTIN gg ll oo tt in ai rr vv ei
JEAN-LUC_COCHARD jj an ll uu kk kk au ch aa rr
JEAN-LUC_COCHARD jj an ll uu kk
JEAN-LUC_COCHARD kk au ch aa rr
```

JEAN-LUC_COCHARD kk au ch aa rr jj an ll uu kk

9.5 Cahier des charges