CSM3
HIIT Mix
Final Report

Colorado School of Mines
Field Session Summer 2012

Benjamin Gilman
Ken “Kip” Shearer
Billy Dixon

Introduction

This product is intended for Elizabeth Hudd, a Science Content Specialist for Jefferson County Public Schools, and Keith Hellman, a Computer Science professor at the Colorado School of Mines. Most athletes will agree that different songs “pump them up” while others slow them down. Liz thought of the idea for this program during a workout where a slow song played during her workout and killed the pace she was attempting to reach. She then came up with the concept of linking music tempo with heart rate to create a uniquely fluid feel during a workout. This application appeals to athletes of every level, primarily the “Average Joe”, who wants to become cardiovascularly fit by training at various intensity levels for maximum benefit.

Product Vision

This product is intended for the use of athletes, personal trainers, and anyone else who wants to get in shape. It will increase the effectiveness of a workout by matching high tempo songs to high intensity intervals and vice versa. The application is made to be used with HIIT, that is High Intensity Interval Training, workouts to achieve optimal performance, however it may also be used with various exercise regimens. The application will create a playlist from the user’s music library, then splice the songs together into a single audio file, which will sync up with a given workout. To do this, it will have to accurately match songs to intervals based on a relationship between tempo and target heart rate. It will also allow users to customize nearly every aspect of their workouts.

Requirements

The HIIT Mix application is designed to enhance a workout by creating a workout-specific song file based on the user's personal music library.
Functional Requirements for the application include:

- A user interface allowing creation and modification of workout mixes, including:
    - A list of separate workout programs to choose for a variety of mixes
        - Each workout is based on High Intensity Interval Training (HIIT)
    - User customization of new and inventive workouts within the interface
    - Audio signals (or cues) to represent changes in workout intervals
        - The user will have the option to record or import recording of audio to represent a change in time (This option may be added at any number of interval transitions)
    - A feedback system (interface) that adjusts music selection based on patterns of the user
        - User has ability to rate an entire workout or individual songs within the application to optimize future workout mix selections

- The ability to analyze and generate songs within the application, including the following requirements:
    - An association between the beats-per-minute of a selected song and the desired heart rate for the interval
        - A song is chosen to most accurately reflect the current pace of the use
    - The ability to generate multiple song files for a desired workout

Non-Functional Requirements include:

- Desktop application environment
    - An simple "one-click" application
    - Development of software that can be used for multiple music libraries
- Ability to analyze an entire music library overnight

Architecture

HIIT Mix is a desktop application that can either be started by double-clicking the executable icon or running it from the command line. Using a recursive algorithm, it sorts through an entire music library, with a root folder selected by the user, to finally output a workout mix in the MyDocuments folder of the user.

The HIIT Mix architecture is primarily based on the Java language environment. To avoid creating complex functions that analyze and manipulate audio data, the project relies on an external application programming interface (API), named “The EchoNest”. The EchoNest company has written different APIs for four separate development languages, including Java and Python. All four languages call a central database managed by EchoNest, which contains analytical information for thirty-million songs.

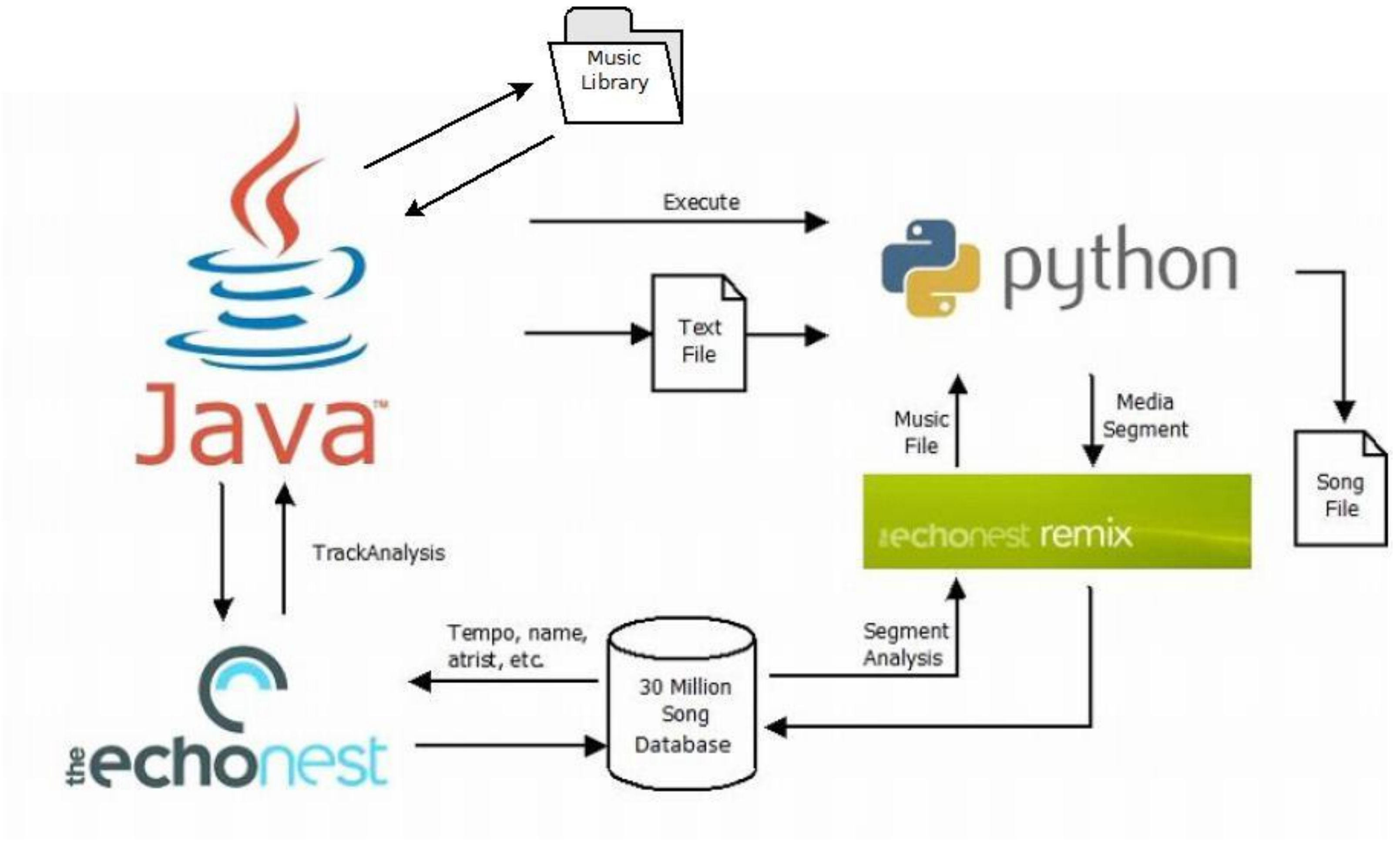
The Java portion of our application calls the EchoNest Java API (see Figure 1) for each song in a music library, returning a string of data for each requested information regarding a song (tempo, name, artist, etc.). In order to keep track of this data without having to make unnecessary calls to the database, these strings are then stored in a read-only text file for future reference.

Once the information in Java (provided by the API) is used to create a final “playlist” of song clips to be spliced into a single object, the information (file location, start time, and end time) are then written to a file to be read by an external Python application (see Figure 1). To do this, the Java portion of the code executes a system call to the command prompt of the user’s system. This call is given one string, which contains four file locations, each separated by spaces:

1. The python implementation .exe file with the installed modules necessary to call the EchoNest API
2. The “Slicer.py” file that performs the functions necessary to splice any number of media files together
3. A temporary “.saves” file which contains the written information described above
4. An output file based on the name of the current workout in the Java application.

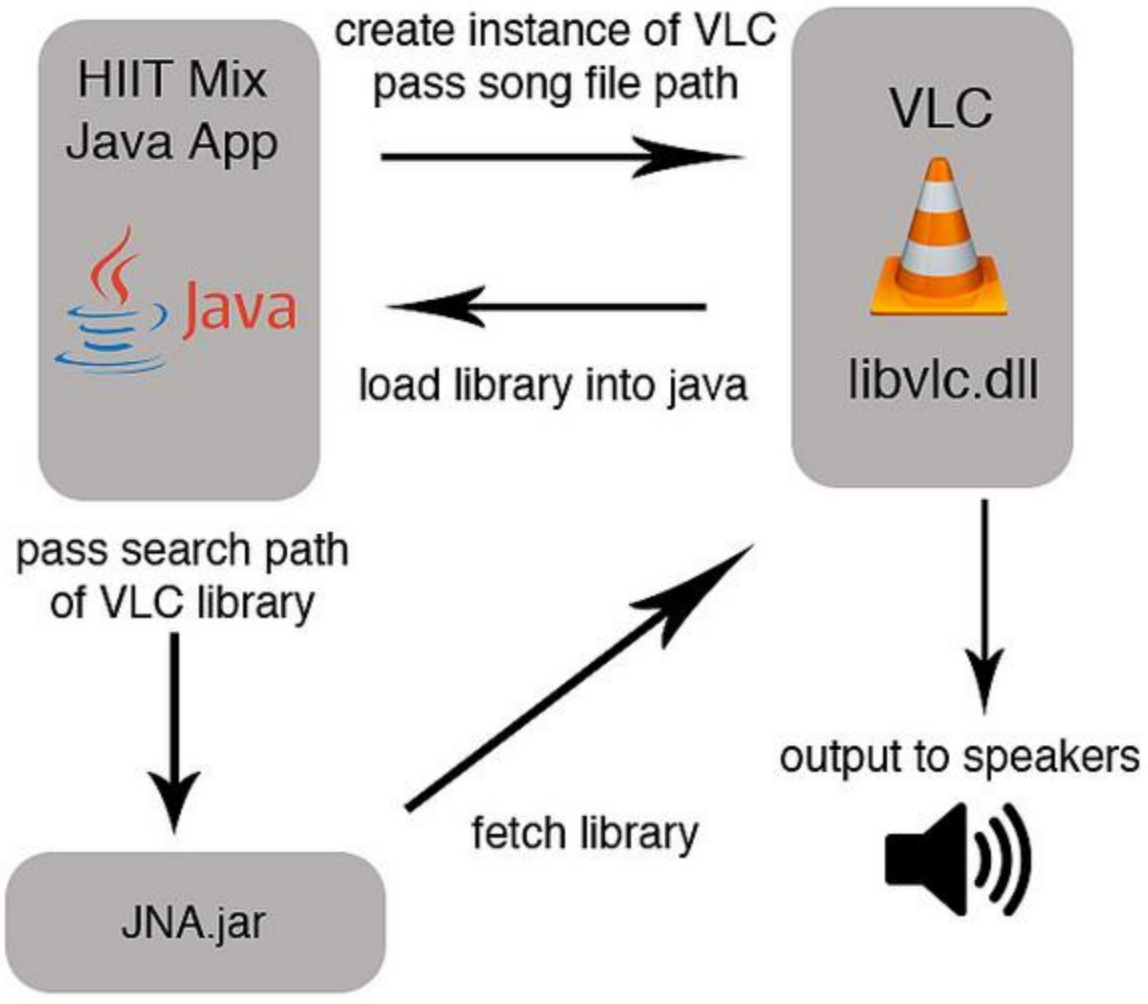
This call to the command line accesses the python compiler, runs the program, and provides enough input for the code to run smoothly.

The Python portion of our application calls the “EchoNest Remix” Python API (see Figure 1). This is the only EchoNest API that provides functions and classes for audio manipulation, albeit it handles analysis in the same manner as the other languages. Using imported Python modules to manage the EchoNest data structures, the programs reads in the file location of a song, which accesses the EchoNest database to then returns a list of strings regarding hundredths-of-seconds-long data of each music file. Using this information, the code appends the desired segments of time to each other for every song. This final data is “assembled,” then “compiled” to an output .mp3 final given by the fourth file location in the command line.



(Figure 1) EchoNest interfaces between Java and Python

Media playback in HIITMix is handled through the use of the java bindings for the VLC Media Player. We chose VLC due to the fact that it is Open Source, has easily implemented Java bindings, and is extremely robust, as it can handle virtually any media file we give it. VLC is written in C++, which is why we need to use the Java Native Access, which is a lightweight approach to calling code across languages, to bind the library to our Java application. In our application, we create an instance of VLC within our Swing interface using the VLC API. This allows us to leverage the power of VLC while it hides much of the complexity involved with media playback. In our MusicPanel class, we create an MP3Player object, which in turn creates an AudioMediaPlayer from the VLC library (see Figure 3). Whenever a song is selected in the playlist panel, its file location is sent from MediaPanel to the MP3Player object, which loads the file into VLC through a simple call to the AudioMediaPlayer's playMedia(String filepath) function, passing the file's absolute file path as an argument (see Figure 2). The AudioMediaPlayer begins playing the song at the beginning, however we do not want the song to start playing just because it was selected in the playlist. Therefore, we have a call to the AudioMediaPlayer's stop() function, which simply stops the file from playing but keeps it loaded in VLC, ready to resume whenever the user clicks the play button. This button acts as a toggle switch, playing the loaded file if not currently playing a song, and pausing the current song if it is already playing. It does this through a function call to the MP3Player from MusicPanel, which in turn calls the AudioMediaPlayer’s play() or pause() functions.



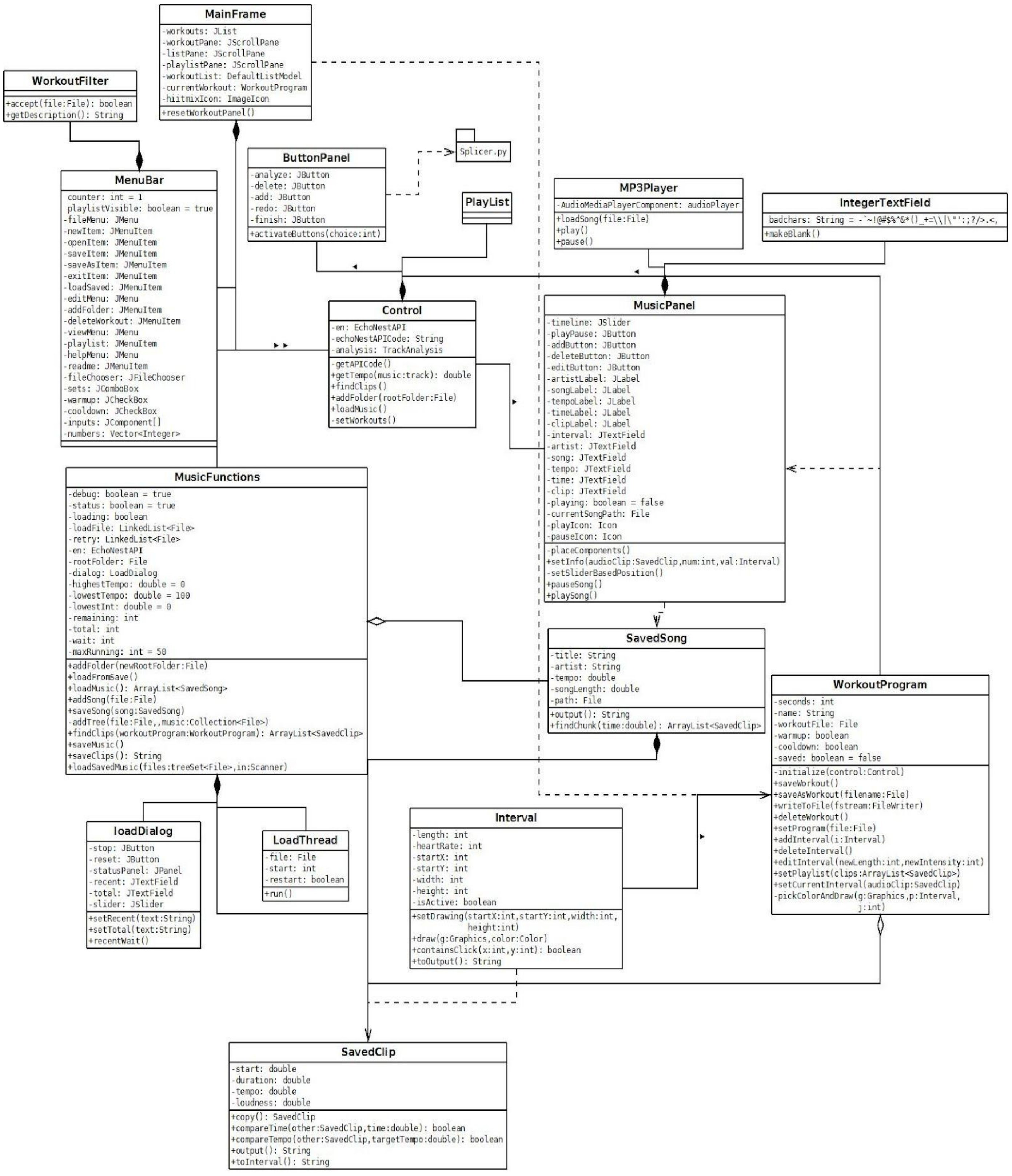
(Figure 2) Interface between Java, VLC, and JNA

There is also a timeline JSlider on the MusicPanel, which allows the user to select the part of the song they want to hear. Due to time constraints, it does not move as the song plays. Using a SliderListener, this slider makes a call to the setPosition(float value) function of the MP3Player’s AudioMediaPlayer, using the position of the slider once the user has released it. This function call works whether a song is playing or paused, however is best used when paused.

Using the VLC API requires that the full VLC application be installed on the user's machine, however we included both the 32-bit and 64-bit versions of VLC with HIITMix so the user is not required to download or install any additional software. In order to use this API, we had to package the vlcj framework as well as the latest version of JNA along with our application, as these .jars are vital in using VLC. Our application must be able to locate the native library for VLC within our include packages and load it using JNA. This can be problematic if VLC is not in the correct place, which is why we include it in our HIITMix folder, as we had to hardcode the location of the library to ensure the library is always found. More information on this can be found in Appendix C.

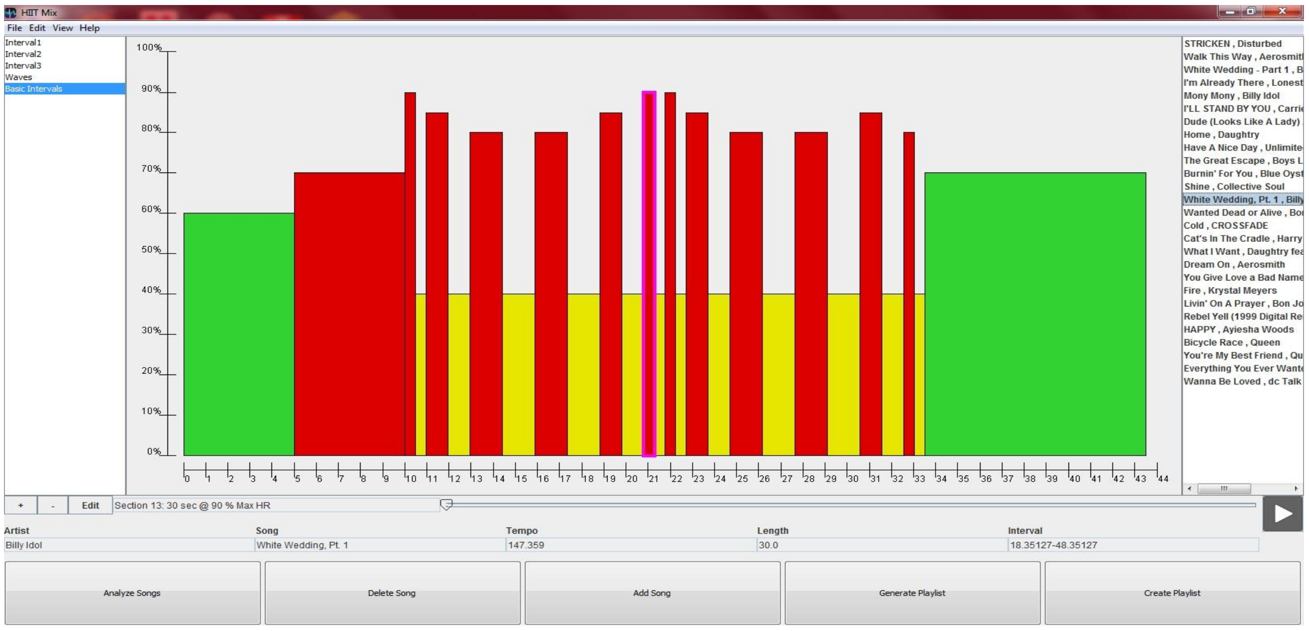
Technical Design

Figure 3 shows the UML diagram for our application which revolves around using the EchoNest API. The MusicFunctions class calls heavily on the API for identifying music passed through it, via threading. Due to the delay from uploading files, a large amount of threading is used to try and make the best use of the client's connection speed. Once values are returned from EchoNest, the data is stored into an ArrayList and later saved to a text file. Due to the private constructors used by EchoNest, we made our own classes, named SavedClips and SavedSong, to be able to save and load data from a text file. The two audio classes contain all the important data returned from the EchoNest database. SavedSong contains the song title, artist, tempo, and an array of SavedClips. SavedClips each contain the SavedSong they belong to, along with the start and end times for the clip. These clips are portions of the song that work well being used on their own so that a clip does not jump in the middle of the chorus or verse.



(Figure 3) UML of HIITMix

Our application begins with the MainFrame, which is our JFrame object that holds all other objects and components (see Figure 3). On the west side of this frame (in Figure 4), we have a JList for the list of workouts, which is nested in a JScrollPane to allow scrolling if the user has a large amount of workouts. In the center panel, we display the currently-selected WorkoutProgram. On the east, we have the playlist panel, which, like on the west side, is a JList nested in a JScrollPane. The south panel houses both the ButtonPanel and the MusicPanel. The ButtonPanel is a JPanel that holds all of the necessary buttons for our application, while the MusicPanel is another JPanel that handles media playback. Finally, a JMenuBar is added to the MainFrame to provide additional functionality without cluttering up the user interface.



Interval1
Interval2
Interval3
Waves
Basic Intervals

STRICKEN , Disturbed
Walk This Way , Aerosmitl
White Wedding - Part 1 , B
I'm Already There , Lonest
Mony Mony , Billy Idol
I'LL STAND BY YOU , Carrie
Dude (Looks Like A Lady)
Home , Daughtry
Have A Nice Day , Unlimite
The Great Escape , Boys L
Burnin' For You , Blue Oyst
Shine , Collective Soul
White Wedding, Pt. 1 , Billy
Wanted Dead or Alive , Bo
Cold , CROSSFADE
Cat's In The Cradle , Harry
What I Want , Daughtry fea
Dream On , Aerosmith
You Give Love a Bad Name
Fire , Krystal Meyers
Livin' On A Prayer , Bon Jo
Rebel Yell (1999 Digital Re
HAPPY , Ayiesha Woods
Bicycle Race , Queen
You're My Best Friend , Qu
Everything You Ever Wante
Wanna Be Loved , dc Talk

Section 13: 30 sec @ 90 % Max HR

Artist	Song	Tempo	Length	Interval
Billy Idol	White Wedding, Pt. 1	147.359	30.0	18.35127-48.35127

(Figure 4) Graphical User Interface, displaying layout of panels and buttons

The user selects a WorkoutProgram from the workout playlist on the west panel. Each WorkoutProgram has an optional warm-up and cooldown, along with an array of any number of intervals with a matching size array of SavedClips. Each interval is composed of an intensity, percentage, and a duration. When chosen, a workout is passed into the MusicFunctions, where a clip is found matching every interval as best it can, comparing tempo to heart-rate percentage and time. The conversion between tempo and heart-rate is simply the highest tempo song matched to the highest heart rate used in the workout, and slowest matched to the lowest heart rate used.

WorkoutPrograms are stored as text files, with the name of the workout at the top, followed by two booleans for checking if it has a warmup and/or cooldown. After this, each line has two integers, the first representing the number of seconds an interval lasts. The second integer indicates the percentage (out of 100) for the target heart-rate of that interval. These files are loaded to the WorkoutProgram class, which creates a unique object. Multiple WorkoutProgram objects are stored in an array in the Control class.

WorkoutPrograms extend JPanel, because they are added and displayed directly in the MainFrame (see Figure 3). Control has a pointer to the currently selected workout, which is needed by the MainFrame when it resets the workout panel after a different workout is selected. This way, the current workout is always displayed properly. Control also handles the creation of a playlist for a particular workout, as well as mapping the playlist save file to the workout save file.

Creating, deleting, and saving workout files is done through the MenuBar class. This class extends JMenuBar, and is added to the MainFrame (see Figure 3). Under the first menu item “File” we have the following options:

New	opens a JDialog where the user can select from a dropdown menu a number of sets, from 1-20, and two JCheckBoxes representing if they would like a warmup and/or cooldown, respectively.
Open	creates an instance of a JFileChooser that allows the user to import an existing .workout file. There is a FileFilter in place here to prevent users from opening any file other than .workout files.
Save	will save the current workout in place, unless it has not yet been saved and therefore has no save file associated with that workout.
Save as	will open another JFileChooser to allow the user to save the current workout with the name and directory of their choosing.

The next menu item in MenuBar is “Edit”

Add Music Folder	opens a JFileChooser that lets the user select a folder that contains music they want to be analyzed. Each music file in this folder is sent to the EchoNest API when the “Analyze Songs” button is pressed.
Load Saved Music	will read from the saved music file in the user’s My Documents folder. It then loads the music files that have been analyzed so far into the playlist when the user clicks the “Generate Playlist” button.
Delete Workout	will only delete a workout if it is not one of the preset workouts included in HIITMix install package.

Finally, we have the “View” and “Help” menu items. “View” simply allows the user to hide the playlist panel to get a better view of the workout graph by clicking the “Hide Playlist” option. “Help” has an option to open the readme file for HIITMix in notepad.

Within the ButtonPanel class is the MusicPanel class, which holds the JSlider for controlling media playback, a play/pause button, and an MP3Player object (see Figure 3). After a playlist of songs has been created, they are displayed on the east panel of the MainFrame in a list corresponding to the appropriate intervals. Each song can also be selected from the list to display the information regarding the clip being used (also included in MusicPanel). Through the use of the VLC API, it is possible to sample the audio from the selected audio clip before finalizing the playlist in the MusicFunctions class. The selected song is passed to the MP3Player class, which creates an instance of the VLC media player that then handles the media playback. When pressed, the MusicPanel’s play/pause button calls the MP3Player object, sending a call to the VLC media player to either play or pause the current song.

In order to load the proper version of VLC, we had to come up with a snippet of code to select the appropriate folder. This code checks if the user has a 64-bit operating system, as indicated by the Program Files (x86) folder. If this is the case, our application loads the 64-bit version of VLC. Otherwise, it loads the 32-bit version. The library file inside VLC is found and loaded using JNA after we pass it the appropriate file’s location.

```
String programFiles32bit = "C://HIITMix//lib//vlc_32bit";
String programFiles64bit = "C://HIITMix//lib//vlc_64bit";
// uses 64-bit VLC if user has 64-bit OS and 64-bit JRE, else uses 32-bit VLC
if(new File("C://Program Files (x86)//Java").exists()
&& new File(programFiles64bit).exists()){
        NativeLibrary.addSearchPath(RuntimeUtil.getLibVlcLibraryName(),
        programFiles64bit);
        Native.loadLibrary(RuntimeUtil.getLibVlcLibraryName(), LibVlc.class);
} else if(new File("C://Program Files//Java").exists()
&& new File(programFiles32bit).exists()){
        NativeLibrary.addSearchPath(RuntimeUtil.getLibVlcLibraryName(),
        programFiles32bit);
        Native.loadLibrary(RuntimeUtil.getLibVlcLibraryName(), LibVlc.class);
} else {
        // user does not have Java installed
        JOptionPane.showMessageDialog(null, "There was a problem
        loading VLC and/or Java.", "Error", JOptionPane.ERROR_MESSAGE);
        System.exit(1);
}
```

When a workout and playlist are satisfactory, they can be finalized, and the playlist is saved to a text file, which is then sent to our python functions. The python call reads the clips from the text file and loads each music file to a list. After converting the files to a .wav format so they can be spliced together in the EchoNest Remix API, they are then converted back into an .mp3 format to be finalized and saved into the My Documents folder. The .mp3 file can be played on any standard MP3 player.

The primary GUI functionality comes from the ButtonPanel, which contains the primary user interaction with the program.

Analyze Songs	Calls MusicFunctions to load all of the files previously loaded from the recursive file locator call. These songs are all added as SavedSongs and written to a text file after the loading is either canceled or completed.
Delete Song	Removes a specifically selected song from the SavedSongs array, and then replaces this gap with another generated song.

Add Song	Opens up a JFileChooser to individually choose a song to add to the array of previously-generated SavedSongs inside of the MusicFunctions class.
Generate Playlist	Sends the currently selected workout from the GUI to MusicFunctions, where SavedClips are matched to intervals to be returned for later use.
Create Playlist	Saves the currently selected clips to a file, which is then passed into python to be spliced together and exported to the final .mp3 file.

Design & Implementation Decisions

We began developing this application in Java, with the intention of creating a multi- platform application that would work on any machine with a JVM. However, during the development process, we ran into many situations where we were forced to decide between specific operating systems. Due to the large install base and the fact that we did not have reliable access to Mac or Linux machines, we decided to tailor our application for the Windows platform. We still wanted to support 32-bit and 64-bit versions of Windows, which became a problem with the use of the VLC API. However, this problem was resolved fairly easily, and the application works on both 32-bit and 64-bit Windows machines.

We chose to use the EchoNest API to acquire song information including the song tempo, official title, artist, length, and “segments” of the song. The segments can be used for “good” splicing points to avoid starting a song in the middle of a chorus, at the very end of the song, or other “bad” times. We also decided to use the EchoNest Remix module for Python because it is the only media-editing API that EchoNest offers, though with the same consistency of information as the API we use for Java. Using the Echo Nest API to identify, analyze, and manipulate songs can take a considerable amount of time, so all of the data collected from the analysis is saved to a text file to be used at a later time. Therefore, it avoids re-sending data to the EchoNest database each time the program loads.

We considered developing a mobile version of the product for Android or iOS. However, this was not requested specifically by the client, and would have limited the overall usability of the end product. Competition applications use specific playlists or songs, but do not implement an association between a user’s heart rate and the beats-per-minute (bpm) of a media file. The competition also require an Android or iOS device, while the goal of our application was to be compatible with any MP3 player.
Our application currently operates as a desktop application. This is also beneficial, in that, most users do not store their entire music library on their mobile devices. In addition, laptop and desktop machines have more processing power, which is necessary for the splicing of a generated playlist. Desktop applications may also be left the run overnight if the user has a large amount of music to analyze or a slow internet connection. Making use of the EchoNest API on a mobile device would not only be slower, but also more costly if the user intends to use the application over a cellular data network with a limited data plan.

Results

The goal of this project was to create a multi-platform desktop application that would analyse the user's music library, generate playlists for HIIT workout routines, and splice the songs in the playlist to produce an MP3 file that can be used on any compatible device. The playlist created would, as closely as possible, match the tempo (BPM) of the song to the target heart rate of each interval in the workout outline. Our application meets all of our client's functional requirements, as it successfully analyses music, creates custom workout routines, and splices songs into an MP3 file for a specific workout. Due to licensing issues however, the EchoNest APIs only allow 120 calls to its server per minute, resulting in a slow return in information. In addition, because of the dependency of EchoNest, the user must always be accessing the internet to run to program successfully. The implementation of audio cues, a song rating system, and a basic artificial intelligence that learns the user's preferences were all features that did not make it into the final product due to time constraints.

Looking Forward

Here are a few items that could be addressed in the next version of HIITMix:

- Improving slicing functionality.
- Making the application multi-platform as originally intended.
- Adding audio cues.
- Improve the MP3 output file's sound quality.

Known Bugs

There are several known issues with the current build of the application:

- When songs are playing, the timeline slider does not move as the song progresses.
- Trying to create a playlist when the workout selected has more intervals than the user has songs analysed results in an error.
- Deleting custom made workouts is error-prone, especially when the workout has not been saved.
- Editing custom workouts is problematic, due to the fact that the WorkoutProgram does not resize after it is initialized. The functionality of the program is not affected, however the workout will not display properly.

Appendix A: Resources

1. VLC was used for media playback within the application. Both 32-bit and 64-bit Windows versions of VLC are packaged in the HIITMix folder to avoid forcing the user to install the program. The packages used can be updated as needed, and are availabe here:
http://www.videolan.org/vlc/index.html#download
2. The Java bindings of the VLC API were used to make use of VLC's functionality within our application. Information on vlcj can be found here:
http://code.google.com/p/vlcj/
3. JNA was necessary to bind the VLC native library in our application, and its two files, jna.jar and platform.jar, are packaged in the HIITMix.jar. These files can be found at:
https://github.com/twall/jna
4. The python framework used for the "EchoNest Remix" API is included in the installation materials, and will take no effect on any Python version installed on a machine. These files should never be updated nor removed.
5. The EchoNest API relies on a developer API code, which can be found in a text file in the installation materials. If there is ever a time where the code is obsolete, you can register a new one to replace it with at: https://developer.echonest.com/account/register
Note: the python code does not directly refer to this document, and needs to be updated in the appropriate line as well

Appendix B: Installation

This application is compatible with 32-bit and 64-bit versions of Windows XP, Vista, and 7. It launches using the Java Virtual Machine, which must be installed in the user's machine prior to use of HIITMix.
To install:
1. Unzip the provided folder and copy the HIITMix folder onto your C: drive. This is the folder containing the executable jar file that will launch HIITMix. You can create a shortcut on your Desktop to this file if desired.
*Note: Do not put this folder in your Program Files as it will not work due to the space in the path name. This causes issues in our python script, which is a vital part of HIITMix.
2. Copy the HIITMixDocuments folder into your My Documents folder.
3. You are now ready to launch HIITMix, assuming you have Java installed on your machine. If you are unsure if Java is installed or if it is up-to-date, head over to http://www.java.com/en/ to update or install Java.
4. Double click on HIITMix.jar to launch.
5. If clicking the jar file fails to launch the application, open Command Prompt and enter the following line:

```
java -jar C:\HIITMix\HIITMix.jar
```

*Note: Do not close the command prompt window until you are finished using HIITMix as this will exit the application.

Appendix C: VLC

HIITMix makes use of vlcj, a 3rd party framework that provides Java bindings for the VLC media player. It uses JNA to load the library of the VLC API, specifically the libvlc.dll file contained within the packaged VLC application downloaded from http://www.videolan.org/. The jars for JNA, jna.jar and platform.jar, as well as vlcj-2.1.0.jar are packaged inside HIITMix.jar, while the packages containing the actual VLC libraries are contained in the HIITMix/lib folder. These jars are imported to the MP3Player class in HIITMix, as shown below.

```
import uk.co.caprica.vlcj.binding.LibVlc; //vlcj
import uk.co.caprica.vlcj.runtime.RuntimeUtil; //vlcj
import com.sun.jna.Native; //jna, platform
import com.sun.jna.NativeLibrary; //jna, platform
```

The VLC library is loaded into the working environment using JNA, which requires the path to the proper version of VLC for the user's operating system. This was accomplished by following and modifying an example application that was found at: http://www.capricasoftware.co.uk/vlcj/tutorial1.php.