Estructura General

Duetto -Sound Lab

Nombre del Proyecto: duetto-SoundLab

Programadores:

* Gabriela
* Carlos Peña
* Hiram Borbolla
* Yasel Febles Torres

Plataforma: Multiplataforma

Lenguaje de Programación: Python versión 2.7

Librerías dependientes: matplotlib, numpy, scipy, PyQt v4.9.5, PyAudio 0.2.7, pyqtgraph 0.9.8, xlwt-0.7.5.

VCS: Git versión 1.8.1.2

# Descripción General

Sound Lab es un software diseñado para el procesamiento de señales acústicas con fines científicos. Es una aplicación de escritorio que ofrece una ventana principal en la que se pueden realizar acciones de edición de señales y otra en la que se puede realizar el cálculo de parámetros y clasificaciones sobre segmentos detectados en la señal. Brinda una interfaz visual altamente configurable.

El software consta de un manual de ayuda y de una documentación para los programadores. El documento DUETTO SOUND LAB ISSUES recoge los errores, contratiempos y los datos de interés para el equipo de desarrollo sobre el lenguaje, el framework o alguna librería que puedan surgir en la programación del sistema.

# Elección de Lenguaje y Librerías

Se decidió utilizar el lenguaje Python para la implementación de este sistema por sus características como lenguaje de alto nivel, su alta portabilidad entre plataformas y sistemas operativos y por el creciente grupo de desarrolladores que posee en el mundo. En el estudio de factibilidad realizado se analizaron varios lenguajes y plataformas. En el análisis final se incluyeron, además de Python, Java y C#. Aunque Java si posee una alta portabilidad al igual que Python fue rechazado por su pobre rendimiento debido a la necesidad del sistema de procesar grandes volúmenes de información y porque las librerías que se necesitaban eran pobremente implementadas en Java. C# fue rechazado por su baja (o complicada) portabilidad entre sistemas operativos y plataformas y por la carencia de librerías para el procesamiento científico de audio. (Cuando se habla de carencia de librerías se refiere a librerías gratuitas)

Las librerías que se utilizaron fueron las siguientes:

* Matplotlib 🡪 Métodos numéricos.
* Numpy 🡪 Métodos numéricos e implementación eficiente de arrays.
* Scipy 🡪 Métodos numéricos.
* PyQt 🡪 Versión del framework Qt(C++) para Python. Se utiliza para la confección de la interfaz visual.
* PyAudio 🡪 Grabación y reproducción de audio. Antes de su elección se valoró utilizar audioop de la librería estándar, Pyglet, el namespace Phonon de PyQt y audiolazy. Se decidió optar por PyAudio por su alto control de los dispositivos de audio y por su transparencia en la reproducción-grabación.
* Pyqtgraph 🡪 Implementación de un graficado eficiente. Objetos adicionales o extendidos para la interfaz visual. Antes de elegirla se realizaban las gráficas con matplotlib pero resultaron demasiado poco escalables en tamaño al crecer los ficheros que se procesaban.
* Xlwt 🡪 Exportación de datos a Excel

# Diseño de clases

El sistema está compuesto por dos grandes módulos: Duetto\_Core y Graphic\_Interface. Como lo indican sus nombres en el primero se incluye todo el procesamiento sobre señales de audio y en el segundo las clases que integran la interfaz visual. Este esquema se propuso para facilitar la reutilización del código en otros productos de Duetto o por terceros. Este diseño permite una fácil extensibilidad y mantenimiento del código. Se intentó obtener una alta modularidad e independencia entre los módulos.

# Duetto\_Core

Este módulo contiene los siguientes sub-módulos

* Audio Signals 🡪 Representación de las señales de audio.
* Clasification 🡪Métodos de clasificación.
* Segmentation 🡪 Métodos de segmentación
* SignalProcessors 🡪 Procesamiento de señales
* Cursors (deprecated) Manejo de cursores

Una documentación más extensa de este módulo puede hallarse en el documento DUETTO CORE GUIDE que se distribuye junto con el presente.

# Graphic\_Interface

Este módulo contiene los siguientes sub-módulos

* Dialogs 🡪 Las ventanas de diálogo que se muestran en las distintas funcionalidades del sistema.
* UI Files 🡪Los ficheros .ui que se generan por el framework PyQt para la generación de ventanas.
* Widgets 🡪 Los distintos controles visuales que se implementan o redefinen en la aplicación.
* Windows 🡪 Las distintas ventanas que componen el sistema.

Una documentación más extensa de este módulo puede hallarse en el documento GRAPHIC INTERFACE GUIDE que se distribuye junto con el presente.

# I18n

La internacionalización (y localización) se realiza mediante el framework PyQt. Todos los controles herederos de QWidget poseen un método tr que recibe un string y devuelve su traducción. Cada aplicación de PyQt permite la carga de un archivo con las traducciones para el idioma destino. En el libro *Rapid-gui-programming-with-python-and-qt* aparece un capítulo dedicado a este tema. A continuación se presenta un resumen de ese capítulo.

Here is how an internationalized application is created.

1. Create the application using QObject.tr() or QApplication.translate() for

all user-visible strings.

2. Modify the application to read in the locale-specific .qm (Qt message) files

at start-up if they are available.

3. Create a .pro file that lists the application’s .ui (Qt Designer) files, its .py

and .pyw source files, and the .ts (translation source) file that it will use.

4. Run pylupdate4 to create the .ts file.

5. Ask the translator to translate the .ts file’s strings using Qt Linguist.

6. Run lrelease to convert the updated .ts file (that contains the translations)

to a .qm file.

And here is how such an application is maintained.

1. Update the application, making sure that all user-visible strings use

QObject.tr() or QApplication.translate().

2. Update the .pro file if necessary—for example, adding any new .ui or .py

files that have been added to the application.

3. Run pylupdate4 to update the .ts file with any new strings.

4. Ask the translator to translate any new strings in the .ts file.

5. Run lrelease to convert the .ts file to a .qm file.