APPENDIX D

SOFTWARE FOR AUDIO ANALYSIS

The process of developing software for audio analysis can generally be split into two steps, the algorithmic design including the prototyping as well as the evaluation and the implementation of "production-quality" software which is made available to customers and users. The final software is commonly implemented in the programming languages *C++* [482] and *Java*.[1] Many developers consider C++ as the language that allows the most workload-efficient implementations and Java as the language that allows a comparably rapid development cycle.

While C++ and Java are also used for algorithm prototyping, other languages such as *Python*,[2] often extended by the packages *NumPy*, *SciPy*, and *IPython*,[3] are more common. *Matlab*[4] (or the largely compatible open-source software *Octave*[5]) provides an environment with a simple script language with a large set of already included routines as well as various possibilities of data visualization. Furthermore, visual audio programming environments such as *Max*[6] or *Pure Data (PD)*[7] are used especially in the context of designing real-time audio algorithms.

[1] Java. http://www.java.com. Last retrieved on Jan. 28, 2012.
[2] Python. http://www.python.org. Last retrieved on Jan. 28, 2012.
[3] NumPy. http://numpy.scipy.org, SciPy. http://www.scipy.org, IPython. http://ipython.org. Last retrieved on Jan. 28, 2012.
[4] Matlab. http://www.mathworks.com/products/matlab. Last retrieved on Jan. 28, 2012.
[5] Octave. http://www.octave.org. Last retrieved on Jan. 28, 2012.
[6] Max. http://cycling74.com/products/max. Last retrieved on Jan. 28, 2012.
[7] Pure Data. http://puredata.info. Last retrieved on Jan. 28, 2012.

The aim of this appendix is to provide a short overview about software supporting the design and implementation of audio analysis algorithms. The solutions presented below include toolboxes, libraries, frameworks, and applications. They cover a multitude of ACA-related tasks, and each solution focuses on improving or accelerating the prototyping or development process in different areas. The main areas of interest can be identified as

- *annotation* of (audio) files and generation of ground truth data,
- *feature extraction*,
- *pattern recognition* and machine learning algorithms, and
- *visualization* of features and properties of (collections of) audio files.

The software may also differ in the level of user expertise (developer vs. non-technical user), in its real-time capabilities, and in the input data sources (e.g., audio vs. MIDI).

D.1 Software Frameworks and Applications

This section presents software frameworks offering comprehensive possibilities for file input and output, signal processing, audio analysis, and machine learning. Thus, they offer the prototyping and possibly the building of ACA systems. These frameworks do not depend on any specific environment and do not require the usage of other software.

D.1.1 Marsyas

Marsyas is a software framework in C++ designed by George Tzanetakis. It offers both real-time and offline processing of audio data. While Marsyas features also audio synthesis and (effect) processing, its emphasis is on MIR. It is written in C++ but allows users access to configuring and using Marsyas, for instance, through a Python front end.

Since its first publication [297], Marsyas has grown into a powerful set of classes which allow to construct basically any MIR-related system [483]. Numerous of the algorithms described in this book are already implemented in Marsyas.

The design of Marsyas is modular; it consists of processing blocks which can be composed into data flow networks. A block can represent different functionality; it can, for instance, be an audio or text file IO, a sound card audio IO or an algorithmic operation on audio samples or features. Marsyas also includes a set of machine learning tools for training and classification.

The latest information and the source code of Marsyas is available from the project's web page.[8]

D.1.2 CLAM

A software framework developed by a research team of the Music Technology Group of Pompeu Fabra University, Spain, is *C++ Framework for Audio and Music (CLAM)*. It is implemented in C++ and its emphasis is on spectral modeling and spectral processing.

The first version of CLAM was presented in 2002 [484, 485]. The basic design principles seem to be similar to Marsyas in the way that the researcher can build networks of

[8]Marsyas. http://marsyas.info. Last retrieved on Jan. 28, 2012.

individual processing blocks. Nowadays, CLAM comes with a visual network editor that allows the user to build the processing networks through a graphical user interface [486]; its main focus remains on audio processing and less on audio analysis. It does, for instance, not include any machine learning blocks.

The latest information and the source code of CLAM is available from the project's web page.[9]

D.1.3 jMIR

The software framework *jMIR*, implemented in Java, is developed and maintained at the Music Technology Group at McGill University. Its focus is on machine-learning-related tasks on music such as musical genre classification and mood classification.

The framework consists of a set of tools for feature extraction and machine learning algorithms as well as tools for meta data annotation. The tools can also be used individually [487, 488]. One of the most notable differences to the functionality of other frameworks is the built-in functionality to extract features directly from symbolic data such as MIDI. It comes with both an audio and a MIDI ground truth data set for musical genre classification.

The latest information and the source code of jMIR is available from the project's web page.[10]

D.1.4 CoMIRVA

CoMIRVA is a Java framework for the visualization of large collections of music data. It is developed at the Johannes Kepler University Linz.

The software offers various possibilities of analyzing and visualizing the relationship (such as the similarity) of music files in large collections. This functionality is complemented by some feature extraction routines and web data mining tools [489].

The latest information and the source code of CoMIRVA is available from the project's web page.[11]

D.1.5 Sonic Visualiser

Sonic Visualiser is a software for the visualization of various properties of one or more audio files. The project was initiated at the Centre for Digital Music at Queen Mary University of London.

Sonic Visualiser was first presented in 2006 and has been under development since then [490]. It offers numerous possibilities to visualize and annotate audio. Compared to the frameworks presented above, it does not focus as much on the algorithm developer but on (non-technical) expert users such as musicologists who need access to objective information on the audio recording. In order to be able to easily integrate new tools providing analysis data for the visualization, it offers a plugin interface called *VAMP* (see below).

The latest information and the source code of Sonic Visualiser is available from the project's web page.[12]

[9]CLAM. http://clam-project.org. Last retrieved on Jan. 28, 2012.
[10]jMIR. http://jmir.sourceforge.net. Last retrieved on Jan. 31, 2012.
[11]CoMIRVA: Collection of Music Information Retrieval and Visualization Applications. www.cp.jku.at/CoMIRVA. Last retrieved on Jan. 31, 2012.
[12]Sonic Visualiser. http://www.sonicvisualiser.org. Last retrieved on Jan. 31, 2012.

D.2 Software Libraries and Toolboxes

In contrast to the frameworks presented above, the following software solutions focus on specific aspects such as feature extraction. Many of them require either a programming environment (for instance, Matlab) or they are libraries to be linked against a custom-designed software.

D.2.1 Feature Extraction

The *MIRtoolbox* is a toolbox for Matlab and has been developed by a team at the University of Jyväskylä. It provides an extensive set of functions for the extraction of low-level features as well as for the analysis of tonal, structural, and temporal properties [491]. The latest information and the source code of the MIRtoolbox is available from the project's web page.[13]

The *libXtract* library has been presented by Bullock from the Birmingham Conservatoire [492]. It is programmed in C and deals with the extraction of low-level features. The libXtract library allows to arbitrarily cascade the feature extraction routines for the computation of subfeatures on different hierarchical levels. The latest information and the source code of libXtract is available from the project's web page.[14]

Content-based Audio and Music Extraction Library (CAMEL) is a comparably new library for feature extraction and aggregation aspiring to become a framework for various MIR-related tasks. It is implemented in C++ and was presented by a team of the University of Lethbridge [493]. The latest information and the source code of CAMEL is available from the project's web page.[15]

Yet Another Audio Feature Extractor (YAAFE) is a command line tool for the extraction of low-level features published by the Telecom ParisTech [494]. It is implemented in C++ but provides Python bindings as well. YAAFE aims at very efficient feature extraction by utilizing a *feature plan parser*, determining and removing redundant processing steps in the calculation of different features. The latest information and the source code of YAAFE is available from the project's web page.[16]

The *Timbre Toolbox* is a Matlab toolbox for the extraction of a large set of (low-level) features. It is the result of a joint effort of IRCAM and McGill University [495]. The source code of the Timbre Toolbox is available online.[17]

SCMIR is an extension for the *SuperCollider* audio programming language. It is developed by Collins [496] and allows access to ACA routines through a high-level programming language. The emphasis is on feature extraction in order to use the extracted information for "creative musical activity." SCMIR offers the extraction of standard features, the analysis of tempo and beat, as well as structural analyses. The latest information and the source code of SCMIR is available from the project's web page.[18]

[13] MIRtoolbox. https://www.jyu.fi/hum/laitokset/musiikki/en/research/coe/materials/mirtoolbox. Last retrieved on Jan. 31, 2012.
[14] libXtract. http://libxtract.sourceforge.net. Last retrieved on Jan. 31, 2012.
[15] CAMEL – A Framework for Audio Analysis. http://camel-framework.sourceforge.net. Last retrieved on Jan. 31, 2012.
[16] Yaafe – audio features extraction. http://yaafe.sourceforge.net. Last retrieved on Jan. 31, 2012.
[17] http://recherche.ircam.fr/pub/timbretoolbox. Last retrieved on Jan. 31, 2012.
[18] SCMIR. http://www.sussex.ac.uk/Users/nc81/code.html. Last retrieved on Jan. 31, 2012.

Maaate is an audio analysis toolkit written in C++. It was designed to perform feature extraction and analysis routines directly on encoded MPEG Layer 1–3 files [497]. The latest information and the source code of Maaate is available from the project's web page.[19]

A library designed for feature extraction in a real-time context is *aubio*. It is written in C++ and enables the user to extract several low-level features as well as both onsets and tempo in a causal way in order to support real-time applications. The latest information and the source code of aubio is available from the project's web page.[20]

OpenSMILE is a C++ library for feature extraction. It addresses both speech processing and music processing by combining features typically used in both domains in one feature extractor [498]. The latest information and the source code of OpenSMILE is available from the project's web page.[21]

D.2.2 Plugin Interfaces

One idea to face the problem of multiple and apparently redundant implementations of the same (low-level) features in various libraries and frameworks is to define a plugin *Application Programmer's Interface (API)*. The advantage of using plugins for feature extraction is the possibility of dynamically loading new plugins at runtime without compilation or static linking, complemented by the simple exchange of plugins (features) between researchers. A plugin API itself is thus a rather general interface definition of how to extract an unknown feature as opposed to the actual implementation of various features.

D.2.2.1 FEAPI

The *Feature Extraction Application Programmer's Interface (FEAPI)*, named here for historic reasons, was a plugin API for the extraction of low-level features. It was a joint effort with participants from the four institutions Ghent University, IRCAM, Technical University of Berlin and zplane.development [499].

The project is not actively maintained or developed anymore. One of the most likely reasons why FEAPI was not accepted by the research community was the lack of host applications. Although a Max port existed and had been used, other (graphical) user interfaces except for a simple command line interface did not exist.

The documentation and source code of FEAPI, complemented by example implementations of several spectral and loudness features, can still be found on the FEAPI project web page.[22]

D.2.2.2 VAMP

VAMP is the only reasonably widespread plugin interface for feature extraction from audio signals. It has been defined and developed by a team of the Centre for Digital Music of the Queen Mary University of London [500]. VAMP plugins can be hosted by the *Sonic Visualiser* (see above). The basic functionality is similar to the functionality of FEAPI.

Several plugins, as well as the latest information and the source code of VAMP is available from the project's web page.[23]

[19]Maaate!. http://maaate.sourceforge.net. Last retrieved on Jan. 31, 2012.
[20]aubio, a library for audio labelling. http://aubio.sourceforge.net. Last retrieved on Jan. 31, 2012.
[21]openSMILE: The Munich Versatile and Fast Open-Source Audio Feature Extractor. http://opensmile.sourceforge.net. Last retrieved on Jan. 31, 2012.
[22]FEAPI. http://feapi.sf.net. Last retrieved on Jan. 31, 2012.
[23]VAMP Plugins: The Vamp audio analysis plugin system. http://vamp-plugins.org. Last retrieved on Jan. 31, 2012.

D.2.3 Other Software

There is a multitude of other software applications and libraries that can be used in the context of ACA. Very important are the various machine learning and data mining solutions; probably the most-cited software is the *Waikato Environment for Knowledge Analysis (WEKA)* software [501].[24] WEKA is a comprehensive collection of machine learning algorithms and data pre-processing tools such as algorithms for regression, classification and clustering. *Torch* provides a Matlab-like environment for machine learning tasks [502].[25] It can be used with the programming language *Lua*.[26] Other libraries focus on specific areas of machine learning; examples are *libsvm*[27] [503] and *The Spider*, a machine learning environment for Matlab.[28] *HTK* is a toolkit for building and manipulating hidden Markov models [62].[29] Due to its focus on speech recognition HTK also offers the extraction of various features.

[24] Weka 3: Data Mining Software in Java. http://www.cs.waikato.ac.nz/ml/weka. Last retrieved on Jan. 31, 2012.
[25] Torch 5. http://torch5.sourceforge.net. Last retrieved on Jan. 31, 2012.
[26] The Programming Language Lua. http://www.lua.org. Last retrieved on Jan. 31, 2012.
[27] LIBSVM – A Library for Support Vector Machines. http://www.csie.ntu.edu.tw/~cjlin/libsvm. Last retrieved on Jan. 31, 2012.
[28] The Spider. http://people.kyb.tuebingen.mpg.de/spider. Last retrieved on Jan. 31, 2012.
[29] HTK Speech Recognition Toolkit. http://htk.eng.cam.ac.uk. Last retrieved on Jan. 31, 2012.