Piper: Audio Feature Extraction in Browser and Mobile Applications

Lucas Thompson[*], Chris Cannam, and Mark Sandler
Centre for Digital Music
Queen Mary, University of London
{lucas.thompson, c.cannam, mark.sandler}@qmul.ac.uk

ABSTRACT

Piper is a protocol for audio analysis and feature extraction. We propose a data schema and API that can be used to support both remote audio feature extraction services and feature extractors loaded directly into a host application. We provide a means of using existing audio feature extractor implementations with this protocol.

In this talk we demonstrate several use-cases for Piper, including an "audio notebook" mobile application using Piper modules to analyse recordings; a web service for remote feature extraction; and the refactoring of an existing desktop application, Sonic Visualiser, to communicate with a Piper service using a simple IPC mechanism.

1. MOTIVATION

Our key motivation is to make use of existing audio feature extractors, for example pitch and chord estimators or beat trackers already implemented as Vamp plugins[1] in C++, while rapidly prototyping applications in a browser context. This is enabled by compiler backends such as Emscripten [13] and emerging standards such as WebAssembly [5], with which one can compile feature extractors as downloadable modules for consumption in a browser with satisfactory performance, and by tools for audio visualisation in browsers such as the Waves-UI libraries [10]. We construct a module interface that uses well-defined data structures and a set of common API verbs, so that we can recompile existing extractors to this interface and also use essentially the same protocol when providing feature extraction services across a network.

2. EXISTING WORK

The extraction of feature descriptors from audio for use in audio and music related web applications, and other online services, is nothing new. Web applications such as Songle carry out a series of analyses on the server, extracting musically meaningful descriptors to inform visualisations of song segments, melody lines, chords, and beat annotations which are aggregated in an interactive time-aligned view in the web application's front end [4]. Similarly, Chordify—a web application for automatically generating interactive and printable chord charts from user provided audio recordings—extracts harmonic and metrical structure descriptors using Vamp feature extraction plugins on the server, which are used as input to a method for finding chord candidates [3]. In the context of desktop applications for specific audio analysis tasks, an application such as Tony, for melody line annotation and transcription, also relies heavily on Vamp plugins, principally the pYin pitch tracker [11]. (A goal of our work is to support this kind of task-focused application in the browser as well.)

There are several open-source JavaScript libraries for low-level feature extraction, including Meyda [12] and JS-Xtract [6] which contain code manually rewritten from existing C++ libraries. A number of approaches also exist in the web audio space for defining audio effect modules in the vein of DAW plugins like VST and AudioUnits. Some are written in pure JavaScript for use in the Web Audio API graph [2], and some experiment with the use of Emscripten and Portable Native Client (PNaCl) as mechanisms for bringing existing implementations in C and C++ to the browser without manually translating the methods to JavaScript [9, 8]. Some approaches relating to cross adaptive audio effects also include feature extraction [7].

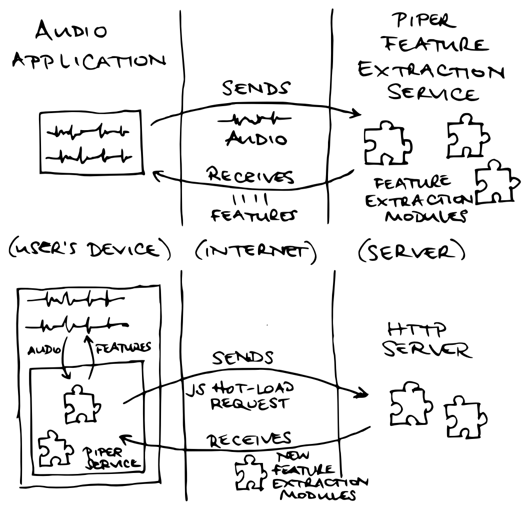

Figure 1: Two architectures supported by Piper
Above, with a central feature-extraction service. Below, with modules hot-loaded into a client-side Javascript application.

[*]Supported by EPSRC Platform Grant EP/K009559/1
[1]http://vamp-plugins.org

3. THE PIPER PROTOCOL

Piper is a language-agnostic protocol for communicating with a feature extraction service, which captures the life cycle of extracting feature descriptors from blocks of audio samples in a small set of API verbs. Feature extraction services can exist as modules within a client application, for scenarios similar to the Tony application, or on a server, for applications more like Chordify. Communicating with a feature extraction service in this way provides a clear separation of concerns in an application's code-base, which may correspond to a useful separation between components in different languages or on different hardware. For example, version 3.0 of the Sonic Visualiser application[1] uses Piper to operate Vamp plugin feature extractors in order to move them out of the main process for reliability purposes.

A Piper service responds to 5 verbs: *list, load, configure, process*, and *finish*.

A client application first queries the available feature extraction methods using the *list* method. It then communicates with a feature extractor using a series of stateful requests, whose parameters are informed by the dataflow dependencies used when operating a Vamp plugin:

- *load* — Instantiates the requested extractor. Returns metadata describing the outputs of the extractor, details of configurable parameters, and a unique handle for referring to the extractor instance in subsequent requests.
- *configure* — Sets up the extractor with parameter settings and framing information for the audio input. Returns metadata describing the shape of the feature outputs (corresponding to matrices, vectors etc).
- *process* — Given a block of audio samples and a timestamp, calculates and returns a structure to the client containing features derived from that block, of the form described in the configure response. This method is called multiple times in sequence, as needed, until the audio is finished.
- *finish* — Notifies the end of the audio stream, allowing processing to finish, clean up, and return any remaining calculated features.

This stateful approach supports interaction with a feature extraction service at a low level, necessary for applications like Sonic Visualiser where audio is read block-by-block from disk and features extracted in a streaming fashion.

We also propose a stateless, higher level API, where given an entire audio file and details of the desired extractor configuration, the framing of the audio happens server side and the features can be returned to the client either in a streaming fashion, as they're calculated, or aggregated in a single response after all processing, without the need for the more fine-grained life cycle requests.

4. PIPER LIBRARIES

We provide a number of open-source libraries[2] relating to the Piper protocol, including:

- A JavaScript library for writing client and servers;
- A C++ library for existing Vamp host environments, allowing for communicating with Piper servers;
- C++ headers for adapting existing Vamp plugins into Piper modules, either as native libraries or via Emscripten or WebAssembly;
- JSON and Cap'n Proto schemas defining the request and response payloads.

[2]http://github.com/piper-audio/

5. REFERENCES

[1] C. Cannam, C. Landone, and M. Sandler. Sonic visualiser: An open source application for viewing, analysing, and annotating music audio files. In *Proceedings of the ACM Multimedia 2010 International Conference*, pages 1467–1468, 2010.

[2] H. Choi and J. Berger. WAAX: web audio API extension. In *13th International Conference on New Interfaces for Musical Expression*, pages 499–502, 2013.

[3] W. B. de Haas, J. P. Magalhães, and F. Wiering. Improving audio chord transcription by exploiting harmonic and metric knowledge. In *Proceedings of the 13th International Society for Music Information Retrieval Conference*, pages 295–300, 2012.

[4] M. Goto, K. Yoshii, H. Fujihara, M. Mauch, and T. Nakano. Songle: A web service for active music listening improved by user contributions. In *Proceedings of the 12th International Society for Music Information Retrieval Conference*, pages 311–316, 2011.

[5] A. Haas, A. Rossberg, D. Schuff, B. L. Titzer, D. Gohman, L. Wagner, A. Zakai, J. Bastien, and M. Holman. Bringing the web up to speed with webassembly. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2017.

[6] N. Jillings, J. Bullock, and R. Stables. Js-xtract: A realtime audio feature extraction library for the web. In *Late-Breaking/Demo Session of the 17th International Society for Music Information Retrieval Conference*, 2016.

[7] N. Jillings, Y. Wang, J. D. Reiss, and R. Stables. Jsap: A plugin standard for the web audio api with intelligent functionality. In *Audio Engineering Society Convention 141*, 2016.

[8] J. Kleimola. Daw plugins for web browsers. In *Proceedings of the 1st Web Audio Conference*, 2015.

[9] J. Kleimola and O. Larkin. Web audio modules. In *Proceedings of the 12th Sound and Music Computing Conference*, 2015.

[10] B. Matuszewski, N. Schnell, and S. Goldszmidt. Interactive audiovisual rendering of recorded audio and related data with the wavesjs building blocks. In *Proceedings of the 2nd Web Audio Conference*, 2016.

[11] M. Mauch, C. Cannam, R. Bittner, G. Fazekas, J. Salamon, J. Dai, J. Bello, and S. Dixon. Computer-aided melody note transcription using the tony software: Accuracy and efficiency. In *Proceedings of the First International Conference on Technologies for Music Notation and Representation*, 2015.

[12] H. Rawlinson, N. Segal, and J. Fiala. Meyda: An audio feature extraction library for the web audio api. In *Proceedings of the 1st Web Audio Conference*, 2015.

[13] A. Zakai. Emscripten: an llvm-to-javascript compiler. In *Companion to the 26th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pages 301–312, 2011.