Audio Processor Outline

Overview

- Work with binary files and predefined formats.
- Manipulate data in various formats using arrays/vectors.
- Develop a terminal-based user interface.
- Utilize inheritance, abstract classes, interfaces, and composition.
- Implement a modular design following good development principles.

Project Description

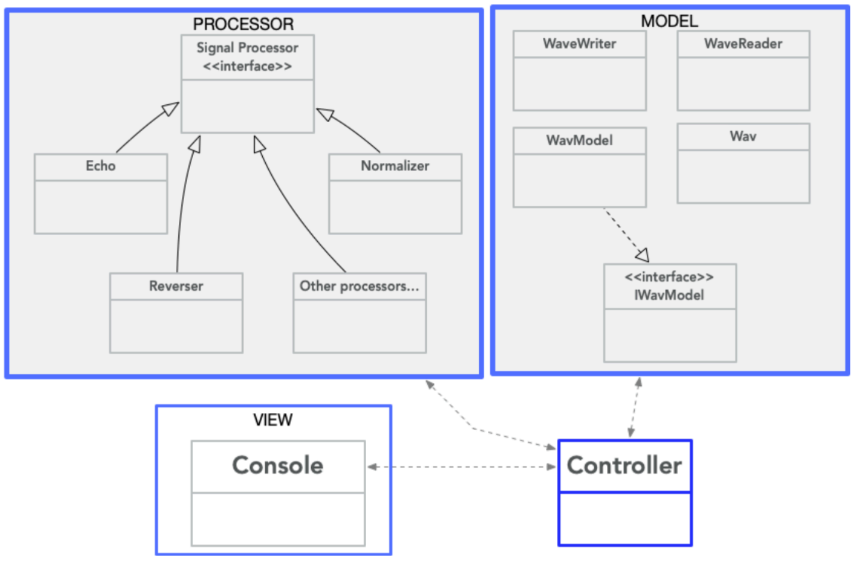
This project entails the creation of an application capable of loading audio data from WAV files, applying common audio effects such as echo and normalization, and saving the modified files for playback. The application will feature a console user interface and a set of processors for handling audio manipulation.

Background

- Digital audio files comprise numerous samples, which measure the amplitude of signals from microphones or electronic instruments at specific moments.
- Quantization is the process of converting sampled values into binary numbers.
- This process is commonly known as pulse code modulation (PCM).
- Sample rates
  - Modern digital audio systems commonly record at sample rates of 44,100 or 48,000 samples per second per channel.
  - Stereo recordings entail either 88,200 or 176,400 samples per second, respectively.
  - The sample rate directly influences the system's frequency response.
- Quantization (bit depth)
  - The bit depth determines the accuracy and noise level of a sample.
  - Common sample formats include 8, 16, 24, and 32 bits.
  - WAV files utilized in this project will be either 8 or 16 bits and may be mono or stereo.
  - See https://www.izotope.com/en/learn/digital-audio-basics-sample-rate-and-bit-depth.html
- File formats and compression
  - Audio files are binary, not text-based.
  - They can be compressed to reduce file size, but we will focus on uncompressed files.
  - Common compressed formats include MP3 and AAC.
  - Our project will specifically work with uncompressed WAV files.
  - See format for a wav file can be found here: http://soundfile.sapp.org/doc/WaveFormat/

Architecture

- The Model-View-Controller (MVC) pattern originated in Smalltalk for desktop applications.
- It has since become a dominant architecture for web applications.
- In this project, MVC serves as the prototype for the application architecture.
- Model
  - The model represents the system's data.
  - In database-backed applications, it acts as an interface between the application code and the database.
  - In our project, the model provides an interface to the WAV file data.
- View
  - The view manages displaying output and handling user input.
  - In the basic level, it utilizes console I/O.
  - For the challenge level, it employs the UI presented by the Qt framework.
- Controller
  - The controller oversees control flow and facilitates communication between the model and the view.
  - Various implementations of MVC may differ in how the controller is utilized.
  - In our project, the controller manages the application flow.
- Processor module
  - The processor module operates independently of the MVC pattern.
  - While some frameworks integrate similar functionality into the controller, our implementation keeps the processor code separate.
  - The controller utilizes the processor module as required.
  - The diagram below is an illustration of the system's major components.



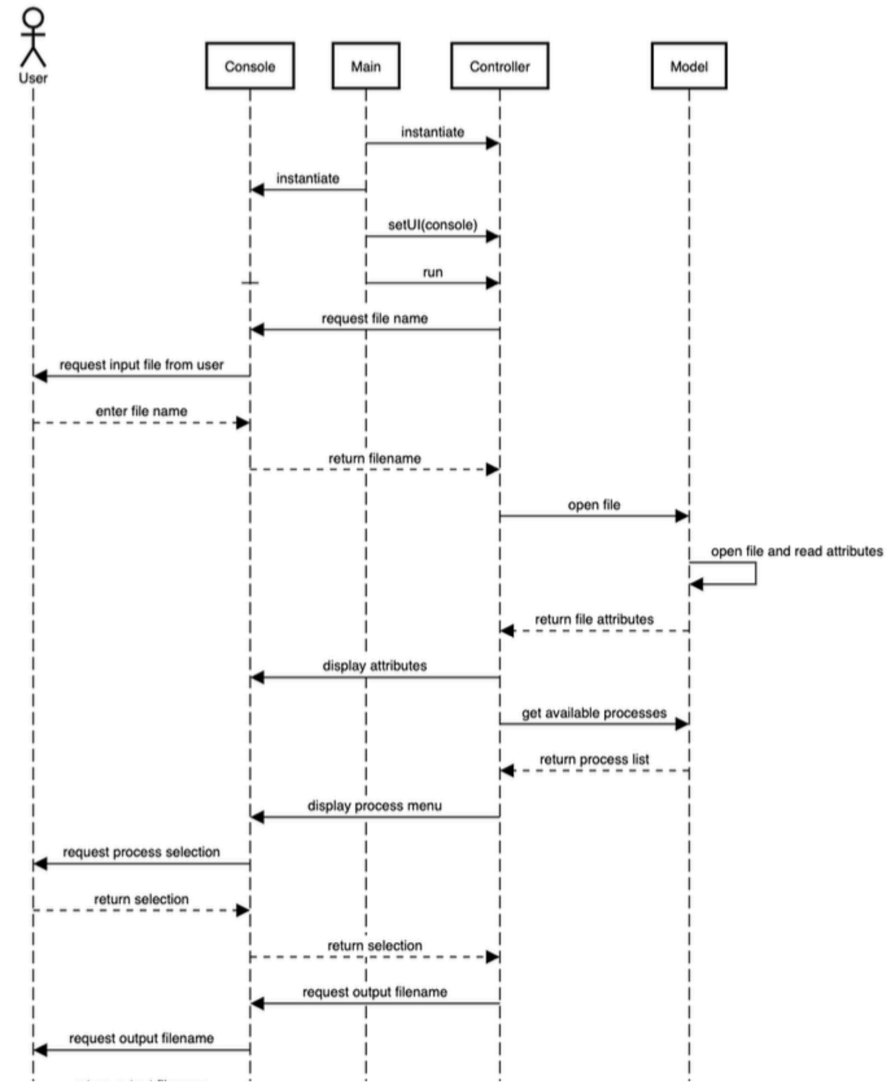
Managing Audio Data

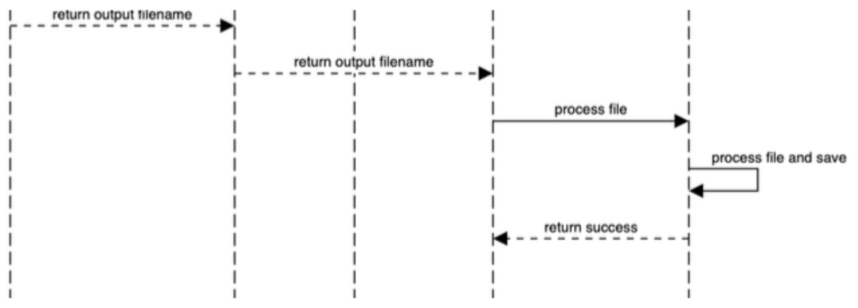
- Audio data in wav files is stored as integers. For processing purposes, however, integers can be problematic. This is in part because 8 bit data, for example, has its zero value at

128, while 16 bit data uses both positive and negative numbers with 0 representing the 0 value. The best approach is to
    - convert the integer values to floating point format
    - perform all processing using the floating point values
    - convert the floating point values to integers when writing out the result of processing
- The task involves processing 8 and 16-bit mono and stereo files. To manage the different formats, the project will employ templates, inheritance, or composition/aggregation.

Basic Level Requirements

- The basic level presents the user with a console interface.
- Program Flow
    - The sequence diagram illustrates the program flow and communication between primary objects.
    - It offers an overview of each object's responsibilities.
    - Specific functions required for implementing the flow are customizable.
    - Error handling, such as invalid filename warnings, is crucial, despite not being depicted in the diagram.



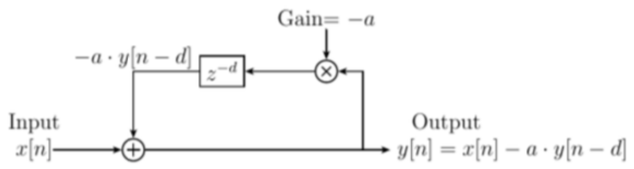


- File Metadata
  - The data to be displayed by the console:
    - file name
    - sample rate
    - bits per sample

        - stereo or mono
        - length of the file in seconds

Processors to be implemented

- Normalization
    - Algorithm: The largest sample value in the data is found, and then the data is scaled so that that value is the maximum possible value. This maximizes the amplitude of the final waveform.
    - Example for floating point data
        - Original data: n = {0,0.2,0.4,0,-0.3}
        - Largest absolute value: 0.4, so scaling value s = 1/0.4 = 2.5
        - Scaled result: n_scaled = {0, 0.5,1,0,-0.75}
- Echo
    - Algorithm: Samples are copied, scaled, and added to later locations in the sample buffer to create an echo effect.
    - Feedback architecture


Gain= $-a$
$-a \cdot y[n-d]$
z^{-d}
Input
$x[n]$
Output
$y[n] = x[n] - a \cdot y[n-d]$