Digitally Distributed Orchestra: DDO

Andrea Compton, Ashley Hedberg, and Tyler Lubeck

December 5, 2014

Contents

1 Project Goals

Our goal for this project was to create a digitally distributed orchestra. Given a representation of a song (i.e. digitized sheet music), one computer would assign various client machines a note to play. The client machines would then play the song together.

At a minimum, we wanted our orchestra to be able to play a hard-coded sequence of frequencies and durations in unison. Our stretch goals included being able to play any song written in a specific format. This would involve handling songs which played notes in multiple octaves as well as those containing sharps and/or flats.

2 Final Design

2.1 Client-Server Relationship

For this project, we need to implement some notion of clients communicating with a central server. We considered two options for this.

2.1.1 Option 1: Twisted

The first was using the Twisted Python package. Given that parsing of the song file and playing the audio must be done in Python, using this option would allow the entire project to be implemented in one language. Twisted has a built-in notion of connections being made and lost, which would allow our server to more easily keep track of clients and distribute song notes accordingly. It also has a notion of executing tasks at specific time deltas, which may help enforce a consistent concept of time among the clients. However, none of us have used this framework before, and it involves some fairly low-level asynchronous calls, which may prove to be a challenge.

2.1.2 Option 2: Erlang

The second was implementing the server/client relationship in Erlang and the file/audio parsing in Python. This option would allow us to take advantage of Erlangs built-in message passing for note distribution. File parsing and sound generation would be more easily accomplished in Python, and we could use the ErlPort package to communicate between the two languages. However, we would have to implement the concept of client connections being lost ourselves.

2.1.3 Decision

We opted to use the first method. Being able to redistribute notes when a client drops seems like an important requirement for a digitally distributed orchestra. This functionality is built in to the Twisted servers, whereas we would have to implement it ourselves in the Erlang clients.

2.2 Sheet Music Representation

2.2.1 Inspiration from Guitar Tabs

Our initial text-based representation of a song was the following:

```
Song Title
Beats per minute (an integer)
Note 1: ---X---X---
Note 2: --XX---X---
...
```

Note 1, 2, etc. will be A, B, and so on. A dash represents a beat where the given note does not play, and an X represents a beat where the given note does play.

2.2.2 Refinement of Long Note Representation

After we began implementing the parser for this format, we realized that this was not the best representation. We modified our song format to include the dash character (-), numbers, and the letter `h`. A dash represents one beat of silence. Numbers represent the number of times the note plays in one beat. This allows us to represent quarter notes, eighth notes, and so on. The `h` indicates a hold of the previous note, and allows us to represent notes lasting longer than one beat. "Hot Cross Buns," for example, looked like this in the new format:

```
hot cross buns
60
B:1--1----1---
A:-1----22-1--
G:--1h22----1h
```

We then discovered that handling held notes with this format was cumbersome. If a 1 was found while parsing, it was difficult to discern whether the note played for one beat or was supposed to be held over multiple beats. As a result, we again modified our format so that all beats over which a note was supposed to be held were designated by the `h` character. This led us to the following representation of "Hot Cross Buns":

```
hot cross buns
60
B:1--1----1---
A:-1----22-1--
G:--hh22----hh
```