Étude et développement d'un serveur de message à réponse vocale

Paul Tripodi

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

CENTRE AGREE

DE GRENOBLE (C.U.E.F.A.)

MEMOIRE

Présenté en vue d'obtenir

LE DIPLOME D'INGENIEUR C.N.A.M.

en

ELECTRONIQUE

par

Paul TRIPODI

ETUDE ET DEVELOPPEMENT D'UN SERVEUR DE MESSAGE A REPONSE VOCALE

Les travaux relatifs au présent mémoire ont été effectués à l'Institut de la Communication Parlée de Grenoble (I.C.P.) sous la direction de Mr Jean Marc Dolmazon et de Mr Jean Claude Caërou.

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

CENTRE AGREE

DE GRENOBLE (C.U.E.F.A.)

MEMOIRE

Présenté en vue d'obtenir

LE DIPLOME D'INGENIEUR C.N.A.M.

en

ELECTRONIQUE

par

Paul TRIPODI

ETUDE ET DEVELOPPEMENT D'UN SERVEUR DE MESSAGE A REPONSE VOCALE

Les travaux relatifs au présent mémoire ont été effectués à l'Institut de la Communication Parlée de Grenoble (I.C.P.) sous la direction de Mr Jean Marc Dolmazon et de Mr Jean Claude Caërou.

Remerciements :

Le travail présenté dans ce mémoire a été réalisé à l'Institut de la Communication Parlée de Grenoble (I.C.P.). Cet institut (unité associée au C.N.R.S.) dépend de deux universités : l'Institut National Polytechnique de Grenoble (par l'E.N.S.E.R.G.) et l'université STENDHAL.

Je remercie vivement Mr Jean Marc Dolmazon directeur de l'institut de la communication parlée, de m'avoir accueilli à l'I.C.P. J'ai beaucoup apprécié l'ambiance chaleureuse et amicale qu'il sait créer dans le laboratoire.

Je remercie Mr Moreau, professeur responsable du C.N.A.M. à Paris, de présider le jury de ce mémoire.

Je remercie également Mr Baribaud, directeur de l'E.N.S.E.R.G. et responsable du cycle C C.N.A.M. dans la spécialité électronique, de m'avoir aidé dans la recherche d'un laboratoire et de participer au jury.

Je tiens à remercier tout particulièrement Mr Jean Claude Caërou, maître de conférence à l'I.N.P.G., qui a supervisé mes travaux depuis mon arrivée au laboratoire. Il a su par son enthousiasme et son efficacité m'aider dans la conduite de mon projet et la rédaction de mon mémoire.

Mr Gérard Bailly, responsable de la synthèse à l'I.C.P. m'a toujours fait bénéficier de ses connaissances en synthèse de la parole et en traitement du signal, je lui exprime ici ma profonde gratitude.

Je remercie Mr Vincent Corcoles, ingénieur dans l'entreprise Merlin Gérin, d'avoir accepté de juger mon travail et de participer au jury.

Enfin je remercie toutes les personnes travaillant à l'I.C.P. pour la sympathie et l'accueil chaleureux que j'ai trouvé en arrivant au laboratoire. Je n'oublierai pas l'esprit de coopération et de bienveillance qui caractérise l'Institut de la Communication Parlée.

Je remercie Dominique, mon épouse, pour l'aide précieuse qu'elle m'a donnée et la patience qu'elle a eue pendant toutes ces années de cours du soir.

SOMMAIRE

1. Introduction.

La synthèse de la parole est de plus en plus présente dans la vie quotidienne, nous la trouvons notamment dans les annonces de message, les traducteurs, les automobiles et les jeux. Elle est le résultat des travaux de recherche entrepris dans plusieurs disciplines (acoustique, anatomique, physiologique, etc.). Sa mise en oeuvre nécessite la connaissance des mécanismes de production, et de perception. D'autre part, les progrès technologiques de l'électronique et de l'informatique permettent l'adaptation de la synthèse sur de petits systèmes.

Le développement de la synthèse a longtemps été guidé par la nécessité de réduire le débit de la transmission du signal de parole. Elle est aujourd'hui motivée par un besoin plus vaste : celui du dialogue homme-machine.

Le serveur de message, que nous avons réalisé, utilise les travaux effectués à l'I.C.P. par l'équipe synthèse. Parmi ces travaux, nous pouvons signaler la validation des connaissances acquises dans le domaine de la parole, et la création d'outils d'analyse et de synthèse. Le principal élément, que nous utilisons, est le logiciel orienté objet COMPOST. Il permet un traitement de la parole depuis le texte écrit, jusqu'à sa sortie vocale. La structure des programmes COMPOST (scénario) est composée de règles et d'appels de fonction. La synthèse vocale générée par ces programmes commande un synthétiseur à formants.

Les applications de notre serveur de message à réponse vocale se situent dans le cadre d'un dialogue homme-machine. Le but principal est la formation de messages parlés, en fonction d'informations contenues dans des bases de données. Les messages parlés sont plus directs et plus naturels que la lecture d'informations sur un écran. Pour donner une grande souplesse à notre système, nous avons choisi de rendre la composition des messages directement accessible aux utilisateurs. Ainsi, les requêtes et les définitions de formats sont passées par l'intermédiaire de commandes écrites.

Une présentation du projet est proposée dans le chapitre 2. Elle définie le contexte et les méthodes qui ont régit le développement du produit. Nous rappelons, au chapitre 3, quelques notions de synthèse vocale ; elles sont indispensables pour la compréhension des autres parties de ce mémoire.
La conversion texte-parole est présentée au chapitre 4. Nous détaillons le concept de COMPOST à travers les notions d'objets, de classes, de règles, de grammaires, de scénario et d'arborescence. Pour clore ce chapitre nous décrivons le fonctionnement du synthétiseur à formants.

Le chapitre 5 traite des bases de données. Les enregistrements sont composés de champs, dont la forme du contenu n'est pas obligatoirement compatible avec le logiciel COMPOST. Il faut donc prévoir des syntaxes d'écriture et des outils de conversion pour la mise en oeuvre des messages. Cette partie importante du serveur est traitée dans le chapitre 6. Elle montre le choix que nous avons fait pour la définition des formats de message. Les liens à établir, entre la gestion du serveur et le traitement de la parole sont le fil conducteur de cette démarche. Les syntaxes d'écriture et de conversion des champs sont expliquées par des exemples concrets.
L'intégration d'un interface utilisateur procure un confort d'utilisation appréciable. Cette partie est décrite au chapitre 7. Elle introduit de façon formelle les notions de commandes et d'indépendance des modules. La personnalisation de l'interface est entièrement configurable par un fichier texte.

L'organisation complète du serveur est étudiée dans le chapitre 8. Elle présente l'architecture, la description des commandes, ainsi qu'une évaluation des performances du traitement de la parole. Elle amène une conclusion sur les orientations possibles de notre serveur de message.

2. Présentation du projet

2.1. Serveur de message à réponse vocale.

Le but du serveur à synthèse vocale est de fournir des messages parlés, en fonction d'informations contenues dans des bases de données. Les requêtes (ou commandes) sont passées sous une forme écrite à l'aide du clavier. Parmi les applications possibles citons :

- Un agenda à réponse vocale donnant l'emploi du temps selon les sélections de l'utilisateur.
- La lecture de fichiers de personnes donnant les coordonnées, les adresses, et les numéros de téléphone.
- La lecture de fichiers pour gestion de stock de composants.
- Etc.

Ce serveur est destiné à promouvoir les travaux de l'équipe synthèse, en ce qui concerne le logiciel de traitement de la parole COMPOST, ainsi que les recherches sur la synthèse à formants par règles du français.

Un soin tout particulier est apporté à la convivialité du serveur, par une interface utilisateur à menus. Les utilisations très diverses, que l'on peut faire du serveur de messages, ont conduit à une conception très ouverte et modifiable. Cette flexibilité peut s'exercer à plusieurs niveaux :

- Sur la définition des formats de messages, par des commandes.

- A travers le traitement de la parole, par l'écriture de règles dans le langage du logiciel orienté objet COMPOST.
- Par l'écriture de fonctions complexes en langage C.

L'architecture interne du produit est modulaire et tient compte des impératifs liés à la structure du logiciel COMPOST (appels des fonctions, passage des paramètres, etc.)

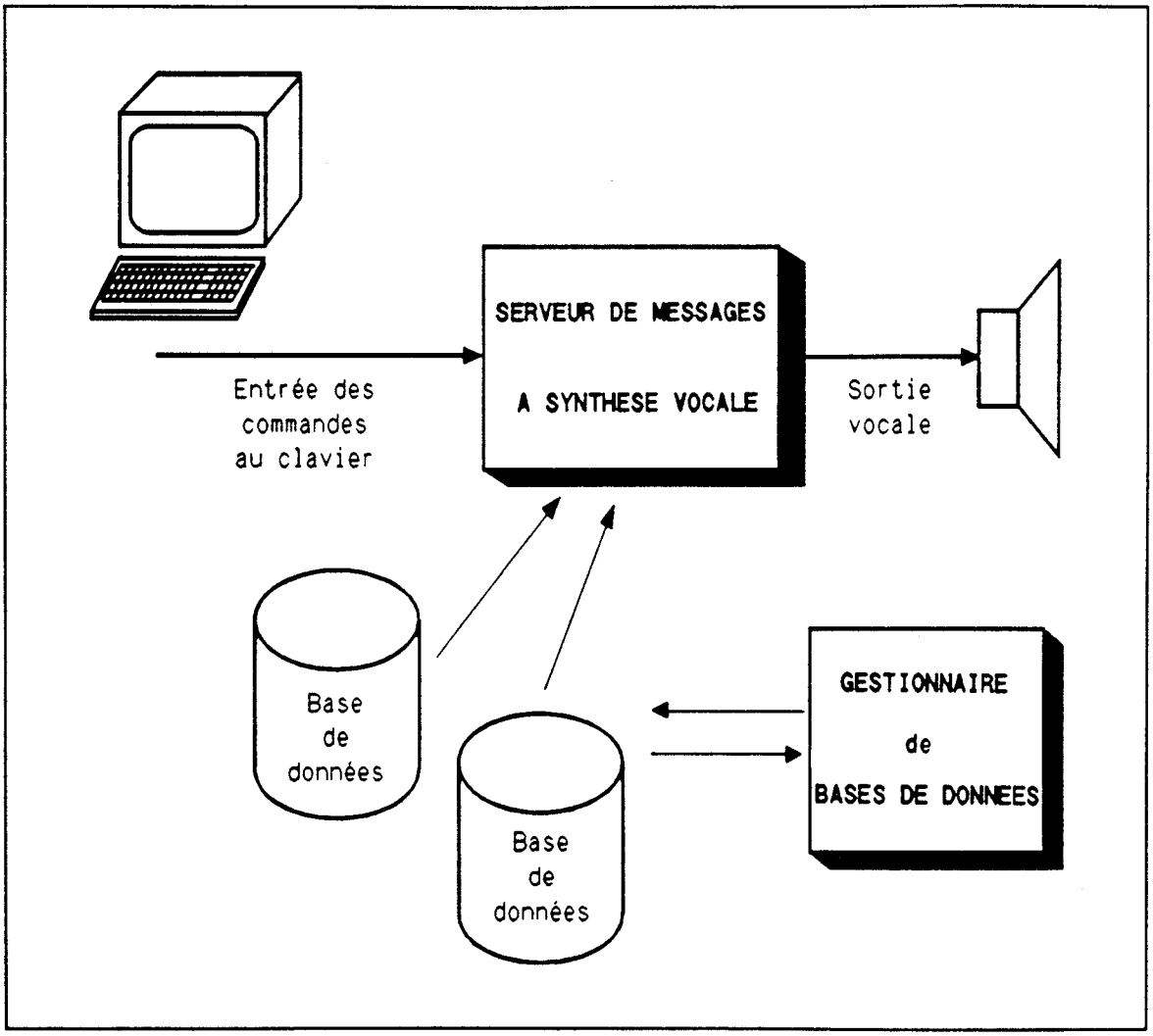


Figure 2.1. *Organisation générale du serveur de message.*

La figure 2.1 montre l'organisation générale de notre serveur avec l'entrée représentée par un clavier. Un écran assure l'interface visuel avec l'utilisateur. Le serveur commande une sortie vocale. Les bases de données sont gérées de façon indépendante, mais doivent être lues par le serveur. La seule contrainte, sur le choix du gestionnaire de base de donnée, est le format des fichiers qui doit être compatible avec le format

standard "DBase III" (.dbf). Dans le cadre de notre projet, c'est un "petit" système de gestion du commerce que nous avons utilisé.

2.2. Contexte matériel.

Ce serveur fonctionne sur un ordinateur individuel (I.B.M. ou compatible de type AT) muni de 640 koctets de mémoire vive et d'un disque dur.

La sortie analogique est obtenue par une carte de traitement de signal et d'interface analogique "OROS AU21". Cette carte est connectée avec le bus interne du micro-ordinateur. Elle est adressée dans l'espace des entrées sorties de l'ordinateur (typiquement 784_{10} ou 920_{10}). L'architecture de cette carte est donnée en ANNEXE 1. Nous retrouvons pour la partie logique : le processeur de signal et ses deux blocs de mémoire. Le bus du PC commande des registres à travers un interface. La partie analogique est composée de deux modules :

- Un module d'entrée pour l'acquisition de signaux, avec amplification, filtrage et échantillonnage (il n'est pas utilisé pour notre application).
- Un module de sortie qui comprend : un convertisseur numérique-analogique, un filtrage et un amplificateur de sortie.

Le signal de sortie est envoyé à un haut-parleur. Comme l'échantillonnage de notre synthétiseur se fait à une fréquence de 10kHz, le filtre passe-bas interne à la carte (de fréquence de coupure à 20kHz) est inadapté à notre application. Il faut donc ajouter, dans le logiciel de traitement du signal, un filtre interpolateur afin de porter la fréquence d'échantillonnage du signal de sortie à 40 kHz.

Pour des applications particulières cette sortie pourra être connectée au réseau commuté téléphonique ou sur un réseau d'interphones.

2.3. Méthodes d'analyses des fonctions.

Les fonctions nécessaires au développement de notre projet sont écrites en langage C. Elle sont compilées et liées dans un module appelé par le logiciel hôte COMPOST. La conception modulaire des fonctions autorise un assemblage semblable à celui pratiqué avec des composants électroniques. L'analyse de l'ensemble du projet a

suivi une démarche descendante, depuis le projet global jusqu'au codage, en partant d'une conception générale vers une conception détaillée. Le test et l'assemblage des fonctions sont pratiqués en suivant une démarche inverse, notamment le test unitaire de chaque fonction. Dans la suite du mémoire nous représentons, lorsque cela est nécessaire à la compréhension, les fonctions sous une forme modulaire, et le séquencement par une écriture algorithmique.

2.3.1. Approche modulaire.

La méthode, que nous avons choisie pour l'étude et le développement du serveur de message, suit une approche modulaire du type S.A.D.T. (Structured Analysis and Design Technique). Cette démarche n'est par limitée à l'électronique ou à l'informatique, mais peut s'appliquer à toutes sortes d'analyses. Le principe, de cette méthode, est basé sur une analyse descendante, en décomposant des blocs complexes en blocs de plus en plus petits. La représentation imposée d'un bloc est donnée en figure 2.2, elle privilégie l'imbrication des fonctions et les flux de données.

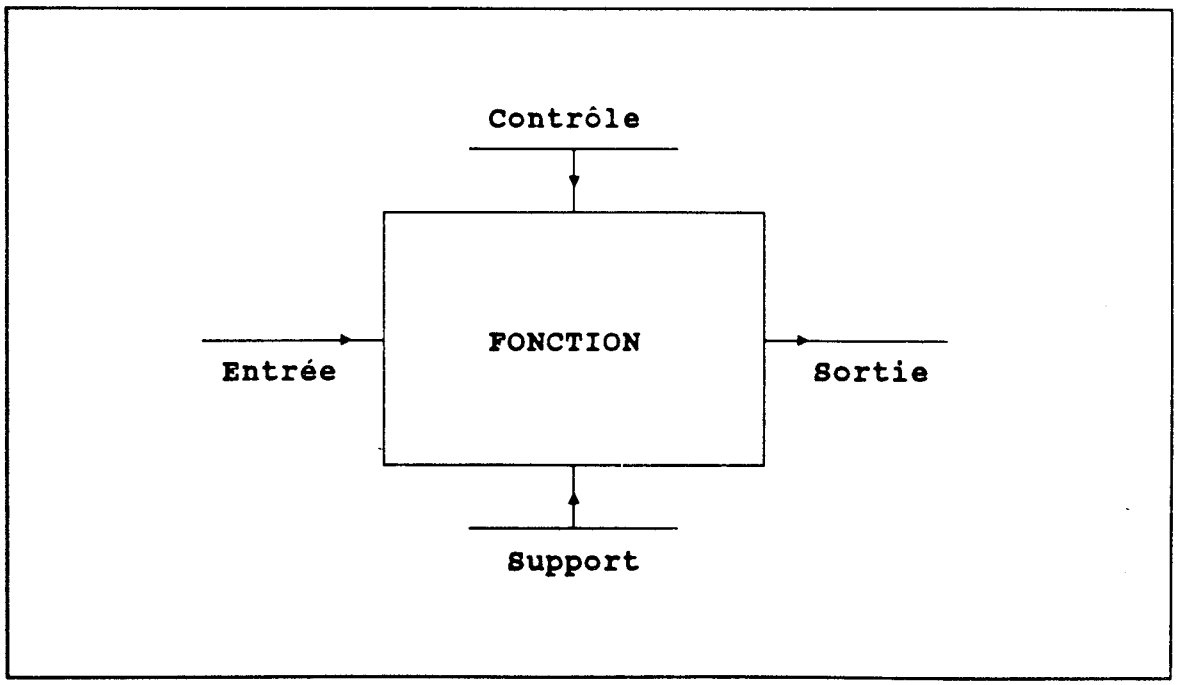


Figure 2.2. *Représentation modulaire S.A.D.T.*

2.3.2. Représentation algorithmique.

Le complément indispensable à une analyse structurelle par bloc est l'ordonnancement temporel des opérations. Nous utilisons, pour cette représentation algorithmique des mots-clés, que nous retrouvons dans, la pluparts des langages (Si, Tant que ...). Les algorithmes, que nous décrivons dans ce mémoire, montrent l'architecture générale et le fonctionnement du serveur de message.

2.4. Développement.

Afin de faciliter la recherche et la lecture, lors de la maintenance, les fonctions du serveur (hors COMPOST) sont regroupées dans plusieurs fichiers. Aux fichiers de programme, deux fichiers d'entête, sont joints :

- Le fichier pour le prototypage des fonctions,
- Le fichier pour les variables globales partagées par un grand nombre de fonctions.

Nous donnons ci-dessous la liste et le contenu de ces fichiers.

Fichiers de programme (fonctions C) :

- base.c , fonctions de gestion des appels de COMPOST,
- message.c , fonctions générant le message au format COMPOST,
- lect_dbf.c , fonctions de lecture des bases de données,
- lect_ent.c , fonctions de lecture des fichiers d'entrée (d'initialisation et de commandes),
- fonc_com.c , fonctions de gestion des commandes,
- chaines.c , fonctions de traitement des chaînes de caractères (conversions),
- formats.c , fonctions de traitement des formats paramétrables,
- ecran_u.c , fonctions de l'interface utilisateur,
- agenda.c , fonctions pour le format à caractère fixe (agenda).

Fichiers d'entête :

- base.h , définition des constantes (MACROS), définition des nouveaux types de données (structures), et prototypage des fonctions,
- var_base.h , variables globales (externes) partagées par plusieurs fonctions.

3. Eléments de synthèse vocale.

3.1. Signal vocal.

Les sons de la parole sont générés par les vibrations des cordes vocales (à une fréquence fondamentale F0) pour les sons dit voisés, et par l'écoulement rapide de l'air au niveau d'une constriction du conduit vocal pour les sons non voisés. Dans tous les cas, le spectre de l'onde produit par la source d'excitation est modifié par les résonances des cavités pharyngienne, buccale et nasale, puis par le rayonnement aux lèvres et aux narines (≈ +6 db/octave). Une modélisation de l'appareil phonatoire est représentée sur la figure 3.1, elle met en évidence la succession des principales cavités et leur couplage.

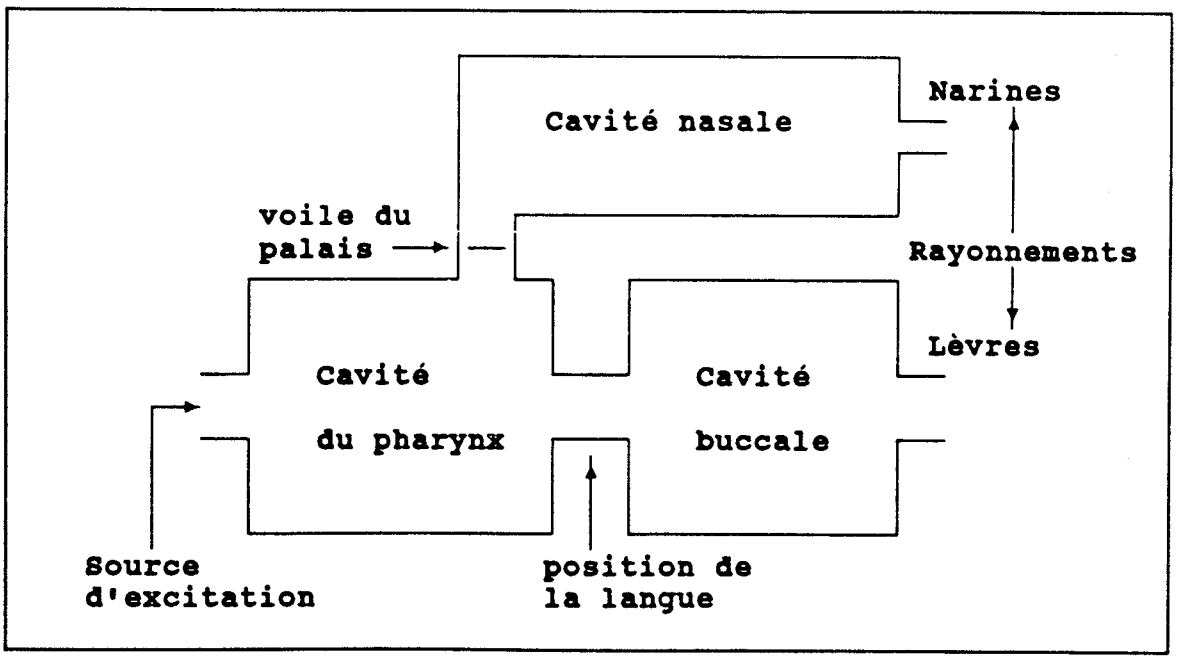


Figure 3.1. *Modélisation de l'appareil phonatoire.*

Les variations de géométrie, de volume et de couplage entre cavités, modifient les fréquences de résonance du conduit vocal. Ainsi, pour les sons voisés, le spectre du signal de parole issu de l'appareil phonatoire, présente des maximums qui sont appelés formants (et des minimums antiformants) [12].

Le plus petit élément sonore que l'on puisse identifier est le phonème. Il permet d'élaborer et de reconnaître des mots. Chaque langue a une quantité de phonèmes bien définie, avec des possibilités d'association plus ou moins étendues. Pour le français, les phonèmes, voyelles et consonnes, sont au nombre de 37, leur liste est représentée sur la figure 3.2.

Consonnes :

[p] paie	[t] taie	[k] quai
[b] baie	[d] dais	[g] gai
[m] mais	[n] nez	[ɲ] gagner
[f] fait	[s] sait	[ʃ] chez
[v] vais	[z] zéro	[ʒ] geai
[w] ouais	[ɥ] huer	[j] yéyé
	[l] lait	[R] raie

Voyelles :

[i] lit	[y] lu	[u] loup
[e] les	[ø] leu	[o] lot
[ɛ] lait	[œ] leur	[ɔ] lotte
[a] là	[ə] le	
[ɛ̃] lin	[ã] lent	[õ] long

Figure 3.2. *Les phonèmes du français.*

Par une approche acoustique du signal vocal nous observons que le message ne peut pas être considéré comme une simple juxtaposition de phonèmes isolés. Une grande partie de l'information se trouve dans la liaison des phonèmes, pour cette raison le groupement de deux phonèmes, nommé "diphone", est utilisé comme méthodes de synthèse.

3.2. Réduction du débit et de la taille mémoire.

La quantité d'information contenue dans le signal de parole est très élevée. Les variations du spectre sont plus lentes et montrent une redondance du signal. Les premiers synthétiseurs ont été étudiés pour réduire la quantité d'information à transmettre en reconstituant le signal de parole. L'échantillonnage d'un signal de parole, qui est de l'ordre de 10 kHz, peut être réduit à ≈ 100Hz (période de renouvellement de 5 à 20ms) lorsque les échantillons représentent des éléments du spectre (ou des éléments acoustiques). Le nombre de ces éléments est de 10 à 20, nous obtenons alors un gain global de 10. Aujourd'hui, les systèmes de synthèse ne sont plus limités à la transformation d'éléments acoustiques en signal de parole, mais acceptent comme entrée des éléments phonétiques et prosodiques.

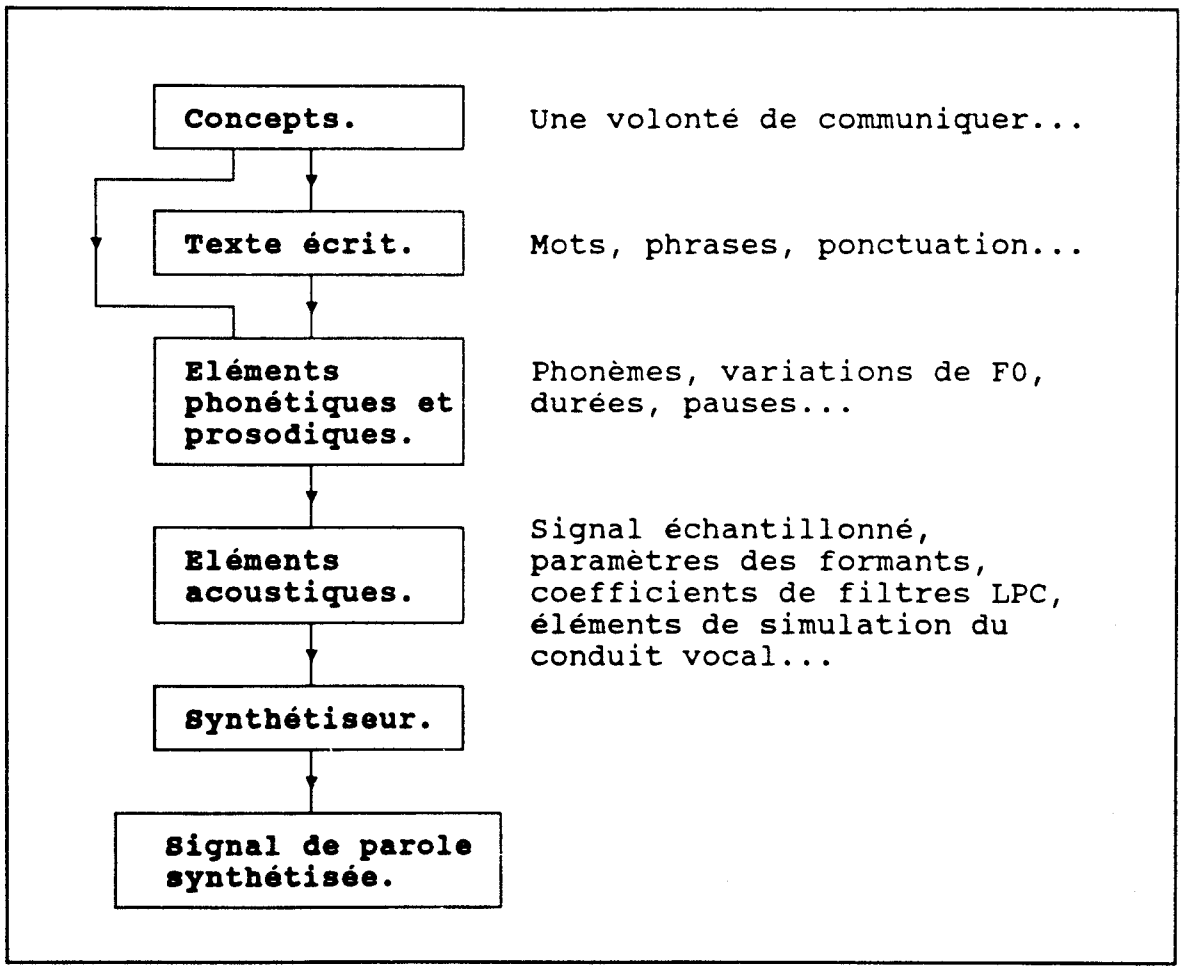


Figure 3.3. *Etapes de la synthèse vocale.*

Ces méthodes de synthèse qui utilisent des niveaux linguistiques, réduisent encore la quantité d'information à stocker ou à transmettre. Sur la figure 3.3 nous voyons les niveaux de transformation d'un message parlé, depuis le concept jusqu'au signal de parole. Le processus

de synthèse peut commencer quelque soit le niveau, cependant, plus la synthèse est proche du signal de sortie, et plus le débit d'information à fournir est grand. Si le débit et le stockage diminuent, le traitement local et l'élaboration du signal deviennent plus complexes. Dans la plupart des cas la parole est reconstituée soit, par concaténation d'éléments stockés, souvent sous forme de diphone pour conserver les transitions des éléments acoustiques entre les phonèmes, soit par des règles qui interviennent dans toutes les étapes de la synthèse. Sur la figure 3.4 sont représentés des débits indicatifs de chaque niveau de synthèse, en élément/s et en bits/s.

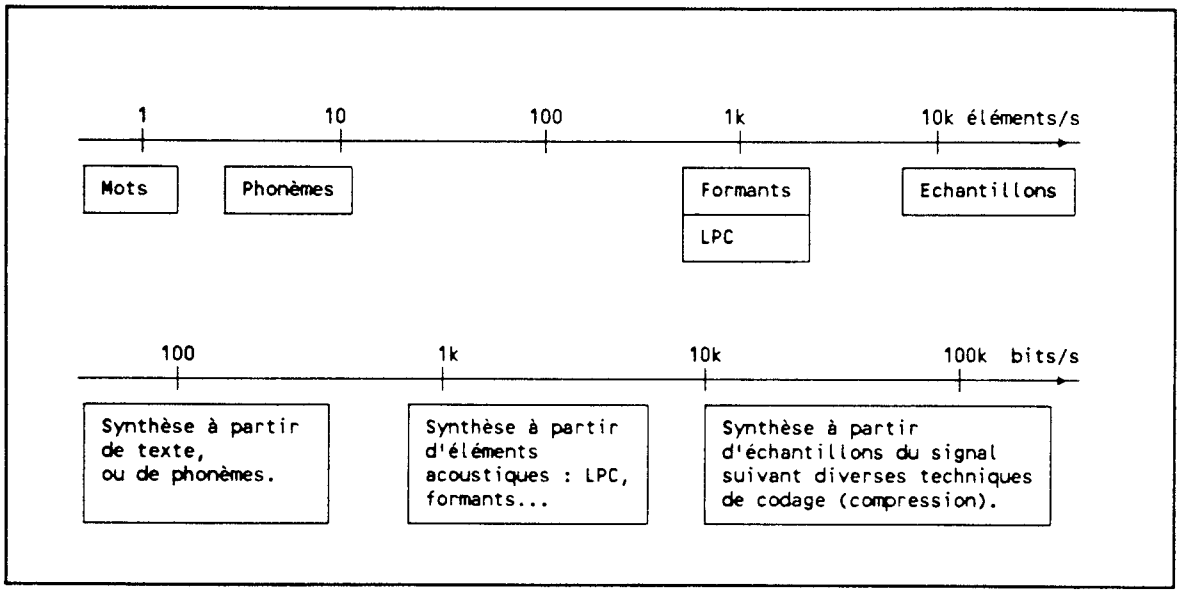


Figure 3.4. *Débits de l'information du signal de parole.*

3.3. Différents types de synthétiseurs.

Dans cette partie nous allons décrire les techniques de synthèse les plus connues, sans chercher à introduire de classification trop rigide entre les différents types.

3.3.1. Synthétiseurs à canaux.

Le but des synthétiseurs à canaux (ou vocodeurs) est de transmettre un signal de parole, avec une bande passante plus réduite. La partie analyse est constituée d'un banc de filtres, d'un détecteur de fréquence fondamentale, et d'un discriminateur pour le dosage voisé/non-voisé du signal de parole. Chaque voie du banc de filtre comprend un filtre passe-bande, suivi d'un redresseur et d'un filtre passe-bas, de façon à transmettre l'énergie du signal de chaque bande. Le nombre de canaux est de l'ordre de 15, la largeur des bandes

passantes peut être fixe ou variable, avec des pas plus petits pour les basses fréquences, et des pas plus grands pour les hautes fréquences ; le rafraîchissement des valeurs se fait à une période de 10 à 20 ms. La partie synthèse du récepteur fait l'opération inverse, les paramètres de F0 et de la décision voisé/non-voisé commandent des générateurs périodique (voisé) et aléatoire (non-voisé) [12] [40]. Ces générateurs alimentent un banc de modulateurs et de filtres qui reçoivent les amplitudes des énergies de chaque bande. La somme des signaux de ce banc constitue le signal de parole synthétisée (figure 3.5).

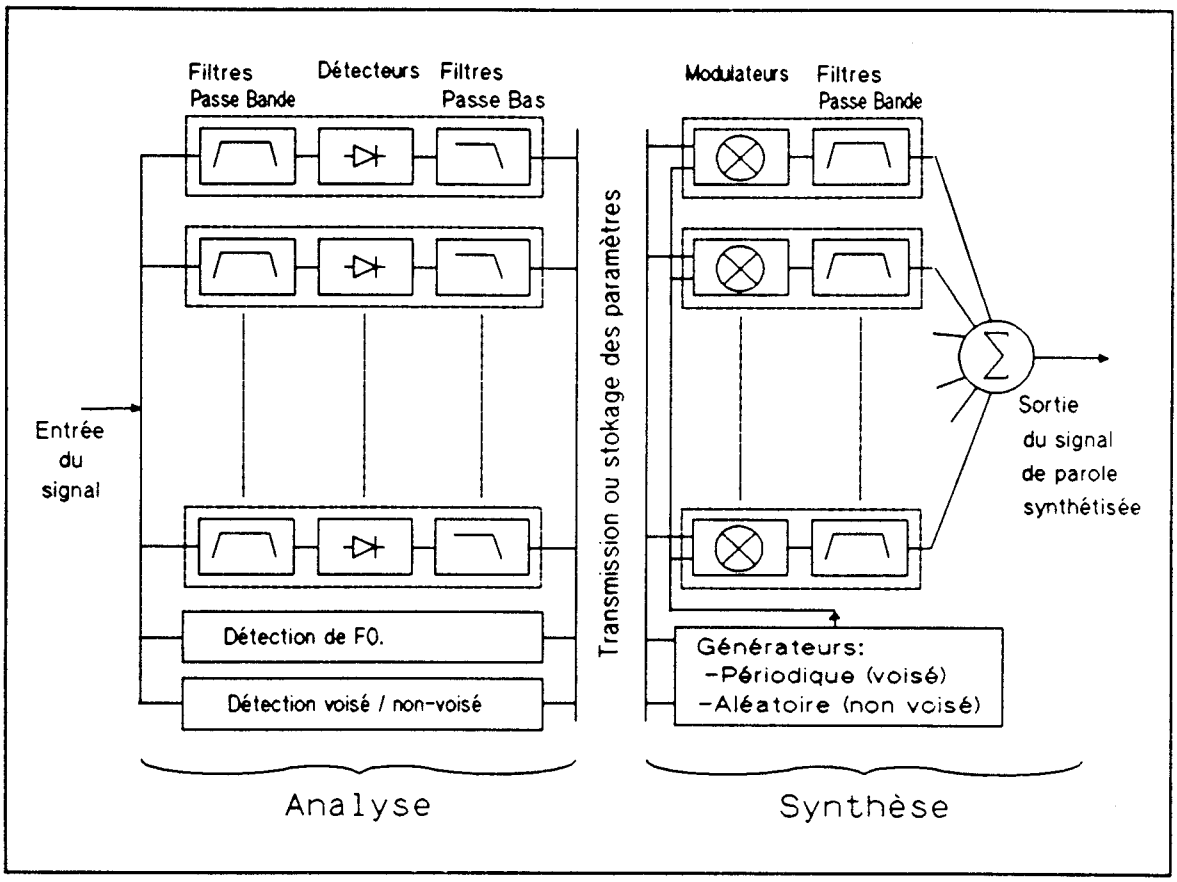


Figure 3.5. *Vocodeur à canaux.*

Le vocodeur à canaux est performant pour le traitement automatique de l'analyse synthèse du signal de parole. Mais étant donnée la qualité peu naturelle de la parole synthétisée, il n'est pratiquement plus utilisé aujourd'hui, ce défaut étant lié au nombre de filtres, et à la difficulté de doser le paramètre voisé/non-voisé du signal.

3.3.2. Synthétiseurs à prédiction linéaire (LPC).

Dans les systèmes de synthèse à prédiction linéaire, nous considérons que, sur une fenêtre où le signal est quasi stationnaire, la valeur d'un échantillon du signal peut être prédite

par une combinaison linéaire des n échantillons précédents. A l'analyse, les valeurs des coefficients, correspondant aux pôles d'un filtre numérique, sont calculées par un algorithme qui minimise l'erreur de prédiction (au sens des moindres carrés) entre le signal d'entrée et le signal prédit pour une fenêtre donnée (figure 3.6) [11] [12] [15] [21] [37].

Les calculs de l'analyse seront conduits par les notations suivantes :

S_n est la valeur d'un échantillon de rang n du signal réel,

$\hat{S}_n$ est la valeur d'un échantillon de rang n prédit et $\hat{S}_{(z)}$ la transformée en Z :

$$\hat{S}_n = \sum_{k=1}^{p} a_k \hat{S}_{n-k} \quad ; \quad \hat{S}_{(z)} = \frac{1}{1 - \sum_{k=1}^{p} a_k Z^{-k}}$$

$E_n = S_n - \hat{S}_n$ est l'erreur à minimiser.

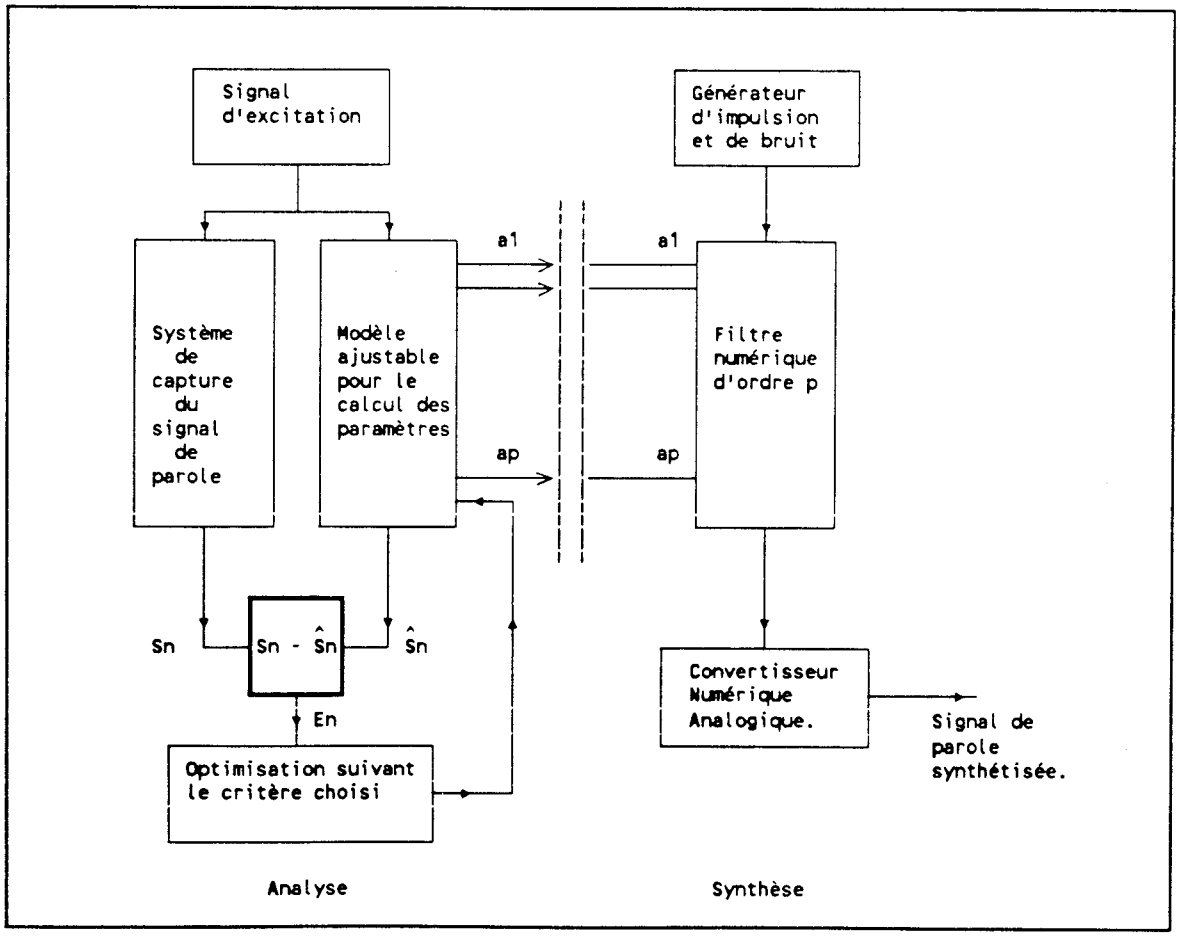


Figure 3.6. *Analyse et synthèse en prédiction linéaire.*

Les coefficients a_1 à a_p sont transmis ou stockés comme pour les autres synthétiseurs. Les deux méthodes les plus courantes, pour la résolution de ces systèmes, sont la covariance et l'autocorrélation. Le nombre de coefficients nécessaires est de l'ordre de 10 à 15, et la période de renouvellement se situe entre 5 et 20 ms. La fonction de transfert de ce système ne comporte que des pôles, les zéros seront approchés par une combinaison de plusieurs pôles successifs.

Dans un synthétiseur à prédiction linéaire un filtre numérique d'ordre p est excité par un générateur d'impulsions ou par un bruit blanc (voisé/non-voisé). Ce filtre numérique construit les échantillons du signal de sortie, en fonction des échantillons précédents et des coefficients de prédiction linéaire. La structure de ces filtres est souvent une structure en treillis (méthode d'Itakura-Saito) où deux vagues de signaux se déplacent en sens inverses, l'une portant le signal direct et l'autre le signal rétrograde. La valeur de ces signaux est calculée au niveau de chaque cellule, par un calcul matriciel à deux multiplicateurs (figure 3.7) [14].

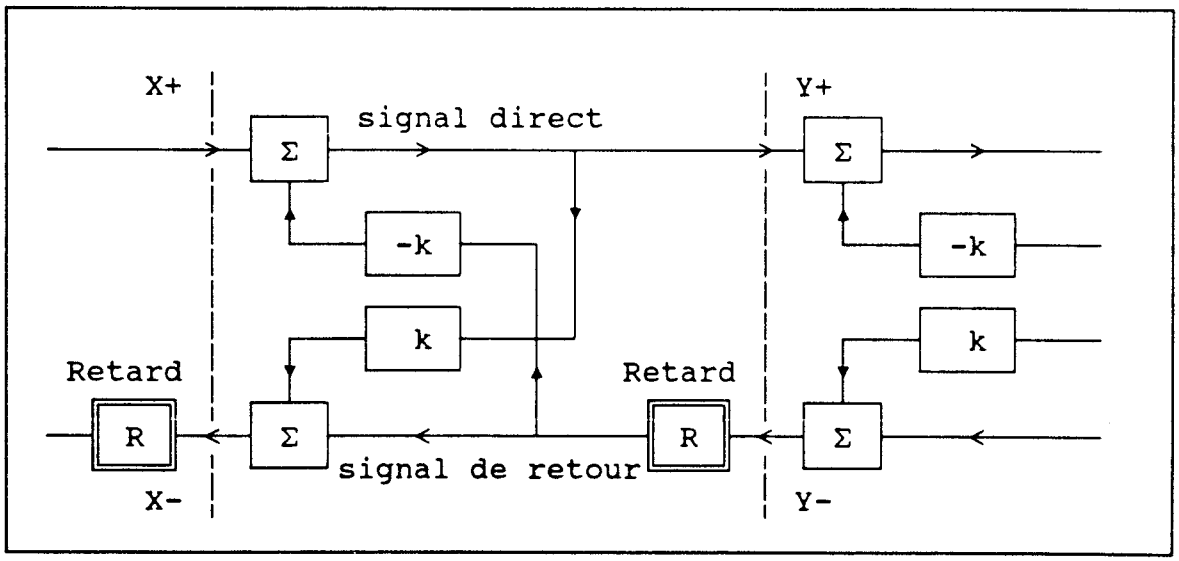


Figure 3.7. *Cellule de filtre en treillis pour la synthèse LPC.*

La synthèse à prédiction linéaire est très utilisée grâce à l'automatisation possible de l'ensemble analyse-synthèse. La qualité de la parole synthétisée dépend du nombre de coefficients, mais aussi de la qualité de la source d'excitation. Pour améliorer la qualité, certains systèmes utilisent la méthode multi-impulsionnelle qui permet de modifier les impulsions de la source plusieurs fois pendant une période de rafraîchissement des coefficients (L.P.C) [15].

3.3.3. Synthétiseurs à formants.

Les synthétiseurs à formants sont basés sur la simulation de la fonction de transfert du conduit vocal. Cette technique suppose que le spectre peut être réduit à une suite de pôles (résonances) et de zéros (anti-résonances). Les filtres utilisés pour réaliser cette fonction sont en général des filtres passe-bas résonateurs du second ordre, dont on paramètre la fréquence centrale, la bande passante et l'amplitude. La connexion de ces filtres peut être : cascade (ou série), parallèle ou mixte. La source d'excitation est constituée par deux générateurs, le premier fournit un signal représentant l'onde glottique, et le deuxième un bruit blanc pour les sons non voisés (Figure 3.8). Les architectures mixtes sont souvent constituées de trois canaux : le canal nasal est paramétré par un formant particulier et éventuellement un anti-formant (zéro). Le canal vocal reçoit la majeure partie des paramètres. Le canal de bruit est commandé par un "formant" de fréquence élevée. L'extraction automatique des formants n'est pas toujours facile et il faut souvent retoucher les trajectoires à la main dans le cas des structures parallèles. Mais en synthèse, la commande de ces synthétiseurs se prête bien à l'application de règles. Il est alors possible de grouper les calculs de trajectoires en différentes équations mathématiques. Ce type de synthétiseur sera utilisé dans notre application, il est dérivé de celui étudié par KLATT [27]. Nous le détaillerons dans les chapitres suivants.

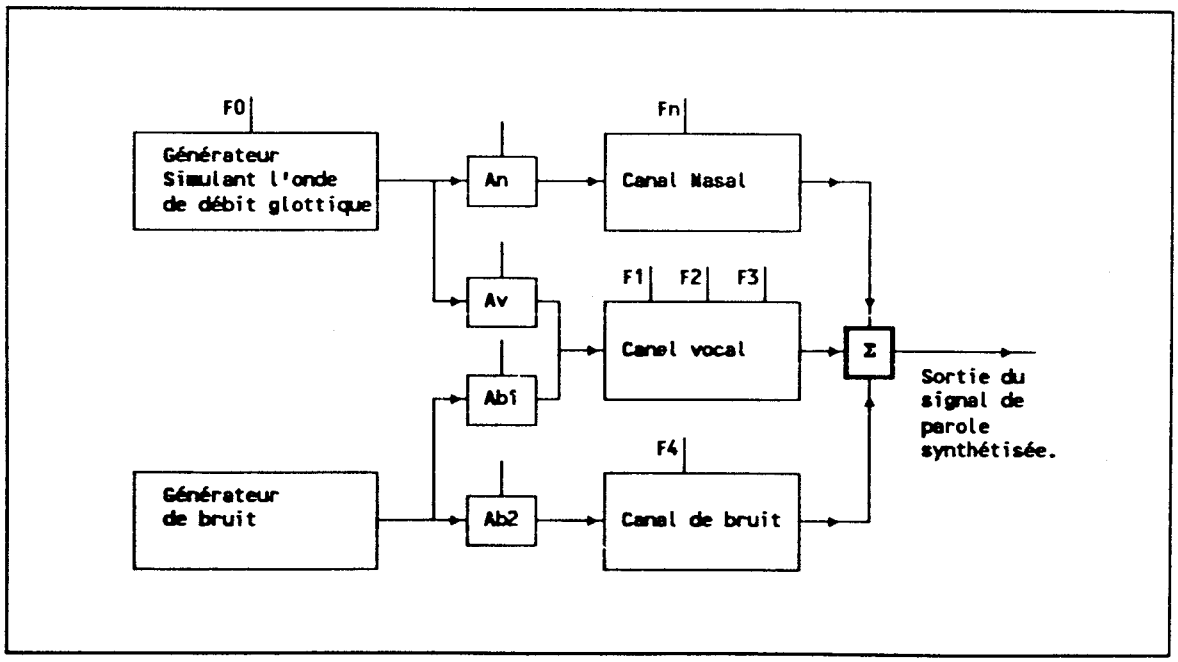


Figure 3.8. *Synthétiseur à formants.*

3.3.4. Synthétiseurs par concaténation d'éléments stockés.

Sous cette appellation se trouve toute une famille de synthétiseurs. Ils se distinguent par le type d'éléments stockés et par la méthode de concaténation. Nous trouverons par exemple, les formes d'ondes où une période du signal pour chaque son est mémorisée. Les phonèmes sont reconstitués par duplication et concaténation de ces ondes. Les éléments stockés sont, soit les échantillons du signal, soit des coefficients de filtres L.P.C. Pour les synthétiseurs plus complexes, le stockage concerne des durées entières de plusieurs phonèmes (polysons) ; nous atteignons ainsi une qualité de parole très naturelle, puisque les transitions entre phonèmes sont bien restituées. Enfin les raccordements de ces éléments sont constitués par un recouvrement et la synchronisation des signaux. Cette technique appelée PSOLA (Pitch Synchronous Over Lap and Add) [8] [16] autorise le traitement de la prosodie ; elle permet le contrôle de la période fondamentale (PITCH) ainsi que de l'énergie du signal. Une représentation de cette technique est donnée sur le schéma de la figure 3.9.

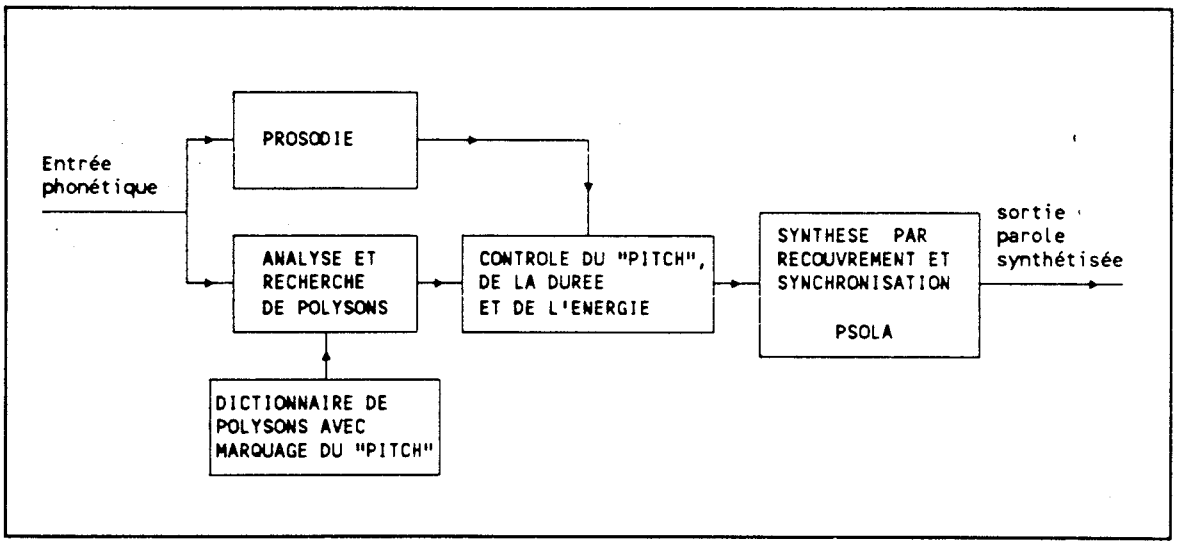


Figure 3.9. *Synthèse par recouvrement et synchronisation PSOLA.*

3.3.5. Les simulateurs du conduit vocal.

Appelée aussi synthèse articulatoire, cette technique modélise la production et la propagation du son dans le conduit vocal. Les variations de dimension du conduit tiennent compte des mouvements de l'appareil articulatoire (mâchoires, langue, voile du palais), et des caractéristiques des parois [12]. Cette simulation est souvent transcrite en une série de cellules LC, où chaque étage représente une petite partie du conduit. Cette méthode est très prometteuse mais aujourd'hui la quantité de calcul nécessaire, et le manque de données pour une synthèse complète, en font un système encore expérimental.

3.4. Prosodie.

La prosodie (ou "la musique que fait la voix") est une composante importante de la communication vocale. Sa fonction principale est de donner la position du locuteur par rapport à son discours. Les informations qu'elle indique sont :

- la compétence linguistique du locuteur (marquage des unités linguistiques),
- l'identité et l'état psychologique,
- la provenance sociologique,
- l'intentionnalité du discours ...

Ces informations sont véhiculées essentiellement par trois paramètres acoustiques :

- les variations mélodiques liées à la fréquence fondamentale de vibration des cordes vocales, que nous noterons F0.
- les variations de vitesse d'élocution (durée des phonèmes ou des syllabes),
- les variations d'intensité.

Les phénomènes liés aux variations mélodiques sont souvent utilisés en synthèse. Ils contribuent à une meilleure intelligibilité de la parole. Ces variations de F0 peuvent avoir plusieurs origines.

La ligne de déclinaison rend compte de la baisse régulière de la fréquence fondamentale entre le début et la fin d'une phrase ou d'un groupe de souffle. Elle donne en synthèse un aspect plus naturel à la voix.

La modalité d'une phrase est caractérisée par le contour mélodique. Bien que, l'analyse d'une structure mélodique complète soit complexe, nous pouvons remarquer que pour une phrase déclarative, la variation mélodique est descendante sur la dernière syllabe, alors que pour une phrase interrogative totale, cette variation est montante.

La micro-prosodie (ou micro-mélodie) est représentée par de petites variations de F0 se limitant aux phonèmes. Un exemple de micro-prosodie se trouve sur certaines des consonnes voisées (b, d, g, v, z, j ...), où un affaiblissement de la pression de sous-glottique entraîne une baisse de la fréquence fondamentale, de 10 à 20Hz par rapport aux voyelles les plus proches.

L'accent intervient au niveau de la syllabe suivant les mots ou selon le ton que l'on veut donner à la phrase.

En synthèse tous ces éléments sont groupés pour déterminer les valeurs de la fréquence fondamentale [12] [4] [5] [9] [19] [30].

4. La conversion texte-parole.

La méthode de synthèse choisie, pour générer la sortie vocale du serveur de message, est la synthèse par règles du français. La mise en oeuvre de cette synthèse nécessite des outils de développement et de recherche offrant une grande souplesse d'utilisation. Ce but est atteint par le logiciel de traitement de la parole COMPOST.

4.1. Le logiciel COMPOST.

COMPOST est un compilateur orienté objets qui est destiné essentiellement à la synthèse de la parole à partir de texte [1] [23]. Toutes les étapes, du texte écrit, jusqu'à la sortie de la parole synthétisée, sont traitées séquenciellement ; elles correspondent, soit à un appel de fonction (call), soit à l'exécution de règles de grammaire. Cette suite de traitements, appelée "scénario", crée une arborescence où chaque branche est une instance d'objet. Les objets et les règles sont définis dans des fichiers différents, leur écriture suit la syntaxe imposée par le langage orienté objet. Les fonctions appelées par le scénario peuvent avoir des buts très divers. Elles servent, notamment, à relier les traitements exercés par les règles à l'environnement extérieur, ou à initialiser l'application. Nous pouvons distinguer deux catégories de fonctions : d'une part celles qui sont intégrées au logiciel COMPOST de base, tels que la lecture d'un fichier d'entrée ou l'affichage de la trace de l'arborescence, d'autre part les fonctions spécifiques à l'application. Ces dernières fonctions doivent aussi être compilées puis liées avec le logiciel COMPOST de base.

4.2. Architecture COMPOST.

Le schéma bloc de COMPOST et de son environnement est représenté en figure 4.1. Nous pouvons remarquer les fonctions de l'application qui font partie intégrante du logiciel, et les fichiers objets et grammaires qui peuvent être modifiés à tout moment par un langage programmation orienté objet.

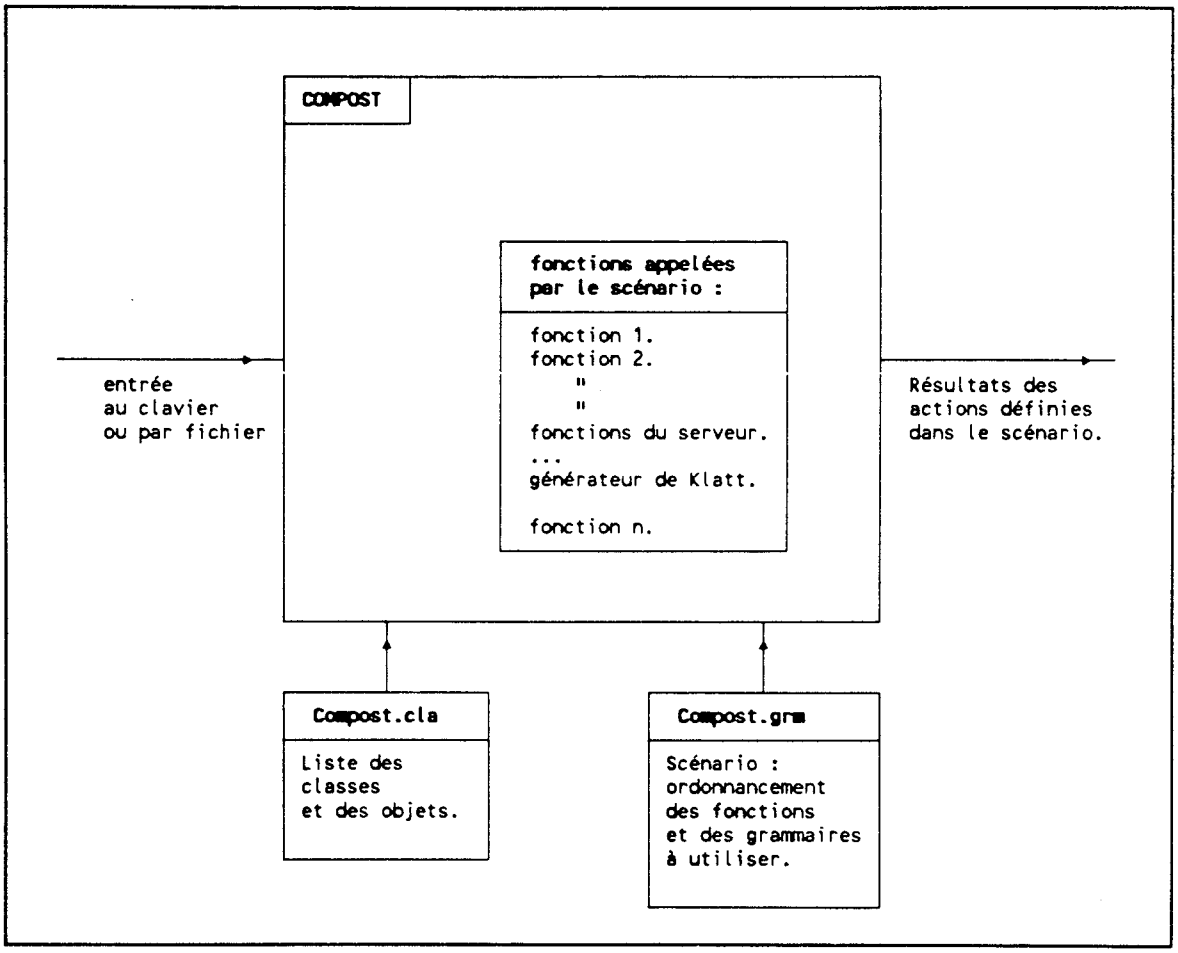


Figure 4.1. *Organisation de COMPOST.*

La structure d'un scénario comporte : une partie initialisation qui fait essentiellement appel à des fonctions, le corps du programme qui boucle tant que la dernière entrée n'est pas détectée, et éventuellement une partie fin après le corps permettant de sortir proprement en libérant les zones mémoire allouées dynamiquement. Le corps est délimité par les deux mots clé CORPS et FINCORPS, on y trouvera des appels de fonctions et toute la définition de l'arborescence.

Pour être interprétés par COMPOST, les fichiers (texte) "objets.cps" et "scénario.cps" doivent être compilés par un exécutable "clasgram.exe" qui donnera deux nouveaux fichiers "compost.cla" et "compost.grm" (Figure 4.2).

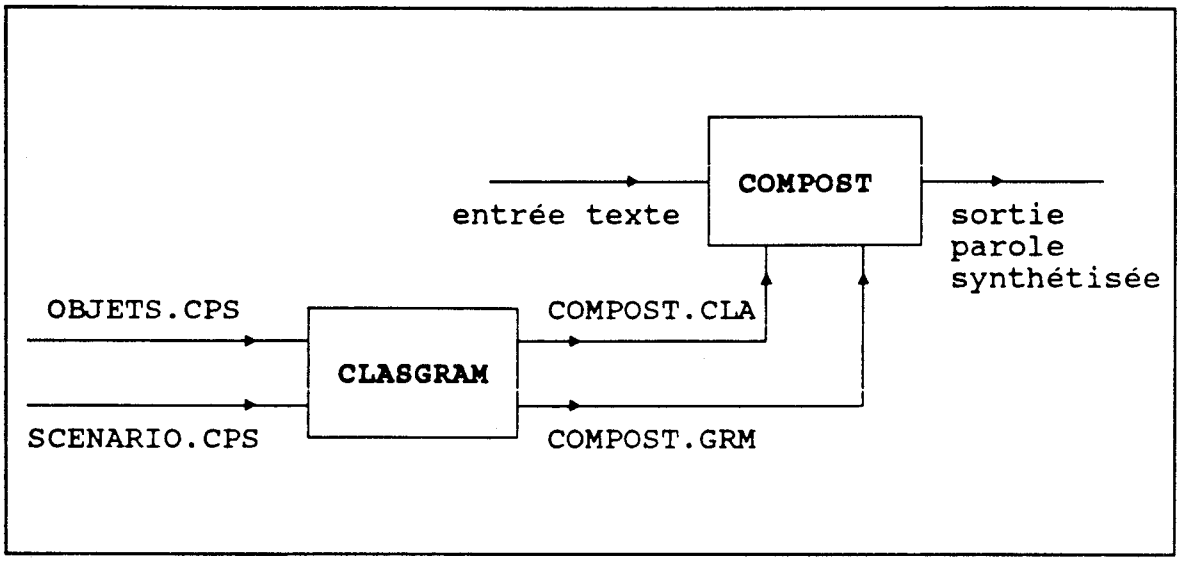


Figure 4.2. *Compilation des fichiers COMPOST.*

4.2.1. Définition des objets.

Les objets déclarés dans un fichier texte sont regroupés en classes, afin de bien marquer les différentes étapes de la synthèse. Nous trouverons dans l'ordre d'utilisation, les classes : phrases, mots, caractères alphanumériques (graph), les phonèmes, puis les trajectoires des formants (cibles). A l'intérieur de chaque classe, il est possible de définir des traits, et des indices qui pourront être attribués aux objets (voir exemples ci-dessous). Les traits permettent de qualifier et de grouper des objets, afin de constituer par la suite des règles plus simples et plus générales. Les objets sont affectés d'un trait par le verbe "est". Les indices sont les paramètres des objets, ils sont décrits en valeurs : minimum, maximum et par défaut, les valeurs sont attribuées aux objets par le verbe "vaut".

Exemples d'écriture de classes :

```
classe mot
        objet (Inc,Det,Nom,Num, ... Pt,Vrb,Vrg)
        trait (SUJ,MAS,FEM,NEU,SNG,PLU ...)
finclasse
```

Dans la classe "mot" nous trouvons les objets : déterminants (Det), nom (Nom), verbe (Vrb) etc. "Inc" (inconnu) indique un objet dont la nature n'a pas encore été attribuée. Les traits de ces mots peuvent être : sujet (SUJ), masculin (MAS), pluriel (PLU), etc.

```
classe phon
        objet (a,e,i, ... p,t,k,b,d ... g~,n~,_)
        trait (voi,nas,occ,cns,liq,ouv,lien, ...)
        indice (duree dans [20,500,100],  mf0 dans [-50,  1,  1], ...)
        a   est  ouv    vaut duree= 90; ...
        b   vaut duree= 70, mf0 = -10; ...
finclasse
```

La classe "phon" contient les phonèmes du français ainsi que le silence. Ils peuvent être affectés des traits distinctifs : voisé (voi), occlusive (occ), consonne (cns), etc.

```
classe cible
  objet (X0,X1,X3,X2G,X2D)
  trait (final)
  indice(      F1 dans [  0,5000,   0],
               F2 dans [  0,5000,   0],
               F3 dans [  0,5000,   0],
               F4 dans [  0,5000,   0], ... )
finclasse
```

Les objets de la classe "cible" correspondent aux types de trajectoires de liaisons. Elles prennent les valeurs d'indices correspondant aux paramètres du synthétiseur à formant.

La liste du fichier objet.cps, qui contient tous les objets de notre application, est donnée en ANNEXE 5.

4.2.2. Ecriture des règles et des grammaires.

Les règles sont groupées dans des grammaires, afin de bien marquer toutes les étapes du traitement séquentiel. L'écriture des règles suit le format :

Nom de règle = Partie Gauche (-> Partie droite / Contexte Gauche + Contexte Droit) ;

La partie gauche donne la partie droite, si le contexte gauche et le contexte droit sont vérifiés. Il faut cependant noter que l'existence de contexte n'est pas toujours obligatoire. Par exemple, dans la traduction alphanumérique phonétique la règle " EAU -> o ; " est valable quelque soit le contexte.

La définition d'une grammaire précise les classes des quatre éléments des règles.
Syntaxe d'écriture des grammaires :

```
grammaire  nom [classe PG -> classe PD  / classe CG + classe CD ]
        nom de règle = règle 1
        nom de règle = règle 2
        ...
fin  grammaire
```

Lorsque le nombre de grammaires nécessaires à une application devient très grand, il est possible, pour faciliter l'écriture et la lecture du fichier principal du scénario, d'écrire des grammaires dans des fichiers auxiliaires, et de les inclure par des appels à l'endroit de leurs interventions. L'appel de ces fichiers se fait, comme en langage C, par la syntaxe suivante : " #include nom de fichier ".
Le fichier contenant un exemple de scénario de notre application est montré en ANNEXE 6.

4.2.3. Ecriture d'un scénario.

Exemple de structure d'un programme COMPOST :

call Initialisation de compost

call Initialisation de l'application

CORPS

call lecture de l'entrée

grammaire Traduction graphèmes phonèmes
Règles ...
fin grammaire

grammaire Syllabation
Règles ...
fin grammaire

grammaire Génération des trajectoires des formants
Règles ...
fin grammaire

grammaire Duplication et Raccordement des cibles entre les phonèmes
Règles ...
fin grammaire

grammaire Prosodie
Règles ...
fin grammaire

call contrôle de la carte de sortie

FINCORPS

4.3. Arborescence et synthèse de la parole.

Les règles, qui agissent au cours du traitement du texte écrit vers la sortie du signal de parole, ne construisent pas obligatoirement une arborescence linéaire et ascendante. Par exemple, les transformations d'un texte de deux mots "LE CHEMIN" effectuées par l'application des règles de chaque grammaire (figure 4.3), nous amène à faire les remarques suivantes :

- la traduction orthographique phonétique ne génère pas de niveau supplémentaire, les mots qui dominaient des graphèmes englobent maintenant des phonèmes,
- la syllabation intercale un niveau supplémentaire entre les mots et les phonèmes, les mots dominent les syllabes, qui à leur tour dominent des phonèmes,
- la génération des trajectoires prolonge l'arborescence depuis les phonèmes, ainsi les phonèmes dominent des cibles,
- les autres grammaires opèrent des modifications locales, notamment les trois dernières ajoutent des branches aux extrémités.

Texte initial :
Det (LE)Nom (CHEMIN)#

TOPF : (traduction orthographique phonétique)
Det (lq)Nom (s^qme~)#

ELISION : (modification des sons du "E")
Det (lx^)Nom (s^x^me~)#

Syllabation : (groupement des phonèmes en syllabes)
Det (Syl (lx^))Nom (Syl (s^x^)Syl (me~))#

Generation : (Affectation des trajectoires des formants)
Det (Syl (l (X1)x^ (X1)))Nom (Syl (s^ (X1)x^ (X1))Syl (m (X1)e~ (X3X1)))#

Duplication : (duplication des trajectoires en début et fin du phonème)
Det (Syl (l (X1X3)x^ (X1X3)))Nom (Syl (s^ (X1X3)x^ (X1X3))Syl (m (X1X3)e~ (X3X1X3)))#

F0_BASE : (modification de la prosodie)
Det (Syl (l (X1X1X3)x^ (X1X3)))Nom (Syl (s^ (X1X3)x^ (X1X3))Syl (m (X1X3X2G)e~ (X3X1X3))#

Figure 4.3. *Arborescence créée par un scénario COMPOST.*

Les niveaux de l'arborescence sont visibles par les parenthèses ; ainsi dans l'exemple donné, les grammaires ont défini les dépendances suivantes :

mot(syl(phon(cible cible ...) phon ...) syl(phon(cible)) ...)

que nous pouvons représenter aussi sous une forme verticale (figure 4.4).

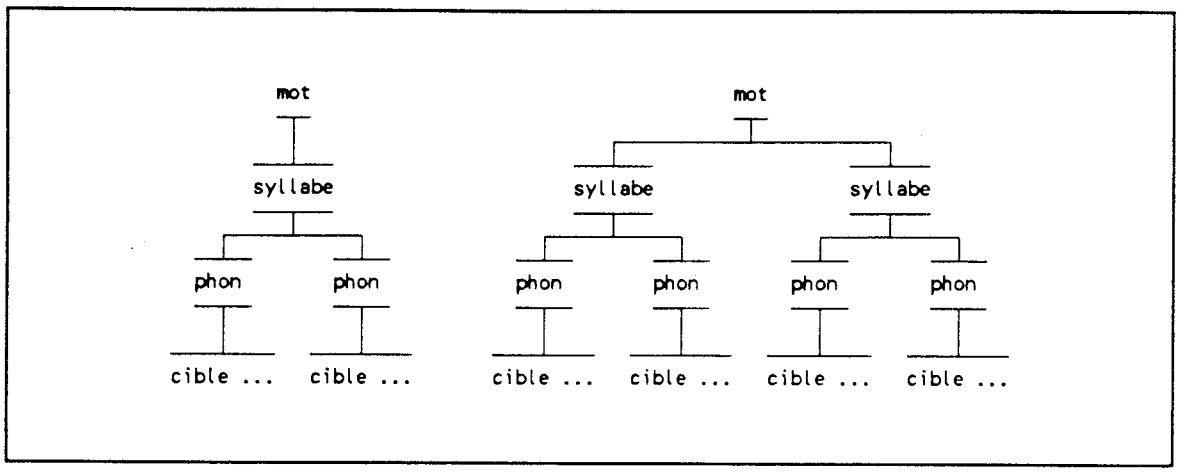


Figure 4.4. *Dépendances des classes lors d'un traitement COMPOST.*

4.4. Fonctions importantes pour la synthèse à formants.

Le synthétiseur que nous utilisons est semblable au modèle décrit par KLATT (1980) [27] (ANNEXE 2). Il a été utilisé aux cours de différents travaux sur la synthèse de la parole, et s'est enrichi d'améliorations, notamment dans la modélisation de la source d'excitation [2] [3] [17] [32].

Ces fonctions prolongent le traitement de la parole en transformant les derniers éléments de l'arborescence (cibles) en paramètres (formants ...). Le synthétiseur est prévu pour recevoir des paramètres dont la période d'échantillonnage est fixée à 5 ms, alors que, les paramètres fournis par les phonèmes sont positionnés à des intervalles très variables. Il faut donc une fonction d'interpolation pour calculer les valeurs intermédiaires. En moyenne, un phonème possède 2 à 3 positions de cibles par paramètre, après duplication et coarticulation.

Les interpolations, entre cibles, sont effectuées par des trajectoires droites ou représentant des fonctions polynomiales. La figure 4.5 montre l'indépendance qu'il peut y avoir entre les positions temporelles des cibles de formants différents. Nous pouvons avoir des calculs qui opèrent sur des durées différentes, par exemple, les positions des cibles pour F0_BASE sont modifiées beaucoup moins souvent que celles des autres paramètres (F1, F2...).

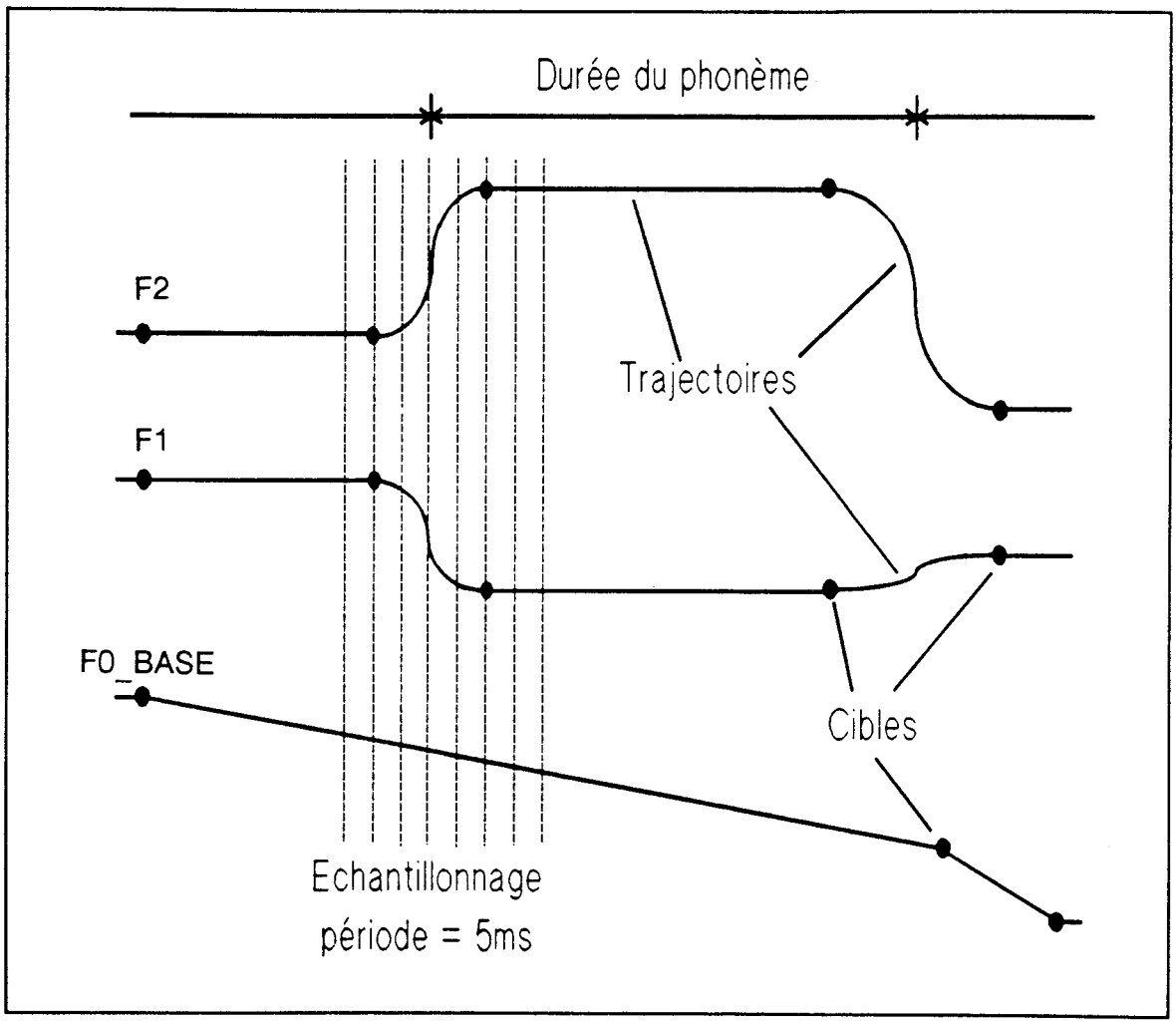


Figure 4.5. *Générateur de trajectoires.*

Les types de trajectoires entre cibles, qui peuvent être utilisées, sont :

- X0, un échelon ou valeur absolue,
- X1, une droite entre les deux cibles,

- X2G, courbe présentant une dérivée nulle à gauche,
- X2D, courbe présentant une dérivée nulle à droite,
- X3, courbe présentant des dérivées nulles à droite et à gauche.

La figure 4.6 montre graphiquement ces types de trajectoires.

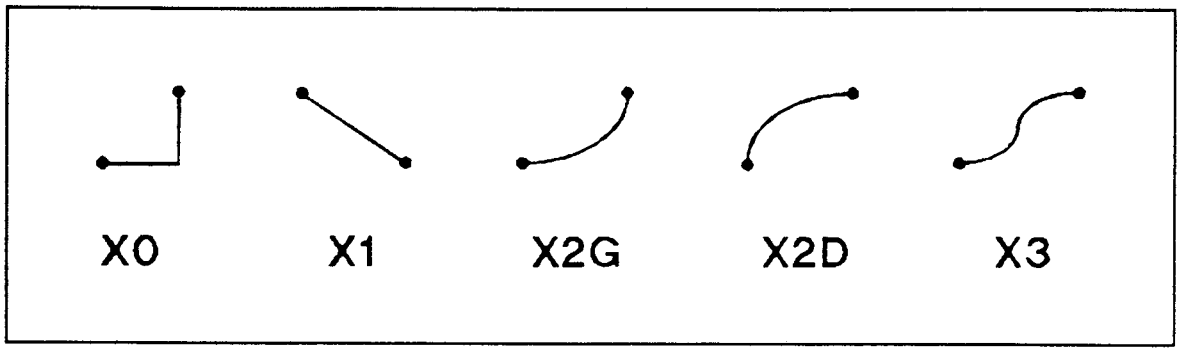


Figure 4.6. *Types de trajectoires.*

Le nombre de paramètres du synthétiseur à formants que nous utilisons (schéma en ANNEXE 2), est de 24. Ils sont définis dans le fichier d'objets sous la forme d'indices dans la classe des cibles. Le paramètre F0 n'est pas défini comme un indice. Il est calculé en faisant la somme des trois indices lors de la constitution des trames de paramètres (modèle de Fujisaki).

Composition du paramètre F0 :

- F0_BASE, fréquence de référence de F0. Elle sera utilisée dans les règles pour modéliser la ligne de déclinaison et le contour mélodique,

- ACCENT, variations de F0 sur certaines syllabes accentuées,

- MICRO, variations relative de F0 pour la micro-prosodie sur certain phonèmes. Elle est négative pour les consonnes voisées (v, z, j, f, s, *ʃ*), et positive pour les consonnes plosives non voisées (p, t, k).

Quatre paramètres modélisent la source vocale :

- F0 , fréquence fondamentale,
- U0 , amplitude de la source,
- QD , coefficient de dissymétrie,
- QO , coefficient d'ouverture.

Le graphique de la figure 4.7 nous montre la signification et le calcul des ces paramètres.

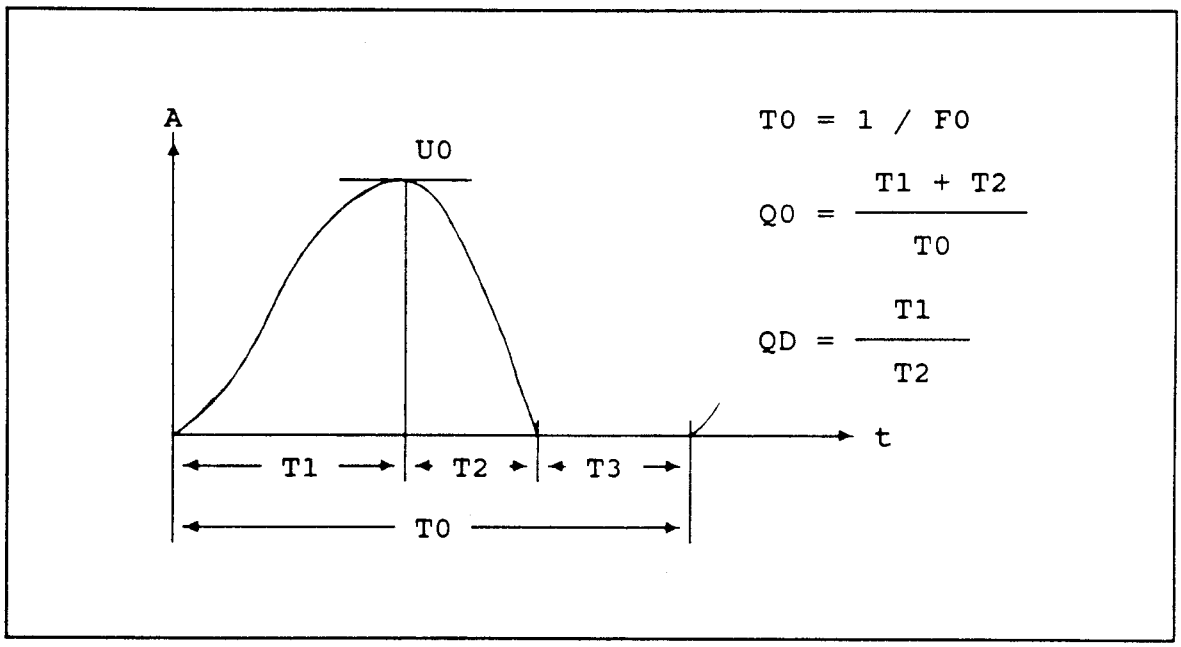


Figure 4.7. *Modélisation de la source vocale.*

Le générateur de bruit est caractérisé par deux indices :

- AH , amplitude de l'aspiration pour le canal F1 et nasal,
- AF , amplitude des fricatives pour le canal vocal et de canal de bruit (F5).

les paramètres du canal nasal sont indicés par :

- FNP, fréquence du formant nasal (pôle),
- BNP, bande passante du formant nasal (pôle),
- FNZ, fréquence de l'anti-formant nasal (zéro),
- BNZ, fréquence de l'anti_formant nasal (zéro).

Les paramètres des formants sont définis par les indices :

- F1, F2, F3, F1, F5, fréquences de résonance des formants,
- B1, B2, B3, B4, B5, bandes passantes des formants,
- A2, A3, A4, A5, coefficients d'amplitude des filtres résonateurs (sauf pour F1 qui reçoit directement le signal, contrôlé, de la source vocale).

Le schéma bloc et la réponse en fréquence d'un filtre (numérique) résonateur du second ordre, utilisé pour reconstituer les formants, sont représentés sur la figure 4.8.

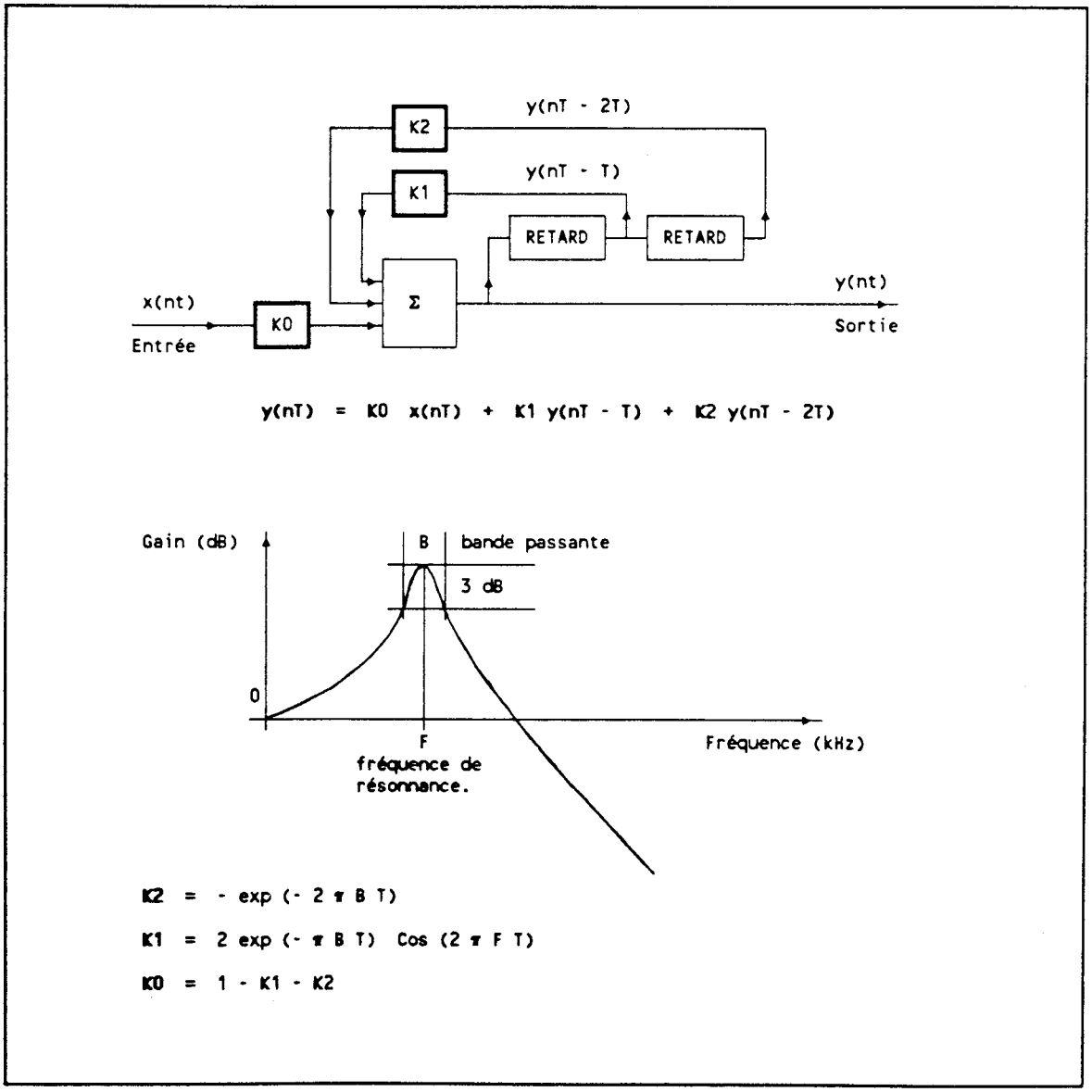


Figure 4.8. *Filtre numérique pour les formants (passe-bas résonateur).*

Nous montrons ci dessous, sous la forme d'un algorithme, la fonction qui génère les trajectoires, et commande le synthétiseur de KLATT.

Algorithme de génération des trajectoires et de commande de la carte de traitement du signal :

Initialisation de la carte de traitement de signal ;
Initialisation du début des trajectoires ;
Tant que (le dernier objet de la phrase n'est pas détecté)
    Lecture de l'objet suivant ;
        Si (l'objet est un phonème)
            Lecture de sa durée et Calcul de la durée totale ;
        Sinon
            Si (l'objet est une cible)
                Lecture des traits et des indices ;
                Génération d'une trajectoire pour chaque indice
                (Allocation mémoire) ;
            Fin Si
        Fin Si
Fin Tant que
Connexion des trajectoires (par les pointeurs: suivant, précédant) ;
Sauvegarde des pointeurs de début de phrase pour chaque indice ;
Faire
    Positionnement des pointeurs en début de phrase ;
    Faire
        Synthèse du message (passage de trames de tous les paramètres) ;
        Traitement des pauses ;
    Tant que (la fin du message n'est pas atteinte)
    Fin Faire
Tant que (les traitements des répétitions et des pauses sont actifs)
Fin Faire

Libération de la mémoire ;

Au niveau de la conception du logiciel, les trajectoires sont caractérisées par des structures, dans lesquelles sont définies :

- le type et la longueur de la trajectoire,
- les valeurs de début et de fin du paramètre ainsi que des variables de calcul pour les valeurs intermédiaire,
- deux pointeurs, sur les structures (suivante et précédente) permettent la connexion des trajectoires d'un même paramètre.

Toutes ces structures sont positionnées par des allocations mémoire lors de l'initialisation des trajectoires, puis cette partie de la mémoire est automatiquement libérée à la sortie de la fonction.

La sauvegarde des pointeurs de début de phrase est indispensable. Considérés comme des points d'entrée, ils adressent les premières structures du cycle, les autres sont adressés récursivement par les pointeurs internes "suivants" des structures précédentes. un repositionnement en début de phrase permet ainsi une relecture ou la libération de la mémoire.

Le traitement des pauses est paramétré par une variable globale (commande "pause"). La détection, du point d'intervention des arrêts, est faite au niveau du phonème "_" (silence) par la valeur de U0. Si la variable pause est active et U0 en dessous d'un certain seuil, la diffusion du message s'arrête jusqu'à une prochaine frappe au clavier.

5. Gestion des bases de données.

Les informations contenues dans les bases de données constituent le principal support des messages. Elles sont sélectionnées et rangées avant d'être converties par le serveur et intégrées dans un message parlé. Afin d'offrir une grande diversité d'utilisations et de formats de messages, ces informations peuvent se trouver dans plusieurs bases de données (10 par application). Ainsi une fonction agenda peut prendre en compte les agendas individuels de plusieurs personnes.

5.1. Choix du système de gestion des bases de données.

Le format des fichiers de bases de données, que nous avons choisi, est un format compatible avec les systèmes "dbase III", "dbase III+" et "dbase IV". Celui ci est parmi les plus répandus sur les micro-ordinateurs personnels. Il permet d'extraire des messages avec des bases de données déjà existantes, et de rendre ainsi notre serveur immédiatement opérationnel. La diffusion de ce format nous autorise, dans un premier temps, à ne pas intégrer de gestionnaire de bases de données complet, qui prendrait beaucoup de place mémoire, mais d'utiliser un gestionnaire séparé, les deux fonctions pouvant être indépendantes.

5.2. Constitution des fichiers bases de données.

La structure d'un fichier base de données ".dbf" est constituée par une entête, et par des enregistrements mis à la suite dans l'ordre d'écriture. L'entête comprend deux parties. La première de taille fixe comporte des renseignements généraux, tels que la date de création, la taille de l'entête, et le nombre d'enregistrements. La seconde décrit les champs, avec pour chacun, le nom, le type et la taille. La taille de l'entête dépend du nombre de champs de la base de données. Les enregistrements sont composés par tous les champs définis dans l'entête, plus un caractère de validation situé au début : "espace" pour les enregistrements

actifs et "*" pour ceux qui sont éliminés. Dans notre application, les entêtes des bases de données sont stockées dans des structures au moment de l'initialisation, ainsi nous pouvons à tous instants, connaître les caractéristiques des bases de données, sans avoir à lire les entêtes à chaque accès. La fonction de lecture d'un champ a comme arguments d'entrée : les numéros de base (0 à 9 pour 10 bases maximum), d'enregistrement et de champ. Elle calcule la position du champ et prend le nombre de caractères correspondants à la longueur, en fonction des éléments des structures des bases de données.

Exemple de calcul de la position d'un champ :

Position du champ = Taille de l'entête
+ N° d'enregistrement * Taille d'un enregistrement
+ Σ des champs précédents dans l'enregistrement.

Structures du début de l'entête en langage C :

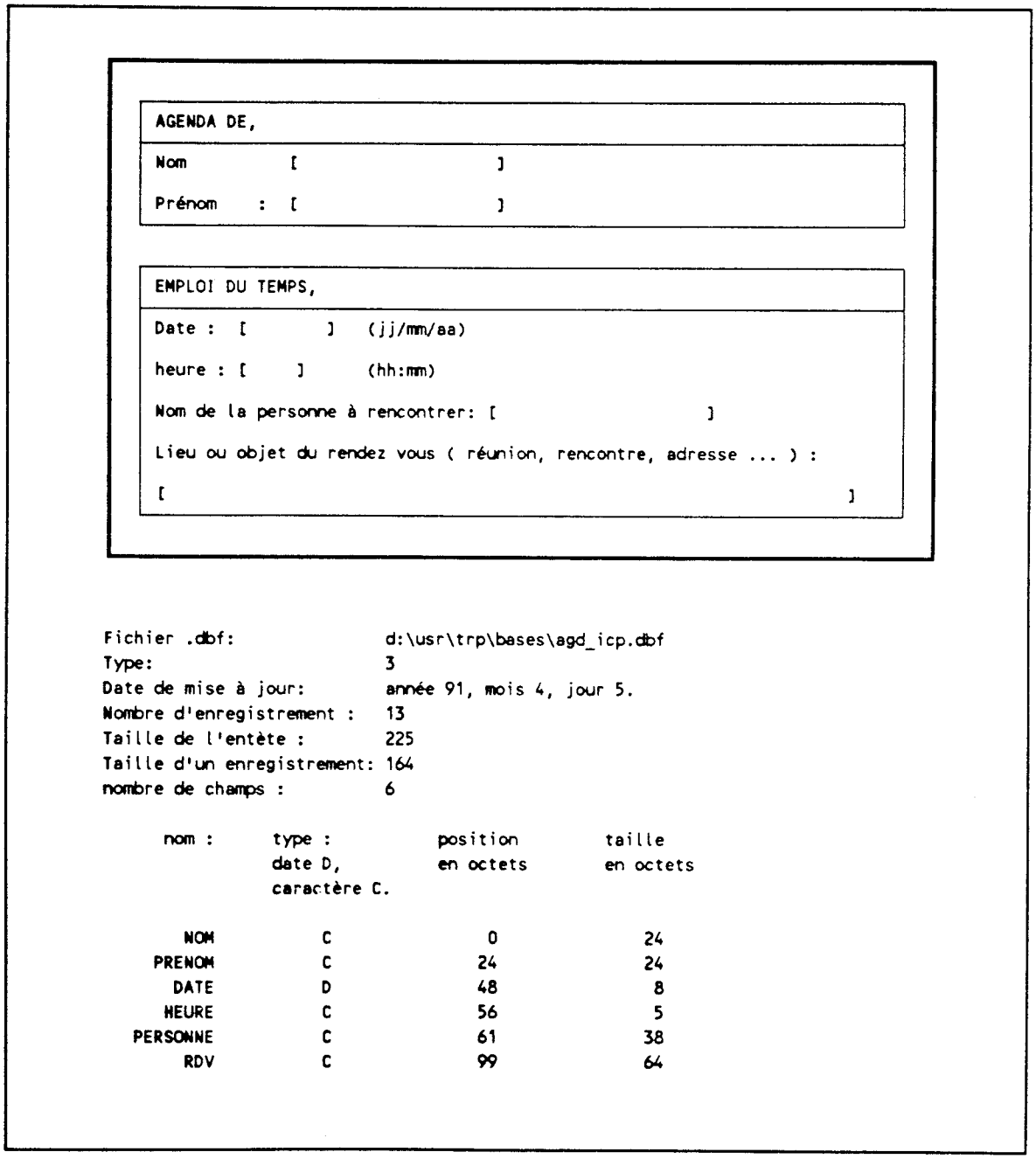
(les éléments faisant directement partie de l'entête sont soulignés):

```
typedef struct d_dbf
    {
    char nom_f_dbf[DIM_DEF_FICHIER] ; /* Nom et chemin du fichier base.     */
    FILE *fichier_dbf ;               /* Pointeur du fichier base.          */
    unsigned char  f_dbt ;            /* Test de la présence d'un fichier .dbt.  */
                                      /* 03 -> non  ;  83h -> 131 -> oui (1 octet). */
    unsigned char  annee ;            /* Date de la dernière mise à jour (3 octets). */
    unsigned char  mois  ;            /*                                    */
    unsigned char  jour  ;            /*                                    */
    unsigned long  nb_enr ;           /* Nombre d'enregistrements (4 octets).  */
    unsigned short  dim_ent ;         /* Taille de l'entête (2 octets).     */
    unsigned short  dim_enr ;         /* Taille d'un enregistrement (2 octets).  */
    unsigned short  nb_ch ;           /* Nombre de champs.                  */
    unsigned short  nb_ndx ;          /* Nombre de fichiers index associés.  */
    } Ddbf ;
```

Struture de définition d'un champ:

```
typedef struct champ
    {
    char nom_champ[11] ;              /* Nom du champ (10 octets + 1).      */
    char type_champ ;          /* Type du champ (1 octet).               */
    unsigned short ad_org ;           /* Adresse par rapport à l'origine (2 octets) */
    unsigned short dim_champ ;        /* Taille du champ (1 octets).        */
    } Champ ;
```

La figure 5.1 présente un écran du gestionnaire de bases de données et les informations de l'entête chargées dans les structures. Cet écran est prévu pour la composition d'un agenda. Les contenus des champs sont capturés sur les parties grisées entre crochets. Les informations recueillies dans les entêtes permettent de connaitre à tout moment la constitution de la base (par exemple nom et taille des champs), sans avoir à ouvrir le fichier.



```
AGENDA DE,
Nom          [                    ]
Prénom    :  [                    ]

EMPLOI DU TEMPS,
Date :  [        ]   (jj/mm/aa)
heure : [     ]      (hh:mm)
Nom de la personne à rencontrer: [                    ]
Lieu ou objet du rendez vous ( réunion, rencontre, adresse ... ) :
[                                                              ]
```

```
Fichier .dbf:              d:\usr\trp\bases\agd_icp.dbf
Type:                      3
Date de mise à jour:       année 91, mois 4, jour 5.
Nombre d'enregistrement :  13
Taille de l'entète :       225
Taille d'un enregistrement: 164
nombre de champs :         6

      nom :     type :          position        taille
                date D,         en octets       en octets
                caractère C.

        NOM         C              0              24
     PRENOM         C             24              24
       DATE         D             48               8
      HEURE         C             56               5
  PERSONNE          C             61              38
        RDV         C             99              64
```

Figure 5.1. *Ecran de gestion et entète d'une base de données.*

Si les bases sont petites, la recherche de données peut se faire séquentiellement en lisant chaque enregistrement, par contre pour les bases importantes, la recherche doit se faire par un fichier auxiliaire d'index, où les enregistrements sont plus réduits (en général un seul champ) et rangés par ordre alphabétique.

5.3. Fonctions développées en langage C pour les bases de données.

5.3.1. Recherches dans les bases de données.

La sélection des enregistrements de la base de données nécessite l'utilisation de fonctions spécifiques de recherche, de sélection, de tri, etc. La première étape consiste à rechercher les bases de données, qui correspondent au format de sortie. Pour cela, les noms des arguments doivent se retrouver dans les noms des champs, nous les appellerons champs de recherche. Ensuite pour les champs de lecture se trouvant dans la description du message (voir l'écriture des formats chapitre 6 page 43), nous considérons que leurs présences dans les bases doivent coïncider avec celles des champs de recherche. Ces champs pourront avoir un caractère obligatoire ou facultatif. Dans les recherches, sans niveau relationnel, tous les champs obligatoires doivent se trouver dans le même fichier que les champs des arguments. On peut noter aussi, que les arguments et les champs de lecture peuvent être communs. La sélection des champs, quant à elle, compare les valeurs des arguments et les contenus des champs correspondants. Le tri range, suivant des critères alphabétiques ou temporels, les coordonnées des enregistrements.
Nous proposons ci dessous un algorithme de recherche sans niveau relationnel entre les bases de données :

```
Recherche_base
        Pour (chaque base de données)
                Recherche des arguments dans les noms des champs ;
                Si (oui)
                        Recherche des noms des champs de lecture ;
                        Si (oui)
                                Recherche et Sélection des enregistrements ;
                        Fin Si
                Fin Si
        Fin Pour
        Tri des enregistrements ;
Fin Recherche_base
```

Les recherches, à un niveau relationnel, donnent la possibilité de séparer les champs de lecture dans des fichiers différents, les arguments devant toujours être dans le même fichier pour que la sélection soit possible, et ne pas alourdir l'algorithme. La relation entre les bases de données va se faire par les noms des arguments, afin que les champs de lecture soient repérés dans les bases de données. Le déroulement des opérations se fait de la manière suivante : d'abord recherche et présélection des enregistrements, selon les arguments pour en préciser le contenu. Puis avec les arguments présélectionnés, une nouvelle recherche dans les bases de données contenant les champs de lecture, donnera la sélection définitive.

Algorithme de recherche à un niveau relationnel :

Recherche_base_1

Pour (chaque base de données)
    Recherche des arguments dans les noms des champs ;
    Si (oui)
        Recherche et **Présélection** des enregistrements ;

        Pour (chaque enregistrement présélectionné)
            Précision du contenu des arguments ;
            Recherche des arguments dans les noms des champs ;
            Si (il en existe au moins un)
                Recherche des noms des champs de lecture ;
                Si (oui)
                    Recherche et **Sélection** des enregistrements ;
                Fin Si
            Fin Si
        Fin Pour

    Fin Si
Fin Pour

Tri des enregistrements ;

Fin Recherche_base_1

Pour mieux illustrer les différences entre les deux recherches, la figure 5.2 montre un exemple de recherche aléatoire des numéros de poste des personnes travaillant dans une ville donnée.

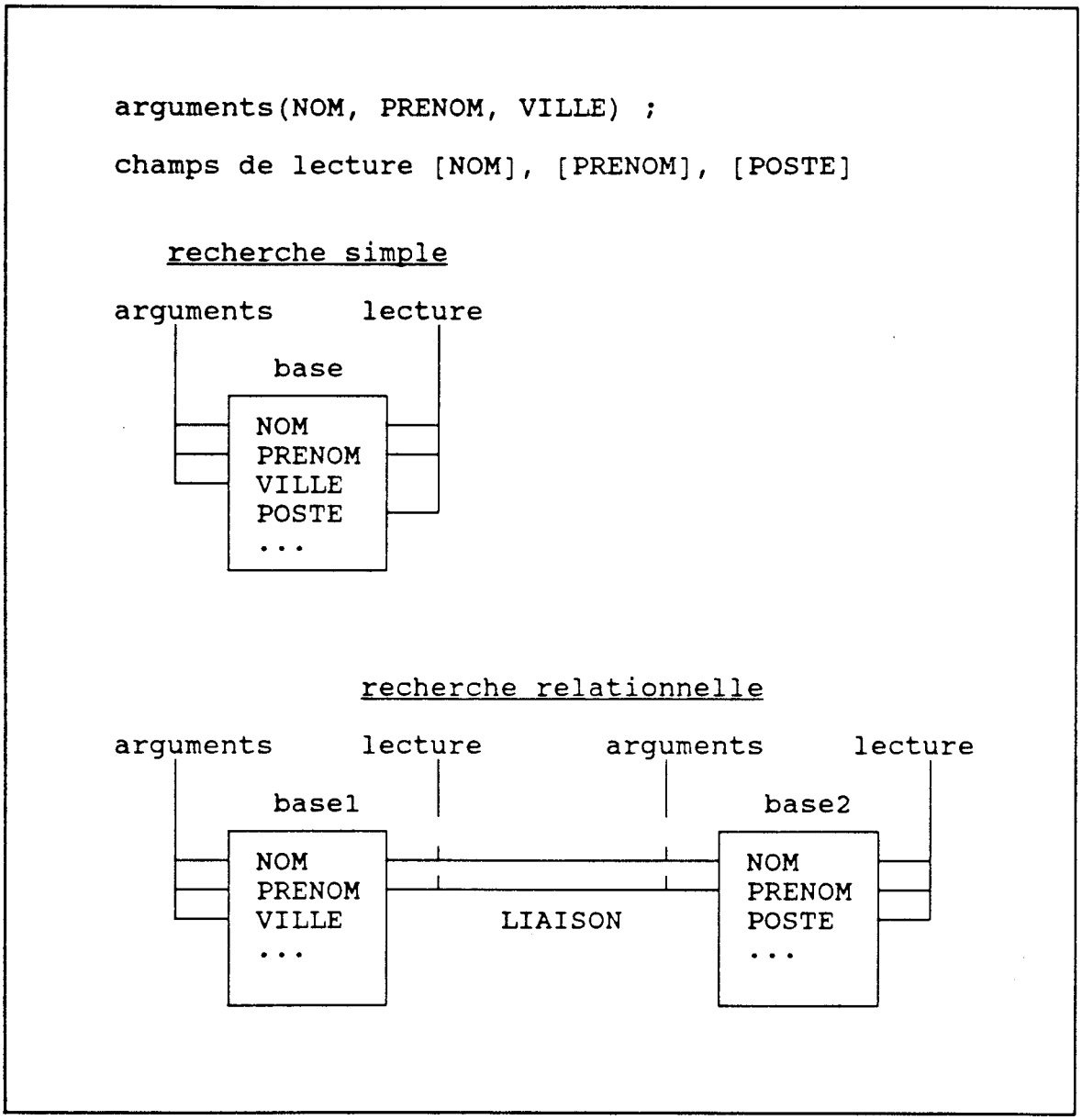


Figure 5.2. *Comparaison des types de recherche.*

5.3.2. Fonction de comparaison.

Toutes les recherches ont besoin d'une fonction de comparaison de chaînes de caractères. Celle qui a été développée pour notre serveur a quelques caractéristiques particulières, et nous la retrouverons dans plusieurs fonctions de notre programme. Cette fonction a comme entrées les deux chaînes à comparer, plus un paramètre numérique qui

autorise un certain nombre de différences (ou d'erreurs). Bien entendu, si ce paramètre est à zéro, la comparaison est totale, et, pour des valeurs supérieures, il va nous permettre une recherche plus souple, notamment en ce qui concerne les noms propres, où l'orthographe est quelque fois incertaine. Le retour de cette fonction est le signe de la comparaison (1 ou -1) si le nombre de différences est dépassé, et zéro dans les autres cas. La comparaison, qui opère par mots, ne fait pas de différence entre les caractères majuscules et minuscules, les lettres accentuées sont considérées sans accent, le "ç" comme un "C", et les espaces ne sont pas comptabilisés. Si une erreur est détectée, elle est automatiquement ajoutée, ensuite le système essaie de recaler les mots pour ne pas grossir sans raison le nombre d'erreurs. Cette remise en phase est utile dans le cas de lettres doublées, ou de consonnes muettes à l'intérieur d'un mot (voir exemple figure 5.3). Enfin, pour mener avec succès une recherche aléatoire, il est prévu des caractères "Jokers", un "*" remplace :

- une lettre s'il est à l'intérieur d'un mot,
- la fin d'un mot si aucune lettre ne le suit,
- un mot s'il est précédé et suivi par des espaces,
- la fin de la chaîne s'il est dans la séquence "_*".

Mot de référence en base de données : SELECTION

Mot à comparer	Résultat	Paramètre Erreur max	Erreurs
sélection	0	0	0
*	0	0	0
se*	0	0	0
recherche	-1	0	8
sélction	0	1	1

Figure 5.3. *Comparaison d'un mot.*

5.3.3. Fonctions de tri.

Deux fonctions de tri ont été écrites. L'une range par ordre alphabétique, l'autre range par ordre temporel, deux tables de sélection. Les éléments de ces tables sont des structures qui contiennent les coordonnées des enregistrements (numéros de base de données et d'enregistrement), auxquelles s'ajoute au moins un élément de comparaison, un nombre représentant le temps (date et heure) pour le tri temporel, ou des chaînes de caractères pour le tri alphabétique. Le temps est un nombre long calculé par des fonctions de la bibliothèque C, il représente le nombre de secondes depuis 1900.

Exemples de structures utilisées pour le tri et leurs écritures en langage C :

Base de données	Enregistrement	TEMPS
1	56	2070000001
0	10	2070000322

```
typedef struct selection_tempo
    {
    unsigned short   num_base ;   /* Numéro de la base de données.   */
    unsigned short   num_enr ;    /* Numéro de l'enregistrement.     */
    time_t  val_temps ;           /* valeur du temps date + heure.   */
    } Selection_t ;
```

Base de données	Enregistrement	Chaîne (1)	Chaîne (2)
5	23	XXX	YYY
2	76	AAA	BBB

```
typedef struct selection_alpha
    {
    unsigned short   num_base ;   /* Numéro de la base de données.   */
    unsigned long    num_enr ;    /* Numéro de l'enregistrement.     */
    char             arg1[9] ;    /* Première chaîne de comparaison. */
    char             arg2[9] ;    /* Seconde chaîne de comparaison.  */
    } Selection_Alpha ;
```

L'algorithme de tri rapide, que nous utilisons pour le classement de ces tables, est donné ci-dessous. La fonction échange utilise une structure identique à celle de la table, pour faire le tampon lors des copies. La fonction de comparaison de chaînes de caractères est celle que nous avons étudiée au paragraphe 5.3.2. Elle retourne les valeurs -1, 0 ou 1 en fonction du résultat.

Le tri opère sur trois boucles imbriquées. La première détermine l'écart en le divisant par deux à chaque passage (dichotomie). La deuxième parcourt tous les éléments entre la valeur de l'écart et le dernier élément. La troisième compare les élément inférieurs à l'élément parcouru par la boucle précédente, en suivant des intervalles équivalents à l'écart, et elle les échange s'ils ne sont pas dans l'ordre.

Afin de clarifier l'écriture de cet algorithme, nous allons utiliser des indicateurs : écart pour la première boucle, i pour la deuxième et j pour la troisième.

TRI (une table de n éléments)

```
Pour (un écart = n/2) jusqu'à (un écart > 0)
        Pour (i = écart jusqu'à i < n )
                Pour (j = i - écart) jusqu'à ( j >= 0 et Si (Comparaison(des l'éléments de
                                                        rangs [j] et [j + écart]) > 0))

                        Echange     (élément j -> tampon,
                                     élément j + écart -> élément j,
                                     tampon -> élément j + écart) ;

                        j = j - écart ;
                Fin Pour
                i = i + 1 ;
        Fin Pour
        écart = écart / 2 ;
Fin Pour
```

6. Mise en œuvre des messages.

6.1. Définition des messages.

Dans ce chapitre nous allons traiter une des parties les plus importantes de notre projet, puisqu'elle est directement en relation avec les applications des utilisateurs. La constitution des messages passe par une phase préalable, où l'on va définir le format du message qui sera écouté. Les formats de sortie concernent les messages qui contiennent des recherches de champs dans les bases données. Les autres fonctions, comme la prise au clavier pour une écoute immédiate, sont des commandes internes et fixes. La définition du format doit tenir compte du contenu des champs des bases de données, afin de prévoir des conversions nécessaires. La création ou la modification de ces messages peut se faire à tout moment, sous la forme d'une commande, ou dans un fichier (de configuration notamment).

6.2. Traitement des messages.

6.2.1. Syntaxe d'écriture des formats.

La définition des formats est écrite dans une chaîne de caractères commençant par le mot clé "FORMAT", afin de pouvoir être détectée dans un fichier de configuration ou lors d'une commande, suivi de la description du format proprement dit. Le premier élément du format est le nom (ou titre) suivi de la liste des arguments entre parenthèses, et séparés entre eux par des virgules. Les noms des arguments correspondent aux noms des champs prévus pour la recherche des enregistrements à sélectionner. Il faut donc, que tous les champs passés en arguments soient présents dans la base de données, pour que la sélection soit possible. Le

dernier bloc du format est la constitution du message. Il est écrit entre accolades. Le corps du message est composé de parties fixes écrites directement dans la chaîne de caractères, et de parties variables correspondant aux champs de lecture. Les noms de ces champs sont écrits entre crochets à l'endroit d'intervention de leurs contenus. Le message est découpé en zones, afin de rendre l'énoncé plus proche de la réalité, et d'éliminer les répétitions inutiles. Ainsi chaque zone est influencée par les contenus des champs qui la compose. Par exemple, si les contenus des champs sont identiques à ceux des champs précédents, la partie du message défini dans cette zone n' est pas prononcée. De la même façon, les zones, qui ne contiennent pas de champ de lecture, ne sont dites qu'une seule fois, lors du premier passage. Le caractère choisi pour la séparation des zones est la barre verticale (code ASCII 124).

Modèle d'écriture des formats :

FORMAT nom (Argument1, argument2, ...) { texte fixe [champ de lecture 1] texte fixe | texte fixe [champ de lecture 2] ... | ... }

6.2.2. Adaptation des messages et conversion des champs.

Les éléments qui sont passés à COMPOST doivent être des objets prévus dans les classes du fichier "objets.cps". Les notions d'objets de classes et arborescence ont été décrites dans le chapitre 4 (traitement de la parole). L'écriture est de la forme suivante :

nom de l'objet de classe "mot" (suite d'objets de classe "graph" ou "phon")

exemples : Det(LE), Aux(EST), Num(1991), Det(lx^) ...

Dans certains cas, la classe "mot" peut être remplacée par une classe spéciale en vue d'une conversion sous COMPOST.

exemple : Tel(76570000)

Afin de proposer un large éventail de possibilités, nous avons prévu de faire les conversions, soit par l'intermédiaire de fonctions préétablies (écrites en langage C), soit par le transport vers COMPOST du contenu des champs ou de texte fixe sous la forme d'un objet

spécifique, puis de faire directement le traitement par des règles. Parmi les fonctions, nous disposons des conversions de texte libre en mots de classe "Inc" et "Num", ou en mots de classe "Inc" uniquement, auxquelles il faut ajouter les conversions des champs date et heure en partie de message.

Comme il faut différencier, dans la définition du message, les parties de texte fixe et les champs, nous avons institué des syntaxes d'écritures différentes.

Exemples de modèles d'écritures pour le texte fixe :

- En objets COMPOST de classe orthographique "graph".

  Det(LE) Nom(N°) Inc(DE) Nom(RUE)

(Les lettres minuscules sont converties en majuscules).

- En objets COMPOST de classe phonétique "Phon"

  Det(@ lx^) Nom(@ nymero) Inc(@ dx^) Nom(@ ry)

(le "@" indique que la chaîne qui suit ne subit pas de conversion).

- En portion de phrase convertie en mots majuscules (Inc) et numériques (Num).

  MN(le N° de rue)

(La séquence "MN", devant le texte parenthésé, indique la conversion)
Le résultat de la conversion est : **Inc(LE) Inc(N°) Inc(DE) Inc(RUE)**.

Sachant que la traduction orthographique phonétique de notre application n'utilise pas de dictionnaire très complet (faute de place mémoire), il est important de veiller dans le dernier cas à ne pas introduire d'ambiguïté, notamment dans la distinction nom et verbe (Nom(COUVENT) et Vrb(COUVENT)). Les écritures en classe "mot" différentes de "Inc" (inconnues) donnent des traductions plus sûres. Dans les parties de texte fixe, les trois types d'écritures peuvent être utilisés mais à l'intérieur d'un mot une seule classe peut être choisie.

syntaxes d'écriture pour les conversions des champs :

[champ_x] La conversion par défaut est "majuscule, numérique". Chaque mot du champ devient "Inc" ou "Num".

Xxx(MN[champ]) Même conversion que ci-dessus mais explicitée. La notation Xxx peut être un mot mnémonique qui ne sera pas utilisé, "MN" est l'identificateur de conversion (majuscule, numérique).

Npr(MC[nom]) Conversion en majuscules puis transmission dans un objet de classe mot. "Npr" est l'objet nom propre, MC est l'indentificateur de conversion majuscule.

Xxx(D[date]) Conversion de la date "AAAAMMJJ" en séquence "Det(LE) Num(JJ) Nom(JANVIER ...) Num(AAAA)". Le caractère D devant le champ indique la conversion.

Xxx(H[HEURE]) Conversion de l'heure "HH:MM" en séquence "Num(HH) Nom(HEURE) Num(MM)", (Indentificateur: H).

Tel([tel]) Pas de conversion locale mais récupération du champ sous une classe reconnue par COMPOST.

Si nous reprenons le dernier exemple, le remplacement du nom du champ par son contenu peut donner la chaîne suivante: **Tel(76570000).** Transmise à COMPOST puis modifiée par une grammaire de conversion, cette chaîne se transformera en portion de message de la forme :

Num(76) Inc(|) Num(57) Inc(|) Num(00) Inc(|) Num(00)

Les numéros sont groupés par deux et une pause ou un arrêt sont intercalés entre les prononciations pour mieux comprendre le contenu.

Après avoir vu comment composer des formats de message, nous pouvons donner des exemples d'écriture :

- Lecture des numéros de téléphone dans une base de données contenant au moins les champs nom, prénom, et tel_tra (numéro de téléphone du lieu de travail).

Format telephone(nom, prenom) { Nom(NUME6ROS) Pcn(DE) Nom(TE6LE6PHONE) | Nom(CELUI) Pcn(DE) [prenom] [nom] Vrb(EST) Tel([tel_tra]) }

Ce message est constitué de deux zones, la première ne comporte pas de champ, alors que la deuxième a deux champs vers une conversion majuscules/numérique (nom, prénom) et un champ à transfert direct vers COMPOST (tel_tra). La première zone est un générique et elle ne sera prononcée qu'une seule fois. On notera aussi que les caractères accentués sont suivis par un chiffre (6 : pour l'accent aigu) afin de les différencier dans la classe "graph".

- Une variante de ce message peut être écrite, plus simplement, sur une seule zone :

Format telephone(nom, prenom) { MN(le N° de téléphone de) [prenom][nom] Vrb(est) Tel(tel_tra) }

Nous voyons ici une conversion sur du texte fixe du début du message. Le verbe "est" garde la classe "Vrb" pour ne pas être confondu avec le nom "EST" (point cardinal).

6.3. Processus de traitement des formats.

Nous allons décrire ici l'ordonnancement des fonctions qui construisent les messages. La première d'entre elles est la détection et le chargement du format, le mot clé "format" est détecté, puis la chaîne de description est chargée en mémoire et le titre est repéré. Ensuite, à chaque passage de commande le nom est comparé aux titres des formats. Si une comparaison est effective le format est déclaré actif, et le restera jusqu'à la fin de la commande. Au début de chaque détection de format, il y a initialisation des paramètres de recherche et de sortie tels que : les noms des arguments, les noms des champs de lecture, le positionnement des zones, etc. Cette initialisation prépare les recherches dans les bases de données et la segmentation du message. Lorsque la sélection et le tri des enregistrements sont effectués, les messages sont constitués par les résultats des parcours du format actif. A chaque accès à la constitution des messages, une zone du format est parcourue et testée. Si le contenu de ses champs (non vides et non répétés) ou si sa position temporelle (premier passage) sont pertinents, alors la mise en forme du message peut être effectuée.
Nous pouvons résumer cette analyse avec un algorithme à deux boucles, dont la première englobe les enregistrements sélectionnés, et la seconde les zones du format.

Algorithme de sortie des messages :

Pour (chaque enregistrement)
    Pour (chaque zone)
        Si (la pertinence du message est bonne)

            Constitution du message ;

        Fin Si
    Fin pour
Fin pour

Nous poursuivons plus en détail l'analyse de la lecture du texte qui constitue le message. L'identification des éléments du texte est faite par des caractères fixés et ne devant servir uniquement qu'à cet usage ({, }, (,), [,], |). Pour cette fonction, nous utiliserons aussi une présentation sous forme d'algorithme, montrant le mécanisme de transformation vers une phrase compréhensible par COMPOST.

Algorithme de constitution de message :

Début de phrase

Positionnement en début de zone ;

Tant que (la fin de zone n'est pas détectée)

Tant que (la position d'un champ n'est pas détectée : '[')
Lecture de la classe (avant parenthèse ouverte) ;
Lecture du mot (entre parenthèses) ;
Si (la classe et le mot sont complets)
Si (la classe n'est pas un indicateur de conversion)
Ajout au message (classe et mot) ;
Sinon
Conversion puis Ajout au message ;
Fin Si
Fin Si
Fin Tant que

Si (la classe et le mot sont encore actifs (non utilisés dans la boucle précédente))
Si (le mot devant le nom de champ est un indicateur de conversion)
Convertir le champ et l'**Ajouter** au message ;
Sinon
Ajout au message (classe et champ) ;
Fin Si
Sinon
Conversion (par défaut: majuscule, numérique) et Ajout au message ;
Fin si

Fin Tant que

Fin de phrase

6.4. Passage des paramètres au logiciel COMPOST.

Les objets COMPOST peuvent être créés soit par des fonctions de création d'instances, soit par le passage d'une chaîne de caractères. Nous détaillons ici, la deuxième possibilité qui donne des facilités de contrôle plus évidentes. Cette chaîne est constituée par des ajouts successifs de mots dominés par des classes, et répondant aux consignes établies dans le format qui a été sélectionné. Ce lien principal, entre les fonctions propres du serveur et COMPOST qui prend le relais pour la transformation en signal de parole, montre l'indépendance des deux modules.

6.5. Evolutions des commandes et des formats.

La définition de messages complexes demande un plus large éventail de possibilités de tests et de contrôle. La première approche peut être une augmentation du dialogue avec la machine, lors de messages sur les états de fonctionnement. Nous pouvons avoir des indications sur le nombre d'enregistrements sélectionnés, ou des messages indiquant les opérations pour la suite des commandes. Les autres évolutions sont la dépendance des messages en fonction de la valeur des arguments ou les réponses alternatives lorsque les champs sont vides.

6.5.1. Messages de contrôle.

Le dialogue direct avec les états de fonctionnement se fait par des messages paramètrables au niveau du "scénario" COMPOST. Des objets spéciaux sont définis dans une classe de contrôle. Ces noms sont connus par le programme principal, et ils ne peuvent pas être changés facilement, alors que le contenu de la transformation en message est entièrement libre à l'utilisateur.

Exemples d'écriture et de transformation :

```
classe contrôle
        objet(Vide,Dep...)
finclasse
```

passage des paramètres à COMPOST : Vide() ou Dep(),

Règles de transformation au début du scénario :

grammaire messages

vide = Vide() -> Inc(PAS) Pcn(DE) Nom(MESSAGE) ;

dep = Dep() -> Inc(TROP) Inc(D'ENREGISTREMENTS) Adq(SE6LECTIONNE6S)
Vrg(,) Vrb(PRE6CISEZ) Det(LA) Nom(COMMANDE) Inc(!) ;

fingrammaire

6.5.2. Messages conditionnels sur les arguments.

La présentation des messages, notamment dans la partie générique, peut amener une compréhension plus efficace du contenu. Pour modifier la phrase, dans le traitement de la parole COMPOST, nous passons les valeurs des arguments dans des classes d'objets arguments. Ces objets "Arg_" sont rappelés dans la définition du format au niveau de leur intervention (par zones). Ce mode d'écriture est très lié à une application donnée, il faut concevoir le message dans sa globalité (définition du format, test sur les arguments, règles de transformation, etc.). Nous donnons dans la suite, un exemple de modification de la partie générique d'un message, en fonction de la valeur du premier argument d'une commande. Ce message concerne la sortie des références des personnes de l'institut.

1) Définition du format :

FORMAT ref(nom,prenom) { Nom(Références) Arg1() Nom(l'institut) |
[prenom][nom][reference] }

2) Déclaration des classes :

classe arg
objet(Arg1,Arg2 ...)
finclasse

3) Ecriture des règles :

grammaire argument

```
arg1_1  =  Arg1(*)          ->  Pcn(DE) Inc(TOUTES) Det(LES)
                                Nom(PERSONNES) Pcn(DE) ;

arg1_2  =  Arg1(graph*[1,20] *)  ->  Pcn(DE) Inc(TOUTES) Det(LES)
                Nom(PERSONNES) Inc(DONT) Det(LE) Nom(NOM)
                Vrb(COMMENCE) Inc(PAR) Inc(#2) Pcn(DE) ;

arg1_3  =  Arg1(#A)         ->  Pcn(DE) Nom(PERSONNE) Pcn(DE) ;
```

fingrammaire

Les modifications effectuées par ces règles apportent trois types de messages, elles dépendent des recherches : totales (avec l'argument "*"), semi-sélectives (avec un début suivi de "*") ou entièrement sélectives si le caractère "*" n'est pas présent). Nous résumons ci-dessous le contenu des messages par des exemples d'arguments.

ref * , *

Référence **de toutes les personnes de** l'institut |
[prenom][nom][reference] | [prenom][nom][reference] | ...

ref D* , *

Référence **de toutes les personnes dont le nom commence par D à** l'institut |
[prenom] D... [reference] | [prenom] D... [reference] | ...

ref Dupont , *

Référence **de personnes de** l'institut |
[prenom] Dupont [reference] | ...

6.5.3. Réponses alternatives sur les champs.

Nous avons vu que si le champs était vide la partie du message qui le concerne (zone), n'est pas pertinente. Cependant, dans certains cas, il est intéressant d'avoir une indication sur le contenu des champs. Par exemple, si le champ est laissé volontairement vide,

nous pouvons le charger d'un message. Dans d'autres utilisations, où un champ vide est dans la même zone que d'autres champs, il faut préserver la structure de la phrase en remplissant par des informations fixes. Nous allons définir ces alternatives, dans le format, par une syntaxe appropriée. l'alternative est limitée par les signes "<" et ">", Le nom du champ est toujours entre crochets, et séparé de la partie de remplacement par une virgule.

Exemple d'écriture dans un format :

Format réunion(nom,prenom,date)
{ Det(le) [date] (rencontre avec) [personne] **<[lieu], MN(le lieu n'est pas précisé) >** }

Si le champ [lieu] est vide la phrase "le lieu n'est pas précisé" remplace le contenu du champ.

7. Interface utilisateur.

7.1. Analyse de la fonction.

Les solutions adoptées, pour l'interface utilisateur, répondent à des critères de maintenance et de flexibilité. Notamment son rôle principal, qui est l'habillage de notre application, est complètement paramétrable par un fichier de configuration. Les informations présentes sur l'écran permettent un accès direct aux principales fonctions par deux lignes de menus, et l'acquisition des arguments est faite dans des fenêtres à l'aide d'un mini-éditeur. La liaison avec le reste de l'application doit être la plus indépendante possible, le serveur doit donc être capable de fonctionner sans interface utilisateur. L'affichage à l'écran doit être compatible avec plusieurs ordinateurs, et être facilement adapté à d'autres systèmes. Pour cela nous avons choisi la norme ANSI. Cette norme définit des caractères d'échappements, qui envoyés sur le flux de sortie, fixent les attributs de l'affichage et la position du curseur. La liste des commandes ANSI est donnée en ANNEXE 3, ainsi que des exemples de son utilisation en langage C.

7.2. Création des menus, des fenêtres, et de l'aide.

Les éléments que nous pouvons voir à l'écran sont les deux lignes de menus, les fenêtres d'édition des arguments, et les fenêtres d'information ou d'aide. A cela, il faut ajouter la fenêtre de logo qui s'affiche lorsque le système est au repos (attente), et deux lignes supplémentaires : une ligne d'états facultative et une ligne de commandes directes.
Les lignes de menus, appelées menus hauts et menus bas, offrent la possibilité de séparer deux types de fonctionnalités bien distinctes. Nous avons d'une part, les utilitaires comme les appels des fenêtres "AIDE", les définitions de formats temporaires, le passage de chaîne pour

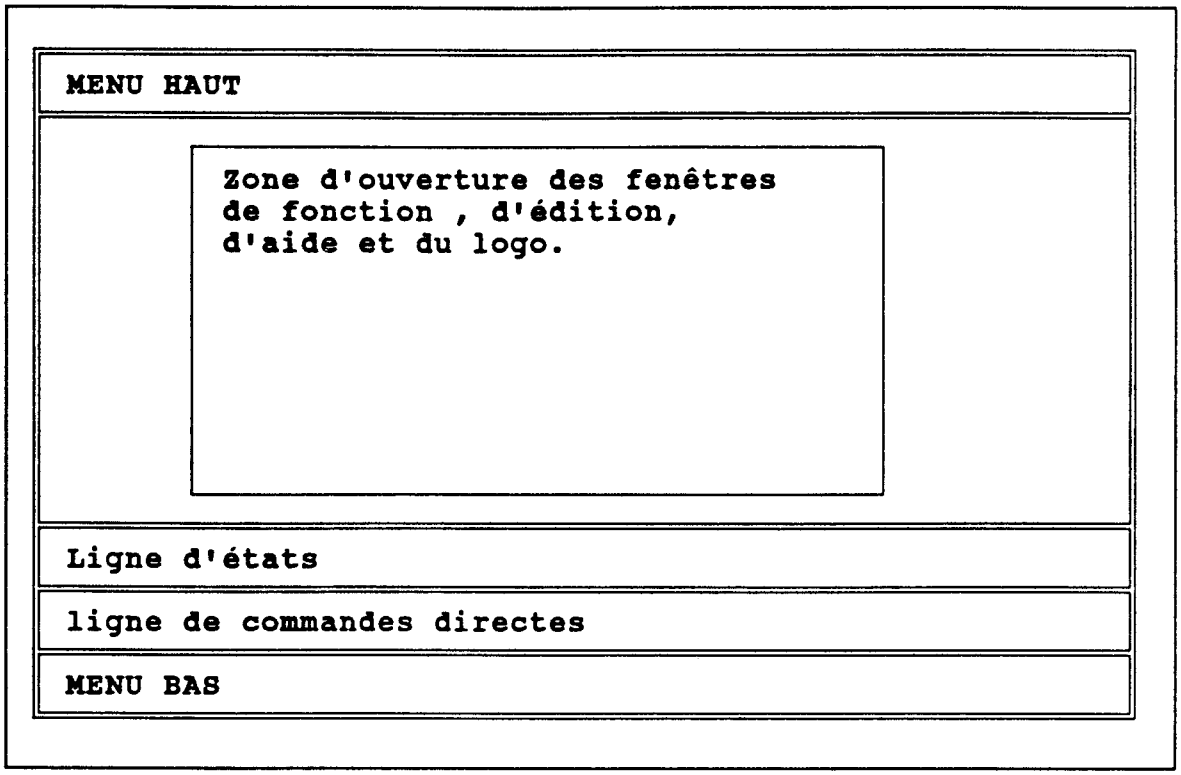


Figure 7.1. *Disposition initiale de l'écran.*

le logiciel COMPOST, la sortie de l'application, etc. D'autre part, nous avons les fonctions directement concernées par le serveur, c'est à dire celles qui vont générer des messages suivant les formats, et les recherches dans les bases de données. La figure 7.1 présente une disposition possible de tous ces éléments sur l'écran. Chaque élément d'un menu est constitué de la façon suivante : le début représente la touche de fonction associée, ensuite le corps indique le titre ou le mnémonique correspondant. Un des caractères du corps sera choisi pour la sélection, son affichage aura une couleur ou une apparence différente.

exemple:

F1-Aide

- Touche de fonction : F1.
- Corps : Aide.
- Caractère de Sélection : A.

Chaque appel de fonction, décrite dans les barres de menus, peut donner lieu à l'ouverture d'une fenêtre, pour acquérir les arguments qui seront transmis à l'application. Cette fenêtre se compose d'une partie fixe, qui aidera à présenter la fonction et à commenter les arguments. Si cette fenêtre n'est pas suffisante pour décrire la fonction, ou s'il n'est pas nécessaire d'alourdir la présentation, il est possible de définir une fenêtre d'aide qui est appelée par la fonction locale "AIDE". Cette fenêtre donne des indications sur la fonction du menu courant, ou une aide générale si aucun menu n'est actif.

7.3. Interprétation des commandes.

Au niveau fonctionnel, l'interface utilisateur se comporte comme un système en état de veille. La gestion du clavier aiguille les informations vers des traitements locaux, ou vers les fonctions de retour qui génèrent la sortie d'une ligne de commande. Le schéma bloc de l'interpréteur de commandes est représenté en figure 7.2, nous pouvons remarquer que l'appel de la fonction arrive sur les traitements locaux puisqu'il y a toujours initialisation ou restitution de l'écran.

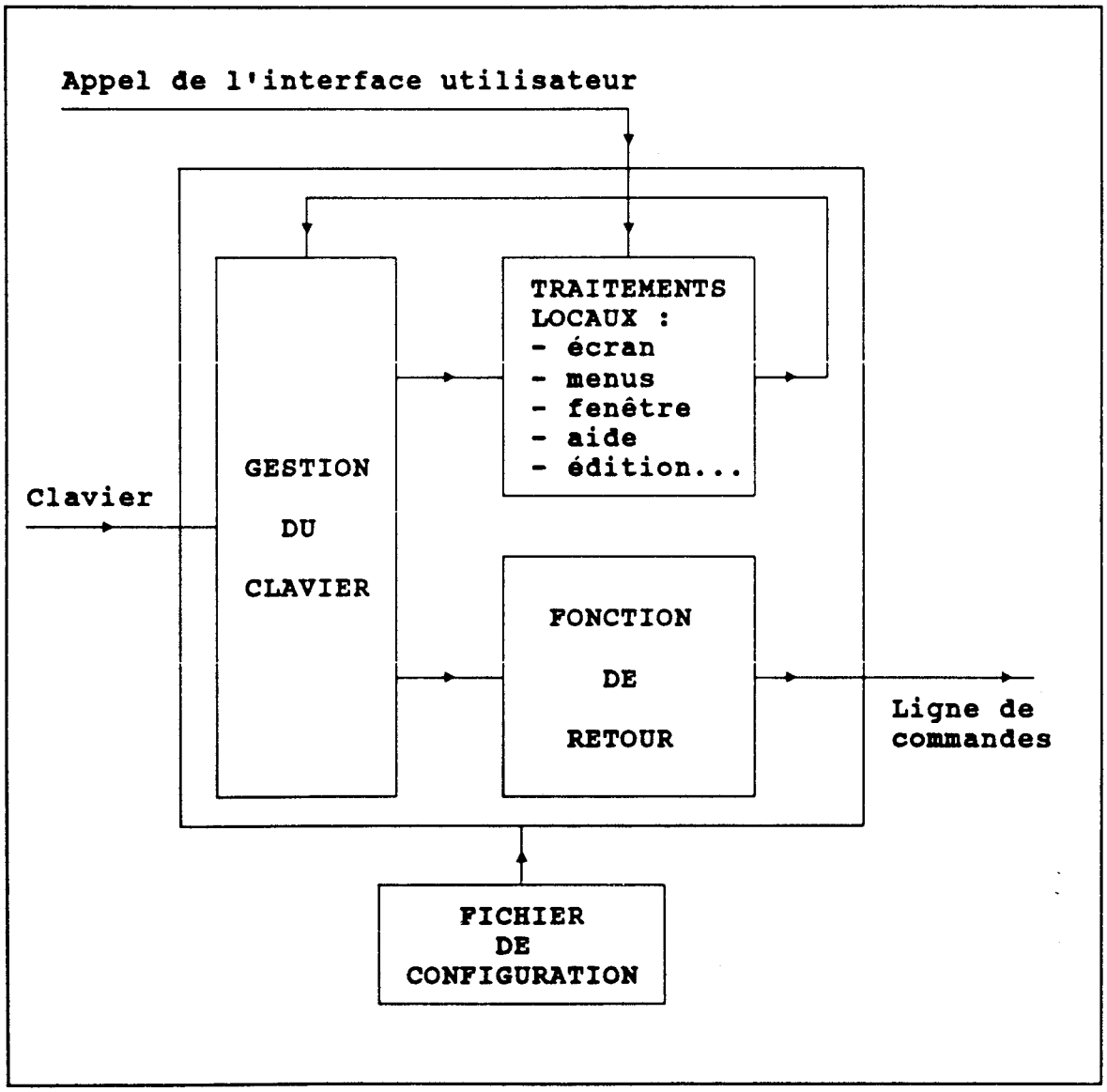


Figure 7.2. *Interpréteur de commandes.*

7.3.1. Indépendance de la fonction.

Le seul lien qui existe entre l'interface utilisateur et le reste de l'application est une chaîne de caractères que nous appellerons aussi ligne de commande ; elle émane de l'entrée clavier, ou de la lecture d'un fichier de commandes. Pour construire ces commandes, la fonction séquence de l'interface utilisateur, lorsqu'elle est appelée, fait la concaténation du nom de la fonction suivi par les arguments séparés par des espaces et des virgules. Nous rappelons ci-dessous le format d'une commande :

nom_de_fonction argument1 , argument2 , ...

7.4. Paramétrage des éléments par un fichier de configuration.

Les paramètres prévus dans le fichier de configuration définissent tous les éléments accessibles depuis l'écran, mais aussi des paramètres du serveur comme les formats de message (ANNEXE 4). L'écriture des valeurs des paramètres est spécifique à chaque type, la description d'une barre de menus étant totalement différente de la définition de l'aide. La détection d'une "commande" se fait par l'intermédiaire de mots clé, et l'ordre d'écriture n'est imposé que dans la définition des menus, elle doit être écrite avant celles des fenêtres et des aides, afin de créer les liaisons avec les noms de fonction. Les paramètres manipulés dans ce fichier sont des positions, des dimensions, des couleurs, des chaînes de caractères, et des noms de variables ou de fonctions. Les chaînes de caractères sont écrites entre des guillemets puisqu'elles peuvent comporter des espaces, alors que les couleurs et les paramètres numériques sont séparés par des espaces et des virgules. Des commentaires peuvent être introduits entre les définitions, ils doivent commencer et finir par un "#". Les mots clé, détectés par la lecture du fichier de configuration, sont les suivants : ECRAN, LOGO, MENU_HAUT, MENU_BAS, LIGNE_COMMANDES, LIGNE_ETATS, FENETRE, AIDE et FORMAT. Ils sont définis ci-dessous :

"ECRAN" est suivi des couleurs de :

- fond de l'écran,
- fond du logo, caractères du logo,
- fond des fenêtres, caractères des fenêtres,
- fond des aides et caractères des aides.

Exemple : **ECRAN bleu, noir, rouge, ...**

"LOGO" introduit d'abord 4 nombres : les positions initiales (ligne et colonne), et les nombres de lignes et de colonnes. Ensuite viennent les chaînes de carractères qui doivent correspondre au nombre de lignes, dont la longueur ne doit pas dépasser le nombre de colonnes.

Exemple : **LOGO 4, 10, 18, 60**
" chaîne de desciption du logo "
...
suite de 18 chaînes de 60 caractères.

Les "MENU_HAUT" et "MENU_BAS" sont définis par deux parties. La première partie fixe la présentation en indiquant : le nombre d'items, la ligne d'écran qu'ils occupent, puis les couleurs du fond, des caractères de texte, des touches de fonction et des caratères de sélection. La deuxième partie décrit le contenu des menus. Elles est composée de séquences correspondant au nombre d'items. Chaque séquence comprend : le nom de fonction, la chaîne affichée à l'écran, et la chaîne d'affichage de la touche de fonction. Le nom de fonction est le lien, qui fera communiquer le menu, les fenêtres, l'aide, les formats, et le passage des commandes au serveur. La chaîne d'affichage écran est totalement libre à l'utilisateur, il sera donc possible de configurer un écran même en langue étrangère. Les items peuvent être activés par :

- Un caractère de sélection, précédé par une "{" dans la chaîne d'affichage,
- Une touche de fonction.

Nous avons donc les possibilités, "F1" à "F10" pour les accès directs, et "AF1" à "AF10", "CF1" à "CF10" et "SF1" à "SF10" pour les accès à deux touches avec "Alt", "Ctrl" et "Shift".

Exemple : **MENU_HAUT 5, 1, blanc, noir, rouge, bleu**
aide "-{Aide " "F1"
quitte "-{Quit " "F3"
format "-For{mat " "SF4"
agenda "-Agen{da " "AF6"
sequence "-Dé{part " "CF2"

"LIGNE_COMMANDES" positionne la ligne des commandes directes, et fixe les couleurs du fond, et des caractères de l'invite et du texte ; l'invite est écrite dans une chaîne précédée du mot "prompt".

La ligne d'états montre des indicateurs de fonctionnement, comme le nombre d'enregistrements sélectionnés ou le mode d'écoute (pause), les paramètres qui suivent le mot clé "LIGNE_ETATS" sont le nombre d'items, la position de la ligne, la couleur du fond, des caractères des items et réponses. La description des différents items de la ligne correspond au format suivant : nom de la variable, chaîne d'affichage associée à la variable, chaîne de la valeur positive (ex: "oui") et chaîne de la valeur négative (ex: "non"). Pour d'autres indications l'affichage est fait en valeurs numériques.

Exemple : **LIGNE_ETATS 0, 23, bleu, blanc, noir**
pause " PAUSE " "Oui" "Non"
nb_sélection " ENR = "

Les fenêtres de capture des arguments "FENETRE" sont décrites par deux parties. La première est caractérisée par :

- Le nom de la fonction associée, ou du format de message,
- Le positionnement et la dimension des fenêtres (ligne d'origine, colonne d'origine puis le nombre de lignes et de colonnes).

La deuxième partie décrit le contenu des fenêtres par des chaînes de caractères. Les lignes qui ne génèrent pas d'arguments, commencent par le caractère "(". Elles servent à la présentation des fenêtres à l'écran. Les autres sont décrites par un nom et une chaîne d'affichage pouvant rappeler le nom de l'argument. Les noms ne sont présents qu'à titre indicatif, en fait les arguments doivent êtres définis dans le même ordre d'apparition que dans la description des formats.

Exemple :

```
FENETRE adresse  10,  10,  7,  48
   (   " COORDONNEES DES MEMBRES DU CLUB DE VOL A VOILE "
   (   " ------------------------------------------- "
   (   ""
   nom      " Nom            : "
   (        ""
   prenom   " Prénom         : "
   (        ""
```

La définition de l'aide "AIDE" est la même que celle des fenêtres (position de l'origine puis nombre de lignes et de colonnes), les chaînes de caractères sont définies par lignes. Elles doivent correspondre au nombre de lignes, et être d'une longueur égale au nombre de colonnes. Contrairement aux autres paramètres, les aides ne sont pas chargées en mémoire lors de l'initialisation, mais sont identifiées par leurs positions dans le fichier.

Enfin, la description des formats est identifiée par le mot clé "FORMAT", et écrite suivant la syntaxe développée au chapitre 6. Si le format doit correspondre à l'appel par menu et par fenêtre, le nom doit être rappelé, dans les autres cas les formats sont aussi enregistrés, et leurs appels sont effectués par la ligne de commande.

Pour illustrer la correspondance entre les menus, les fenêtres, l'aide et les formats nous allons montrer un exemple de description d'un message de numéros de téléphone dans un fichier de configuration.

Exemple d'éléments d'un fichier :

```
MENU_HAUT   4,  1,  blanc,  noir,  rouge,  bleu

    aide                "-{Aide  "          "F1"
    quitte              "-{Quit  "          "F3"
    format              "-For{mat  "        "F4"
    sequence            "-{Envoi  "         "F5"


MENU_BAS   3,  25,  blanc,  noir,  rouge,  bleu

    agenda              "-Agen{da  "        "F6"
    telephone           "-{Tel  "           "F7"
    references          "-{Ref  "           "F8"


LIGNE_COMMANDES  24,  cyan,  noir,  noir

prompt  ">>>>"


FENETRE telephone  10,  10,  7,  42

   (        "  NUMEROS DE TELEPHONE ET DE TELECOPIEUR  "
   (        " ---------------------------------------- "
   (        ""
   nom      " Nom            : "
   (        ""
   prenom   " Prénom         : "
   (        ""


AIDE    telephone      6,  12,  9,  51,

"╔═══════════════════════════════════════════════╗"
"║  AIDE  générale sur l'utilisation de téléphone ║"
"║  ---------------------------------------      ║"
"║                                               ║"
"║ * A l'écoute les numéros sont groupés par deux ║"
"║                                               ║"
"║ * Pour un arrêt entre les groupes taper "pause"║"
"║                                               ║"
"╚═══════════════════════════════════════════════╝"


FORMAT  telephone(nom, prenom) { Det(LE) Nom(NUME6RO)
Pcn(DE) Nom(TE6LE6PHONE) Pcn(DE) [prenom][nom] Vrb(est)
Tel([tel_tra]) }
```

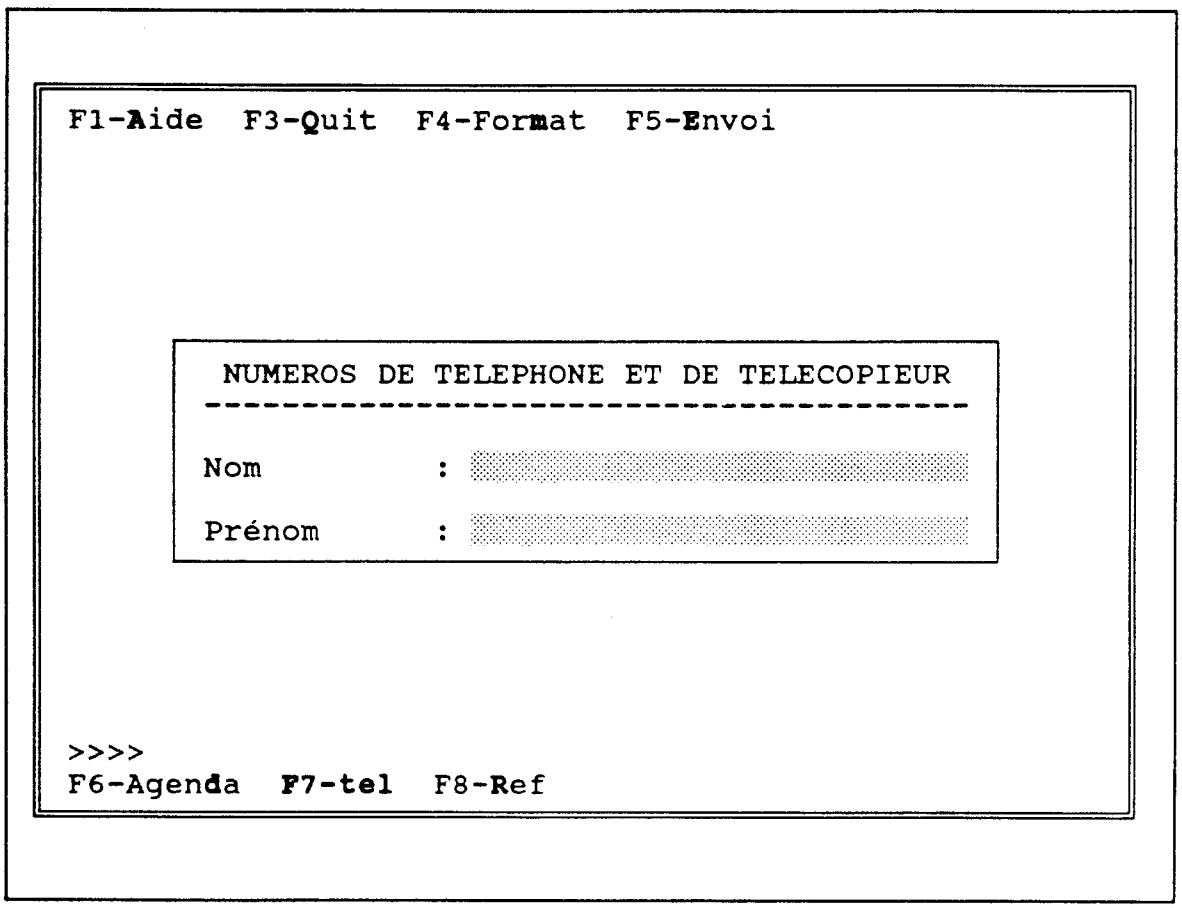


Figure 7.3. *Ecran défini dans le fichier de configuration.*

Les éléments du fichier de configuration donnent l'écran de la figure 7.3. Le curseur ne se positionne que sur les parties grisées, en face des noms d'arguments, où l'on peut saisir des arguments avec le mini éditeur. La ligne de commande qui en découle est de la forme :

telephone argument nom , argument prenom

Pour avoir les numéros de téléphone de toutes les personnes figurant dans les bases de données, il faut utiliser les caractères joker "*", d'où la syntaxe :

telephone * , *

8. Organisation complète du serveur de messages à réponse vocale.

Dans ce chapitre, une vision globale du serveur nous montre l'architecture et le fonctionnement. Les différents aspects que nous avons abordés dans les chapitres précédents (synthèse et traitement de la parole, bases de données, création de messages) sont regroupés dans la présentation d'un produit complet.

8.1. Architecture.

Les liens, qui existent entre les différentes parties et l'environnement extérieur, mettent en évidence la conception modulaire du programme. Les blocs les plus visibles sont, bien entendu, COMPOST et la gestion du serveur (figure 8.1).

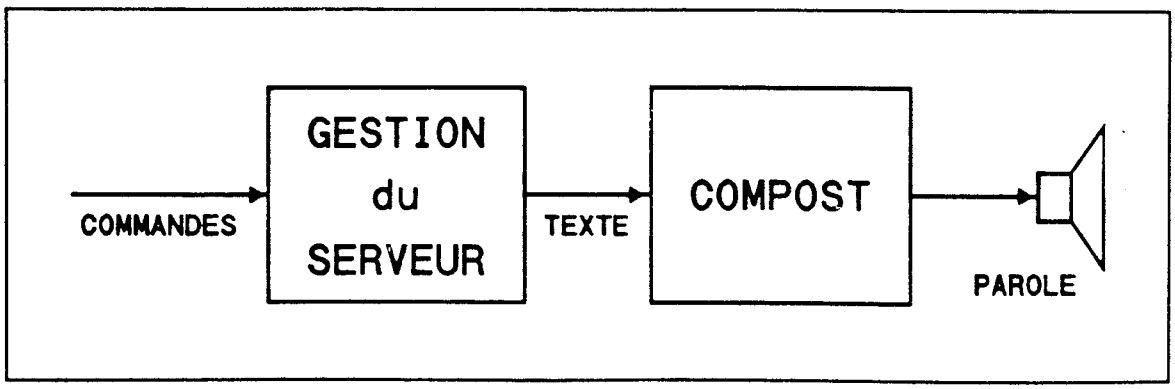


Figure 8.1. *Serveur de message à réponse vocale.*

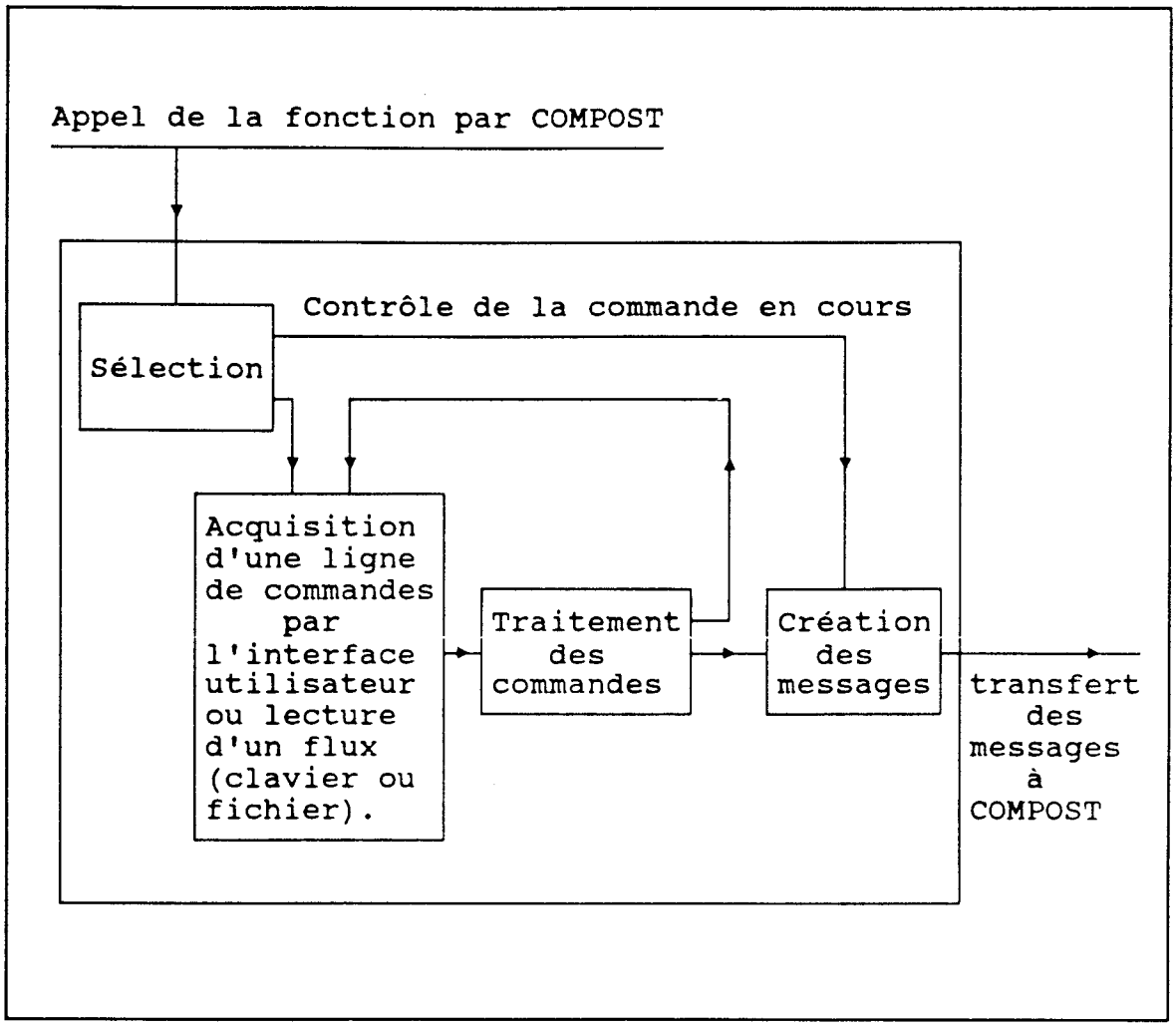


Figure 8.2. *Liaison des principaux modules.*

Le serveur est constitué des modules principaux, tels que la lecture des bases de données, le paramétrage des messages par formats, ou la création des messages. Ces modules font appel à des fonctions (de comparaisons, de transformation, de tri, etc.) dont le support et les données les plus fréquents sont des chaînes de caractères. Etant donné que les commandes ne génèrent pas toujours des messages, le programme doit boucler pour éviter les messages vides. Les commandes, qui font des accès aux bases de données, peuvent fournir des quantités importantes de messages. Dans ce cas, l'appel de la fonction par COMPOST n'aboutit pas sur l'acquisition des commandes, mais poursuit celle qui est en cours en générant la suite des messages. Une sélection, à l'entrée de la fonction, aiguille l'appel vers les différentes phases du traitement. La figure 8.2 montre l'architecture globale de la sélection et de la sortie des messages vers COMPOST. L'appel des fonctions et le séquencement des

opérations font apparaître les contraintes logicielles, notamment le fait que le programme appelant est COMPOST et non le serveur. Ce concept oblige nos fonctions à garder un certain nombre de variables et d'indicateurs en mémoire entre chaque appel par COMPOST.

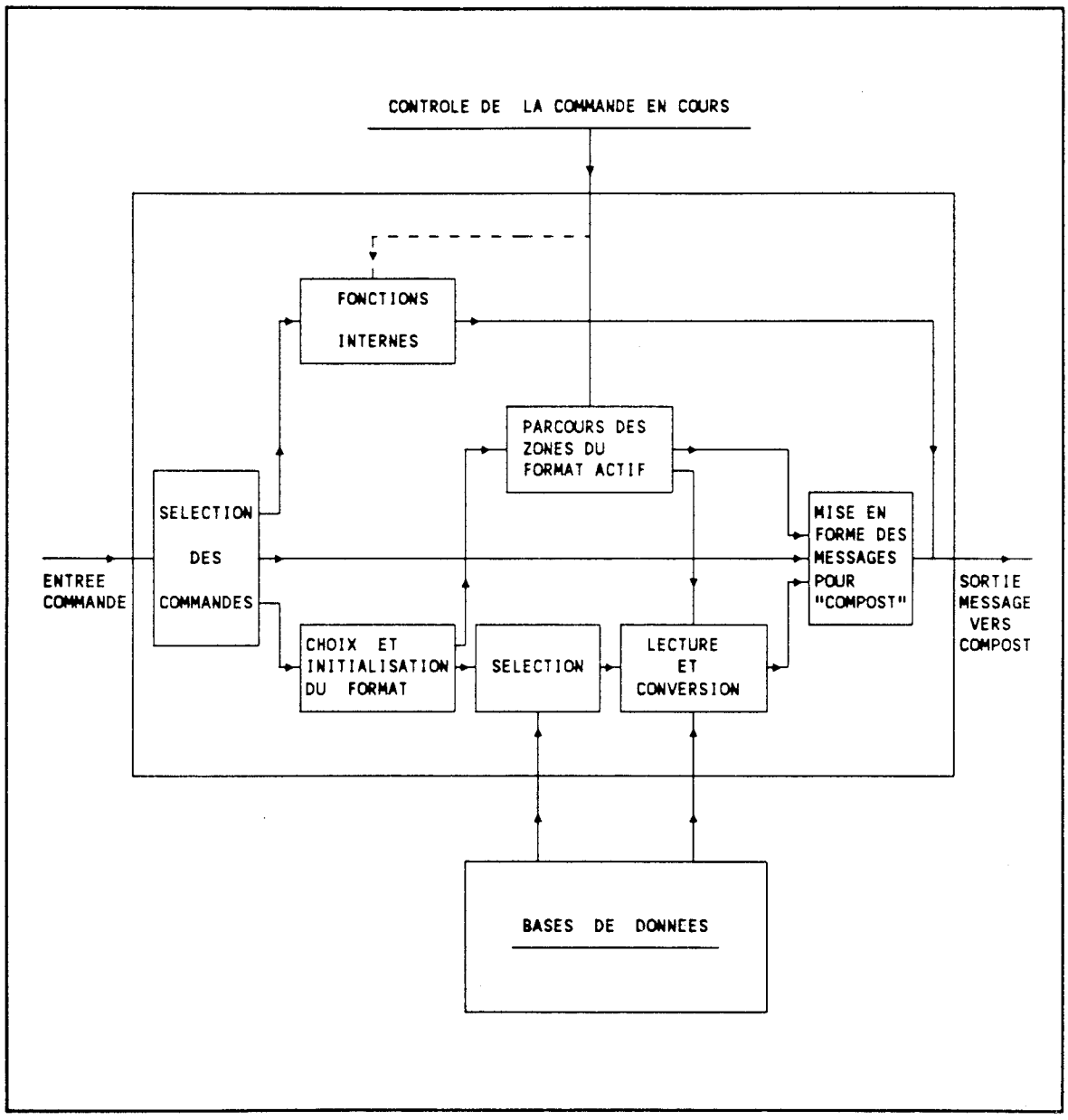


Figure 8.3. *Architecture de la création des messages.*

Le schéma bloc de la figure 8.3 montre le fonctionnement et la structure interne de la création des messages. L'entrée est une chaîne de caractère représentant une commande qui génère obligatoirement une sortie de message. La sortie transfère les messages à

COMPOST également sous la forme d'une chaîne considérée alors, comme une variable globale. Les commandes, sélectionnées en fonction de leur type, alimentent des séquences de traitement spécifiques :

- Les plus simples opèrent un transfert direct, avec une conversion possible de la chaîne passée en argument.
- Les autres ont un fonctionnement plus complexe ; par exemple, la création des messages, suivant un format pré-enregistré, entraîne une phase d'initialisation des paramètres des champs et des zones, ainsi que les recherches et le tri des enregistrements.

Tant que la commande n'est pas terminée (plus de messages), le contrôle de cette fonction permet de créer des messages conformes au format, et au contenu des champs des enregistrements sélectionnés.

Notons enfin, que les fonctions internes ont un fonctionnement spécifique qui ne fait pas obligatoirement appel aux mêmes procédures que les commandes par format. La création des messages est directe ainsi que leur mise en forme pour COMPOST.

8.2. Séquence des opérations.

Nous proposons maintenant, pour illustrer le fonctionnement général, un algorithme qui englobe les appels de fonctions faites par COMPOST, le caractère "→" indique les points d'entrée des fonctions. Comme tout programme informatique, la partie COMPOST et la partie serveur sont constituées par une phase d'initialisation, et par un corps de programme qui boucle, dans la plupart des cas, jusqu'à la fin. Cet algorithme correspond au scénario COMPOST, les fonctions, que nous détaillons, génèrent les messages d'entrée qui seront convertis en parole. Cette conversion est effectuée par une phase importante du fonctionnement de notre serveur ; l'application des règles de grammaire (représentée en encadré).

Algorithme du fonctionnement général du serveur de messages :

→ **Initialisation** de compost ;

→ **Initialisation** du serveur ;

Tant que (la dernière entrée n'est pas détectée) (CORPS)

→ **FONCTIONS DU SERVEUR**

Si (aucune commande n'est active)

Tant que (une sortie de commande ne rend pas de message)

Lecture d'une commande ;

Si (la commande est locale)

Exécuter la commande ;

Sinon

Si (La commande se réfère à un format)

Initialisation du message en fonction du format ;

Sélection des enregistrements ;

Fin Si

Fin Si

Fin Tant que

Fin si

Sortie des messages vers compost ;

Fin FONCTIONS DU SERVEUR

Application des règles contenues dans les différentes grammaires (Création de l'arborescence);

→ **SYNTHESE DU MESSAGE** **(sortie des paramètres vers la carte de traitement du signal)**

Fin Tant que (FINCORPS)

8.3. Commandes du serveur.

Comme tous les systèmes, le serveur doit acquérir des informations en entrée, et fournir des informations ou faire des actions en sortie. Les entrées de notre serveur sont des commandes qui agissent à plusieurs niveaux dans la structure du programme. Les commandes sont de deux types. Les premières génèrent des messages, et sont directement liées aux applications du serveur. Les autres sont des commandes de contrôle, et ne génèrent pas de message parlé.

8.3.1. Commandes générant des messages parlés.

1) Génération de messages directs sans accès aux bases de données.

Ces commandes sont utilisées en mode interactif pour évaluer la prononciation des messages et le résultat des conversions.

• Passage d'une chaîne au format COMPOST sans aucune modification :

ch_compost < chaîne compost >

ex : ch_compost Det(LE) Nom(MESSAGE) ...

• Passage d'une chaîne qui opère une conversion des mots en majuscules et numériques ; le résultat est une suite d'objets de classe "mot" dominant des objets de classe "graph".

synt_graph < chaîne à convertir >

ex : synt_graph le message N° 1

action : Inc(LE) Inc(MESSAGE) Inc(N°) Num(1)

• Passage d'une chaîne qui opère une séparation des mots sans conversion ; le résultat est une suite d'objets de classe "mot" dominant des objets de classe "graph" ou "phon". Cette commande offre la possibilité d'une écriture phonétique dans une chaîne.

synt_phon < chaîne à séparer >

ex : synt_phon lx^ MESSAGE N° 1

action : Inc(lx^) Inc(MESSAGE) Inc(N°) Num(1)

2) Fonctions paramétrables avec accès aux bases de données.

Ces commandes nécessitent deux actions. La première opération est l'enregistrement du format du message. La seconde est l'appel de la commande par le nom qui a été défini lors de l'écriture du format, suivi par les valeurs des arguments dans l'ordre de leur apparition. Un enregistrement reste valable pendant toute la durée de l'utilisation.

format nom(liste des arguments) { corps du message }

nom < valeur de l'argument 1 > , < valeur de l'argument 2 > ...

ex : format code(nom, prénom) { corps du message }

code * , *

Cette fonction est caractéristique de notre serveur. Par les possibilités d'adaptation qu'elle offre, elle peut répondre aux besoins les plus divers.

3) Fonctions locales et fixes utilisant l'accès aux bases de données.

Ces fonctions sont écrites en langage C et liées avec le serveur. Elles nécessitent une connaissance approfondie de l'architecture du programme. L'utilité de ces fonctions est justifiée, lorsque les limites de l'écriture des formats se fait sentir, ou si une stratégie de recherche particulière dans les bases de données est indispensable. Dans ce cas, la complexité de la fonction n'est limitée que par la machine utilisée. A titre d'exemple, nous avons développé une fonction agenda avec des tests sur les valeurs des arguments, et des tris temporels sur les enregistrements. Cependant une limite apparait ; elle est due à l'encombrement mémoire que pourrait provoquer la multiplication de ces fonctions.

agenda nom , prénom , date de début , date de fin

8.3.2. Commandes ne générant pas de messages parlés.

1) Commandes de contrôle.

• lecture des champs et visualisation à l'écran.

ch < N° de base > , < N° d'enregistrement > , < N° de champ >

• lecture d'enregistrements et visualisation à l'écran.

enr < N° de base > , < N° d'enregistrement >

• arrêt des commandes et sortie

fin

• appel d'un fichier de commandes

$ < nom de fichier >

2) Paramètrage des valeurs (variables globales).

• mise en activité des pauses, cette variable permet l'arrêt du message lorsque le caractère d'identification est rencontré,

pause **< > ou O ou Y ou 1** activée

pause **0 ou N** désactivée

• affectation du nombre de différences autorisées pour la recherche

err **< nombre d'erreurs maximum >**

8.4. Fichiers périphériques.

Une partie importante des éléments de l'environnement extérieur est constituée par les fichiers périphériques. Comme il est logique de garder jusqu'au bout la démarche de la séparation modulaire, les fichiers de COMPOST et du serveur sont différents.

Nous avons pour les entrées COMPOST :

• Les fichiers obligatoires pour la programmation objet et l'écriture du scénario (compost.cls et compost.grm).

• Le fichier "klatt.lod" téléchargé sur la carte de traitement du signal pour la faire fonctionner en synthétiseur à formants.

Les fichiers sorties constituent essentiellement des traces de fonctionnement, leur ouverture est autorisée par les demandes d'option à l'appel du programme :

• Le fichier "compost.pho" garde la trace de l'arborescence. Son utilité ultérieure est intéressante pour la visualisation graphique et la modification des trajectoires.

• Le fichier "compost.tra" garde les traces du déroulement du logiciel COMPOST,

Il est surtout utilisé lors du développement.

• Le fichier de paramètres "compost.frm" mémorise sur le disque toutes les trames des échantillons.

Les fichiers d'entrée du serveur sont :

• Le fichier d'initialisation, qui contient les nom des fichiers des bases de données

• Le fichier de configuration qui est directement lié à l'interface utilisateur.

• Le fichier de commandes pour les appels automatiques du serveur de messages.

En sortie un fichier de traces rend compte des commandes qui se sont produites lors du dernier appel. Enfin les fichiers de bases constituent le support principal du serveur de messages. Une représentation des fichiers est donnée sur la figure 8.4

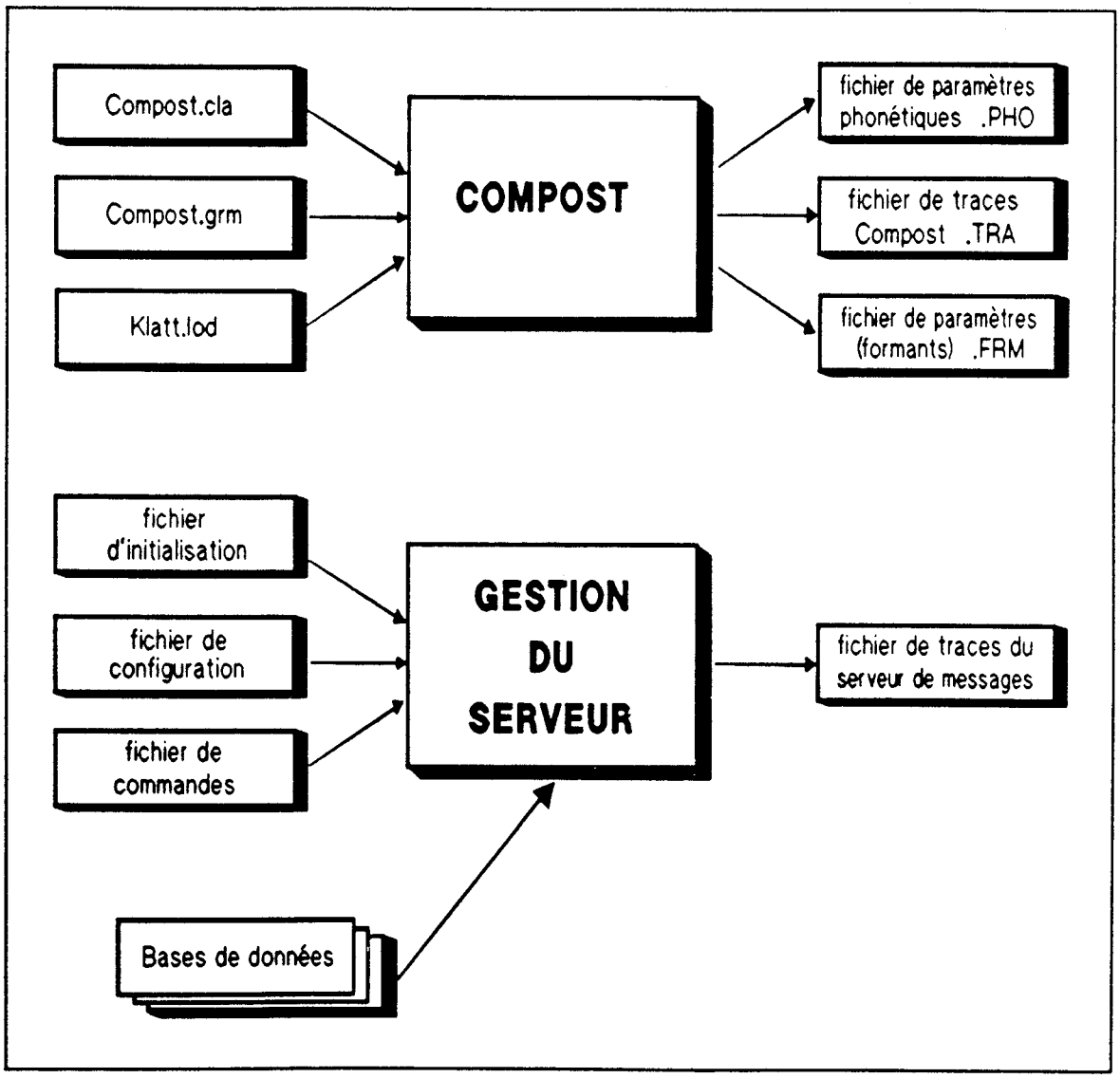


Figure 8.4. *Fichiers d'entrée et de sortie du serveur.*

8.5. Format de sortie à caractère fixe.

Les fonctions internes, écrites directement en langage C, offrent des possibilités très diverses. Leur paramétrage ne dépend que de la volonté du programmeur, elles peuvent être très ouvertes ou complètement figées. Nous allons décrire, dans ce paragraphe, une fonction qui a servi de prototype lors du développement du projet. Cette fonction est un agenda, elle a pour but de donner l'emploi du temps des personnes dont le nom est passé en arguments. La structure du message précise la date, la personne à rencontrer, l'heure et l'objet du rendez-vous. Les noms des champs nécessaires à cette fonction sont : NOM, PRENOM, DATE, HEURE, PERSONNE et RDV. Le nom de la fonction est "agenda" et les arguments sont le nom, le prénom, la date de début et la date de fin de la recherche. La partie générique du message est composée en fonction de la valeur des arguments "dates".

8.5.1. Utilisation des arguments temporels de l'agenda.

Pour la fonction agenda, il n'est pas obligatoire de préciser les dates, si la recherche concerne le jour de l'utilisation. L'écriture des arguments de la date est faite sous une forme numérique, selon l'ordre et les habitudes françaises : JJ MM AAAA (ou AA). La notation des éléments n'est pas obligatoirement complète, elle est, dans ce cas, reconstituée dans l'ordre par l'année le mois ou le jour courants.

exemple:

	-->	jour, mois et année courants
JJ	-->	JJ mois et année courants
JJ MM	-->	JJ MM et année courante

La présence du caractère joker "*" indique que la recherche n'est pas limitée. Les exemples suivants montrent quelques cas de recherche :

nom , prénom , * , 2 1 91	donne l'agenda de toutes les dates précédant le 1 janvier 1991,
nom , prénom , 10 12 1990 , *	donne l'agenda de toutes les dates depuis le 10 décembre 1990,
nom , prénom , * (, *)	la recherche est totale.

Pour plus de précision dans les recherches, il est possible de séparer les journées, en matins ou après-midi par les séquences "ma (ou av)" et "ap" ajoutées dans les arguments "dates". Les comparaisons et le tri tiennent compte aussi des champs "heure".

8.5.2. Constitution des messages.

Comme nous l'avons annoncé plus haut, les parties génériques des messages de cette fonction sont influencées par les valeurs des arguments "dates". Pour plus de clarté, l'énoncé de ces messages sera exprimé de façon littérale (sans tenir compte des conversions COMPOST obligatoires).

Liste de messages génériques en fonction des arguments :

- nom , prénom , * --> "Emploi du temps total" ,
- nom , prénom , 1 1 91 , * --> "Emploi du temps depuis le "1er janvier 1991",
- nom , prénom , * , 31 12 91 ap --> "Emploi du temps jusqu'au" "31 décembre 1991" "aprés midi",
- nom , prénom , 1 9 1990 av --> "Emploi du temps du" "10 septembre 1991" "matin",
- nom , prénom , 1 3 91 , 10 3 91 --> "Emploi du temps du 1 mars 1991" "au 10 mars 1991"

Les portions de message sont gérées par le contenu des champs qui la composent. Des champs identiques à ceux du passage précédant ne sont pas prononcés. Par exemple, la date d'un jour n'est prononcée qu'une seule fois, et si l'agenda ne concerne qu'une seule personne, ses prénoms et noms ne seront dits qu'une seule fois. La structure du format de messages est composée de quatre zones. La première, générique, ne contient pas de champs mais elle est variable suivant le contexte des arguments. La deuxième fait appel au champ date. La troisième contient les champs prénom, nom, heure et celui de la personne à rencontrer. La quatrième zone indique le lieu de rencontre. Nous rappelons dans la description qui suit, les portions de messages, les dépendances et les champs qui peuvent intervenir dans la constitution d'un message.

zone 1 (générique)

Partie fixe : " Emploi du temps ".
Parties contextuelles : "total", "du", "au", "depuis le", "jusqu'au", "matin", "après midi".
Parties variables : argument "date de début" et argument "date de fin".

zone 2 (annonce de la date)

Partie fixe : "le".
Partie variable : contenu du champ [date].

Le contenu du champ doit être différent de celui du précédent, pour générer une sortie parlée de cette zone.

zone 3 (rencontre des personnes)

Séquence des parties :

variables	champs [prénom] et [nom]
fixe	"rencontre avec"
variable	champ [personne]
fixe	"à"
variable	champ [heure]

La sortie parlée de cette zone dépend du contenu des champs [prénom] et [nom].

zone 4 (précisions sur le "rendez vous")

Une seule partie, variable, constitue cette zone : le champ du rendez-vous [RDV]. Elle indique le motif et le lieu de la rencontre.

Les recherches sont sans niveau relationnel, il faut donc que tous les champs soient présents dans chaque base de données prévue pour l'application.

8.6. Règles COMPOST spécifiques au serveur de messages.

Les concepts généraux de COMPOST ont été spécifiés au chapitre 4. Nous allons ici montrer quelques règles de transformation, notamment celles qui ont servi d'exemples, à plusieurs reprises dans ce mémoire. Pour certaines conversions, nous avons défini une classe supplémentaire "champ" au dessus de la classe "mot", afin de bien séparer les actions et les grammaires, lors de l'écriture du scénario. D'autre part, le caractère choisi pour identifier l'arrêt est le | dans la classe "graph".

définition de la classe champ : (avec les objets arrêt, date, fax, heure et téléphone) :

```
classe  champ
        objet (Art, Date, Fax, Heu, Tel)
finclasse
```

8.6.1. Conversion des numéros téléphoniques.

Donnons en exemple les règles de séparation des numéros de téléphone deux chiffres par deux chiffres (habitude française), avec un ajout de pauses et d'arrêts entre les groupes. Par exemple la chaîne d'arrivée à COMPOST est "Tel(76570000)".
La grammaire agit sur les objets des classe "graph" pour les parties et les contextes. Nous savons aussi que les règles sont appliquées, tant que cela est possible, de gauche à droite.

```
grammaire TEL  (graph -> graph / graph + graph , 1)
        gr_2   =   [chif] [chif]  ->  Inc(|) Num(#1#2)  /  Tel([chif]*[0,20] + [chif]*[0,20]) ;
fingrammaire
```

Cette règle indique que deux chiffres donnent un arrêt et un groupe de deux chiffres, s'ils se trouvent entre un début de mot "Tel(" (suivi éventuellement de chiffres déjà convertis entre 0 et 20) et une suite de chiffres pas encore convertis entre 0 et 20. Les #1 et #2 transfèrent les deux chiffres de la partie gauche vers la partie droite. La chaîne résultante est :

```
Tel( Inc(|) Num(76) Inc(|) Num(57) Inc(|) Num(00) Inc(|) Num(00) )
```

A ce texte nous pouvons enlever le niveau Tel( ), qui devient inutile, par une autre règle, le résultat sera alors :

Inc(|) Num(76) Inc(|) Num(57) Inc(|) Num(00) Inc(|) Num(00)

8.6.2. Traitements des arrêts (Pause).

Nous avons choisi de traiter les arrêts de deux façons différentes sans perturber les règles déjà existantes. La première par le caractère de classe "graph" "|", qui peut être inséré dans du texte lors du traitement COMPOST. La seconde par l'objet "Art" lorsque nous voulons le passer depuis les formats de message (le caractère "|" étant réservé à la coupure de zone).

Les régles de transformation dans ce cas sont simples :

```
grammaire PAUSE1 ( graph -> phon / phon + phon , 1 )
        Arret   =  |  ->  <_, duree = 500> (X1, INSTANT=10, UO=-7, AF=1, AH=1);
fingrammaire

grammaire PAUSE2 ( phon -> phon / phon + phon , 1 )
        ART     =  _  ->        <_, duree = 500> (X1, INSTANT=10, UO=-7, AF=1, AH=1) /
                                 Art(phon*[0,10] + phon*[0,10]) ;
fingrammaire
```

La première procède à une transformation orthographique phonétique, sans condition sur les contextes. Alors que la deuxième reste dans la classe "phon", les contextes à l'intérieur de l'arrêt sont chargés de phonèmes pour agrandir l'éventail des possibilités. Dans les deux cas le résultat est identique ; le phonème "_" (silence) se voit imposer la durée et les "cibles" qu'il domine. La différence essentielle entre l'arrêt et une pause normale est la valeur du paramètre U0. En effet, lorsque la commande pause est activée, un test sur la valeur de U0 permet d'arrêter la parole jusqu'à la frappe d'un caractère au clavier. Si la pause n'est pas active, alors l'arrêt n'est pas pris en compte.

8.6.3. Contrôle de la prosodie.

Les règles qui s'appliquent à la prosodie modifient les valeurs de F0. Dans notre synthétiseur, F0 est la somme de trois éléments indépendants : F0_BASE, MICRO et ACCENT. "F0_BASE" supporte les règles agissant sur la ligne de déclinaison et le contour prosodique. Les deux exemples, que nous montrons, concernent les phrases déclaratives et interrogatives. Au début de la phrase F0_BASE est fixée à une valeur haute (140 Hz), puis par une trajectoire X1 (droite) il atteint le point bas (120 Hz) de la ligne de déclinaison sur l'avant dernière syllabe. Le contour prosodique de la dernière syllabe est défini en fonction de la ponctuation. Un point ou une absence de ponctuation produit une variation descendante (90 Hz) par une trajectoire en "X1" ou en "X2D" (dérivée nulle à droite). Un point d'interrogation définit une trajectoire montante (170 Hz) en "X2G" (dérivée nulle à gauche). Comme toutes les ponctuations génèrent un silence, par la traduction orthographique-phonétique, il faut garder une trace des points particuliers (? ou !), jusqu'au moment de l'écriture des règles les concernant. Le traitement de la ponctuation doit donc se faire au moins sur deux grammaires : une grammaire incluant la traduction orthographique-phonétique avant la syllabation, et une grammaire traitant les variations de F0_BASE en fonction des syllabes, après la syllabation et la génération des trajectoires.

exemple de traitement de la ponctuation :

Les premières grammaires transforment le point "?" en séquence "_?" séparée du dernier mot, afin de garder, avec le phonème silence, la trace interrogative.

```
grammaire PONCT    [ graph -> graph / graph + graph , 1 ]
   pg_in =     ?     ->    ) Pt(?? / graph +
                       ->    ?? / mot(+       ;
fingrammaire

grammaire AJOUT [ graph -> phon / graph + graph , 1 ]
   pau_? =     ?     ->    <_,duree=200>  / + ?  ;
fingrammaire
```

Traduction orthographique-phonétique ;

Syllabation en fonction des phonèmes ;

Génération des trajectoires ;

Les grammaires suivantes traitent les variations de F0_BASE en fonction des syllabes et des traces de ponctuation.

```
grammaire BASE_F0 [phon -> phon / phon + phon , 1]

    /*  Régle de début de phrase : initialisation de F0_BASE ainsi que ACCENT et
    MICRO.  */
    deb_p  = mot(Syl (phon(#A)    ->    #1(#2(#3 (<X1, F0_BASE=140, MICRO=-1,
           ACCENT=1, INSTANT=10> #A) / ## + ;

    /*  Régle de fin de phrase déclarative.  */
    fdec2  = Syl(phon(#A) phon*[0,3] phon(#B)))
           ->     #1(#2 (<X2D, F0_BASE=120,INSTANT=0> #A) #3 #4
           (#B <X1,[final], F0_BASE=90,INSTANT=0>))) / + {mot(Syl(_)) ##, ##} ;

    /*  Règle de fin de phrase interrogative.  */
    fint2  = Syl(phon(#A) phon*[0,5])) Pt( Syl (phon(#B)  phon*[0,1]
           ->    #1(#2 (<X2G, F0_BASE=120,INSTANT=0> #A) #3)
           #4(#5 (#6(#B<X1, F0_BASE=170,INSTANT=0>) #7  / + ?) ;

fingrammaire
```

Nous rappelons que dans l'écriture des règles, les "#" suivis de chiffres de la partie droite correspondent aux objets de la partie gauche, alors que les "#" suivis de lettre transmettent l'arborescence qu'ils dominent, de la partie gauche vers la partie droite. Les accolades indiquent des choix multiples, par exemple le contexte gauche de la régle de fin de phrase déclarative peut être un "##" (fin de phrase), ou un phonème isolé avant la fin de phrase.

"MICRO" modifie la micro prosodie (phonème), et "ACCENT" dépend de certaines syllabes accentuées. Nous trouverons toutes les régles qui se réfèrent à la prosodie en ANNEXE 6 dans le fichier du scénario.

8.7. Evaluation du serveur.

Le but de notre projet était de faire fonctionner, sur un micro-ordinateur personnel de grande diffusion, une application contenant un système complet de synthèse de la parole. Il faut maintenant évaluer, suivant différents critères, les performances de notre serveur. Les éléments qui peuvent limiter les capacités sont la taille de mémoire vive disponible, et la rapidité du micro-ordinateur lors du traitement de la parole.

8.7.1. Occupation de la mémoire.

La taille mémoire du micro-ordinateur, qui a servi pour les tests, est de 640 koctets. Mais la mémoire réellement disponible n'est plus que de 490 koctets, lorsque les pilotes de périphériques (device drivers), le système d'exploitation, et les programmes résidents sont chargés, et les variables d'environnement initialisées. Une des préoccupations majeures, lors du développement, est de limiter la dimension du code (programme exécutable), afin de conserver la possibilité de faire de la synthèse de la parole dans de bonnes conditions. Le code généré par les fonctions de la gestion du serveur est équivalent à celui de COMPOST. Une compilation qui optimise la compacité du code donne un programme exécutable dont la taille est de l'ordre de 150 koctets.

Le bilan de l'occupation mémoire par le serveur est le suivant :

- Programme exécutable	compost.exe	150	koctets
- Liste des objets	compost.cla	16	koctets
- Grammaires	compost.grm	94	koctets
- Variables globales (estimation)		50	koctets
- Pile pour variables des fonctions		20	koctets

Soit un total de 320 koctets pour les parties stables de l'occupation mémoire.

Il reste donc **170 koctets** (490 - 320) pour le traitement de la parole. Cet espace va être essentiellement occupé par des structures, sous la forme d'allocations dynamiques de la mémoire. Ces structures correspondent à un point cible de chaque paramètre. Elles sont constituées des données nécessaires aux positionnements et aux calculs des valeurs des paramètres envoyés dans les trames de contrôle de la carte de traitement du signal.

Le calcul de l'occupation moyenne de la mémoire pour une seconde de parole, peut être conduit avec les valeurs suivantes :

- le débit moyen de phonèmes est de l'ordre de 6 par seconde,
- le nombre de paramètres, concernés par les structures, est de 26 par phonème,
- le nombre moyen de cibles est de 2,5 par paramètre et par phonème,
- la taille d'une structure est de 40 octets,

L'occupation mémoire est donc le produit de tous ces éléments, elle est de l'ordre de :

15600 octets par seconde de parole

Dans le contexte que nous venons de décrire, la durée (longueur) maximale de la phrase prononcée est de l'ordre de :

170000 / 15600 = **10,8 secondes**

Nous pouvons aussi exprimer cette longueur en nombre maximum de phonèmes :

10,8 * 6 = **64 phonèmes**

Les mesures faites lors des essais confirment ces résultats. En effet, les phrases, dont la durée ne dépasse pas 10s, ont toutes été synthétisées avec succès, alors que des phrases, qui dépassent les 10s, présentent parfois des problèmes d'allocation mémoire. Il n'a pas été possible, toujours avec la même configuration, de synthétiser des phrases dont la longueur était supérieure à 14s.

Cette limitation est tout à fait acceptable pour le serveur de message : une phrase de 10 secondes est assez longue pour une bonne compréhension des messages. Pour les messages longs il est possible de les sectionner en zones (par groupes de souffle par exemple) dans la définition des formats.

8.7.2. Temps de traitement.

Le temps de traitement est un paramètre important pour le confort de l'utilisateur. Une attente trop longue, entre le passage d'une commande et le retour de la phrase synthétisée, rendrait le serveur inexploitable. Les mesures de temps ont été effectuées sur un micro-ordinateur équipé d'un microprocesseur "INTEL 386", et d'une horloge fonctionnant à 20 MHz. Le temps de recherche dans les bases de données n'est pas pris en compte puisqu'elles étaient assez petites (moins de 100 enregistrements). Le temps de traitement est significatif, et les relevés ont montré qu'il était légèrement supérieur à la durée de la phrase.

Relevés des temps pour des phrases types :

Phrase : " Les numéros de téléphone sont dans la base de données"
Temps de traitement : 5,5 s
Durée du message : 5 s

Phrase : " Les numéros de téléphone sont dans la base de données utilisée comme fichier de référence "
Temps de traitement : 9 s
Durée du message : 7,5 s

Phrase : " Laboratoire de traitement du signal de parole à l'institut de la communication parlée"
Temps de traitement : 10 s
Durée du message : 9 s

Phrase : " Enumération de nombres 76 57 86 78 23 11 49 "
Temps de traitement : 15 s
Durée du message : 11 s

L'ajout d'un nombre supplémentaire (87) a provoqué un problème d'allocation mémoire. La limite de longueur du message est atteinte.

Vérifions à présent, sur deux exemples, la contribution des messages en nombre de phonèmes. Ces exemples sont directement issus du traitement (fichier de trace), ils montrent la traduction orthographique phonétique.

Exemple 1 :

Commande : **synth_graph** Les numéros de téléphone sont dans la base de données

Chaîne passée à COMPOST : Inc(_) Inc(LES) Inc(NUME6ROS) Inc(DE) Inc(TE6LE6PHONE) Inc(SONT) Inc(DANS) Inc(LES) Inc(BASES) Inc(DE) Inc(DONNE6ES)#

Transformation en chaîne phonétique : Inc(_) Inc(le) Inc(nymero^) Inc(dx^) Inc(telefo^n) Inc(so~) Inc(da~) Inc(le) Inc(baz) Inc (dx^) Inc (do^ne)#

Le nombre de phonèmes est de 33 et la durée du message est 5s, nous avons donc pour cet exemple un débit de :

(33/5) **6,6 phonèmes / seconde**

Exemple 2 :

Commande : **synth_graph** énumération de nombres 76 57 86 78 23 11 49

Chaîne passée à COMPOST : Inc(_) Inc(E6NUME6RATION) Inc(DE) Inc(NOMBRES) Num(76) Num(57) Num(86) Num(78) Num(23) Num(11) Num(49) #

Transformation en chaîne phonétique : Inc (_) Inc(enymerasjo~) Inc(dx^) Inc(no~br) Num(swasa~t) Num(sx^z) Num(se~ka~t) Num(sx^t) Num(katr) Num(ve~) Num(sis) Num(swasa~t) Num(diz) Num(hit) Num(ve~t) Num(trwa) Num(o~z) Num(kara~t) Num(nx^f)#

Pour cette phrase le nombre de phonème est 72 et le débit est égal à :

(72/11) **6,5 phonèmes / seconde**

Ces valeurs sont conformes (légèrement supérieures) aux hypothèses de départ. Nous avions prévue un débit moyen de 6 phonèmes par seconde et un nombre moyen maximum de 64 phonèmes.

8.7.3. Qualité de la parole synthétisée.

Bien que cette notion soit très subjective, la parole synthétisée par le serveur de messages est intelligible. L'aspect naturel de la voix a été nettement amélioré par l'introduction des règles de prosodie, les plus importantes étant la ligne de déclinaison et le contour mélodique des dernières syllabes. Les recherches actuelles, sur la synthèse à formants par règles du français, vont permettre une amélioration sensible de la qualité de la parole synthétisée.

2. Conclusion.

Le serveur de messages à réponse vocale, que nous avons réalisé, répond entièrement aux objectifs que nous nous étions fixés. La parole synthétisée est de bonne qualité, et les phrases prononcées sont compréhensibles et naturelles. La composition des messages est directement accessible aux utilisateurs. Une interface utilisateur apporte une convivialité appréciable en minimisant les opérations. Toutes ces fonctions sont implantés sur un ordinateur personnel du type A.T. avec 640 koctets de mémoire. Cependant les temps de traitement ne sont pas négligeables et la nécessité d'étendre les fonctionnalités se fait sentir. Une évolution sensible des performances impliquera obligatoirement une amélioration des ordinateurs support. Elle passera par une augmentation de la vitesse et de puissance de calcul. D'autre part l'extension, et l'exploitation de mémoire RAM supplémentaire, autoriseront des fonctionnalités plus entendues (notamment pour la gestion des bases de données), et une longueur de message plus élevée.

Les applications du serveur de message peuvent être très variées, celles que nous avons réalisées sont locales (agenda, numéros de téléphone, référence, etc.). Pour des extensions, il sera utile de prévoir des fonctions qui contrôlent l'environnement extérieur. Par exemple, nous pouvons prévoir la connexion au réseau téléphonique, et composer les numéros de téléphone avant l'émission des messages. Une autre extension consisterait à utiliser à la place du clavier et de l'écran local un minitel distant pour effectuer des requêtes. Les lignes téléphoniques véhiculeraient alors, les commandes sous une forme binaire, et la voie synthétisée par un signal analogique. Cette approche implique une adaptation matérielle du produit.

L'architecture modulaire permet, grâce au concept de COMPOST, d'intégrer facilement les progrès effectués par la recherche dans le domaine de la synthèse de la parole. Il est tout à fait concevable, dans des versions futures, de remplacer la synthèse à formants par une synthèse à prédiction linéaire, ou une synthèse PSOLA, sans modifier la gestion du serveur.

Une augmentation des possibilités matérielles en taille mémoire, et en rapidité de calcul permettra d'utiliser un analyseur syntaxique performant. La synthèse de texte libre se fera alors avec de bonnes performances. Cette amélioration concerne essentiellement le traitement des ambiguïtés, lors de la traduction orthographique phonétique.

L'intégration de marqueurs de prosodie dans le traitement COMPOST, et d'un analyseur prosodique complexe, améliorera la génération automatique de la prosodie. Ils contribueront ainsi, à donner une qualité plus naturelle à la voix synthétisée, par un contrôle plus précis de la tonalité, de l'énergie et du rythme.

La définition du format des messages est un élément important du serveur. Elle satisfait aujourd'hui une grande partie des besoins. Il est cependant possible de la faire évoluer vers un dialogue homme-machine plus naturel et plus complet. Les orientations que nous proposons sont :

- une gamme plus importante de conversions,
- l'intégration de dépendance inter-zones à l'intérieur d'un message.

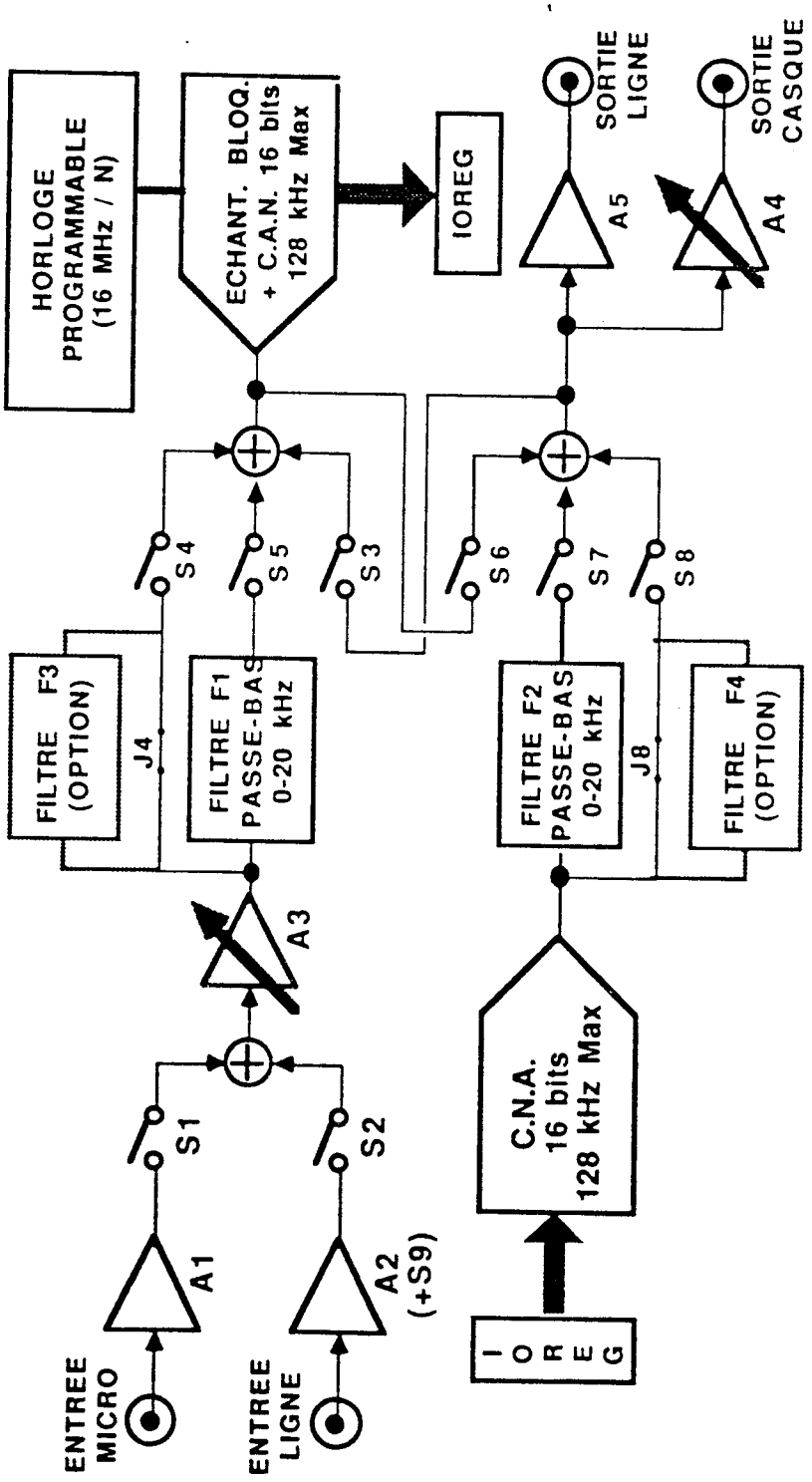
L'introduction de stratégies de recherche, de sélection et de tri spécifiques à chaque format, apporterait une grande souplesse dans l'utilisation du serveur. Dans un but de compatibilité, cet ajout pourra faire intervenir un interpréteur de langage spécialisé dans la gestion des bases de données, par exemple S.Q.L. (Stuctured Query Language).

Le temps de traitement de la parole est du même ordre que la durée du message (quelque secondes). Bien que cette attente ne soit pas gênante pour l'écoute et la compréhension du message, il parait souhaitable de rendre la parole plus continue. Cette relative équivalence des temps de traitement et de parole, nous conduit à proposer le parallélisme pour l'architecture matérielle des prochains produits.

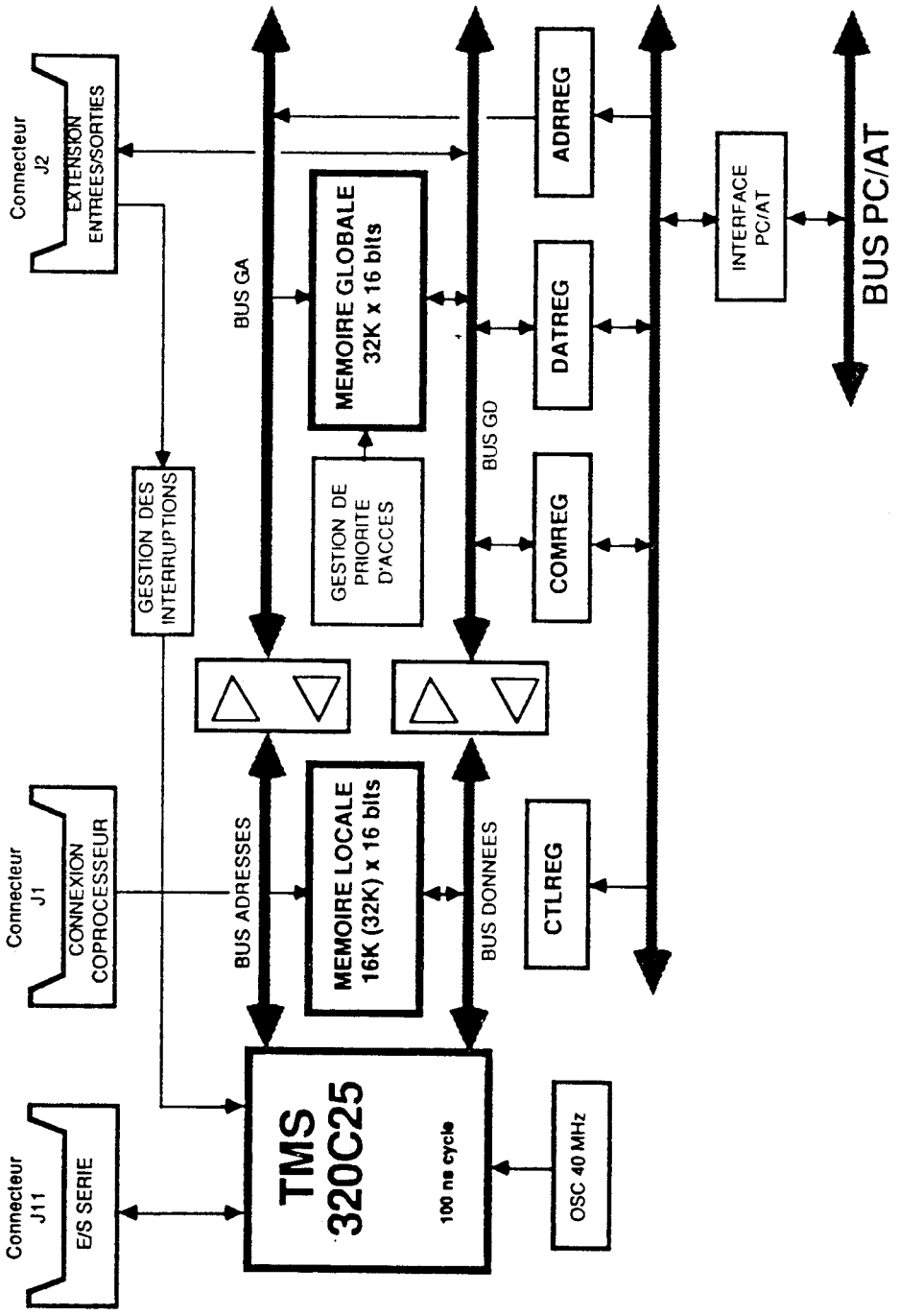
Enfin nous ne pouvons terminer ce mémoire sans faire appel à la reconnaissance de la parole. Une introduction progressive, au niveau de l'interface utilisateur, de la reconnaissance de mots isolés peut remplacer les barres de menu pour la sélection des commandes. Une deuxième étape introduirait une reconnaissance plus complète pour la détection des commandes et la prise des arguments. Il serait, alors, possible d'envisager un dialogue sans clavier, ni écran. Pour un bon confort d'utilisation, un tel système devra fonctionner sur des stations de travail performantes.

ANNEXE 1 : Architecture de la carte OROS-AU21.

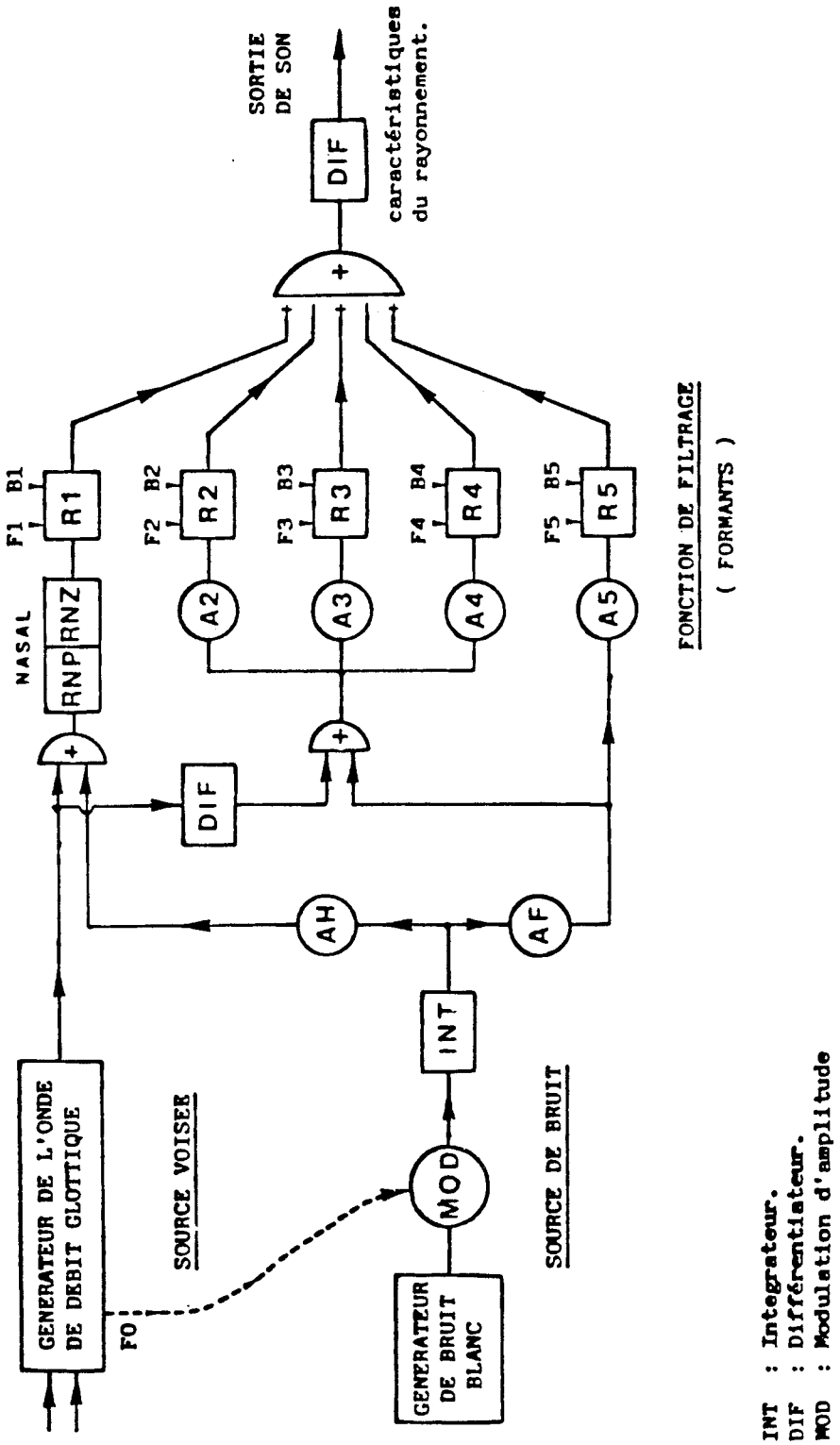
PARTIE ANALOGIQUE :



PARTIE NUMERIQUE :



ANNEXE 2 : Schéma du synthétiseur à formants.



ANNEXE 3 : Séquences d'échappement ANSI.

Les séquences que nous allons décrire sont actives si le pilote de périphérique (DEVICE) ANSI.SYS est déclaré dans le fichier de configuration CONFIG.SYS. Les actions concernent le curseur, les fonction du curseur, l'effacement et la sélection des modes. Les paramètres : ligne, colonne, nombre et sélection, sont passés sous forme numérique décimale.

les séquences ANSI commencent toujours par le code d'échappement ESC (27 en décimale ou 1B en hexadécimale) suivi d'un "[". Ces fonctionnalités sont extraites des normes ANSI 3.64-1979 et ISO 6429.

Positionnement du curseur : ESC[ligne;colonne**f** ou ESC[ligne;colonne**H**

Déplacements du curseur :

vers le haut,	ESC[nombre**A**
vers le bas,	ESC[nombre**B**
vers le droite,	ESC[nombre**C**
vers la gauche,	ESC[nombre**D**

Demande de la position : ESC[**6n**

la récupération est sous la forme ESC[ligne;colonne**R** dans le tampon d'entrée.

Sauvegarde de la position du curseur : ESC[**s**

Restauration de la position du curseur : ESC[**u**

Effacement écran : ESC[**2J**

Effacement ligne : ESC[**K**

Mode d'affichage: ESC[=sélection**h** ou ESC[=**h** ou ESC[=**0h** ou ESC[**?7h**

Annulation du mode d'affichage: ESC[=sélection**l** ou ESC[=**l** ou ESC[=**0l** ou ESC[?7l

Les valeurs de la sélection sont :

0	40 x 25	caractères,	monochrome,
1	40 x 25	caractères,	couleur,
2	80 x 25	caractères,	monochrome,
3	80 x 25	caractères,	couleur,
4	320 x 200	points,	couleur,
5	320 x 200	points,	monochrome,
6	640 x 200	points,	monochrome,
7			

paramètres graphiques : ESC[sélection; ... ;sélection**m**

valeur des sélection :

attributs :	0	normaux,	4	souligné,
	1	surintensité (gras),	5	clignotement,
	2	demie densité,	6	clignotement rapide,
	3	Italiques,	7	vidéo inverse,
			8	invisible.

couleurs :

	caractères,	fond,
noir	30	40
rouge	31	41
vert	32	42
jaune	33	43
bleu	34	44
magenta	35	45
cyan	36	46
blanc	37	47

Certains ordinateurs acceptent des commandes supplémentaires telles que la redéfinition des touches du clavier. ESC[nombre; ... ;"Chaîne"; ...nombre**p**

le premier code correspond à la touche et les suivants sont la séquence de code lorsque la touche est sélectionnée.

Exemple d'utilisation des séquences ANSI en langage C.

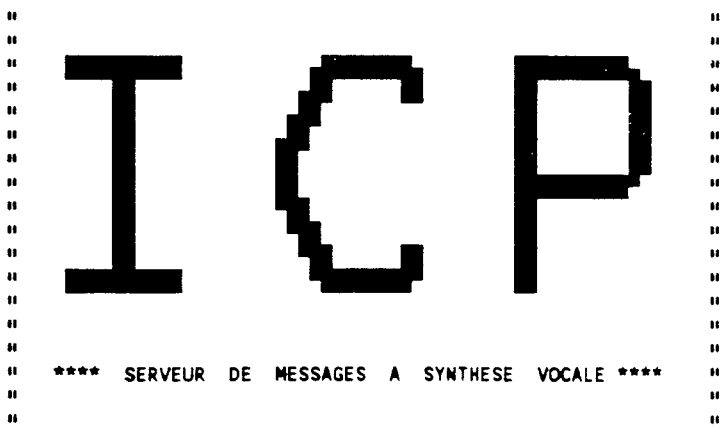
Utilisations de MACROS (#define).

```
#define  FANSI(s)      printf("%c[%s", 27, s)   /* macros de passage des fonctions ANSI     */
#define  CLS           FANSI("2J")              /* effacement ecran                        */
#define  EFL           FANSI("K")               /* effacement d'une ligne                  */

#define  NOIR          0       /* couleurs de base */
#define  ROUGE         1
#define  VERT          2
#define  JAUNE         3
#define  BLEU          4
#define  MAGENTA       5
#define  CYAN          6
#define  BLANC         7

#define  CAR           30+          /* attribution des couleurs aux caractères.    */
#define  FOND          40+          /* attribution des couleurs au fond.           */

#define  NORM          0            /* attributs de luminosité : normal            */
#define  GRAS          1            /*                  sur-intensité              */
#define  DEMI          2            /*                  demie intensité            */
#define  INV           7            /*                  inversion vidéo            */

      /* Position du curseur en valeurs absolues (ligne, colonne). */
#define  CU(l, c)      sprintf(esc,"%d;%dH", l, c); FANSI(esc)
      /* déplacements du curseur */
#define  CUH(n)        sprintf(esc,"%dA",n); FANSI(esc)      /* haut    */
#define  CUB(n)        sprintf(esc,"%dB",n); FANSI(esc)      /* bas     */
#define  CUG(n)        sprintf(esc,"%dD",n); FANSI(esc)      /* gauche  */
#define  CUD(n)        sprintf(esc,"%dC",n); FANSI(esc)      /* droit   */

/* définition d'une séquense (attribut, couleur des caractères, couleur du fond) */
#define  COULEUR(a, c, f)   sprintf(esc,"%d;%d;%dm", a, c, f); FANSI(esc)

char  s[20], esc[20];

int fonction ()
      { ...
            CLS                                    /* effacement écran   */
            COULEUR(CAR ROUGE, FOND NOIR, NORM)    /* couleurs           */
            CU(10,20);                  /* position du curseur: ligne 10, colonne 20 */
      ...
      }
```

ANNEXE 4 : Fichier de configuration.

```
#*****************************************************************************
   Fichier : IU.CFG
   Type :    TEXTE
   But :     CONFIGURATION DE L'INTERFACE UTILISATEUR DU SERVEUR DE MESSAGES
   Auteur :  Paul  TRIPODI
   Date :    2  AVRIL  1991
 *****************************************************************************#



# Fichier de ressources pour l'application agenda et serveur à réponse vocale  #


# Nombre maximum d'items par menu 8,                                          #
# Longueur maximum de l'ensemble des textes de menu 80                        #
# Ecriture des chaines de caractères :  "chaines"                             #
# Ecriture des commentaires : #commentaire#                                   #

# Couleurs possibles :                                                        #
# noir, rouge, vert, jaune, bleu, magenta, cyan, blanc .                      #

#                                                                             #
# SYNTAXE :                                                                   #
#                                                                             #


#  ECRAN                                                                      #
#  couleur de fond de l'écran,                                                #
#  couleur de fond et couleur des caractères du logo,                         #
#  couleur de fond et couleur des caractères des fenêtres,                    #
#  couleur de fond et couleur des caractères de l'aide,                       #
#  couleur de fond et couleur des caractères des items des menus sélectionnés #


#  LOGO                                                                       #
#  position initiale ligne,  position initiale colonne,                       #
#  nombre de ligne, nombre de colonnes                                        #
#  Suite des chaines de définition du logo (le nombre est égal au nombre de   #
#  lignes et la longueur au nombre de colonnes                                #


#  MENU_HAUT nombre d'items, n° de ligne, couleur du fond, couleur du texte,  #
#  couleur de l'indicateur de touches de fonctions, couleur du caractère de   #
#  sélection.                                                                 #
#  Séquences de description (le nombre doit être égal au nombre d'items)      #
#  Contenus des séquences de description :                                    #
#  nom de la fonction  "chaine d'affichage à l'écran" "touche de fonction".   #
#  Le caractère de sélection est précédé d'une accolade.                      #


#  MENU_BAS  (même forme que MENU_HAUT)                                       #
```

```
#  LIGNE_COMMANDES n° de ligne, couleur du fond, couleur du texte de l'invite  #
#  (prompt), couleur du texte de commande.                                     #


#  LIGNE_ETATS nombre d'items, n° de ligne, couleur du fond, couleur du texte  #
#  fixe, couleur des réponses.                                                 #
#  Séquences de définition des items.                                          #
#  Les séquences sont composées par :  le nom de la variable                   #
#  "la chaîne affichée à l'écran"  "la chaîne pour la réponse affirmative"     #
#  "la chaîne pour la réponse négative".                                       #
#  Dans le cas de réponse numérique ces deux dernières chaînes sont annulées   #


#  FENETRE                                                                     #
#  position initiale ligne,  position initiale colonne,                        #
#  nombre de ligne, nombre de colonnes                                         #
#  Séquences des chaînes d'affichage.                                          #
#  Les chaînes attendant des arguments sont précédées d'un nom                 #
#  les autres (chaînes de présentation) sont précédées d'une parenthèse.       #


#  AIDE
#  position initiale ligne,  position initiale colonne,                        #
#  nombre de ligne, nombre de colonnes                                         #
#  Suite des chaînes de définition de l'aide (le nombre est égal au nombre de  #
#  lignes et la longueur au nombre de colonnes                                 #


#  FORMAT  nom de la fonction (liste des arguments) ( description du message ) #


# ------------------------------------------------------------------------------ #


#              *********          CONFIGURATION          *********              #


ECRAN  bleu,
       noir, rouge,
       blanc, noir, bleu,
       cyan, noir,
       rouge, noir


LOGO    4, 10, 18, 60
```



```
MENU_HAUT   5   ,     1   ,   blanc ,   noir ,   rouge ,    bleu
    aide                "-(Aide  "        "F1"
    quitte              "-(Quit  "        "F3"
    sequence            "-(Go  "          "F5"
    format              "-For(mat  "      "F4"
#    gestion_fichiers    "-(Fichiers  "    "F2" #
    synt_graph          "-(Synt_graph  "  "AF6"
#    synt_phon           "-(Synt_phon"     "AF7" #


MENU_BAS   6,25, blanc, noir, rouge, bleu
    agenda              "-Agen(da  "      "F6"
    tel                 "-(Tel  "         "F7"
    tel_fax             "-Tel-(Fax  "     "F8"
#    references          "-(Ref  "         "F8" #
    ch_compost          "-(Compost  "     "F9"
    adr                 "-Adre(sse  "     "CF1"
    travail             "-Tra(vail  "     "CF2"


LIGNE_COMMANDES  24, cyan, noir, noir
    prompt    ">>>>"


#              nb d'items      ligne,   fond,   texte,   réponses    #
LIGNE_ETATS    4               23       bleu,   blanc,   noir
    index           "index "    "0"          "N"
    erreur          " ; Err = "
    nb_selection    " ; enreg = "
    pause           " ; Pause " "0"          "N"
    trait-en-cours " "          " ; BUSY "   " ; READY"


FENETRE    synt_graph  10, 10, 6, 60
   (     ""
   (     "                  SYNTHESE  DE TEXTE LIBRE                  "
   (     "         ----------------------------------------           "
   (     ""
   chaines  " Texte : "
   (     ""


FENETRE    ch_compost  10, 10, 6, 60
   (     ""
   (     "             SYNTHESE  D'UNE  CHAINE  COMPOST              "
   (     "         ----------------------------------------           "
   (     ""
   chaines  " Chaines : "
   (     ""


FENETRE    synt_phon  8, 6, 8, 68
   (     ""
   (     "                      SYNTHESE DE TEXTE LIBRE :                  "
   (     "                 ------------------------------------            "
   (     "  Les caractères minuscules simples sont traités en phonèmes.     "
```

```
(     " Les caractères majuscules ou accentués sont traités en graphèmes. "
(     ""
chaines  " Texte : "
(     ""


FENETRE    format  6, 4, 10, 72
   (   ""
   (  "                DEFINITION D'UN FORMAT TEMPORAIRE  "
   (  "              ************************************** "
   (   ""
   (  " nom (arg1, arg2, ...)(  Det(LE) ... [champ]  |  Nom(SUITE) [champ] )"
   (  ""
   format1   "  "
   format2   "  "
   format3   "  "
   (          ""


FENETRE   agenda  10 , 10 , 6 , 60
   (         ""
   nom       " Nom           : "
   prenom    " Prénom        : "
   date_d    " date 1        : "
   date_f    " date 2        : "
   (         ""


FENETRE tel_fax  10 , 10 , 7 , 42
   (         "  NUMEROS DE TELEPHONE ET DE TELECOPIEUR  "
   (         " ---------------------------------------- "
   (         ""
   nom       " Nom           : "
   (         ""
   prenom    " Prénom        : "
   (         ""


FENETRE    tel  10 , 10 , 6 , 40
   (         "           NUMEROS DE TELEPHONE         "
   (         " ------------------------------------ "
   (          ""
   nom       " Nom           : "
   prenom    " Prénom        : "
   (          ""


FENETRE    adr  10 , 10 , 6 , 40
   (         "           LIEU  DE RESIDENCE         "
   (         " ------------------------------------ "
   (          ""
   nom       " Nom           : "
   prenom    " Prénom        : "
   (          ""
```

```
FENETRE    travail  10 , 10 , 6 , 40
   (        "   LIEU DE L'ACTIVITE PROFESSIONNELLE   "
   (        " ------------------------------------- "
   (        ""
   nom      " Nom           : "
   prenom   " Prénom        : "
   (        ""


FENETRE    references  10 , 10 , 4 , 40
   (        ""
   nom      " Nom           : "
   prenom   " Prénom        : "
   (        ""


AIDE   agenda   3 , 5 , 19 , 71 ,

"
"            AIDE  générale sur l'utilisation de l'agenda
"            ------------------------------------------
"
" * édition    :
" Le caractère '*' peut remplacer, suivant l'endroit où il se trouve,
" une lettre , un mot, la fin d'un mot ou d'une phrase.
" Pour la recherche il n'y a pas de distinction entre lettres
" majuscules et minuscules et entre caractères simples et accentués.
" La date (1) correspond à la date de début de la recherche .
" La date (2) correspond à la date de fin de la recherche.
" Le format de la date est jj mm AAAA (19 12 90 ou 19 12 1990).
" Les valeurs non précisées seront initialisées avec la date courante.
"
" * fonctions  :
" Pour lancer une recherche sélectionnez 'GO'.
" Pour arrêter une recherche utilisez 'ESC' ou 'FIN'.
" Pour revenir à la fenêtre d'édition sélectionnez 'Agenda'.
"


AIDE   tel_fax   5 , 15 , 9 , 51 ,

"
"  AIDE  générale sur l'utilisation de tel_fax
"  -------------------------------------------
"
" * Les nombres sont composés de 2 chiffres.
"
" * La commande pause autorise les arrets.
"


AIDE    aide    5 , 5 , 9 , 62 ,

"
" L'accès aux commandes se fait par :
"      - les touches de fonction
"      - par les barres de menus avec PgUp ou PgDn puis
"        par le caractère de sélection  ou  les flèches
"        gauche et droite suivi d'un CR.
"
"
"
```

```
FORMAT  nom (nom, prenom) (  [prenom] [nom]  )

FORMAT  telephone (nom, prenom) ( Det(LE) Nom(NUME6RO) Pcn(DE) Nom(TE6LE6PHONE)
Pcn(DE) [prenom] [nom] Vrg(,) Vrb(est)  Tel([tel_tra])  )

FORMAT  tel (nom, prenom) ( Det(LES) Nom(NUME6RO) Pcn(DE) Nom(TE6LE6PHONE) Vrg(,) |
Inc(CELUI) Pcn(DE) [prenom] [nom] Vrg(,) Aux(est)  Vrg(,)  Tel([tel_tra])  )

FORMAT  tel2 (nom, prenom) ( Det(LE) Nom(NUME6RO) Pcn(DE) Nom(TE6LE6PHONE)
Pcn(DE) [prenom] [nom] | Vrb(est)  Tel([tel_tra])  )

FORMAT  tel_fax(nom, prenom,tel_tra) ( Det(LE) Nom(N°) Pcn(DE) Nom(téléphone)
Pcn(DE) [prenom] [nom] Vrb(est) Tel([tel_tra]) | Inc(et le N° de fax est) Tel([fax]) )

FORMAT  tel_fax3(nom, prenom, tel_tra, fax) ( Det(LE) Nom(NUME6RO) Pcn(DE)
Nom(TE6LE6PHONE) Pcn(DE) [prenom] [nom] Vrb(est) Det(LE) [tel_tra] | Inc(et) Det(LE)
Nom(NUME6RO) Pcn(DE) Nom(@FAX) Vrb(est) Tel([fax]) )

FORMAT adr (nom, prenom) ( L( le lieu de résidence de ) [prenom] [nom] Inc(_) Aux(EST)
Inc(_) [lieu] [*code] [*ville] | [*pays])

FORMAT travail (nom, prenom) ( L( le lieu de travail de ) [prenom] [nom] Inc(_) Aux(EST)
Inc(_) [lieu] [*code] [*ville] | [*pays])
```

ANNEXE 5 : Fichier de déclaration des objets.

OBJETS.CPS

```
/*****************************************************************************
   Fichier COMPOST : OBJETS.CPS
   But :             DECLARATION DES OBJETS
   Auteur :          G. BAILLY
   Date :            14-Juin-88
   Rques:            Le tilde ~ doit être déclaré comme objet (BLANC)
 *****************************************************************************/

/*** Declaration des objets ***/
classe phrase
        objet (Phrase)
        indice (type dans [0, 2, 0])
finclasse

classe marque
        objet (Nul,Id,It,Dg,Dd)
        trait (faible)
        Id,It,Dg,Dd est -faible;
finclasse


classe champ
        objet (Date,Heu,Tel,Fax,Art)
finclasse


classe mot
        objet (Inc,Adq,Adi,Adv,Aux,Cco,Ccs,Det,Inf,Li,Ne,Nom,Npr,Num,
               Pas,Pcn,Pnp,Pp,Pps,Ppt,Pre,Pri,Prl,Pt,Vo,Vrb,Vrg)
        trait (CONTENU,SUJ,ACC,DAT,GEN,MAS,FEM,NEU,1ER,2ME,3ME,SNG,PLU,
               AVOIR,ETRE)
        Aux,Cco,Ccs,Det,Li,Ne,Pas,Pcn,Pp,Pre,Pri,Prl,Pt,Vrg est -CONTENU;
        Adq,Adi,Adv,Inf,Nom,Npr,Num,Pps,Ppt,Vrb,Pnp,Vo est CONTENU;
finclasse

classe syllabe
        objet (Syl)
        trait (acc)
        indice ( nbp dans [0 ,20 ,0] )
        Syl est -acc;
finclasse

classe phon
        objet (a,e,i,u,y,o,x,q,x^,e^,o^,a^,
               a~,e~,o~,x~,
               p,t,k,b,d,g,f,s,s^,v,z,z^,
               j,w,h,l,r,
               m,n,g~,n~,_)
```

```
trait (voc,voi,nas,occ,cns,liq,ouv,lien)
indice (duree dans [20,500,100],
        enr   dans [  0, 80,  0],
        mf0   dans [-50,  1,  1],
        ok    dans [  0,  1,  0] )

a   est  ouv    vaut duree= 90, enr=29;
e               vaut duree= 80, enr=22;
i   est -ouv    vaut duree= 70, enr=19;
u   est -ouv    vaut duree= 70, enr=20;
y   est -ouv    vaut duree= 70, enr=22;
o               vaut duree= 90, enr=28;
x               vaut duree= 80, enr=26;
q               vaut duree= 70, enr=20;
e^              vaut duree= 90, enr=26;
o^  est  ouv    vaut duree= 90, enr=27;
a^  est  ouv    vaut duree= 90, enr=29;
x^              vaut duree= 80, enr=28;
a~  est  ouv    vaut duree=110, enr=27;
e~              vaut duree=110, enr=28;
o~  est  ouv    vaut duree=110, enr=26;
x~              vaut duree=110, enr=27;
j   est -ouv    vaut duree= 60, enr=21;
w   est -ouv    vaut duree= 45, enr=18;
h   est -ouv    vaut duree= 50, enr=18;


p  vaut duree= 60;
t  vaut duree= 70;
k  vaut duree= 70;
b  vaut duree= 70, enr=14, mf0 = -10;
d  vaut duree= 70, enr=12, mf0 = -15;
g  vaut duree= 70, enr=12, mf0 = -20;
v  vaut duree= 80, enr=13, mf0 = -10;
z  vaut duree= 80, enr=12, mf0 = -15;
z^ vaut duree= 90, enr=22, mf0 = -20;
f  vaut duree= 90, enr=11;
s  vaut duree= 90, enr= 9;
s^ vaut duree=100, enr=26;
l  vaut duree= 50, enr=20;
r  vaut duree= 45, enr=17, mf0 = -10;
m  vaut duree= 80, enr=25, mf0 = -10;
n  vaut duree= 80, enr=24;
g~ vaut duree= 70, enr=20;
n~ vaut duree= 70, enr=20;
_  vaut duree= 50, enr=0;

a, e, i, u, y, o, x, q, e^, o^, a^, x^
       est  voc, voi,-nas,-occ,-cns,-liq;
a~, e~, o~, x~
       est  voc, voi, nas,-occ,-cns,-liq;
b,d,g  est -voc, voi,-nas, occ,-cns,-liq;
p,t,k  est -voc,-voi,-nas, occ,-cns,-liq;
v,z,z^ est -voc, voi,-nas,-occ, cns,-liq;
f,s,s^ est -voc,-voi,-nas,-occ, cns,-liq;
l,r    est -voc, voi,-nas,-occ,-cns, liq;
j,w,h  est  voc, voi,-nas,-occ,-cns, liq;
m,n    est -voc, voi, nas,-occ,-cns,-liq;
g~,n~  est -voc, voi, nas, occ,-cns,-liq;
_      est      -voi,-nas,-occ,-cns,-liq;

finclasse
```

```
classe graph
        objet (A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z,
               0,1,2,3,4,5,6,7,8,9,',",!,?,:,~,¦,°,
               \. ,\, ,\$ ,\% ,\+ ,\- ,\* ,\/ ,\=, \(, \), \; )
        trait (maj,voy,let,accent,chif,pct,cnas,cliq,chif60,chif80,chif10,chif11)
        A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z,
        0,1,2,3,4,5,6,7,8,9,',",!,?,:,
        \; ,\. ,\, ,\$ ,\% ,\+ ,\- ,\* ,\/ ,\=, \(, \), ~ est -maj;
        0,1,2,3,4,5 est -chif60,-chif80,-chif10;
        0,2,3,4,5 est -chif11;
        1,7,9 est chif11;
        7,9 est chif10;
        8,9 est chif80,-chif60;
        6,7 est chif60,-chif80;
        6,8 est -chif10,-chif11;
        M,N est cnas;
        R,L est cliq;
        B,C,D,F,G,H,J,K,L,P,Q,R,S,T,V,W,X,Z est -cnas;
        B,C,D,F,G,H,J,K,M,N,P,Q,S,T,V,W,X,Z est -cliq;
        0,1,2,3,4,5,6,7,8,9 est chif,-let,-pct,-voy;
        2,3,4,5,6 est let,accent;       /* ACCENTS ET CEDILLE */
        B,C,D,F,G,H,J,K,L,M,N,P,Q,R,S,T,V,W,X,Z est let,-chif,-pct,-voy,
            -chif10,-chif80,-chif60,-accent;
        A,E,I,O,U,Y est let,-chif,-pct,voy,-chif10,-chif80,-chif60,-accent;
        \;,",',!,?,:,\.,\,,\$,\%,\+,\-,\*,\/,\=,\(,\),~  est -voy,-let,-chif,pct,
            -chif10,-chif80,-chif60;
finclasse

/* les types de trajectoires sont indiqués par les objets génériques */
classe cible
  objet (X0,X1,X3,X2G,X2D)
  trait (final)
  indice (/* PARAMETRES FORMANTIQUES */
          F0_BASE dans [  60, 500, 60],
          F1 dans [   0,5000,    0],
          F2 dans [   0,5000,    0],
          F3 dans [   0,5000,    0],
          F4 dans [   0,5000,    0],
          F5 dans [   0,5000,    0],
          B1 dans [   0, 300,    0],
          B2 dans [   0, 300,    0],
          B3 dans [   0, 300,    0],
          B4 dans [   0, 400,    0],
          B5 dans [   0, 500,    0],
          A2 dans [ -10,  50,  -10],
          A3 dans [ -10,  40,  -10],
          A4 dans [ -10,  30,  -10],
          A5 dans [ -10,  20,  -10],
          U0 dans [ -10,  60,  -10],
          Q0 dans [   0, 100,    0],
          Q0 dans [   0, 100,    0],
          AF dans [ -10,  70,  -10],
          AH dans [ -10,  50,  -10],
          FNZ dans [   0,1000,   0],
          FNP dans [   0,2000,   0],
          BNZ dans [   0, 400,   0],
          BNP dans [   0, 400,   0],
          MICRO dans [ -50, 50,  0],
          ACCENT dans [ 0, 100,  0],
          /* POSITIONNEMENT DES CIBLES */
          INSTANT dans [-100,1000,  0])
finclasse
```

ANNEXE 6 : Fichiers du scénario COMPOST.

SCENARIO.CPS

```
/*************************************************************************
   Fichier COMPOST : SCENARIO.CPS
   But :             Scénario du serveur de messages (agenda)
   Auteur :          Paul  TRIPODI
   Date :            1  Juin  1991
   Remarques :       Ce fichier appelle 3 autres fichiers (#include) :
                                    TOPF_I.CPS
                                    SYLF_I.CPS
                                    GENF_I.CPS
 *************************************************************************/


call TRON ;

call Init_Base ;     /* Initialisation du serveur  */


CORPS


 call Base ;         /* Appel des fonctions du serveur */


/*---------------*/   /*  Début des GRAMMAIRES  (régles)  */


  grammaire TEL2     [ graph -> graph / graph + graph , 1 ]
     gr_3    =   [chif] [chif]  ->   Inc(¦) Num(#1#2)  / Tel([chif]*[0,20]  + [chif]*[0,20] ) ;
     gr_1    =   [chif])  ->  ) Num( ¦ #1)  / Tel( [chif]*[0,20] +  ;
  fingrammaire


  grammaire PONCT     [ graph -> graph / graph + graph , 1 ]
     pg_in =  ?   -> ) Pt(?? / graph +
                  -> ?? /  +       ;
     pp_in =  ?   -> ) Pt(?? / phon + ;
     pauex =  !   -> ) Pt(!! / graph + ;
     paupt =  \.  -> ) Pt(\.  / [let]*[2,2] + ;
  fingrammaire


  grammaire AJOUT [ graph -> phon / graph + graph , 1 ]
     pau_? = ? -> <_,duree=200>  / + ?  ;
     pau_! = ! -> <_,duree=200>  / + !  ;
  fingrammaire
```

```
/* Intégration du fichier contenant les règles de transcription orthographique-phonétique */

  #include TOPF_I.CPS


  grammaire PAUSE_FIN [ graph -> phon / graph + graph , 1 ]
     pau_p =  phon _)  ->  #1) Inc(#2)     / +  ##  ;
  fingrammaire


  grammaire ELISION [phon -> phon / phon + phon , 1]
     elQ   = q ->     / [voc,-liq][-voc]*[0,20] + )
               -> x^ ;
     elnum = Num() -> ;
  fingrammaire


/* Fichier contenant les règles de syllabation */

  #include SYLF_I.CPS


/* Fichier contenant les règles de génération des trajectoires */

  #include GENF_I.CPS


  grammaire BASE_F0 [phon -> phon / phon + phon , 1]
     deb_p   = mot(Syl (phon(#A)
               -> #1(#2(#3 (<X1, F0_BASE=140, MICRO=-1, ACCENT=1, INSTANT=10> #A) / ## +    ;

     fdec1   = Syl((-_)(#A)))
               -> #1(#2 (<X2D, F0_BASE=120,INSTANT=0>
                  <X1,[final], F0_BASE=90,INSTANT=0> #A)))  / + (mot(Syl(_))  ##, ##) ;

     fdec2   = Syl(phon(#A) phon*[0,3] phon(#B)))
               -> #1(#2 (<X2D, F0_BASE=120,INSTANT=0> #A) #3
                  #4(#B <X1,[final], F0_BASE=90,INSTANT=0>))) / + (mot(Syl(_)) ##, ##) ;

     fint2   = Syl(phon(#A) phon*[0,5])) mot( Syl (phon(#B)  phon*[0,1]
               -> #1(#2 (<X2G, F0_BASE=120,INSTANT=0> #A) #3)
                  #4(#5 (#6(#8<X1, F0_BASE=170,INSTANT=0>) #7  / + ?) ;
  fingrammaire


  grammaire FIN [graph -> phon / phon + phon , 1]
     fin?  =  ? ->     ;
     fin!  =  ! ->     ;
  fingrammaire


  grammaire DureesP [phon -> phon / phon + phon , 1 ]
     DurP= phon(#A) -> <#1,duree=duree*1.6> (#A);
  fingrammaire
```

```
  grammaire MICRO_ACCENT [phon -> phon / phon + phon , 1]
     mic_p = < phon , [-voc]  , mf0 < 0 , #m = mf0 , #d= duree > (#A)
             -> #1 ( #A <X2D , MICRO = -1 , INSTANT = 1 >
                        <X2G , MICRO = #m , INSTANT = #d/2  >
                        <X1  , [final] , MICRO = -1 , INSTANT = -5 > ) ;


     acc_t = < (p, t, k) ,  #t = 15 , #d= duree > (#A)
             -> #1 ( #A <X2D , ACCENT = 1 , INSTANT = 1 >
                        <X2G , ACCENT = #t , INSTANT = #d/2  >
                        <X1  , [final] , ACCENT = 1 , INSTANT = -5 > ) ;
  fingrammaire


  grammaire PAUSE [graph -> phon / phon + phon , 1]
/**** GENERATION DU Silence ****/
Arret =  ¦ -> <_,duree = 500>(<X1,INSTANT=10,U0=-7,AF=1,AH=1>) ;
  fingrammaire


  grammaire PAUSE2 [phon -> phon / phon + phon , 1]
/**** GENERATION DU Silence ****/
ART =  _ -> <_,duree = 500>(<X1,INSTANT=10,U0=-7,AF=1,AH=1>)  /  Art( phon*[0,10] + phon*[0,10] ) ;
  fingrammaire


/*-----------------*/   /*  Fin des GRAMMAIRES  (règles)  */


  call Gen_klatt(784);   /*  Appel de la fonction synthètiseur de "KLATT"
                             (et contrôle de la carte AU21), l'adresse est 784 (10) */


  /*call TRACER;*/


/*----------------*/


FINCORPS
```

Eléments du fichier de traduction orthographique-phonétique.

TOPF_I.CPS

```
/*************************************************************************
   Fichier COMPOST : TOPF_I.CPS
   Type :            INCLUDE
   But :             ABREVIATIONS, NOMBRES ET TRADUCTION GRAPHEME-PHONEME
   Langue :          FRANCAIS
   Auteur :          G. BAILLY
   Date :            20-Juillet-90
 *************************************************************************/

/*** GRAMMAIRE DE TRADUCTION GRAPHEMES -> PHONEMES ***/

grammaire TOPF [graph -> phon / graph + graph , 1]

/*** TRADUCTION DE 0 ***/
cf0 =  0 -> zero / Num( + )
       -> )Num(di)Num(mil)Num( / [chif11] + [chif]*[3])
       -> )Num(mil)Num(        /          + [chif]*[3])
       -> )Num(dis             / [chif11] + )
       -> ;

/*** TRADUCTION DU 1 ***/
cf1 =  1 -> x~ / Num( + )
       -> )Cco(e^)Num(x~)Num(mil)Num(  / (2,3,4,5,6) + [chif]*[3])
       -> )Cco(e^)Num(x~               / (2,3,4,5,6) + )
       -> )Cco(e^)Num(o~z)Num(mil)Num( / 7           + [chif]*[3])
       -> )Num(o~z)Num(mil)Num(        / (1,9)       + [chif]*[3])
       -> )Num(x~)Num(mil)Num(         / (8,0) + [chif]*[3])
       -> )Num(mil)Num(                /       + [chif]*[3])
       -> )Num(sa~)Num(                /       + ([chif]*[5],[chif]*[2]))
       -> )Cco(e^)Num(o~z               / 7     + )
       -> )Num(o~z                     / (9,1) + )
       -> ;

   "
   "
   "
c60 =  [chif60] -> )Num(swasa~t)Num(      / + ([chif],[chif]*[4])) ;
c80 =  [chif80] -> )Num(katr)Num(ve~)Num( / + ([chif],[chif]*[4])) ;

/*** TRADUCTION DU 6 ***/
cf6 =  6 -> )Num(se^z)Num(mil)Num( / [chif11] + [chif]*[3])
       -> )Num(si)Num(mil)Num(  /          + [chif]*[3])
       -> )Num(si)Num(sa~)Num(  /          + ([chif]*[2],[chif]*[5]))
       -> )Num(sx^z)Num(        / [chif11] + )
       -> )Num(sis ;

/*** TRADUCTION DU 7 ***/
cf7 =  7 -> )Num(dis)Num(sx^t)Num(mil)Num( / [chif11] + [chif]*[3])
       -> )Num(sx^t)Num(mil)Num(         /          + [chif]*[3])
       -> )Num(sx^t)Num(sa~)Num(         /          + ([chif]*[2],[chif]*[5]))
       -> )Num(dis)Num(sx^t)Num(         / [chif11] + )
       -> )Num(sx^t ;
```

```
     "
     "
     "

/*** ABREVIATIONS ***/
abr1 = mot(ST)        -> Adq(se~) ;
abr2 = mot(STE)       -> Adq(se~t) ;
abr3 = mot(MR)        -> Nom(mx^sjx^) ;
abr4 = mot(M(E,)LLE) -> Nom(madmwaze^l) ;
abr5 = mot(DR)        -> Nom(do^ktx^r) ;
    "
    "
abr17= mot(\=)        -> Nom(egal) ;
abr18= mot(\$)        -> Nom(do^lar) ;
abr19= mot(\%)        -> Nom(pursa~) ;
abr20= mot(N°)        -> Nom(nymero) ;
abr21= mot(1ER)       -> Nom(prxmie) ;


/*** DICTIONNAIRE D'EXCEPTIONS ***/

/*** LES APOSTROPHES ***/
aposC = mot(C')mot( -> #4(s ;
aposD = mot(D')mot( -> #4(d ;
aposJ = mot(J')mot( -> #4(z^ ;
     "
     "
     "

/*** EPPELATION ***/
eB    = B -> be       / ( mot( , \. ) + \. ;
eC    = C -> se       / ( mot( , \. ) + \. ;
eD    = D -> de       / ( mot( , \. ) + \. ;
eF    = F -> e^f      / ( mot( , \. ) + \. ;
eG    = G -> z^e      / ( mot( , \. ) + \. ;
eH    = H -> as^      / ( mot( , \. ) + \. ;
    "
    "
    "

/*** EXCEPTIONS ***/
exA   = A    -> a    / + (MSTER,MM,NN,I4)          /* AMMONIAQUE */
             -> e^   / + Y[+voy]                   /* PAYE */
             -> o^   / H + LL                      /* HALL */
             -> a    ;                             /* PAPA */
exAYE = AYE  -> e^ji / BB +                        /* ABBAYE */
             -> aj   / B + ;                       /* COBAYE */
exAY  = AY   -> aj   / M + ON                      /* MAYONNAISE */
             -> e^ji / P + S ;                     /* PAYSAGE */
exAI  = AI   -> ai   / + 4                         /* LAI4C */
             -> a    / + (LL,LS*[0,1]))            /* RAILLER */
             -> x^   / F + S[voy]                  /* FAISONS */
             -> e^   ;                             /* FAIRE */
exAI3 = AI3  -> e^   ;                             /* CONNAI3TRE */
exAU  = AU   -> o    ;                             /* SAUR */
    "
    "
    "
    "
```

```
exCH  = CH   ->    / Nom(ALMANA +                    /* ALMANACH */
             -> k  / + (R,L,N)                       /* CHLORE */
             -> s^ / Nom(TRA + E6E                   /* TRACHE6E */
             -> k  / (TRA,PSY) +                     /* TRACHE6I4TE */
             -> k  / (AR,STO) + (E6,A)               /* ARCHE6OLOGUE */
             -> k  / (ON,E6,DI) + O                  /* E6CHOGRAPHIE */
             -> k  / OR + (ESTRE,IDE6)               /* ORCHIDE6E */
             -> k  / + E6O                           /* CHE6OPS */
             -> k  / (E,O) + S*[0,1])                /* VARECH */
             -> k  / + A(O,RIS,LD,LC,NA)             /* CHAOS */
             -> k  / E6 + (IL,IRO,L,M,N)             /* CHE6NOPODE */
             -> s^ / + IROUBLE                       /* CHIROUBLE */
             -> k  / + I(ANTI,ASMA,RO,RA,T)          /* CHIANTI */
             -> k  / + O(E,L,N,R)                    /* CHOEUR */
             -> k  / JOA + I(N,M)                    /* JOACHIM */
             -> s^ ;                                 /* CHAT */

CHEN  = CHEN -> ke^n / LI + ;                        /* LICHEN */
exC5  = C5   -> s ;                                  /* BESANC5ON */
    "
    "

exI   = I    -> i   / CR + AILLE                     /* CRIAILLERIE */
             ->     / ILL + [voy]                    /* QUINCAILLIER */
             -> j   / (LL,RR,TH) + [voy]             /* RALLIER */
             -> ij  / [-voy,-cnas](R,L) + ER         /* CRIER */
             -> i   / + ES*[0,1])                    /* INERTIE */
             -> i   / (Q,G)U +                       /* ANGUILLE */
             -> ij  / + ENTS                         /* PATIENT */
             ->     / (U,U2,U3,A,A2,A3) + LL         /* ANGUILLE */
             -> ij  / [-cliq][cliq] + [voy]          /* CRIA */
             -> j   / + <-E,[voy]>                   /* LIA */
             -> i   ;                                /* PIMENT */
     "
     "

/*** EXCEPTIONS DU P ***/
exP   = P    -> p / mot((CA,STO) +                   /* CAP */
             -> / + S*[0,1])                         /* COUP */
             -> / P +                                /* FRAPPER */
             -> p ;                                  /* PAPA */
exPT  = PT   -> t / (BA,COM) +                       /* COMPTER */
             -> t / mot(S + (IE2ME,))                /* SEPT */
             -> pt / mot(DOM +                       /* DOMPTER */
             -> / + S*[0,1]) ;                       /* PROMPT */
exPH  = PH   -> f ;                                  /* PHRASE */
     "
     "

/*** EXCEPTIONS DU Y ***/
aY    = Y    -> ai / DR +                            /* DRY */
             -> j / + [voy]                          /* PAYONS */
             -> i ;                                  /* PYTHON */
exYN  = YN   -> e~ / + [-voy] ;                      /* SYNTHE2SE */
exYM  = YM   -> im / G + N                           /* GYMNASTIQUE */
             -> e~ / + [-voy] ;                      /* THYM */
    "
    "

fingrammaire
```

Eléments du fichier de génération des trajectoires.

GENF_I.CPS

```
/*************************************************************************
   Fichier COMPOST : GENF_I.CPS
   Type :            INCLUDE
   But :             GENERATION DES TRAJECTOIRES POUR SYNTHETISUR A FORMANTS
   Langue :          FRANCAIS
   Auteur :          G. Mhania
   Date :            20-Juillet-90
 *************************************************************************/


grammaire Combinaison [phon -> phon / phon + phon , 1]
nj=  <g~,#d=duree> -> <n,duree=#d*0.65> <j,duree=#d*0.35> ;
fingrammaire


grammaire Generation [phon -> phon / phon + phon , 1]


/**** GENERATION DU a ****/
G_a=  a  -> #1(<X1,INSTANT=30,
                U0=40,AF=0,QD=32,AH=0,
                FNP=900,BNP=500,FNZ=900,BNZ=500,QO=80,
                F1=622, B1=75,
                F2=1244,B2=60,A2=37,
                F3=2510,B3=154,A3=24,
                F4=3330,B4=136,A4=18,
                F5=4296,B5=189,A5=7>);

/**** GENERATION DU u ****/
G_u=  u  -> #1(<X1,INSTANT=30,
                U0=39,AF=0,AH=0,QD=18,
                FNP=900,BNP=500,FNZ=900,BNZ=500,QO=80,
                F1=244, B1=60,
                F2=752, B2=60,A2=37,
                F3=2108,B3=94,A3=13,
                F4=3053,B4=111,A4=10,
                F5=4152,B5=196,A5=5>);

/**** GENERATION DU e ****/
G_e=  e  -> #1(<X1,INSTANT=30,
                U0=40,AF=1,AH=0,QD=29,
                FNP=900,BNP=500,FNZ=900,BNZ=500,QO=80,
                F1=319, B1=55,
                F3=2000,B3=84,A3=20,
                F2=2700,B2=80,A2=26,
                F4=3368,B4=102,A4=19,
                F5=4346,B5=132,A5=9>);
```

```
/**** GENERATION DU o ****/
G_o=  o  -> #1(<X1,INSTANT=30,
                U0=36,AF=0,AH=0,QD=28,
                FNP=900,BNP=500,FNZ=900,BNZ=500,QO=80,
                F1=370,B1=53,
                F2=797,B2=63,A2=31,
                F3=2260,B3=116,A3=11,
                F4=3000,B4=100,A4=11,
                F5=4059,B5=161,A5=4>);

/**** GENERATION DU i ****/
G_i=  i  -> #1(<X1,INSTANT=30,
                U0=35,AF=0,AH=0,QD=25,
                FNP=900,BNP=500,FNZ=900,BNZ=500,QO=80,
                F1=250, B1=55,
                F3=2000,B3=60, A3=23,
                F2=3000,B2=100,A2=10,
                F4=3400,B4=100,A4=25,
                F5=4490,B5=148,A5=12>);

       "
       "
       "
       "

/**** GENERATION DES NASALES ****/
G_an= a~ -> #1(<X3,INSTANT=20,
                U0=40,AF=20,AH=0,QO=80,QD=25,
                FNP=815,BNP=100,FNZ=900,BNZ=500,
                F1=515, B1=60,
                F2=1350,B2=60, A2=14,
                F3=2100,B3=300,A3=24,
                F4=2768,B4=60, A4=20,
                F5=3270,B5=100,A5=30>
               <X1,INSTANT=40,U0=36,AF=20,AH=0,QO=80,QD=25,
                FNP=825,BNP=20,FNZ=900,BNZ=500,
                F1=530, B1=130,
                F2=1000,B2=150,A2=29,
                F3=2200,B3=300,A3=33,
                F4=2800,B4=100,A4=12,
                F5=3440,B5=140,A5=20>);

G_in= e~ -> #1(<X3,INSTANT=20,
                U0=40,AF=10,AH=0,QO=80,QD=20,
                FNP=900,BNP=500,FNZ=900,BNZ=500>
               <X1,INSTANT=40,
                U0=39,AF=15,AH=0,QO=80,QD=15,
                FNP=1400,BNP=300,FNZ=900,BNZ=100,
                F1=800, B1=150,
                F2=1500,B2=130,A2=32,
                F3=2200,B3=100,A3=25,
                F4=3300,B4=150,A4=30,
                F5=3600,B5=150,A5=30>);

       "
       "
       "
```

```
**** GENERATION du b ****/
G_b=  b  -> #1(<X1, INSTANT=35,
                AF=0,QD=19,AH=0,
                U0=35, FNP=900,BNP=400,FNZ=900,BNZ=400, QO=100,
                F1= 100,B1=40,
                F2= 800,B2=100,A2=10,
                F3=1800,B3=120,A3=20,
                F4=2944,B4=200,A4=20,
                F5=4531,B5=400,A5=0>
               <X0,[final],INSTANT=-10,AF=0>
               <X1,[final],INSTANT=-5,AF=30>
                );

        "
        "
        "

/**** GENERATION du p ****/
/* rque: le VOT est fixe à 5 ms */
G_p=  p  -> #1(<X1,INSTANT=8,
                QD=10,AH=33,
                U0=-5, FNP=900,BNP=400,FNZ=900,BNZ=400,QO=100,
                F1=100, B1=150,
                F2=1112,B2=154,A2=2,
                F3=2181,B3=225,A3=2,
                F4=3247,B4=490,A4=0,
                F5=4363,B5=490,A5=2>
               <X0,[final],INSTANT=-10,AF=0>
               <X1,[final],INSTANT=-5,AF=30>
                );

        "
        "
        "

/**** GENERATION DU f ****/
G_f=  f  -> #1(<X1,INSTANT=20,
                U0=0,AF=26,AH=5,QO=100,QD=10,
                FNP=900,BNP=400,FNZ=900,BNZ=400,
                F1=501, B1=200,
                F2=1318,B2=200,A2=35,
                F3=2250,B3=200,A3=34,
                F4=3159,B4=200,A4=34,
                F5=4154,B5=223,A5=-5> );
        "
        "
        "

/**** GENERATION DU v ****/
G_v=  v  -> #1(<X1, INSTANT=25,
                AF=0,QD=19,AH=0,
                U0=31, FNP=900,BNP=400,FNZ=900,BNZ=400,QO=80,
                F1=246, B1=80,
                F2=1000,B2=150,A2=15,
                F3=2005,B3=160,A3=7,
                F4=2944,B4=200,A4=8,
                F5=4531,B5=400,A5=-5> );

       "
       "
       "
```

```
/**** GENERATION DU r ****/
G_r=  <r,[-voi]> -> #1(<X1,INSTANT=20,U0=30,AF=26,AH=0,QO=80,QD=11,
                  FNP=900,BNP=500,FNZ=900,BNZ=500,
                  F1=249, B1=60,
                  F2=1108,B2=86, A2=33,
                  F3=2300,B3=139,A3=26,
                  F4=3431,B4=182,A4=38,
                  F5=4747,B5=217,A5=8> );

G_rnv= <r,[voi]> -> #1(<X1,INSTANT=30,U0=33,AF=15,AH=0,QO=80,QD=15,
                  FNP=900,BNP=500,FNZ=900,BNZ=500,
                  F1=249, B1=90,
                  F2=1108,B2=90, A2=33,
                  F3=2300,B3=140,A3=24,
                  F4=3431,B4=160,A4=30,
                  F5=4747,B5=200,A5=5> );

/**** GENERATION du l  ****/
G_l=   l  -> #1(<X1,INSTANT=30,
                   AF=11,QD=22,AH=0,QO=80,U0=31,
                   FNP=900,BNP=400,FNZ=900,BNZ=400,
                   F1=261,B1=90,
                   F2=1647,B2=122,A2=20,
                   F3=2104,B3=100,A3=15,
                   F4=3460,B4=143,A4=6,
                   F5=4000,B5=191,A5= 2> );


/**** GENERATION du j  ****/
G_j=   j  -> #1(<X1,  INSTANT=30,
                   AF=15,QD=18,AH=0,QO=50,U0=32,
                   FNP=900,BNP=400,FNZ=900,BNZ=400,
                   F1=300,B1=85,
                   F2=2100,B2=82,A2=31,
                   F3=2500,B3=100,A3=18,
                   F4=3236,B4=107,A4=17,
                   F5=3691,B5=151,A5= 17> );

        "
        "
        "

/**** GENERATION DE n  ****/
G_n= n  -> #1(<X1,INSTANT=10,
                  AF=0,AH=0,QO=100,QD=26,U0=37,
                  FNP=900,BNP=500,FNZ=900,BNZ=500,
                  F1=245, B1=87,
                  F2=1450,B2=50,A2=15,
                  F3=2600,B3=154,A3=20,
                  F4=3363,B4=132,A4=9,
                  F5=4200,B5=160,A5=2>);

        "
        "
        "

/**** GENERATION DU Silence ****/
G_Sil=  _ -> #1(<X1,INSTANT=10,U0=-5,AF=1,AH=1>);


fingrammaire
```

```
grammaire Coarticulation [phon -> phon / phon + phon , 1 ]
/* toute occlusive entre deux voyelles voit son locus (F1..F3) modifie par
   un barycentre de son locus intrinseque et les cibles formantiques des
   voyelles affectees des coefficients 3, 1 et 1 */
Occ=      [voc](<cible,#a1=F1,#a2=F2,#a3=F3>)
          [occ](cible cible*[2])
          [voc](<cible,#p1=F1,#p2=F2,#p3=F3>)
          ->
          #1(#2)
          #3(<#4,F1=(3*F1+#a1+#p1)/5,
                 F2=(3*F2+#a2+#p2)/5,
                 F3=(3*F3+#a3+#p3)/5> #5 )
          #6(#7);
/* le silence en debut de phrase recupere les cibles formantiques du
   phoneme qui suit */
Sild=     _(cible) phon (<cible,#f1=F1,#f2=F2,#f3=F3,#f4=F4,#f5=F5> ->
          #1(<#2,F1=#f1,F2=#f2,F3=#f3,F4=#f4,F5=#f5>) #3 (#4 / ## + ;
/* le silence en fin de phrase recupere les cibles formantiques du
   phoneme qui precede */
Silf=     phon (cible*[0,10] <cible,#f1=F1,#f2=F2,#f3=F3,#f4=F4,#f5=F5>) _(cible ->
          #1 (#2#3) #4(<#5,F1=#f1,F2=#f2,F3=#f3,F4=#f4,F5=#f5> / + ## ;

fingrammaire


grammaire Duplication [phon -> phon / phon + phon , 1 ]
/* positionnement des cibles finales :
   pour les occlusives on impose AF=0 au debut
   pour les voyelles nasales, on duplique la deuxieme cible
   pour les voyelles nasales, on ne touche pas à la premiere cible
   pour les autres, on duplique la premiere */
Dup= cible -> <#1,AF=0> <#1=X1,[final],INSTANT=-INSTANT> / [occ] ( +
           -> #1 <#1=X3,[final],INSTANT=-20> / [nas,voc] ( cible + )
           -> #1 / [nas,voc] ( +
           -> #1 <#1=X3,[final],INSTANT=-INSTANT> / phon ( + ;
fingrammaire

grammaire Explosion [phon -> phon / phon + phon , 1 ]

/* on preleve les parametres d'excitation de la premiere cible de la voyelle
   suivant une occlusive non voisee et on fait demarrer ces parametres au
   tout debut de la voyelle (evite la transition molle des formants) : permet
   de percevoir les transitions formantiques */
Expl= <cible,#u0=U0,#qd=QD,#af=AF> -> <X0,U0=#u0,QD=#qd,AF=#af,INSTANT=0>
                                      #1 / [occ] [voc] ( + ;

fingrammaire
```

Eléments du fichier de syllabation.

SYLF_I.CPS

```
/*****************************************************************************
   Fichier COMPOST : SYLF.CPS
   Type :            INCLUDE
   But :             SYLLABATION
   Langue :          FRANCAIS
   Auteur :          G. BAILLY
   Date :            20-Juillet-90
 *****************************************************************************/


grammaire Syllabation [phon -> phon / phon + phon , 1 ]

lim=   phon        -> Syl(#1) /       mot( + )
                   -> Syl(#1  /       mot( +
                   -> #1)     /            + ) ;
fin=   [-voc]      -> #1      /            + [-voc]*[0,3] ) ;
excs=  [-liq,-voc] -> #1)Syl( /            + [-liq,-voc] [-voc]*[0,3] [voc];
sv=    [voc,liq]   -> )Syl(#1 /      [voc] + [voc] ;
exr=   [liq,-voc]  -> #1)Syl( /            + [-voc];
def=   phon        -> )Syl(#1 / [voc,-liq] + ;

fingrammaire
```

Bibliographie.

[1] ALISSALI Mamoun et BAILLY Gérard, *COMPOST : un serveur multilingue*, 8e Congrès R.F.I.A'91, LYON Villeurbanne, novembre 1991.

[2] AWAD Selim, *Synthétiseur à formant en temps réel : étude d'une source d'excitation élaborée, codage optimum des paramètres de commande,* Thèse de Docteur Ingénieur, Institut National Polytechnique de Grenoble, décembre 1983.

[3] BADIN Pierre, *Analyse de la synthèse à formants. Application à la synthèse des contoïdes constrictives du français. Discution d'une méthode de détermination des indices accoustiques de la parole*, Thèse de Docteur Ingénieur, Institut National Polytechnique de Grenoble, mars 1983.

[4] BAILLY Gérard, *Un modèle de congruence relationnel pour la synthèse de la prosodie du francais*, 15e J.E.P, Aix en Provence mai 1986, pp. 75-78.

[5] BAILLY Gérard, *Multiparametric generation of french prosody from unrestricted text,* I.C.A.S.S.P. Proc., Vol 4, Tokyo, avril 1986, pp. 2419-2422.

[6] BAILLY G, MURILLO C, AL DAKKAK O, GUERIN B, *A TEXT-TO-SPEECH synthesis system for french by formant synthesis,* SPEECH'88, 7th F.A.S.E. Symposium, Edinburgh, Août 1988, pp. 225-260.

[7] BAILLY G et TRAN A, *A rule-compiler for Speech Synthesis,* Eurospeech 89, Paris, Septembre 1989, pp. 136-139.

[8] BARBE T, *Méthodologie et outils pour la mise en oeuvre automatique d'une synthèse de parole de haute qualité*, Thèse de Docteur Ingénieur, Institut National Polytechnique de Grenoble, décembre 1990.

[9] BARBOSA Plinio Almeida, *Génération Automatique de la prosodie du Français*, Rapport de D.E.A Institut National Polytechnique de Grenoble, 1991.

[10] BAUDRY Marc, Synthèse automatique de la parole, *Les techniques de l'ingénieur*, H1960, septembre 1985.

[11] BOITE René et KUNT Murat, *Traitement de la parole*, Lausanne, Presses polytechniques romandes, 1987.

[12] CALLIOPE (Collectif de 36 auteurs), *La parole et son traitement automatique*, Paris, MASSON, 1989.

[13] CARRE René, La parole artificielle, *La recherche*, Volume 6, **N° 54**, mars 1975, pp. 221-229.

[14] CATIER Eric, La parole : synthèse, *Toute l'életronique*, **N° 490**, janvier 1984, pp. 56-67.

[15] CHARPENTIER F et MOULINES E, Nouvelles techniques de synthèse de la parole, *L'écho des recherches*, **N° 137**, 3e trimestre 1989, pp. 5-13.

[16] CHARPENTIER F et STELLA M, *Synthèse à partir de texte par modification et concatétation de forme d'onde*, 15^{e} J.E.P, Aix en Provence, pp. 7-10.

[17] CHENG Yan Ming, *Etude du concept source-filtre interactif pour la synthèse de la parole : Analyse des voyelles nasales,* Thèse de Docteur Ingénieur, Institut National Polytechnique de Grenoble, septembre 1986.

[18] CINARE François, Terminaux vocaux et synthèse de la parole : des puces parlantes, *Minis et micros*, **n° 125**, septembre 1980, pp. 57-63.

[19] DI CRISTO Albert, *De la microprosodie à l'intonosyntaxe*, Publications Université de Provence, Aix en Provence, 1985.

[20] FANT G. *Voice Source Analysis - A Progress Report,* Speech transmission laboratory - Quartely progress and status report, Stockholm, 1979.

[21] GARCIN Philippe, *Les Systhèmes d'analyse-synthèse de la parole par prédiction linéaire : évaluation et amélioration de leur performances sur un signal bruité,* Thèse de Docteur Ingénieur, Institut National Polytechnique de Grenoble, 1981.

[22] GARDARIN George et VALDURIEZ Patrick, *Relationnal Database and Knowledge Bases,* New York, Addison-Wesley Publishing Company, 1989.

[23] GUERTI Mhania et BAILLY Gérard, *Synthesis-by-rule using compost : modeling resonance trajectories,* Eurospeech'91, Genova (Italie), septembre 1991.

[24] HURSCH C.J. HURSCH J.L, *S.Q.L : Le Langage structuré d'interrrogation*, Paris, MASSON, 1990.

[25] HAKAWATI AL DAKKAK Oumayma, *Extraction automatique de paramètres formantiques guidée par le contexte et élaboration des règles de synthèse,* Thèse de Docteur Ingénieur, Institut National Polytechnique de Grenoble, juillet 1988.

[26] KERNIGHAN Brian W. et RITCHIE Dennis M, *Le langage C*, Paris, Masson, 1990.

[27] KLATT Dennis H. ; Software for a cascade/parallel formant synthesizer, *The Journal of the Acoustical Society of America,* Volume 67, **N° 3**, mars 1980.

[28] LILEN Henri, *dBASE IV*, Paris, Edition Radio, 1989.

[29] MAREE Christian et LEDANT Guy, *S.Q.L. Initiation, Programmation et Matrise*, Paris, Armand Colin, 1989.

[30] MARTIN Philippe, *Structure prosodique et structure rythmique pour la synthèse*, 15e J.E.P, Aix en Provence, Mai 1896, pp. 89-91.

[31] POULTON A.S, *Microcomputer speech synthesis and recognition*, Sigma Technical Press, 1983.

[32] QUACH TUAN Ngoc, *Réalisation d'un synthétiseur à formants numérique en temps réel. Caractérisation de la source d'excitation et des transitions d'amplitude,* thèse de Docteur Ingénieur, Institut National Polytechnique de Grenoble, septembre 1986.

[33] QUACH TUAN Ngoc et GERIN Bernard, *Synthétiseur à formant en temps réel : Caractérisation*, 15^{e} J.E.P, Aix en Provence mai 1986, pp. 3-6.

[34] ROUGÉ Daniel, *Nouveau ménemto DBase IV*, Paris, SYBEX, 1989.

[35] ROSS Steven C, *Understanding and Using dBase III plus*, St Paul, West publishing Company, 1987.

[36] SCHAFER Ronald W. et RABINER Lawrence R., System for accoustic formant analysis of voiced speech, *The Journal of the Acoustical Society of America*, Volume 47, **N°2**, 1970.

[37] SÉRIGNAT Jean-François, *Contribution aux recherches sur la communication parlée : travaux sur le vocodeur à autocorrélation, étude et simulation d'un vovodeur à prédiction linéaire*, Thèse de Docteur Ingénieur, Institut National Polytechnique de Grenoble, 1974.

[38] SORIN Christel, *Speech synthesis applications and research : present and future*, 13th International Congress on Acoustics, Belgrade, 1989, pp. 45-55.

[39] SORIN Christel et PERON Roger, *Synthèse à partir du texte et réseau téléphonique : expériences en France*, Conférence AFCET, Paris la Villette, 19 décembre 1989, pp. 59-66.

[40] STELLA M, Synthèse de la parole, *L'écho des recherches*, **N° 115**, 1er trimestre 1984, pp. 21-31.

[41] TREMOLIERE jacques, La synthèse de la parole, *Electronique application*, **N° 62**, pp. 13-27.

[42] YAMASHITA Yoichi, MIZUTANI Naoki et MIZOGUCHI Riichiro, *Concept description for synthetic speech output system*, ETRW on Speech Synthesis, Autrans septembre 1990, pp. 241-244.

[33] QUACH TUAN Ngoc et GERIN Bernard, *Synthétiseur à formant en temps réel : Caractérisation*, 15^{e} J.E.P, Aix en Provence mai 1986, pp. 3-6.

[34] ROUGÉ Daniel, *Nouveau ménemto DBase IV*, Paris, SYBEX, 1989.

[35] ROSS Steven C, *Understanding and Using dBase III plus*, St Paul, West publishing Company, 1987.

[36] SCHAFER Ronald W. et RABINER Lawrence R., System for accoustic formant analysis of voiced speech, *The Journal of the Acoustical Society of America*, Volume 47, **N°2**, 1970.

[37] SÉRIGNAT Jean-François, *Contribution aux recherches sur la communication parlée : travaux sur le vocodeur à autocorrélation, étude et simulation d'un vovodeur à prédiction linéaire*, Thèse de Docteur Ingénieur, Institut National Polytechnique de Grenoble, 1974.

[38] SORIN Christel, *Speech synthesis applications and research : present and future*, 13th International Congress on Acoustics, Belgrade, 1989, pp. 45-55.

[39] SORIN Christel et PERON Roger, *Synthèse à partir du texte et réseau téléphonique : expériences en France*, Conférence AFCET, Paris la Villette, 19 décembre 1989, pp. 59-66.

[40] STELLA M, Synthèse de la parole, *L'écho des recherches*, **N° 115**, 1er trimestre 1984, pp. 21-31.

[41] TREMOLIERE jacques, La synthèse de la parole, *Electronique application*, **N° 62**, pp. 13-27.

[42] YAMASHITA Yoichi, MIZUTANI Naoki et MIZOGUCHI Riichiro, *Concept description for synthetic speech output system*, ETRW on Speech Synthesis, Autrans septembre 1990, pp. 241-244.