System Architecture

The system is broadly divided into two components, the audio engine and iPhone application. The audio engine is implemented as a series of pure-data patch files which provide functionality relating to audio playback, loading and saving audio files and synthesised instruments. The iPhone application is written entirely in Objective-C and provides the application’s user interface and the algorithms for extracting image properties and translating them into drum patterns and melodies.

The following diagram shows the static state of all (non-utility) objects in the application, a detailed description of each follows:



# Audio Engine

The audio engine is broken down into 4 key objects which are represented on the diagram. These are contained within the “.pd” files included in the source repository. Pure-Data introduces new terminology for what object oriented programming calls “objects”. Pd files which are intended to operate as a standalone program are called “patches”. Objects which are intended to be a component in a patch are called “abstractions”, and within either object there may be a number of sub-patches. In the diagram above, SoundSystem is the patch which provides the audio functionality behind our application. Polytracker, Vocoder and SimpleInstrument are abstractions. The pd files may be modified in either Pure-Data or Pure-Data extended which may be downloaded at <http://puredata.info/downloads>.

### SoundSystem

The Objective-C application loads the SoundSystem patch and libpd fetches any required abstractions behind the scenes. The SoundSystem patch provides the interface to all functionality required by the application and does so through responding to messages provided by the message system of libPd. This includes playback controls, controlling song length and instruments, audio playback and capture to file. SoundSystem accepts the following messages:

* startPlayback – The application should send this event to begin playback
* stopPlayback – The application should send this event to stop playback
* pausePlayback – The application should send this event to pause playback, the application itself keeps track of the play, pause and stopped states for updating the UI.
* recordSong – The application should send this event to begin recording to a file. It is important the application sends no other events during this as it could cause various bugs. We solve this issue by displaying a “saving..” dialog and disabling input until completion. Playback progress notifications are used during recording to update a progress bar.
* $0-melodyVolume – Controls the volume of the Vocoder instrument
* $0-drumVolume – Controls the volume of all SimpleInstruments
* $0-numInstruments – Allows instruments to be disabled, currently always set to 5
* $0-length – Set the length of the track, the song pattern is always updated immediately after
* $0-loopPlayback – Enable or disable looping playback
* $0-soundFile0, $0-soundFile1 … - Specify the audio sample used by each instrument, soundFile0 is the Vocoder instrument. The remaining inputs are drum samples.

SoundSystem also outputs the following messages which the application should listen for:

* $0-notifyProgress – Provides notification of playback position within the track, value is the number of notes played.
* stopPlayback – Event fired after the playback reaches the end of the track if repeat is disabled
* recordDone – Event fired after completing recording to a wav file, we had to include a large pause before firing this to allow Pd some time to flush data and close the file as Pd offers no way to receive a notification when the file has been closed.

### PolyTracker

Polytracker is an abstraction which provides a mechanism for packing multiple instrument tracks into a single table. A table in Pd offers functionality similar to array lookups and can be supplied data from an array in the application. Upon receiving a position in the sequence of notes in the left inlet PolyTracker outputs the values of each note for each instrument. The right inlet to polytracker accepts messages “numInstruments” and “length” which control the size of the array and number of notes fired off for each input. PolyTracker provides a global array called “pattern” which should be provided a sequence of values. Sequences are stored in memory such that the sequence of notes for instrument 1 follows directly after the sequence of notes for instrument 0 and so on.

### Vocoder

Vocoder is a complex patch which is a modified version of the phase-vocoder example provided with Pd. The patch is responsible for playback of a soundfile with a modified pitch at run-time. The complexity of this object is in the implementation of time stretching, as the object allows for the length and pitch of an instrument sample to be modified individually. When running on the phone this object consumes a large amount of processing power, therefore we are limited to playing a single pitch modified instrument in our app. In future versions of the app if this could be optimised more instruments could be played concurrently which would allow for a baseline for example. One strategy for optimising this might be to re-implement performance critical objects in libpd using the audio dsp of the iPhone. The left inlet of the vocoder object receives a pitch modification value (measured in cents) and the right inlet receives the filename of the instrument sample.

### SimpleInstrument

SimpleInstrument handles playback of an audio file with modified volume. We use this for percussive instruments such as drums, but otherwise it behaves similarly to the vocoder object. SimpleInstrument requires significantly less processing power than the vocoder instrument and so should be used instead where possible. The left inlet receives the volume of playback, which should lie in the range of 0-1 and the right inlet receives the filename of the instrument sample.

# Objective-C Application

Test Plan