Technical Report

Project Speech Synthesis Using a GA

V 1.0

By Pierre-Yves Hervo

Polytech Nantes High school of Engineering

18/09/2013

Contents

[Introduction 2](#_Toc367353964)

[The architecture 3](#_Toc367353965)

[Quick recap 3](#_Toc367353966)

[2.1 From Java to Praat 3](#_Toc367353967)

[2.2 From Praat to Java 4](#_Toc367353968)

[2.3 Sequencing 4](#_Toc367353969)

[The Genetic algorithm 7](#_Toc367353970)

[The population 7](#_Toc367353971)

[The fitness function 7](#_Toc367353972)

[The selection operator 7](#_Toc367353973)

[The evolutions operators 7](#_Toc367353974)

[The target 7](#_Toc367353975)

[The Evolution of the project 7](#_Toc367353976)

# Introduction

This report contains the technical explanations of the work I have realised for the ICCMR laboratory during my four month internship on the subject of speech synthesis using a Genetic Algorithm.

This report won’t explain the background of the project, it just define the architecture I have designed and its evolution during the project. If you want to learn about the background, please read the report I have written about it.

I will speak a little about the Java code but I won’t explain the implementation in detail. If you want to look at it, the sources are fully documented with javadoc and my own comments.

Both the Java code and the reports are available at the following address: <https://github.com/phervo/ProjetEte2013>.

There is two important points to understand if you want to reproduce or improve the project. The first one is how I designed the communication architecture and the second is how my GA works.

I will start this report with these two points, then I will explain the evolutions I bring during the project and I will conclude.

# The architecture

## Quick recap

The goal of the project is to use a genetic algorithm to found the Praat’s variables values for a specific vowel. For this, I use the formants of the sounds to know if the sound produced by the current individual of the run is close or not to my target. Praat allows me to generate the sound and to get the formants values, so I use this software during the fitness evaluation part of the Genetic algorithm run.

Here is the general idea:

//put the general scheme here

But in practise, the synchronisation of this two softwares is not so simple. I will explain here how I managed to do it.

Giving the Praat's API, we need to use two different ways to connect Java

and Praat. We need to consider one way to communicate between Java and

Praat and another way to communicate from Praat to Java. The first one is

use to send and execute the Praat script and the second to send the answer

from Praat to Java. They both use a different techniques to be executed and

so a particular treatment must be done for each.

## 2.1 From Java to Praat

We want to send a Praat's script to Praat and have it executed. The way

I choose is the software SendPraat. It is a program developed by the same

authors as Praat and available at this address : http://www.fon.hum.uva.

nl/praat/sendpraat.html. It allow to send orders to a running instance

of Praat.

It means we need two programs :

1. A normal Praat software already launched.

2. SendPraat which will give it orders. No need to launch this one, it only

works in command line.

If you give SendPraat the name of a script, it will made Praat launch

and execute it. The only thing left is to make Java executed SendPraat. For

this, I used the Java Runtime Environment which can use the command line

of windows.

Note : I use a SendPraat.exe as I am a windows user but you can compile

the source code yourself to use it in your own operating system. If you want

to make Praat communicate with a C program, you can use the SendPraat

directive, no need to compile source code. For more information, look at the

Praat's API, section Praat scripting. As I was working in Java, the solution

I presented is currently the best.

## 2.2 From Praat to Java

There is only one way to make Praat communicate with another program,

whenever the language is: the sockets.

Sockets are tools used in computer science to make two different program

communicated. For this, they will use the network principles and send net-

work packets to a computer on a speci\_ed port. It is not necessary that the

target is running on another computer, it can be the same and int that case,

we use a local network call localhost. The \_rst program will send a message

to the other specifying the port and the second one will listen will listen to

the port and get the message when it arrived.

Praat allows to send sockets by the directive "sendsocket" but it cant

received sockets from another program. That is why we got to use the Send-

Praat program in the other side. If Praat send a socket then our GA will need

a functionality which always listen to this port and will take the message.

Such functionality basically call a Server. In that purpose, I implemented

a Java server that listen to a speci\_c port to get the message from Praat. I

will describe it in the second part of the report.

## 2.3 Sequencing

There is a problem of sequencing to take care to synchronise Java and Praat.

The problem came from the fact that they are two different thread(program)

running. They both have a different execution's speed. The Java's GA work

very fast, each generation take a few seconds while each sound synthesis take

a few seconds in Praat. For example, it take approximately 12 second to

Praat to generate a 2.0 seconds sound. So there is a problem of speed and

synchronisation.

This is why I should have establish a sequencing order between the two

programs to force the Ga to wait for Praat's answer before going to the next

individual. More precisely, to wait for the server to get the answer from Praat

and store it into the GA. The GA could do the comparison of formants while

it is done.

The only solution was to use semaphores. It is a computing technique for

The only solution was to use semaphores. It is a computing technique for

sequencing tasks. It work on the principle of token. You have a token in a

box, if someone want to do an action he took the token and it released it

when finished. The others wait for the token to be free before doing their

action.

I used in fact two levels of semaphore in the fitness function :

The first on is when the GA start the fitness function, it took the token and

it released it when it had finished the comparison and calculated a matching

mark. It prevent the Ga to launch the fitness function with another candidate

while still running the previous one. For example if Praat is still running a

synthesis, it won’t be able to do the comparison with a empty values and

switch to the next candidate.

The second level is in the function fitness itself, it allow to be sure that the

server had store the message into the GA. As the server is a different thread,

it is a obligation. It forbid the \_tness function to do the comparison with

the reference formant until a value was set by the server. It avoid errors.

Here is the representation in the form of a algorithm. The operations

concerning the semaphores are written with letters whereas the other actions

are written with numbers

fitness function{

A) launch of the function, took the token "function" to avoid the GA to launch 1) create a script with the candidates values

2) send it to praat

3) praat execute the script and calculate the formant values

4) the server get the socket and store the result

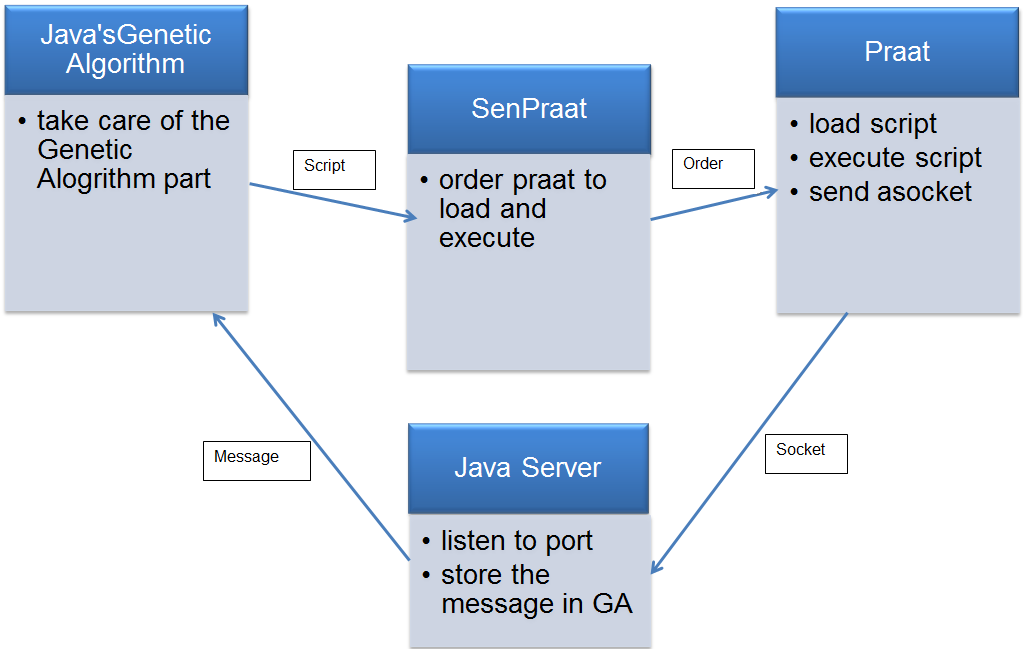
=> release the "praat result stored" token

B) If the token "praat result stored" is available we can continue, else we 5) compare the formants values

6) give a matching result

C) release the token "function".

}



# The Genetic algorithm

I will assume here that the reader is aware of a Genetic algorithm works and know the generic algorithm.

I will in this part explain how I designed each of the different part of my GA to get one that is available to deal with the speech synthesis under Praat.

## The population

To synthesise a sound, we use Praat variables values. There are 29 of them but we can only defined 19 for a vowel, the 10 others will take values by default. It is this combination of vowel that would produce a particular sound, so it is the thing we should make evolve. I design a structure as a list of doubles. The length of this structure is variable, you can define it. It allows using sequence of length 8 or 16, I will came back to that point in the III.

Each of these values corresponds to a particular Praat variable. In Praat, each variable get a specific interval of value. The lungs got one, the masseter got one , etc.

So to fill my sequences, I defined a alphabets with possible values for each and then I pick a value in it. I guaranty that way that

## The fitness function

## The selection operator

## The evolutions operators

## The target

# The Evolution of the project

# Java sources

As I said in the introduction, I won’t speak in detail about the implementation of the solution in java. I will explain the general architecture of the project in order that you understand it and could be able to found quickly the things you are looking for.

First I used the Java Genetic Algorithm’s API : watchmaker Version: **0.7.1** , available here :

<http://watchmaker.uncommons.org/>. This API allow me to design and manipulate my own structures for the candidate, the fitness function and the evolution operators by implementing interfaces. I needed to rewrite some classes to modify the execution as I wanted. As it is an open source project, it was very easy.

I divided the project in different Package with a specific purpose each.

The application package contains the main class to launch the program.

The communication package contains the classes for the java Server, tools for the communication with it and the way to deal with the message from Praat.

The controller package is used by my GUI application according to the MVC design pattern model. It defined all the actions of the GUI components.

The elements package contains all the basics elements I manipulated during the run of the Genetic algorithm. You will find there the definitions for a Sequence or a Formant for example.

The exceptions package contains the exceptions I raised if necessary for some elements.

The genetic algorithm package contains all the classes I had to implement or re-implements to use the watchmaker API, It also contains the GeneticAlgorithmCall class which is the main class of this project. It is the one that use all the others and set the elements of the watchmaker API as I need. It is a very big class but I wanted to keep all the GA’s things in one place.

The message package contains the definition of all the messages I send to Praat or the way to treat the message that comes from Praat after the server get them.

The monitoring package contains all the tools I used to create the CSV file and the results curves at the end of the GA’s run.

The Praat Gestion package contains all the classes needed for the use of the Praat object. As I explain in a point above, it is an object that simulates the Praat software state using the design pattern “State”.

The test package contains both the unitary and the integration tests.

The vue package contains the definitions of the graphic frames used in the MVC deign pattern for the GUI.