COMPOST: a client-server model for applications using text-to-speech systems

Mamoun Alissali and Gérard Bailly
Institut de la Communication Parlée
46, av. Félix Viallet 38031 Grenoble CEDEX - FRANCE
bailly@icp.inpg.fr - http::/www.icp.inpg.fr/~bailly

Abstract

This article presents a Client-Server Model for multilingual text-to-speech synthesis. The server maintains a collection of TTS systems together with related reconfigurable descriptions, called scenarios. Applications of an authorized client can access to this collection via an Ethernet network on a simple request to the server. This server allows the client to customize the TTS processing (language, speaker, speech rate, intonation...) to its requirements by switching between different systems and/or reconfiguring the one it is currently using. The working environment, called COMPOST, has a three layered architecture: the development layer including a powerfull rule-compiler [3] and language-independent processing facilities (linguistic analyzers, PSOLA and Klatt synthesizers ...), the system construction layer including the Scenario Definition Language, and the server layer which has two main components: the process manager and the ressource manager.

1. Introduction

The inspection of existing software environments specialized on TTS, shows that their main objective is to provide development tools to meet the research requirements and the scientific arguments [12, 13, 11]. The little care paid to software engineering issues is behind the lack of flexibility in the resulting TTS systems, which are often dependent of prealable choices like the language, the development environment, the synthesis method and even the hardware [9, 15]. Two prospects emphasize the serious consequences of such dependencies: the relatively new efforts on multilingual TTS, and the fast progress achieved and yet to come in computer software and technology.

The COMPOST approach aims to solve this problem by examining TTS systems from the software engineering point of view. The main issues of modularity and reusability allow the optimisation of the large efforts put to build and use high quality text-to-speech systems. COMPOST integrates into a unique three-layered environment:

a. Development: includes the definition of the central object-based data structure, a rewrite-rule programing language, similar to those already existing in the literature, and a function library for conjoint programing in C.

b. System Construction: centered on a programing language for the description of "scenarios". This language allows to build operative real-time text-to-speech systems. A scenario defines a sequence of modules to be used to construct the TTS system, allowing for run-time selection of modules to be used and their parameters (see section 3).

c. Server: in this layer, available TTS systems are abstracted into services and ressources thus allowing for the implementation of a Client-Server Model. From the application point of view, this model offers an adaptable interface, which includes simple basic requests such as "synthesize (text)", "repeat the last sentence", and customizable scenario-dependent requests such as "set the speech rate to (value)" "spell the last (number) words".

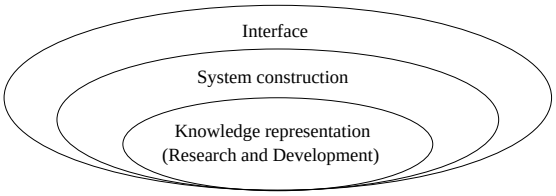

Figure 1: COMPOST: a three-layer environment. The development layer concerns experts in speech research. The system construction layer involves coercitive work to define coherent systems. The server layer abstracts TTS services for the end-user.

2. The development layer

Overview of existing TTS systems and specialized environments, shows that the major difficulty, from the software point of view, is the delicate compromise between efficiency and software quality. High-quality speech synthesis from flat text requires the construction of a complex linguistic and phonological structure. This construction and its memorization then

implies the use of rich data structures and powerful programing tools,

Some currently available systems [12, 13, 10, 14] offer rich data structures, on the cost of extra complexity in the programing language, and decrease of the readability [8]. To solve this problem our approach is to review this compromise in terms of one between the data structure and the programing tools. For the first we use an object-based data structure with implicit hierarchical organisation, allowing thus the incorporation of all desired information while keeping a comprehensive surface representation and minimizing programing requirements. This organisation makes it possible to integrate easily two types of processing: symbolic rewrite rules and the traditional algorithmic programing. This combination virtually covers all programing requirements in TTS [1]. That way, efficiency is guaranted by the use of proper implementation for each sub-task, instead of the classical approach of using one powerful programing language on all levels of TTS system construction.

2.1. The data structure

Objects used in COMPOST are grouped in classes. A class is defined by its unique name and a collection of attributes. Features are variables from seven basic types: feature (plus, minus, neutral), interval, integer, boolean, real, character and string. An example of a declaration of the phoneme and syllable classes is presented:

```
CLASS graph
  OBJECT(A,B,C,D...)
  BOOLEAN -up; /* uppercase */
ENDCLASS
CLASS phon
  OBJECT(a,e,i,a~...)
    FEATURE +voiced, +vocalic, -nasal; ...
    INTEGER duration, energy; ...
    a~ IS +nasal, -vocalic; ...
ENDCLASS
CLASS syllable
  OBJECT(Syl)
    FEATURE -acc;
    REAL lengthening = 0.; ...
ENDCLASS
```

The declaration include object models (generic objects) which define default values of the attributes for each object, and provide automatic instantiation, avoiding thus the use of the somehow encombersome constructor and destructor operators used in Object-Oriented programing languages.

2.2. Programing in COMPOST

During processing objects are grouped in an n-ary tree structure. Construction and various operations on the tree may be done by modules written in either of the two programing languages: algorithmic and rewrite-rules. for the first the wide spread C programing language was enhanced with a specialized function library giving access to the object tree. The second is the COMPOST language which will be described briefly here, for more details refer to [2].

Under COMPOST, rewrite rules have the classical form:

```
Name= Focus -> Target / Left Ctxt + Right Ctxt;
```

They are grouped in grammars that perform object-tree transformation operations such as level insertion, level deletion, object permutation ... to achieve high-level tasks like orthographic-phonetic transcription, syllabation and generation of prosody.

The language includes a restricted form of Regular Expressions pattern matching, enhanced mathematical expressions, and a powerful mark-and-copy mechanism.

For example, the following rule multiplies the duration of the vocalic nucleus of accented syllables by a lengthening factor stored as an attribute of the syllable :

```
dur= <phon;+vocalic,#P>(#A) ->
 <#P;duration=duration*#S.lengthening>(#A) /
  <syllable;#S,+acc>(phon*[0,3] + phon*[0,3]);
```

Algorithmic modules are restrained to used the object-tree structure as their input and output. Using library functions as `CPSgetFeatureValue(object,attribute)` and `CPSputFeatureValue(object,attribute,value)` they can retrieve, manipulate and restore various information to/from the data structure.

2.3. Specification of parametric trajectories

The standard way to control and generate parametric trajectories in COMPOST is to generate "target" objects which are dominated by "time base" objects. The attributes of "target" objects give the controled parameters. The trajectories are specified by sequences of targets connected by transition functions. These functions are given by the names of the objects whereas the position of these targets are given by a particular attribute which gives their delay from the begining or the end of the dominant "time base" object. For example, F0 movements can be described using the following class:

```
CLASS F0_movements
  OBJECT(Spline, Line,...)
    BOOLEAN -end;
    INTEGER F0, delay; ...
ENDCLASS
```

Then a rising F0 from 100 Hz to 150 Hz on a phoneme can be described with three targets as below (of course, the number of targets is unlimited and any complex F0 movements such as tones can be described and delayed. Movements relative to a baseline and a upperline are also implemented easely using `real` attributes):

```
<a;duration=100>(
 <Spline;F0=100,delay=0>
 <Line;F0=150,+end,delay=-5>
 <Line;F0=100,+end,delay=0>) /* reset */
```

3. The system construction layer

3.1. The scenario

Modules constructed in the development layer may be seen as processing blocks providing given functions. The scenario construction allows the combination of these modules in processing sequences (TTS systems). Each scenario is then compiled into initialization tables. These tables can then be downloaded into the Scenario Interpreter (cf. Fig. 2) at run-time as in [9, 15].

One particularly important feature of system construction under COMPOST is the use of a special user-defined data structure, the Static Objects (SOs), that describe "the status of the system", providing two distinct features to the resulting TTS systems:

a. Reconfigurability: The Scenario Description Language provides the possibility of conditional execution of a set of modules, according to the result of tests on the status of the system or the processed message. Tests are performed on the attributes of the SOs, which values can be reset to indicate changes in the status after the execution (see example below).

b. Dynamic Parametrisation: Features of the SOs may be used as parameters to C function calls, and their values may be referenced by rewrite rules. In addition to giving a way for communicating supplementary information between modules by using those attributes as global variables, this mechanisme allows to export system status and parameters to the external world in an abstract form.

For example the message to be processed can be described by a SO as follows:

```
STATIC CLASS input
  OBJECT(text)
    BOOLEAN new,analyzed;
    CHAR[200] in;
ENDCLASS
```

i.e. as attribute `in` to the objet `text`. This attribute will be passed as a parameter to the lexical analyzer.

```
PRECOND <text; new == TRUE, analyzed == FALSE>
  CALL lexical_analyzer(text.in)
  #INCLUDE "presynt.cps"
  CALL syntactic_analyzer()
POSTACT <text; new = FALSE, analyzed = TRUE>;
```

A scenario is compiled into internal description files, that can be interpreted by the Scenario Interpreter, which role is the evaluation of conditions, the application of grammars and C calls and the communication of the data structure between modules.

This organisation provides for development of intuitive interface by the use of an intermediate Command Interpreter which is, conceptually, similar to a shell in the Unix operating system. For example, for the request "synthesize(message)" the command interpreter would execute the sequence:

```
SERVICE synthesize(message):
text.new = TRUE;
text.in = message;
CPSexec();
ENDSERVICE
```

Then it will assign proper values to the concerned attributes and run the TTS system.

3.2. Adding information

The COMPOST input is not restricted to a flat text. It can be enriched by additional information as in the "Focus" part of the rewrite rules. This can be used to add semantic and/or prosodic information such as below (Generic objets for alphanumeric characters are given in uppercase and phonetic symbols in lowercase to avoid specification of the class. Uppercase characters are then distinguished by the attribute `up` as declared in section 2.1):

```
text.in="<Vrb;+emphasis>(<P;+up>RENDS) ceci.";
text.in="Vrb(pr<a~;duration=250>) ceci.";
```

This feature can ease the development of speech generators from the output of intelligent systems such as dialog managers [7], text generators from concepts [17, 16] ... Most of these systems generate representation structures of sentences which can be easily converted into a COMPOST tree-structure [5] predecorated with linguistic objects (syntagmatic organization ...) and related attributes (morphological, lexical, semantic information ...). Richer phonological representations and more acurate prosodic generators may be then established by the domain-related experts.

4. The server layer

With the abstraction provided by the SOs and the Command Interpreter, client applications can use TTS services through an adaptable intuitive interface, as shown above. The generalisation of this concept leads to a Client-Server model providing multi-scenario, multi-client networked operation, that implements simple and efficient use of TTS systems by other applications. The COMPOST Server has two main components, each providing one abstraction of the environment capabilities to the external world.

The first component is the Process Manager, which deals with "services". Services are meant to be intuitive requests by clients, that are interpreted by the

server into operations on one or more of the TTS systems (switching between languages, modification of speech rate) or client-Server dialog and operations (client connected, using French scenario ...). The second component is the Ressource Manager, which allows for abstraction of large data storage (lexicons, sound dictionaries ...), algorithmic processing (entry points of COMPOST C functions), and symbolic ressources like the connection to an audio server. The use of ressources is optimized by use of Object-Oriented techniques like automatic allocation and deallocation and data sharing.

When a client asks for a connection to the COMPOST server the Session Manager offers loading of the scenarios known to the Ressource Manager. The effective connection to one of them creates only a copy of its static objects and working data structure. The Process Manager is also responsible for the dynamic allocation of the COMPOST Interpreter to connected clients (by default, clients alternates at each service request, but explicit locking may be requested by the built-in services LOCK and UNLOCK) as shown in Fig. 2.

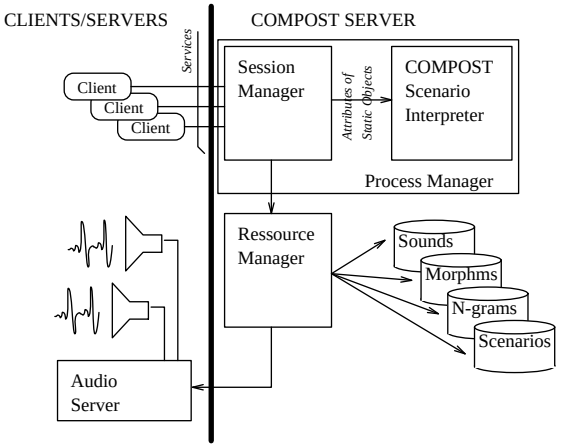

Figure 2: The COMPOST server: The Session Manager deals with services, asks eventually the Resource Manager for ressources and runs the Scenario Interpreter when appropriately parametrized.

5. Conclusions

COMPOST is currently running on UNIX workstations connected via an ETHERNET network. It enables the collection of high-quality modules for building up-to-date TTS. As the modules written in the algorithmic programing language are accessed via a standard library, most of them are language-independent and can be shared by different scenarios using different ressources.

It offers a practical solution for the incremental development of man-machine interfaces using voice: the dialog manager is a client to which the server will offer more quality as the TTS modules improve and more flexibility as the number of services increases.

COMPOST is also a flexible research tool and is currently being used to test new prosodic generators [6] as well as new parametric representations of the speech signal [4].

REFERENCES

[1] Alissali, M. *Architecture logicielle pour la synthèse multilingue de la parole*. PhD thesis, Institut National Polytechnique, Grenoble – France, 1993.

[2] Alissali, M. and Bailly, G. Manuel d'utilisation de compost 2.0. ICP internal report, Grenoble, 1992.

[3] Bailly, G. and Alissali, M. Compost: a server for multilingual text-to-speech system. *Traitement du Signal*, 9(4):359–366, 1992.

[4] Bailly, G. and Guerti, M. Synthesis-by-rule for French. In *12th International Congress of Phonetic Sciences*, pages 506–509, Aix-en-Provence, France, August 1991.

[5] Bailly, G., Perrin, A., and Lepage, Y. Common approaches in speech synthesis and automatic translation of text. *Bulletin du Laboratoire de la Communication Parlée - Grenoble*, 2B:295–311, 1988.

[6] Barbosa, P. and Bailly, G. Generating segmental duration by p-centers. In Auxiette, C., Drake, C., and Gérard, C., editors, *Fourth Rhythm Workshop: Rhythm Perception and Production*, pages 163–168, Bourges, France, 1992. Ville de Bourges.

[7] Bourguet, M.L. *Conception et réalisation d'une interface de dialogue personne-machine multimodale*. PhD thesis, Institut National Polytechnique, Grenoble – France, 1992. sous la direction de Jean Caelen.

[8] Boves, L. Considerations in the design of a multilingual text-to-speech system. *Journal of Phonetics*, 19(1):25–36, 1991.

[9] Carlson, R. and Granström, B. A phonetically oriented programming language for rule description of speech. *Speech Communication*, pages 245–253, 1975.

[10] Carlson, R., Granström, B., and Hunnicutt, S. A multilanguage text-to-speech module. *IEEE International Conference on Acoustics, Speech, and Signal Processing*, pages 1604–1607, 1982.

[11] Coleman, J. "synthesis-by-rule" without segments or rewrite-rules. In Bailly, G. and Benoît, C., editors, *Talking Machines: Theories, Models and Designs*, pages 43–60. Elsevier B.V., 1992.

[12] Hertz, S. From text to speech with srs. *Journal of the Acoustical Society of America*, 72:1155–1170, 1982.

[13] Hertz, S., Kadin, J., and Karplus, K. The delta rule development system for speech synthesis from text. *TrASSP*, 73(11):1589–1601, 1985.

[14] Lazzaretto, S. and Nebbia, L. Scyla: speech compiler for your language. *Proceedings of the European Conference on Speech Communication and Technology*, pages 381–384, 1987.

[15] Olaszy, G., Gordos, G., and Németh, G. The multivox multilingual text-to-speech converter. In Bailly, G. and Benoît, C., editors, *Talking Machines: Theories, Models and Designs*, pages 385–411. Elsevier B.V., 1992.

[16] Yamashita, Y., Mizutani, N., and Mizoguchi, R. Concept representation for synthetic speech ouput system. In Bailly, G. and Benoît, C., editors, *Talking Machines: Theories, Models and Designs*, pages 43–60. Elsevier B.V., 1992.

[17] Young, S.J. and Fallside, F. Speech synthesis from concept: a method for speech output from information systems. *Journal of the Acoustical Society of America*, 66(3):685–695, 1979.