Compost : un serveur de synthèse de parole multilingue

Compost : a Server for Multilingual Text-to-Speech Synthesis

Gérard BAILLY
Institut de la Communication Parlée (ICP),
INPG/Université Stendhal.
UA CNRS n° 368
46 avenue Félix Viallet
3801 Grenoble Cedex

G. Bailly est ingénieur de l'ENSER de Grenoble et a obtenu le doctorat-ingénieur en 1983 à l'INPG. Après un séjour à l'INRS-Télécommunications au Canada, il est chargé de recherches au CNRS en 1986 et s'occupe depuis de la synthèse acoustique à l'ICP. Il a co-édité en 1992 un livre intitulé « Talking Machines : theories, models and designs » consacré aux recherches internationales en synthèse de parole. Ses centres d'intérêt touchent tous les domaines de la synthèse de la parole : de l'analyse linguistique formelle à la modélisation articulatoire en passant par le traitement du signal.

Mamoun ALISSALI
Institut de la Communication Parlée (ICP),
INPG/Université Stendhal.
UA CNRS n° 368,
46 avenue Félix Viallet
38031 Grenoble Cedex

M. Alissali a obtenu le diplôme d'Études Spéciales, profil « Systèmes distribués et parallèles », de l'Institut Supérieur des Sciences Appliquées et de Technologie de Damas (Syrie). Il a passé 3 ans à l'ICP où il soutiendra une thèse de doctorat-INPG à la fin de l'année dans la spécialité « Signal, Image, Parole ». Il s'intéresse à la synthèse de la parole et plus particulièrement à son utilisation en génie logiciel.

RÉSUMÉ

Cet article présente la notion de serveur de synthèse et décrit l'environnement COMPOST qui a été créé pour répondre aux concepts que cette notion développe : l'intégration du dialogue vocal dans les interfaces homme-machine ne sera opérationnel que lorsque les architectures matérielles et logicielles offriront aux concepteurs de systèmes de synthèse et aux spécialistes du génie logiciel un moyen souple de communication. COMPOST propose aux premiers un langage de programmation de systèmes de synthèse multilingues et multi-entrées et aux derniers une infrastructure logicielle basée sur la notion de serveur-client.

Chaque système de synthèse pouvant être utilisé par le serveur est décrit par un scénario écrit dans un langage spécialisé. Dans ce scénario qui décrit les diverses étapes de traitement d'un texte vers son oralisation, peuvent être insérés des points de contrôle permettant à l'utilisateur de fournir une entrée plus riche (synthèse à partir de concepts), changer de voix, changer de langue ou de mode d'élocution (épellation).

Les idées développées dans cet article sont assorties d'exemples concrets extraits d'un système de synthèse du français développé à l'aide de COMPOST.

MOTS CLÉS

Synthèse de la parole, architecture client-serveur, language de programmation spécialisé, représentation centrée-objet, synthèse de parole à partir du texte multilingue.

ABSTRACT

This article presents a server which offers multilingual text-to-speech synthesis to connected clients. It describes the COMPOST environment which has been created to give a complete answer to this challenging idea : voice output will be fully integrated in man-machine dialog systems when hardware and software architectures will offer flexible and powerfull communication tools between text-to-speech systems and dialog systems.

COMPOST consists of a specialised programing language for text-to-speech synthesis. Each text-to-speech system describes the different steps which convert a running text towards its acoustic and/or visual synthesis by means of a main program called a scenario. Each scenario is then compiled and may be downloaded into the working environment of the server. A normalised communication interface enables a real-time dialog between the server and the attached clients i.e. for using a tree-structure as input (synthesis by concepts), changing the voice quality, the language or the spelling mode.

The ideas presented in this article are enlighted by concrete examples extracted from a text-to-speech system for French developped using COMPOST.

KEY WORDS

Speech synthesis, client-server architecture, specialised programing language, objet-oriented design, multilingual text-to-speech synthesis.

1. Introduction

La mise au point d'un système de synthèse de parole à partir du texte pour une langue donnée nécessite plusieurs types de facteurs :

- l'existence de connaissances suffisamment élaborées à tous les niveaux de description de la langue (aussi bien sur le contenu que sur l'expression),
- la mise en collaboration des diverses sources de connaissances (déclaratives ou algorithmiques) par des interfaces et des structures de travail normalisées,
- la définition de la stratégie d'utilisation de ces connaissances (contrôle du séquencement des opérations de transduction texte-parole),
- la gestion des communications entre l'(les) application(s) et cet outil de sortie d'informations.

On voit donc se distinguer deux impératifs importants qui conditionneront l'utilisation intensive des systèmes d'entrées/sorties vocales dans les interfaces de dialogue homme-machine :

a) la facilité de maintenance du système de synthèse dont toute modification devra rester transparente à l'utilisateur du système (client) :

Le système doit être évolutif et pouvoir bénéficier des connaissances plus approfondies susceptibles d'améliorer la qualité de la sortie sonore. Les efforts consentis pour une langue particulière dépendent étroitement de son utilisation économique et scientifique (« industries de la langue »). Cette évolution peut être longue, et les connaissances et stratégies internes au système doivent être interprétables et récupérables par les linguistes, phonéticiens, ingénieurs... non impliqués dans le projet initial. Par ailleurs, on doit pouvoir fournir à une équipe des matériaux (concepts, modèles, algorithmes...) linguistiques de base pour commencer une réalisation nouvelle.

b) la facilité d'utilisation de ces nouveaux systèmes d'entrée/sortie pour lesquels la gestion du dialogue est déjà en soi une tâche suffisamment ardue :

Les variables de contrôle d'un tel système (texte d'entrée, langue, style, locuteur...) doivent être homogènes et réellement multilingues : un tel système ne devra donc pas être un assemblage de plusieurs systèmes monolingues hétérogènes, eux-mêmes constitués de plusieurs modules sans structures communes.

Certaines propositions ont déjà été faites dans ce sens grâce au concept de langages de développement spécialisés pour la synthèse du texte : basés essentiellement sur les systèmes de règles de réécriture couramment employés en phonologie [13, 20], ils proposent des transducteurs symboliques exploitant des sources de connaissances explicites (règles, algorithmes) compilées hors ligne par le concepteur du système.

L'examen des diverses solutions déjà proposées fait apparaître plusieurs lacunes :

- ces systèmes présentent des interfaces de communication médiocres avec l'application (l'interface le plus couramment employé est une chaîne de caractères comportant la chaîne orthographique à synthétiser et des caractères de contrôle sans pouvoir exploiter des connaissances linguistiques plus riches qui pourraient être disponibles dès l'entrée — cf. synthèse à partir de concepts)
- ils proposent soit des langages de puissance trop faible pour implanter des traitements linguistiques complexes [2], [18], [25], [26], soit des compilateurs dédiés à des méthodes et techniques de traitement particulières [7], [9], [14], [19], [21], [28], soit des langages trop puissants [1], [15] (et donc pas assez contraignants et trop complexes à manipuler : le concepteur est presque revenu au niveau d'un langage de programmation classique).

COMPOST répond aux exigences minimales des concepteurs en essayant de les maintenir aussi longtemps que possible dans un cadre de description très coercitif. Il tente en même temps de proposer une standardisation des interfaces vers les applications par la notion de serveur-synthèse, joignant ainsi les développements déjà standardisés (X-Window, Motif... [12]) des interfaces graphiques.

2. COMPOST vu du concepteur de systèmes de synthèse

COMPOST est centré sur un langage de manipulation d'objets complexes organisés dans une **structure de travail arborescente :** ce langage est indépendant de la langue modélisée et décrit les diverses opérations de transduction (transformation) à effectuer pour convertir une structure textuelle (limitée à une chaîne de graphèmes ou non) en une structure décrivant la réalisation phonétique des divers objets sonores du message correspondant. Ces **objets** sont dits **dynamiques** car COMPOST exploite les principes élémentaires de la représentation centrée-objet : ces objets sont des instances d'objets génériques décrivant les différents types d'objets linguistiques susceptibles d'être créés et manipulés par le concepteur. Le concepteur peut donc en créer autant que nécessaire. Chaque objet dynamique hérite, lors de son instanciation, des propriétés de sa classe d'objets. Ces propriétés sont implémentées dans COMPOST sous la forme d'**attributs.** Ces attributs sont restreints dans COMPOST à des valeurs alphanumériques (cf. § 2.1.*b*).

Le séquencement des diverses opérations est appelé **scénario.** Le but de l'interface avec l'application est de paramétrer ce scénario par les attributs d'**objets** dits **statiques** qui sont instanciés une fois pour toutes par COMPOST lors du chargement du scénario concerné. Ces objets et leurs attributs sont donc accessibles par le concepteur grâce au langage COMPOST et par l'(les) utilisateurs (application, serveur de signal...) grâce à l'envoi de messages (voir *fig.* 1).

2.1. LE LANGAGE COMPOST

a) *Les diverses étapes de la conversion texte-signal acoustique*

Un système de synthèse à partir du texte peut être vu

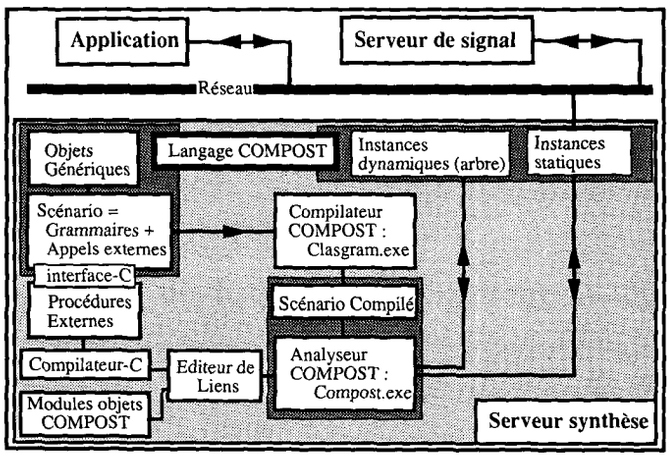

Figure 1. — Organisation logicielle de l'environnement de programmation COMPOST : le concepteur du système dispose d'un langage déclaratif qui inclut des grammaires de règles de réécriture et des appels à des procédures externes paramétrables qui doivent être écrites en C à l'aide d'une interface spécifique. Le scénario préalablement compilé pourra alors être exécuté par l'analyseur et contrôlé par l'utilisateur via les points de contrôle prévus dans le scénario. La transmission de données entre les applications et les divers serveurs se fait au travers du réseau par envoi de messages.

comme un quadruplet (analyse, transfert, génération, synthèse). L'analyse a pour but de fournir une description linguistique du texte d'entrée. La détermination d'une structure formelle depuis la chaîne de graphèmes fournie par l'entrée a de multiples buts : fournir un étiquetage lexical, syntaxique voire sémantique permettant de résoudre les problèmes de phonétisation (des homographes non homophones aux liaisons qui sont conditionnées par la syntaxe) mais aussi fournir une structure du texte qui va contraindre fortement la structure prosodique du message [17]. C'est le rôle du transfert de « traduire » cette structure linguistique en une structure phonologique comprenant l'ensemble des tâches segmentales et supra-segmentales à effectuer. Le module de génération décrit comment cette tâche est actualisée dans le discours : les diverses méthodes vont du « faire semblant » de la synthèse par unités stockées [10], [22], [23] au « faire comme » de la synthèse par règles [7], [11], [16], [21], voire de la synthèse par « contrôle moteur » [5] et cherchent à générer une représentation (paramétrique ou non) du signal de parole suivant une approche plus ou moins anthropomorphique. La dernière étape dite de synthèse se charge alors de convertir cette représentation en signal acoustique via un synthétiseur qui intègre plus ou moins les caractéristiques du système de production humain. On voit que cette transformation consiste essentiellement en une série de transductions de structures : comme dans le cas de la traduction automatique [3], les opérations visent essentiellement à générer une structure dans un certain espace de description, la transformer puis la reprojeter sur un autre espace de description. C'est pourquoi nous avons doté COMPOST d'un langage de transduction d'arborescences capable de donner explicitement sous forme de règles de transformation les transductions élémentaires à effectuer. Ces règles sont regroupées en grammaires de transduction qui décrivent la manière de passer d'un niveau de représentation à un autre par un ensemble de règles concurrentes.

b) *La déclaration des objets génériques*

Le but essentiel de ces transductions est d'organiser un ensemble de symboles dans une structure arborescente par des opérations de création, d'effacement et/ou de réorganisation structurelle. Ces symboles sont éléments d'un vocabulaire terminal fini et peuvent être regroupés en classes naturelles : les niveaux de description de la substance, de la forme du texte ou du message font intervenir des inventaires aussi divers que les caractères d'entrée d'un texte écrit (des caractères alphanumériques aux idéogrammes), les classes lexicales, les types de groupes syntagmatiques, les unités phonologiques (phonèmes, syllabes, moras, tons...) d'une langue donnée.

Nous avons adopté pour ceci une représentation centrée-objet à héritage simple : les symboles sont des intances d'objets génériques munis d'attributs propres à la classe. Ces attributs peuvent être de 7 types différents (les valeurs par défaut sont indiquées entre parenthèses) :

- le type `trait` qui prend trois valeurs : `+`, `-`, `neutre` (par défaut `neutre`)
- le type `bool` qui prend deux valeurs : `faux`, `vrai` (par défaut `faux`)
- le type `entier` qui prend les valeurs de `-32767` à `32767` (par défaut `0`)
- le type `intervalle [min,max]` (par défaut `min`)
- le type `réel` qui dépend de la machine (par défaut `0.0`)
- le type caractère `car` (par défaut le caractère nul `'/0'`)
- le type chaîne `chaine[taille]` (par défaut la chaîne vide `""`)

Évidemment, lors de la déclaration des classes, des valeurs d'initialisation des attributs de certains objets peuvent être précisées (cf. *fig.* 2).

La création d'un symbole dans la structure de travail se fait par instanciation d'un objet d'une certaine classe : une allocation dynamique d'un espace égal à l'ensemble des caractéristiques de l'objet générique est alors effectuée.

Certains objets dits statiques ne sont instanciés qu'une seule fois au chargement d'un scénario : ils servent à

```
classe Phoneme  /* phonèmes du français */
    objet(a,e,i,u,y,...,b,d,g,f,s,...)
     trait voc,nas,...;          /* traits phonologiques: vocalique, nasal... */
     [20, 500] duree = 100;      /* durée du phonème */
     [0, 5] mf0 = 0;             /* creux mélodiques intrinsèques en 1/4 tons */
     .....
    initialisation
     a,i,u,e,y vaut voc, -nas, duree = 90;
     b,d,g vaut -voc, -nas, duree=70;
     b,d vaut mf0=2;
     g vaut mf0=3;                    Partie de la déclaration d'une classe
     .....                            réservée aux initialisations particulières
finclasse
classe Lexeme   /* classes lexicales utilisées */
    objet(Det,Nom,Adv,...)       /* déterminant, nom commun, adverbe,... */
     bool contenu, epel = False; /* nature du mot, mode de phonétisation */
    initialisation
     Nom, Adv vaut contenu = True;
finclasse
statique classe locuteur   /* caractéristiques des locuteurs disponibles */
    objet(loc)
     chaine[20] nom;             /* nom du locuteur */
     [5,80] age, [50,400] f0moy; /* age, fréquence fondamentale moyenne */
finclasse
                                      déclaration d'une partie de l'interface du
                                      serveur : permet de changer de locuteur
```

Figure 2. — Exemple de déclaration d'objets génériques : les classe statiques permettent à l'application de paramétrer le serveur de synthèse. Les mots clés du langage sont figurés en gras.

l'interface de contrôle (cf. § 2.2.*c*). Les valeurs de leurs attributs peuvent cependant être manipulées comme tout autre attribut d'instance.

c) *La structure de travail et sa représentation normalisée*

Les instances des objets dynamiques sont organisées dans une liste parenthésée. Cette contrainte sur l'organisation des instances permet une écriture aisée des règles, une trace lisible de la structure accessible à tout moment et permet aussi à l'application (grâce aux attributs des objets statiques définissant l'interface) de transmettre aussi bien une chaîne orthographique qu'une écriture normalisée d'une structure profonde (cf. *fig.* 3) : l'appel à la procédure LireEntree(chaine) de la bibliothèque standard permettra au scénario de charger la structure de travail avec la structure décrite par chaine.

La représentation normalisée est une chaîne parenthésée : les attributs des instances qui diffèrent de leurs valeurs par défaut sont ajoutées par l'emploi de crochets. Par exemple, les caractères « a » et « A » s'écriront respectivement : <A ; –maj> et <A ; +maj>.

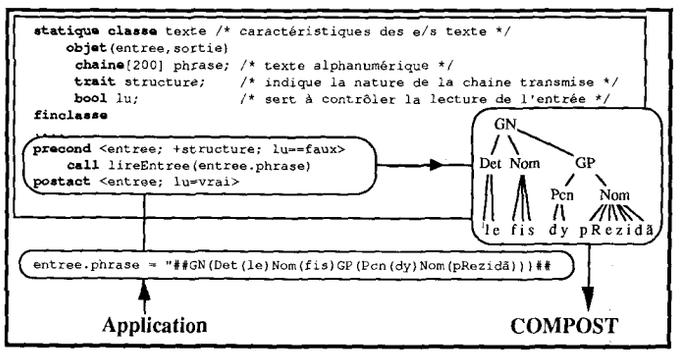

Figure 3. — Exemple d'utilisation de la représentation normalisée et des objets statiques afin de transmettre une structure profonde à un système de synthèse (cas de la synthèse par concepts, de la traduction automatique...). L'ambiguïté du mot « fils » (parent ou câble) est pratiquement impossible à résoudre dans un système de synthèse alors qu'il ne pose aucun problème lorsqu'il est le résultat d'une traduction automatique. Les objets dynamiques terminaux de la structure fournie à COMPOST sont des phonèmes. Ainsi le mot ambigu « fils » sera prononcé /fis/ et non pas /fil/ par le système de synthèse.

d) *Les règles et les grammaires de réécriture*

Les grammaires récursives ou non [4] sont constituées d'un ensemble de règles du type :

Req: PG → PD/CG + CD ; où PG, PD, CG, CD sont des sous-arbres décorés. Ces règles signifient : réécrire la partie gauche PG (appelée aussi le focus) en la partie droite PD (appelée aussi la cible) si les contextes gauche CG et droit CD sont vérifiés. Reg : est le nom de la règle qui apparaîtra dans l'option de trace disponible. Entre PG et PD existe évidemment la possibilité de transfert d'objets et d'attributs grâce à des variables de structure et des marquages. Le principe d'un cycle d'exécution du moteur d'inférence de ces règles est présenté figure 4.

Par exemple, en supposant que les classes d'objets génériques mot, grapheme, phoneme et cible sont définies, la règle suivant :

```
rent: ENT  -> e~ / <Vrb;+sng>( {OB,CON, }VI + )          /* (IL) OBVIENT */
           -> e~ / Vrb( {OB,DE6,RE,CON, }TI + )            /* OBTIENT */
           -> e~ / Vrb( {DE,RE, }VI + )                      /* VIENT */
           ->    / Vrb( graph*[1,20] + );                   /* AIMENT */
```

signifie : dans un verbe conjugué, la chaîne graphémique ENT se prononce en français /ɛ̃/ si la chaîne fait partie des formes des verbes dérivés de « tenir » et « venir » sinon n'est pas prononcée. Dans le cas des homographes non homophones « CONVIENT » (de « convenir » ou de « convier ») et « OBVIENT » (d'« obvenir » ou d'« obvier »), c'est l'analyse grammaticale qui doit au préalable avoir déterminé le nombre du verbe.

Les quatres règles suivantes illustrent la manipulation de la structure arborescente par la notion de marquage. Le marquage permet de désigner les objets de la partie gauche et des contextes qui serviront à reconstruire la partie droite. Ainsi, la règle suivante :

```
rapos: mot(L')<mot; #m>(#A) -> #m(l #A);                  /* Det(L')Nom(AMI) */
```

signifie convertir la suite de graphèmes L' constituant un mot en l qui commencera le mot suivant (le premier nœud mot a été supprimé de la structure), #*m* permet de marquer le deuxième lexème, #A de mémoriser la sous-structure dominée par celui-ci et de les recopier dans la partie droite.

De même, la règle suivante :

```
micro:<phoneme;micro>0,#p>(#A) -> #p (<X2D;F0MICRO=0,TPS=0>
                                      <X2D;F0MICRO=#p.micro,TPS=#p.duree/2>
                                      <X2D;fin=vrai,F0MICRO=0,TPS=-5> #A);
```

signifie marquer tout phonème qui engendre un creux mélodique et mémoriser dans #A sa dépendance et le recopier en ajoutant à sa dépendance ce creux par l'intermédiaire de deux trajectoires paraboliques sur le paramètre F0MICRO partant de 0, allant atteindre une cible égale au creux au milieu du phonème et revenant à 0 à – 5 ms de sa fin (voir *fig.* 5).

Le marquage est le plus usité lors de la manipulation de structures. Cette opération est intensivement utilisée en

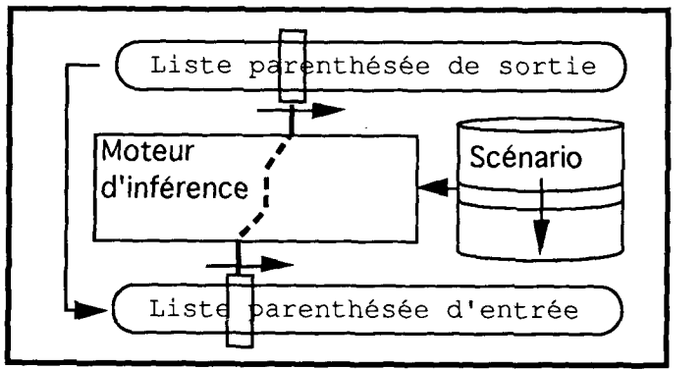

Figure 4. — Principe d'exécution du moteur d'inférence (MI). La portion de code compilé du scénario correspondant à la grammaire est chargée dans les tables d'exécution du MI. Le parcours de l'entrée se fait de gauche à droite sur la liste parenthésée d'entrée. Lorsque l'objet de fin de l'entrée est atteint, la liste parenthésée de sortie est recopiée dans l'entrée en vue de l'application du traitement suivant (call ou grammaire).

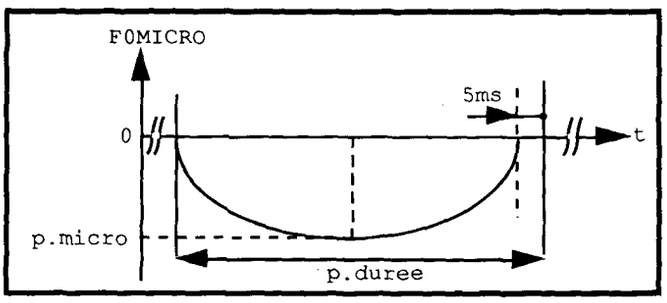

Figure 5. — Résultat de l'application de la règle micro ci-dessus sur l'évolution du paramètre F0MICRO (qui doit être déclaré comme attribut de type entier de l'ensemble des trajectoires comprenant notamment X2G et X2D comme objets génériques). Ce type de modélisation de trajectoires par points cibles connectés par des fonctions de transitions peut être appliqué à la synthèse par formants.

synthèse de parole pour générer ce que l'on appelle la **structure de performance** du message par opposition à la **structure de compétence** produite par l'analyse linguistique du texte source [27]. Cette structure de performance, aussi appelée structure prosodique, permet de hiérarchiser les éléments du discours afin de leur donner dans le discours l'importance qui leur convient. En effet, les paramètres acoustiques tels que durée et intensité des sons, le contour mélodique appliqué à un groupe de mots ainsi que la structure des pauses dans le message sont une mise en valeur de cette hiérarchie sous-jacente. Son calcul requiert la syllabation du message. Supposant que les objets syllabe,marque sont définis, la règle suivante permet d'affecter l'attribut nbsyl de l'objet marque avec le nombre de syllabes dominées :

```
cpts: <marque;#m>(<syllabe;#S>*[1,20]) -> <#m;nbsyl=!S>(#S);
```

La suite de cette opération peut alors consister à regrouper les groupes ainsi marqués dans la limite de 4 syllabes par la règle :

```
grp:  <marque;#m>(#A)<marque;#ms,nbsyl<=4-#m.nbsyl>(#B) -> #m(#A #B);
```

Les appels à des procédures externes permettent d'inclure dans la chaîne de traitement des algorithmes de traitement du signal (par exemple un synthétiseur) voire de traitement de données symboliques non formalisables à l'aide de ce type d'algorithme de réécriture.

2.2. LE SCÉNARIO

Le concept du scénario donne une solution aux problèmes suivants :

- l'intégration de fragments de logiciels complètement indépendants opérant à différentes étapes du processus de synthèse de façon simple pour donner au concepteur du système la flexibilité nécessaire d'utilisation de ces logiciels (par exemple des méthodes de synthèse différentes),
- la simplification du développement de ces logiciels,
- la souplesse et la dynamicité de l'application.

a) *L'intégration de logiciels*

COMPOST est capable d'intégrer deux sortes de modules de traitements : les modules de grammaire écrits dans le langage de programmation introduit dans 2.1, et des modules procéduraux écrit en C. Un scénario COMPOST est, a priori, une suite de grammaires et d'appels de procédures. Les procédures accessibles par COMPOST sont des points d'entrée, reconnus par le compilateur, aux modules procéduraux qui doivent être compilés avec COMPOST.

Dans la suite nous allons développer les idées introduites ci-dessus sur un exemple simpliste. Supposons que la chaîne de synthèse à construire est composée des modules suivants dont on suppose l'existence et le bon fonctionnement :

- la lecture du texte d'entrée : module C qui lit l'entrée du clavier suite à l'appel de lireEntree (point d'entrée du module),
- l'analyse lexicale : module C qui exécute l'accès au lexique. Accessible par la procédure ana_lex,
- l'analyse syntaxique : module C d'analyse syntaxique ayant ana_synt pour point d'entrée,
- la transcription orthographique-phonétique : un ensemble de grammaires COMPOST contenu dans le fichier top.cps,
- une synthèse par règles : ensemble de grammaires COMPOST pour la génération de la description paramétrique pour un synthétiseur à formants, fichier synth.cps,
- le synthétiseur : écrit en C avec synthetiser comme point d'entrée.

L'intégration de ces éléments dans une seule chaîne consiste donc à :

- définir les points d'entrée des modules C non connus par

COMPOST (qui est une opération relativement simple), et recompiler le compilateur COMPOST en lui rajoutant ces modules,

- écrire un scénario de la forme :

```
call lireEntree(entree.chaîne);
call ana_lex();
call ana_synt();
#include " top.cps"
#include " synth.cps"
call synthétiser();
```

Après compilation, l'exécution de ce scénario, en dehors d'une application, aura pour effet l'exécution consécutive de toutes les étapes pour chaque nouvelle entrée : l'utilisation d'une autre méthode de synthèse, par exemple, impliquera uniquement le changement des deux derniers mobules. L'interface par messages à travers les objets statiques et les points de contrôle offre une solution pratique à ce problème (sans avoir à modifier le scénario même) comme on verra plus loin.

b) *L'aide au développement*

Il est évident d'après l'exemple ci-dessus qu'il y a un grand échange d'informations entre les modules de traitement. La solution COMPOST pour réduire le coût de cet échange est d'imposer une structure commune et normalisée : tous les modules viennent lire et écrire dans cette structure construite dans l'espace du moteur d'inférence. Puisqu'il s'agit de transduction d'arborescence, cette structure de données est divisée en deux parties : une représentant l'entrée (l'arborescence à transformer) et une représentant la sortie (le résultat de la transduction). Les modules COMPOST reconnaissent ces deux structures par défaut, tandis que leur accès par les modules C se fait par des appels aux procédures offertes par COMPOST et définies dans l'interface procédurale. Les modules situés aux deux bouts de la chaîne ne sont pas entièrement soumis à cette normalisation : elle n'est pas nécessaire pour l'entrée du premier module (normalement un texte à synthétiser sous format libre) ni pour la sortie du dernier (normalement le signal acoustique ou sa description).

Par exemple, lireEntree dans l'exemple ci-dessus doit construire une structure de données normalisée à partir du texte d'entrée : cette entrée dans ce cas sera une suite d'instances de la classe des caractères alphanumériques. Pour ceci il disposera de procédures de l'interface telles que creer_instance(nom_objet) qui génère une instance de l'objet générique, dont le nom est donné en paramètre, dans la structure de donnée représentant l'entrée. Le module d'analyse lexicale pourra faire appel à des procédures telles que : trouver_instance(num_instance) pour balayer l'entrée, modifier_attribut_instance (nom_attribut, val_attribut) pour inclure certaines informations extraites du lexique dans la structure, etc.

c) *L'interface de contrôle*

Dans la première version de COMPOST [4], le contrôle de l'exécution du scénario était relativement sommaire : le scénario était divisé en trois modules — prologue, corps et épilogue —, le prologue et l'épilogue permettant respectivement d'initialiser et de libérer les environnements dynamiques nécessaires à l'exécution des algorithmes utilisés dans le corps (héritage de la grammaire morphologique d'une langue, d'un dictionnaire de sons d'un locuteur...). Les paramètres de la synthèse étaient donc définis une fois pour toutes dans le scénario lui-même. Le but de l'interface de contrôle présenté ci-dessous est de pouvoir paramétrer l'exécution du scénario depuis l'application.

L'interface de contrôle qui consiste à regrouper les différents modules de traitement dans des blocs dont l'exécution peut être contrôlée par une application à travers l'interface grâce aux objets statiques. Un bloc est constitué d'un ou plusieurs modules, COMPOST et/ou C, précédé d'une précondition et suivi d'une post-action, ces dernières étant optionnelles. Une pré-condition est une liste de tests sur des objets statiques semblables à ceux qu'on peut faire dans la partie gauche ou dans les contextes d'une règle COMPOST. Une post-action est une liste de modifications sur des objets statiques semblables à ceux d'une partie droite de règle. Par exemple, pour n'effectuer la lecture d'entrée, les analyses lexicale et syntaxique et la transcription orthographique-phonétique que lorsqu'il s'agit d'un nouveau texte, on peut utiliser l'objet nouveau de la classe statique indicateurs comme suit :

```
precond < nouveau; texte == vrai>
  call lireEntree(entree.chaine)
  call ana_lex()
  call ana_synt()
  #include " top.cps"
postact < nouveau; texte = faux>
#include "synth.cps"
call synthetiser();
```

Le déroulement de l'exécution de ce scénario dépendra de l'application : quand celle-ci demande de synthétiser un nouveau texte par l'appel de la procédure appropriée de l'interface d'accès aux objets statiques, cette procédure

mettra, automatiquement, nouveau.texte à vrai ce qui déclenchera, au prochain lancement de l'exécution, les traitements de lecture, analyse et transcription. La post-action qui suit le bloc remet cette valeur à faux pour garantir que l'exécution de ces modules ne se refera que sur un autre changement du texte. La programmation d'un tel contrôle est rendue possible par une interface procédurale d'accès aux objets statiques, qui peuvent servir à adapter l'application aux besoins comme dans le cas de modification automatique de nouveau.texte et sans avoir à accéder explicitement à cet objet dans l'application.

Les attributs des objets statiques peuvent aussi servir comme paramètres aux procédures C, le cas échéant. Si par exemple plusieurs dictionnaire de sons sont disponibles, il sera possible de changer interactivement de dictionnaire (avec une démarche semblable à la précédente pour la modification de loc.nom et nouveau.loc) comme suit :

```
precond <nouveau; loc == vrai>
  call init_dico(loc.nom)
postact  <nouveau; loc = faux>;
```

Les attributs de ces objets statiques peuvent en outre apparaître dans les expressions évaluées dans les règles. La paramétrisation du scénario ne se résume donc pas à un arbre de test mais peut influencer plus subtilement la structure résultante. Par exemple, on peut contrôler le débit moyen grâce à l'attribut debit de l'objet statique style en multipliant les durées de tous les phonèmes par ce dernier :

```
debit: <phoneme; #p> -> <#p; duree *= style.debit>;
```

3. COMPOST vu de l'utilisateur du serveur-synthèse

La figure 1 résume parfaitement l'intérêt de la notion de serveur-synthèse. Le client (l'application) après avoir effectué la demande et l'allocation du service, dialogue avec celui-ci par l'intermédiaire d'un serveur de messages. Le serveur de synthèse modifie les attributs des objets statiques en fonction des divers messages. Le scénario étant dans une boucle infinie, toutes les étapes de traitement vérifiant les conditions d'exécution sont effectuées à chaque changement de l'état du système : on peut par exemple envisager de synthétiser la même phrase avec un locuteur différent, changer le débit, la méthode de synthèse sans avoir à refaire deux fois les étapes communes (analyse, transfert...). Il suffit pour ceci de gérer convenablement les attributs de l'objet statique nouveau afin de n'effectuer que les opérations nécessaires à la synthèse courante. Le moteur d'inférence ne s'arrête que lorsque les paramètres actuels (pré-conditions) ne lui permettent d'effectuer aucun traitement. La main est alors rendue au serveur de message qui va organiser alors la prochaine tâche à faire exécuter au scénario.

a) *La paramétrisation de la synthèse*

COMPOST se démarque des systèmes existants par la possibilité offerte à l'utilisateur de communiquer avec le serveur à diverses étapes de traitement : du texte plat à la transmission d'une structure linguistique voire phonologique profonde, COMPOST poursuivra le traitement jusqu'à l'obtention du signal. Ce signal sera alors transmis au serveur du signal [6], [24] par l'intermédiaire d'une pile de fichiers audio : cette pile peut évidemment comprendre des fichiers issus de divers serveurs de synthèse voire d'autres types de signaux (musique, signal de parole naturelle codé...).

D'un point de vue logiciel, il reste à fournir à l'utilisateur du serveur la possibilité d'associer à une macro-instruction du type Synthétiser, Répéter, Eppeler_mot ou Changer_Locuteur la séquence d'opérations sur les objets statiques du scénario correspondant. Ceci permettra à l'utilisateur d'avoir une vue de plus haut niveau de l'interface : ainsi la macro-instruction Eppeler_Mot suppose un algorithme de recherche du mot à un niveau de traitement (par exemple après étiquetage lexical, on peut donner la localisation dans le texte du *n*-ième nom propre), la recopie de la chaîne graphémique dans entree.chaine, l'étiquetage de cette chaîne par un objet de type mot possédant l'attribut epel à vrai et la relance de la synthèse en positionnant nouveau.texte à vrai. Ceci suppose que l'on ait au préalable défini dans la transcription orthographique-phonétique du scénario (top_i.cps) l'ensemble des règles d'épellation du type :

```
epelw≑ W -> dublve / <mot;epel==vrai>(graph*[0,20] + ;
```

On peut donc imaginer les multiples possibilités de précisions sur la nature du traitement ainsi que sur sa paramétrisation que la collaboration entre utilisateurs et concepteurs de systèmes de synthèse peut engendrer. Ces paramétrisations constituent la base de l'incorporation de connaissances a priori sur la nature linguistique, phonologique voire phonétique de l'entrée. C'est cette exploitation qui permet alors d'intégrer notre serveur-synthèse dans un système de dialogue.

b) *La gestion des scénarios*

En admettant l'existence d'un véritable serveur multilingue, c'est-à-dire l'écriture de scénarios pour au moins deux langues distinctes, deux principales possibilités s'offrent

aux concepteurs du serveur : soit une gestion par l'application du chargement du scénario approprié à l'entrée textuelle, soit l'écriture d'un scénario comprenant un algorithme automatique de détection de la langue [8]. Dans ce dernier cas, ce serait le serveur lui-même qui déciderait du scénario le plus approprié à « prononcer » le texte présenté à l'entrée et positionnerait un attribut langue de l'objet statique nouveau.

4. Conclusions

COMPOST essaie de répondre aux problèmes de construction, maintenance et utilisation des systèmes de synthèse de la parole. Il définit un langage de programmation déclarative adaptée à la représentation des connaissances des linguistes et des phonéticiens et introduit la notion de scénario. Cette notion donne au concepteur du système les moyens de tester et de maintenir les étapes de traitement d'un système de synthèse en permettant à l'utilisateur de paramétrer son exécution (changement de langue, locuteur, synthétiseur, etc...) par une interface explicite et interactive.

Les idées introduites par COMPOST pourraient servir de base pour définir un standard de communication et d'écriture de serveur de synthèse. Cependant, comme pour tout système informatique, les concepts COMPOST restent à tester et à approfondir selon les besoins que révèlera la mise en service du système avant de passer à une généralisation de son utilisation : à cette fin, les capacités de COMPOST en synthèse multilingue sont en cours d'évaluation sur plusieurs langues (français, catalan, allemand...).

Remerciements

Ce travail a été rendu possible grâce à un financement initial issu du projet ESPRIT II Multiworks et est actuellement financé par un contrat CNET. Nous remercions particulièrement C. Cabrera, J. Camps, M. Guerti, D. Sos, P. Tripodi, P. Verdier pour leur utilisation intensive et critique de ce serveur.

Manuscrit reçu le 30 janvier 1992.

BIBLIOGRAPHIE

[1] A. Aggoun, 1985, *SYNTHEX : Un système de traitement des connaissances prosodiques pour la synthèse de parole,* Thèse de Doctorat, Université de Rennes, France.

[2] M. Backström, K. Ceder, B. Lyberg, 1989, « PROPHON : an Interactive Environment for Text-to-Speech Conversion », *EuroSpeech'89,* 1, 144-147.

[3] G. Bailly, A. Perrin, Y. Lepage, 1988, « Common approaches in speech synthesis and automatic translation of text », *Bulletin du Laboratoire de la Communication Parlée,* Grenoble, 295-311.

[4] G. Bailly, A. Tran, 1989, « COMPOST : a rule-compiler for speech synthesis », *EuroSpeech'89,* 1, 136-139.

[5] G. Bailly, R. Laboissière, J. L. Schwartz, 1991, « Formant trajectories as audible gestures : an alternative for speech synthesis », *J. of Phonetics,* 19, 9-23.

[6] C. Binding, C. Schmandt, K. Lantz, B. Arons, 1989, « Workstation audio and window-based graphics : similarities and differences », *Olivetti Research Center Techn. Report,* 1-21.

[7] R. Carlson, B. Granström, S. Hunnicutt, 1982, « A multilanguage text-to-speech module », *IEEE Conf. on Acoust. Speech & Sig. Proc.,* 1604-1607.

[8] R. Carlson, K. Elenius, B. Granström, S. Hunnicutt, 1986, « Phonetic properties of the basic vocabulary of five European languages : implications for speech recognition », *IEEE Conf. on Acoust. Speech & Sig. Proc.,* 2763-2766.

[9] R. Carlson, B. Granström, 1990, « An Environment for Multilingual Text-To-Speech Development », *Proceedings of the ESCA Tutorial Day on Speech Synthesis,* Autrans, France.

[10] F. Charpentier, E. Moulines, 1989, « Pitch-synchronous waveform processing techniques for text-to-speech synthesis using diphones », *European Conf. on Speech Com. and Techn.,* 2, 13-19.

[11] C. Coker, O. Fujimura, 1966, « A model for the specification of the vocal tract area function », *J. Acoust. Soc. Am.,* 40, 1271.

[12] J. McCormack, P. Asente, 1988, « An overview of the X Toolkit », *Proceedings of the ACM SIGGRAPH Symposium on User Interface Software,* 9-21.

[13] F. Dell, 1973, *Les règles et les sons, introduction à la phonologie générative,* Hermann, Paris.

[14] P. M. Hansen, 1989, « Syntax, Morphology and Phonology in Text-To-Speech systems », *Annual Report of the Institute of Phonetics,* University of Copenhagen, 23, Copenhagen, Norway.

[15] S. Hertz, 1990, « A Modular approach to multi-dialect and multi-language speech synthesis using the Delta system », *Proceedings of the ESCA Workshop on Speech Synthesis,* Autrans, France.

[16] D. H. Klatt, 1982, « The KLATTalk text-to-speech conversion system », *IEEE Conf. on Acoust., Speech and Sig. Proc.,* 1589-1592.

[17] D. H. Klatt, 1987, Review of text-to-speech conversion for English, *J. Acoust. Soc. Am.,* 82, 737-793.

[18] S. Lazzaretto, L. Nebbia, 1987, « SCYLA : a speech compiler for your language », *EuroSpeech'87,* 381-384.

[19] H. Loman, J. Kerkhoff, L. Boves, 1989, A working environment for speech synthesis by rule, *Proceedings AFN,* 13, 51-63.

[20] I. G. Mattingly, 1981, Phonetic representations and speech synthesis, in *The cognitive representation of speech* (D. R. Ladd & J. Anderson), North-Holland.

[21] G. Olaszy, 1989, MULTIVOX A flexible text-to-speech system for Hungarian, Finnish, German, Esperanto, Italian and other languages for IBM-PC, *EuroSpeech'89,* 2, 525-529.

[22] D. O'Shaughnessy, L. Barbeau, L. Bernardi, D. Archambault, 1988, Diphone speech synthesis, *Speech Communication,* 7, 55-65.

[23] Y. Sagisaka, 1988, Speech synthesis by rule using optimal selection of non-uniform synthesis units, *IEEE Conf. on Acoust. Speech and Sig. Proc.,* 679-682.

[24] D. C. Swinehart, L. Stewart, S. Ornstein, 1984, Adding voice to an office computer network, *Techn. Report CSL,* 83-8, Xerox Research Center.

[25] B. Van Coile, 1989, The DEPES development system for text-to-speech synthesis, *IEEE Conf. on Acoust. Speech & Sig. Proc.,* 250-253.

[26] H. C. Van Leeuwen, 1989, A development tool for linguistic rules, *Computer speech and language,* 3, 83-104.

[27] F. Grosjean, 1983, Les structures de performance en psycholinguistique, *L'année Psychologique,* s3, 513-536.

[28] L. Boves, 1991, Considerations in the design of a multi-lingual text-to-speech system, *J. of Phonetics,* 19, 25-36.