Ringermute: An audio data mining toolkit

Marcel A. Levy   Sergiu M. Dascalu   Frederick C. Harris, Jr.
Department of Computer Science and Engineering

University of Nevada
Reno, NV 89557
mlevy@unr.edu

Abstract
This paper presents Ringermute, an application designed to support audio feature recognition and machine learning, from the training and testing to the deployment phase. By choosing from a combination of feature extraction routines provided by plug-ins, a researcher can quickly produce files for input to standard data mining tools. The best combination of feature-extraction and classifier plugins can then be used to drive a near-real-time application for further testing or production use.
Keywords: Audio, Feature Recognition, Classifier

1 Introduction

Context-aware computing and communication is a relatively recent area of research. The goal is to produce systems and applications that modify their behavior in response to changes in the physical or social environment of the primary user or users [27]. These systems consist of sensors and one or more controllers, which can operate according to behavioral models of variable complexity [15]. Context-aware computing can be seen as a marriage of machine learning with human-computer interaction.

Sensor design is one of several challenges in the field. Tasks that humans find intuitive (face recognition, for example) are still more difficult for machines and can only be done in a narrower range of circumstances, or force larger tolerances for error. Even recognizing the ring of a phone over the sound of conversation or music—a trivial task for most humans—is difficult for a machine [16, 34]. But such real-world sensors are crucial to providing machines with the same context that supports successful human interactions.

Most sensors, and particularly those that operate on audio or video, are given noisy and unpredictable input. While Digital Signal Processing (DSP) is a mature field of research, most of its algorithms are straightforward mathematical transformations and are frequently designed for human decision support, and do not necessarily obviate the need for further analysis. The problem is essentially a machine-learning problem, in that machines are faced with second-order input from DSP algorithms that varies in its complexity. For example, an office environment may be generally quiet and simple to segment into periods of distinct activity types, perhaps even on the basis of volume alone. However, a restaurant, factory floor or even the passenger compartment of an automobile present far more challenging environments. In these cases, spectrum analysis will get us only so far.

Voice recognition has long been a fruitful area of research and development, but audio sensing in support of context-aware systems is relatively less sophisticated. While data mining tools are plentiful, audio feature recognition is still an evolving area of research, and very few audio tools are designed with the needs of machine learning in mind. Even ignoring the vast majority of tools that are designed for music or multimedia production, most audio analysis tools are focused on DSP, and not machine learning tasks.

This paper presents Ringermute, a tool designed to support audio feature recognition and machine learning, from the training and testing to the deployment phase. By choosing from a combination of feature extraction routines provided by plug-ins, a researcher can quickly produce files for input to standard data mining tools. The best combination of feature-extraction and classifier plug-ins can then be used to drive a near-real-time application for further testing or production use.

The problem of audio event and scene classification, and the current state of research, is summarized in Section 2. The intent and scope of the Ringermute system is discussed in more detail in Section 3, and the implementation is covered in Section 4. Two usage scenarios are outlined in Section 5. The results of the project and avenues for future research are found in Section 6.

2 Ringermute Design

While the world certainly does not lack for excellent audio tools [4, 6, 22] or machine-learning applications [33, 35], there does exist a need for an application that:

- Allows researchers to apply machine-learning techniques to live and stored audio data without having to first learn how to use audio and digital signal processing libraries.
- Allows researchers to develop new feature-extraction or classifier plug-ins that are based on existing code, without having to learn much about the preexisting code.

- Allows the rapid creation of an application based on the results of experimental data, without even requiring compilation.
- Is flexible enough to allow further modification and additions.

It may seem that a tool such as MATLAB [17] would suffice to perform research on audio context problems. Although such tools are useful and have a place in the research, their primary limitation is that they were not designed to deliver usable applications in an interactive context, particularly for live audio. And while Waikato Environment for Knowledge Analysis (WEKA)'s tool set [33] is well-adapted to constructing cross-platform applications, the audio framework and feature selection is up to the researcher to provide. Yet the problems of audio input and sound file formats are not so complex that they necessitate re-inventing the wheel. And the fields of auditory context and Computational Auditory Scene Recognition (CASR) provide a rich set of features stable enough to be considered standard as well. The intent is to allow the rapid creation of audio context widgets as conceived by Dey, Abowd and Salber, either at the sensor or Interpreter level [10].

2.1 The Framework

The functionality of Ringermute is contained in three applications: The service (`rimuservice`), the status monitor (`rimutaskbar`) and the feature-extractor (`rimuextract`). `rimutaskbar` serves as the user's GUI interface to the system. It allows the user to view, edit and save settings, start or stop `rimuservice`, and start `rimuextract` within a GUI context. It also displays the current `rimuservice` activity state. `rimuservice` is the engine that is responsible for acquiring the raw audio data (either from hardware input or a sound file), and activating data-processing modules (called Listeners) on the data in turn. The service is also tasked with writing out any data to a file or files. `rimuextract` is a commandline tool that extracts features from a series of audio files and combines them into a single Attribute-Relation File Format (ARFF) file. The relationship between the three applications is seen in Figure 1.

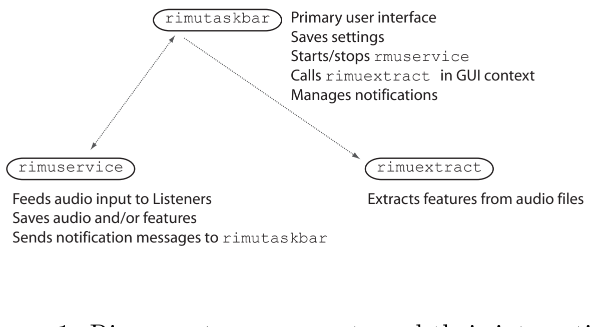

rimutaskbar
Primary user interface
Saves settings
Starts/stops rmuservice
Calls rimuextract in GUI context
Manages notifications
rimuservice
Feeds audio input to Listeners
Saves audio and/or features
Sends notification messages to rimutaskbar
rimuextract
Extracts features from audio files

Figure 1: Ringermute components and their interaction.

2.2 The Central Repository

The key data structure is the central repository, also called Ringermute Central. At its core it is a hash table of pointers to objects containing data (usually arrays of various types). Ringermute service places each frame of audio data in the repository for use by the Listeners, which are responsible for managing the data in their own namespaces. If a Listener requires historical data (the last N frames, for example), it is responsible for keeping this data as well. The Ringermute service only promises to provide the original audio data, and to trigger the Listeners whenever the data changes.

2.3 Listeners

All data-processing and feature extraction (except for the original raw audio data) is performed by Listeners. A set of standard Listeners is provided with Ringermute, and provides basic audio processing, including spectral analysis. But the design of Ringermute is such that third-party modules could easily be written to replace the basic functions. As can be gathered by the description, Listeners are a relatively straightforward implementation of the Observer pattern [11, 12]. A graphic representation of the Listener interface can be seen in Figure 2. Another key point to make is that Ringermute Listeners are designed to be loaded as dynamic plug-ins at runtime. As such, they are required to provide several informational routines that tell Ringermute what data they provide, and what data they depend on. This helps Ringermute determine in which order to alert the Listeners.

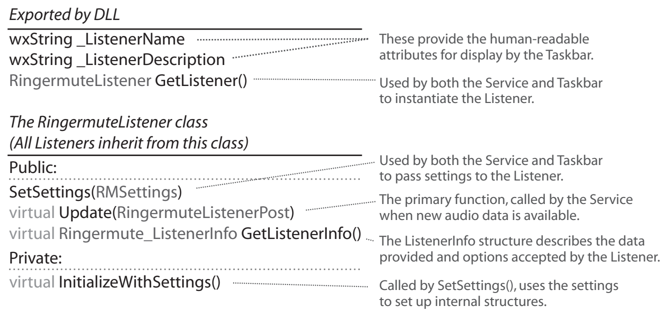

Exported by DLL
wxString _ListenerName
wxString _ListenerDescription
RingermuteListener GetListener()
These provide the human-readable attributes for display by the Taskbar.
Used by both the Service and Taskbar to instantiate the Listener.
The RingermuteListener class (All Listeners inherit from this class)
Public:
SetSettings(RMSettings)
virtual Update(RingermuteListenerPost)
virtual Ringermute_ListenerInfo GetListenerInfo()
Private:
virtual InitializeWithSettings()
Used by both the Service and Taskbar to pass settings to the Listener.
The primary function, called by the Service when new audio data is available.
The ListenerInfo structure describes the data provided and options accepted by the Listener.
Called by SetSettings(), uses the settings to set up internal structures.

Figure 2: The Ringermute Listener specification.

3 Ringermute Implementation

Ringermute's primary components have been implemented in C/C++, using several open-source, cross-platform libraries. One basic requirement for Ringermute was that it run on the three major desktop applications found in research environments: Microsoft Windows, Apple Mac OS X, and Unix/Linux. Several alternatives were considered, including Java and Python. Mindful of Knuth's caution against premature optimization [14], it was felt that the need for responsive and near-real-time audio analysis, especially when functioning as part of a larger context-aware framework, required better performance from the start. While Java has made great strides over the past decade in speed comparisons with C/C++, it is unfortunately still true that it performs at a disadvantage [31, 32]. Python, although it is a more rapid deployment tool than C/C++ and performs nearly as well as Java [25], was eliminated from consideration by its relative obscurity: more people have had experience with C/C++ or Java than with Python. Another advantage of C/C++ is that byte-code based runtime engines can be embedded within the Ringermute engine. This would allow Listeners to make use of the Java-based WEKA [33] machine learning library, for example.

3.1 Audio Input

Given that each major platform implements its own audio API (and in the case of Linux, several different APIs over the years), it was essential to find robust cross-platform libraries in order to read audio data from both a live source and recorded files. As it happens, only two open-source libraries are robust enough, under active development and available on the required platforms: Portaudio [24] and libsndfile [9].

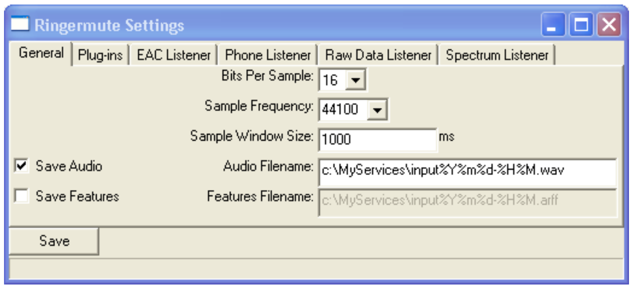
Portaudio Portaudio [24] is an open-source library designed to provide access to the audio hardware on a wide range of platforms, including Microsoft Windows, Apple Mac OS X, several Unix variants and BeOS. It is used by a number of applications, most notably the Audacity sound editor. Portaudio is usually run under a multi-threaded callback scheme, but can be run as single-threaded process with blocking I/O. This is how it has been implemented in the Ringermute service, since it only needs to worry about audio input during its execution.

Libsndfile Since Portaudio does not provide for audio data input from stored files, libsndfile has been used for sound file input and output. Like Portaudio, it is a free, open-source library that runs on a wide range of hardware and operating system combinations. It is capable of reading and writing standard audio formats such as WAVEform audio format (WAV), Audio Interchange File Format (AIFF) and the Sun Unix Audio (AU) file format. Additionally, it can read and write non-audio formats such as the MAT file format used with the open-source MATLAB-compatible numerical application Octave [18], and the file format used by the Hidden Markov Model toolkit HTK [13].

3.2 Graphic User Interface (GUI) Elements

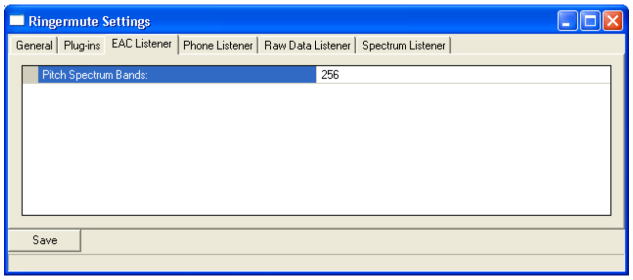
Although the primary application is designed to run as as a background service, a user interface has been developed to allow the user to start and stop the service, control which modules are run, and access settings for each module. The Ringermute status monitor runs as a "system tray" application, and displays an icon in the Microsoft Windows Taskbar, the Macintosh OS X Dock, or in the area specified by freedesktop.org's System Tray protocol [23], which is supported by both GNOME and KDE. This design allows the status monitor to indicate whether the Ringermute service is active, access the settings menu, and also display notifications in a relatively unobtrusive manner. Examples of similar applications include Google Desktop Search and Microsoft AntiSpyware.

Both the primary Ringermute settings (Figure 3) and the settings for each individual plug-in module (Figure 4) are accessed by using "tabbed" windows, which use the metaphor of physical folder or workbook tabs to separate the settings values. The individual plug-in modules do not access the GUI interface directly, but instead indicate to Ringermute what properties are user-controlled. This allows plug-in authors to contribute features without having to learn GUI toolkit routines.



Bits Per Sample: 16
Sample Frequency: 44100
Sample Window Size: 1000 ms
[✓] Save Audio — Audio Filename: c:\MyServices\input%Y%m%d-%H%M.wav
[ ] Save Features — Features Filename: c:\MyServices\input%Y%m%d-%H%M.arff

Figure 3: The Ringermute interface.



Pitch Spectrum Bands: 256

Figure 4: The Ringermute plug-in property interface for the EAC pitch-detection plug-in.

wxWidgets Much of Ringermute's functionality has been implemented using the wxWidgets library, a free, open-source, cross-platform toolkit in use since 1992 [28]. Aside from providing a rich API for GUI applications, wxWidgets also includes several features that made it useful for the Ringermute project. Both the service and status monitor make use of its cross-platform dynamic library features to load the plug-in modules. Settings information for the Ringermute application and the plug-in modules are handled by a cross-platform configuration framework: Under Microsoft Windows, the information is saved to the Registry, and within *.ini files under Mac OS X and Unix/Linux platforms.

3.3 Command-line and Background Applications

Both `rimuextract` and `rimuservice` are fundamentally similar in that they provide a context for Listeners to operate. Their primary difference is the context in which they in turn operate: `rimuextract` is designed to run from the command-line and takes existing audio files as input, while `rimuservice` is designed to run in the background and take live audio as its input. Both applications derive their primary functionality from the Ringermute Central structure mentioned in Section 2.2.

Ringermute Extractor At the moment, `rimuextract` is a fairly simple application. It accepts a list of audio files as individual arguments, and transforms these into a single file containing the features extracted from the audio file, where each line contains the features from a single audio window. The width of the audio window is set using the `rimutaskbar` application. `rimuextract` makes use of the libsndfile [9] library to read the existing audio files, and also makes use of several classes from the wxWidgets library. Although `rimuextract` is designed to run as a command-line application, `rimutaskbar` allows the user to invoke it within a GUI shell as a convenience feature.

Ringermute Service Since `rimuservice` runs as a background process, it runs in a different environment than `rimuextract`. Within Mac OS X and Unix systems `rimuservuice` is to run as a daemonized process. This is usually done by forking the process and killing off the parent, so that the newly "orphaned" process is "adopted" by the `init` process. On the Windows platform, the application runs as a Windows Service instead. At present, only the Windows version has been implemented. A service control API has been written that abstracts the primary interface (start, stop, restart), and concrete subclasses are used to implement platform-specific functionality—this is a fairly canonical example of the Bridge [12] design pattern.

4 Ringermute Usage

The marriage of context-awareness with audio scene and object recognition makes for a somewhat confusing combination of problems, many of which are still outstanding. The overriding issue is the lack of context standards on any platform, let alone platform-independent standards for gathering, reporting and acting on context. This, in part, is why Ringermute is designed the way it is. Although it is by no means a "context server," it does expose some basic notification functions to its context and recognition plugins. Similarly, while it does not include the breadth and depth of audio analysis functions that other systems, such as Music AI Research SYstem for Analysis and Synthesis (MARSYAS) [30] do, it compensates by integrating its recognition features in the GUI. The goal was not to create the perfect application for researchers or users, but to fit the general needs of both groups. In this way it allows promising recognizers to be quickly used in a real-world setting, while also automating some of the tedium of preparing audio for data mining and machine learning. Since Ringermute was designed for two main categories of usage, we will explore them both in this section.

4.1 Feature Extraction and Training

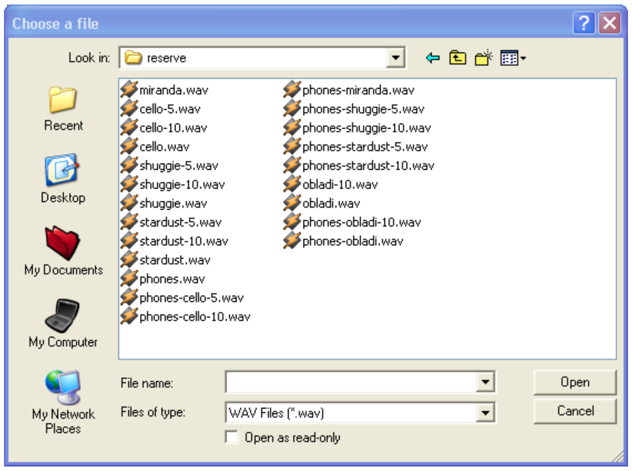
The first step is to gather the audio. Ringermute accepts audio input from the underlying sound API, and can simultaneously save the audio and extract features to an ARFF file. In most cases, the researcher will have recorded audio separately and perhaps prepared it with a tool such as Audacity [6]. In this case, we start with one or more audio files. These will usually be WAV files, but Ringermute is capable of reading all the formats and encodings supported by the open-source library libsndfile. `rimuextract` is a command-line tool that takes the names of audio files as its arguments. Using the central Ringermute settings file, it extracts audio features using the Listener plug-ins mentioned earlier and saves the features to a combined ARFF file. This file consists of a header section that describes the number and data types for each exemplar. This is followed by the data for each exemplar, one per line, in comma-delimited format. The @RELATION line names the dataset, the @ATTRIBUTE lines specify the number and types of the data fields, and the @DATA keyword indicates the end of the header. The exemplars follow, one per line, with comma-separated attributes.

In order to make this file useful for machine learning applications such as WEKA, we must include not only features, but a class, such as "phone ringing." By convention, this is the last field or "attribute" in the list of features. For each audio file, `rimuextract` will search for an accompanying text file that contains the start and stop times for the class in question. For example, if a given audio file is named "example.wav", `rimuextract` will search for a text file named "example.txt" or "example.wav.txt." This text file contains lines that have a floating point number followed by :

```
(start|begin|stop|end)[␣<string>]
```

This is the same format used by Audacity to save its track-labeling feature, so Audacity can be used to visually mark start and stop points for the desired object or scene. The Ringermute Taskbar settings interface (Figure 3) can be used to determine which features are extracted from the file, and can also be used to extract the features without using the command line (Figure 5).

Once the ARFF file has been generated, we have a set of exemplars to use for training or evaluation. For exam-



Choose a file

Look in: reserve

```
miranda.wav            phones-miranda.wav
cello-5.wav            phones-shuggie-5.wav
cello-10.wav           phones-shuggie-10.wav
cello.wav              phones-stardust-5.wav
shuggie-5.wav          phones-stardust-10.wav
shuggie-10.wav         obladi-10.wav
shuggie.wav            obladi.wav
stardust-5.wav         phones-obladi-10.wav
stardust-10.wav        phones-obladi.wav
stardust.wav
phones.wav
phones-cello-5.wav
phones-cello-10.wav
```

File name:

Files of type: WAV Files (*.wav)

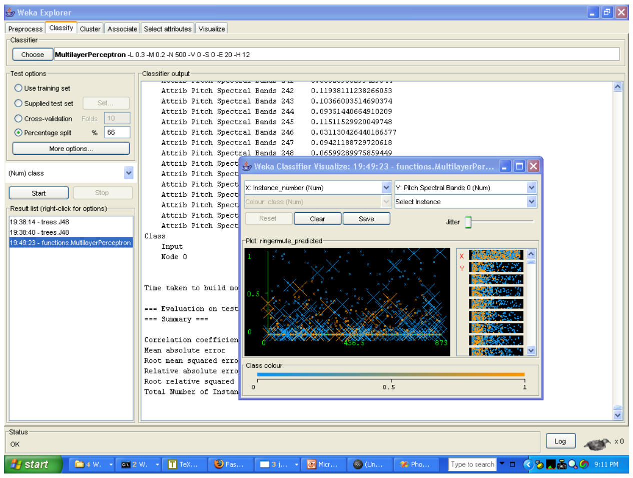
Figure 5: File-chooser window launched from Ringermute taskbar.

ple, WEKA provides a comprehensive interface for data-mining, analysis and experimentation. Figure 6 demonstrates some of WEKA's visualization capabilites, applied to a Ringermute-generated ARFF file. In the figure, we are seeing a visualization of the resulting error rate after training an artificial neural net on a data set. Given that the Ringermute project grew out of a phone-recognition problem, we have written a small neural-net trainer in C++ that accepts ARFF files with a variable number of numeric input attributes and a single numeric output attribute indicating whether a phone is ringing. The ARFF file was generated using the Ringermute system, and the resulting neural net is used by the Ringermute PhoneListener plug-in. This illustrates the power of the Ringermute toolset, in that the output from the PhoneListener can then be used as an extracted feature, either with live audio input or pre-recorded audio files. This allows for stepwise refinement as the actual output from the plugin is compared to the expected output, and any improvements to the neural net can be implemented by simply replacing a configuration file.

4.2 Interactive Usage

As previously mentioned, Ringermute is not a complete context server solution, and in interactive mode is intended to be used as a more sophisticated sensor. The Ringermute Listener plug-ins are responsible for taking data from a central repository, processing it in some way (by calculating a spectrum or applying a neural net, for example), and then leaving output for subsequent plug-ins. External actions and notifications are intended to be performed by Listeners themselves, including action-specific Listeners that only perform actions based on the work of previous Listeners. For example, a useful plugin would be one which paused a given media player if a given sound event was detected.

However, some notification routines are built into the system. The main interactive input loop is run as a Win-



Weka Explorer

Classifier: MultilayerPerceptron -L 0.3 -M 0.2 -N 500 -V 0 -S 0 -E 20 -H 12

Test options: Use training set / Supplied test set / Cross-validation Folds 10 / [✓ Percentage split % 66]

(Num) class

Result list (right-click for options):

```
19:38:14 - trees.J48
19:38:40 - trees.J48
19:49:23 - functions.MultilayerPerceptron
```

Classifier output:

```
    Attrib Pitch Spectral Bands 242    0.11938111238266053
    Attrib Pitch Spectral Bands 243    0.10366003514690374
    Attrib Pitch Spectral Bands 244    0.09351440664910209
    Attrib Pitch Spectral Bands 245    0.11511529920049748
    Attrib Pitch Spectral Bands 246    0.031130426440186577
    Attrib Pitch Spectral Bands 247    0.0942118872972061
    Attrib Pitch Spectral Bands 248    0.06599289975859449
    Attrib Pitch Spect
    Attrib Pitch Spect
    Attrib Pitch Spect
    Attrib Pitch Spect
    Attrib Pitch Spect
    Attrib Pitch Spect
    Attrib Pitch Spect
Class
    Input
    Node 0


Time taken to build mo

=== Evaluation on test
=== Summary ===

Correlation coefficien
Mean absolute error
Root mean squared erro
Relative absolute erro
Root relative squared
Total Number of Instan
```

Weka Classifier Visualize: 19:49:23 - functions.MultilayerPer...

X: Instance_number (Num)   Y: Pitch Spectral Bands 0 (Num)

Colour: class (Num)   Select Instance

Plot: ringermute_predicted

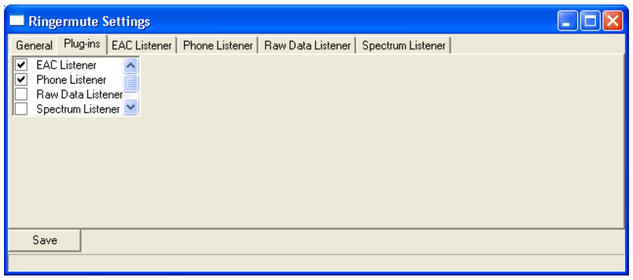
Status: OK

Figure 6: Error visualization in WEKA of results of training of an artificial neural net on ARFF file generated by `rimuextract`.

dows service or Unix daemon process, and can communicate with the taskbar interface via Dynamic Data Exchange (on Windows) or sockets (on Unix) if a GUI pop-up window is needed. Basic SMTP and HTTP notification is also built into the system as a convenience for plug-in authors.

In any event, the first step for basic interactive usage is to launch `rimutaskbar`, the Ringermute taskbar application, right-click on the icon and and select the "Settings" option from the resulting pop-up window.

The resulting view can be seen in Figure 3. For the sake of demonstration, let's say we are interested in live phone detection. Since the phone detection plug-in depends on the EAC pitch-detection plug-in, we want to make sure it is active as well. Figure 7 demonstrates the main plug-in activation list. This shows which plug-ins have been detected, and which ones have been activated.

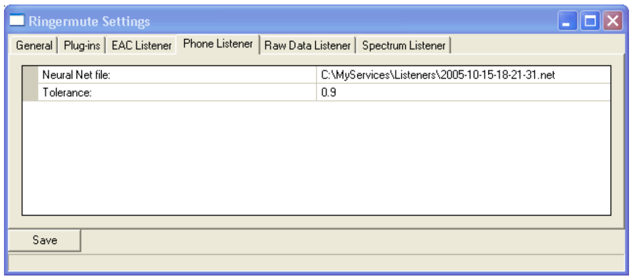


Ringermute Settings

- [x] EAC Listener
- [x] Phone Listener
- [ ] Raw Data Listener
- [ ] Spectrum Listener

Figure 7: Ringermute plug-in list.

We see that the EAC pitch-detection plug-in is active. The properties window for this plug-in was previously shown in Figure 4. We now turn to the properties window for the phone-detection plug-in, shown in Figure 8. We can see that it allows the user to determine both which neural net file to use, and the tolerance level to use when detecting phone rings. In this case, the neural net file is in the format used by the Fast Artificial Neural Network Library (fann). Tolerance refers to the output of the neural net, on a floating-point scale of 0.0 to 1.0, and a tolerance of 0.8 means that any output exceeding 0.8 means a phone ring is present. This allows the user to roughly tune the detection algorithm while the application is running.



Neural Net file:	C:\MyServices\Listeners\2005-10-15-18-21-31.net
Tolerance:	0.9

Figure 8: Properties window for the phone-detection plug-in.

Once the settings have been saved, the user starts the Ringermute listening service with the "Start listening" option on the taskbar, and the service begins taking the audio input and calling the plug-ins. If the phone-detection plug-in's output exceeds the tolerance level, it indicates to other plug-ins that a phone was detected. At present it also instructs the taskbar to pop up a notification window—not an ideal action, but suitable for demonstration purposes. This notification can be disabled in the properties window.

5 Conclusions and Future Work

The primary niche for Ringermute is sensor design. Before systems can make higher-level decisions about physical or social context, they must first sense more basic phenomena and objects. While calendar data stored inside the computer is trivial to read, the desk calendar next to the computer is not, and yet this may contain useful context information as well.

We have seen where Ringermute fits into the spectrum of context-aware research and applications – it is a toolkit that addresses some of the issues involved in audio object and scene recognition, particularly in concert with machine learning and data mining applications. It has been designed to integrate with users in their natural environment: The graphical user interface. The combination of a plug-in interface with feature extraction allows for the refinement of recognition plug-ins by testing them on live and pre-recorded audio, but also allows their output data to be used as input by subsequent plug-ins, either to perform external actions or additional analysis.

5.1 Similar Projects and Existing Tools

MARSYAS MARSYAS is an existing framework designed to support audio analysis research [30]. As such, it is not a single application, but includes several command-line applications useful for audio analysis. It shares some goals and features with Ringermute. The predominant metaphor is one of a pipeline, similar to the UNIX concept of pipes, and MARSYAS. Since it is a framework, it is designed to be used by a number of different applications, but no single included application contains all the features in the Ringermute design. Since it has a larger collection of audio feature extraction routines, MARSYAS is an excellent tool to use in the creation of Ringermute plug-ins. The primary difference, then, between Ringermute and MARSYAS is that Ringermute is designed to be a user-friendly, graphical "control panel" for auditory context-awareness research. MARSYAS is a general-purpose library and framework for auditory analysis in general.

Sound Ruler Sound Ruler is an application that has been developed to meet the needs of bioacoustics researchers [4, 22]. It operates on recorded sound files, and allows the user to recognize and label audio sequences (such as animal calls), either manually or automatically. It offers a large feature set, including the display and graphing of audio data in various forms, including both the time and frequency domains.

One of SoundRuler's most interesting features is the ability to automatically recognize audio sequences. It does so with a correlation technique: Given an example of the sequence, it seeks to find sequences that match the exemplar's envelope. While this is certainly useful in the context of animal calls, it does not allow for recognition over a general *class* of sequences: the general class of bullfrog calls, for example, versus the specific class of European tree frog *Hyla arborea.* In addition, the recognition algorithm does not appear to be robust enough to recognize all calls individually. Finally, the algorithm itself and the included audio features are not designed to be extensible by others: The application itself is a monolithic one. However, it demonstrates the utility of sound analysis applications in general.

CLAM On the other end of the spectrum from Sound Ruler, C++ Library for Audio and Music (CLAM) [1, 2] is a framework for audio signal processing. Along with classes and routines for input, processing and analysis, CLAM provides such ancillary functions as data serialization and visualization. Data types range from low-level signal components to higher-level units of analysis, such as phrasing and segmentation [3]. While it provides a wide range of components and features for digital signal applications, CLAM is not an application as such. Neither is it simply a library, in that it provides a conceptual model along with its library functions [1]. A key difference between CLAM and Ringermute, aside from the features CLAM offers, is that CLAM requires more effort to install and deploy in an application. It has been designed to meet the widest range of audio applications, not just context-aware computing. As such it is certainly possible to make use of CLAM's features within the context of a

Ringermute Listener.

Audacity Strictly speaking, Audacity [6] is a sound editing application, and so on first glance may not compare very well to Ringermute at all. However, aside from its obvious utility in preparing sound files for analysis, it includes several analysis visualization features in the program itself, and supports extension through a plug-in interface. Aside from waveform visualization, Audacity includes a spectrum view and a pitch-detection visualization based on work by Tolonen and Karjalainen [29]. Given its feature set, Audacity is a natural candidate to provide supporting features to the Ringermute project—one problem that Ringermute does not solve directly is the issue of labeling sound segments for training purposes. Audacity's interface allows manual labeling, and Aubio, a separate audio labeling project [5, 20, 21], can create label tracks automatically based on audio signal events, such as the beginning of musical notes.

5.2 Future Improvements

Even though Ringermute is a working and usable tool, there are many opportunities for future refinement. For example, although Ringermute has been built with cross-platform libraries, it has only been developed and tested on the Windows platform. The immediate goal is to produce working executables for the Linux and Mac OS X platforms as well. Documentation, particular API documentation for those seeking to build Ringermute Listeners, is lacking as well.

Plug-in Dependencies More sophisticated use of plug-ins would require the system to be aware of plug-in dependencies – at the moment plug-ins are loaded and activated in the order they are discovered by the operating system (alphabetical, in the case of Windows). Plug-ins already provide "provision" information to the system by enumerating the configuration. Adding dependency information and handling would be a fairly simple task.

RRDtool Integration RRDtool is an open-source system for logging and graphing time-series data [19]. Besides a library and command-line tools, it also offers bindings for many popular languages, including Python, Perl and Ruby. Using RRDtool as the data store in Ringermute would allow plug-ins to make use of a wider range of existing applications and libraries designed for time-series data.

Multithreading In its current form, Ringermute performs blocking reads on the audio input, and must wait for all the plug-in modules to execute before reading another input window. A multi-threaded version would simultaneously read audio input to a buffer and execute a plug-in loop. Modules that were not dependent on the execution of earlier modules could run simultaneously. This sub-project would require modification of the data-writing portion as well.

Scripting Interface Although the plug-in model allows cooperation by researchers without requiring knowledge of GUI or audio libraries, it still requires a separate compilation step for each platform. An ideal situation would be to allow researchers to write plug-ins that can be run on multiple platforms. Audacity, for example, allows the use of plug-ins written in the audio synthesis language Nyquist [8], which contains some features useful in analysis applications. Many languages are designed to be easily incorporated into C/C++ applications: For example, existing gaming engines use languages such as Python and Lua.

Mobile Devices Aside from the office or workgroup scenarios, the mobile environment offers the largest set of interesting applications for context-aware computing [26, 7]. It also offers a new set of challenges to researchers, although these barriers are rapidly disappearing with the advent of cheaper and more powerful mobile devices. Even so, another project would be to adapt Ringermute to the technical limitations and requirements of mobile devices, such as a the PocketPC or Palm OS.

References

[1] Xavier Amatriain. *An Object-Oriented Metamodel for Digital Signal Processing.* PhD thesis, Pompeau Fabra University, 2004.

[2] Xavier Amatriain, Pau Arum, Maarten de Boer, David Garca, Miquel Ramrez, Xavier Rubio, and Enrique Robledo. Clam: C++ library for audio and music. http://www.iua.upf.es/mtg/clam/.

[3] Xavier Amatriain, Pau Arum, and Miguel Ramrez. Clam, yet another library for audio and music processing? In *OOPSLA '02: Companion of the 17th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 46–47, New York, NY, USA, 2002. ACM Press.

[4] M. A. Bee. Sound ruler acoustical analysis: a free, open code, multi-platform sound analysis and graphing package. *Bioacoustics*, 14:171–178, 2004.

[5] Paul Brossier. Aubio: A library for audio labelling. http://aubio.piem.org/.

[6] Matt Brubeck, Joshua Haberman, and Dominici Mazzoni. Audacity: Free audio editor and recorder. http://audacity.sourceforge.net.

[7] Brian Clarkson, Nitin Sawhney, and Alex Pentland. Auditory context awareness via wearable computing. In *Proceedings of the 1998 Workshop on Perceptual User Interfaces (PUI'98)*, San Francisco, CA, November 1998.

[8] Roger B. Dannenberg. The implementation of nyquist, a sound synthesis language. *Computer Music Journal*, 21:71–82, 1997.

[9] Erik de Castro Lopo. libsndfile. http://www.mega-nerd.com/libsndfile/.

[10] Anind K. Dey, Gregory D. Abowd, and Daniel Salber. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Human-Computer Interaction*, 16:97–166, 2001.

[11] Eric Freeman, Elisabeth Freeman, Kathy Sierra, and Bert Bates. *Head First Design Patterns*. O'Reilly Media, Inc., 2004.

[12] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns*. Addison-Wesley Professional, 1995.

[13] Htk speech recognition toolkit. http://htk.eng.cam.ac.uk/.

[14] Donald E. Knuth. Computer programming as an art. *Communications of the ACM*, 17(12):667–673, 1974.

[15] Ajay Kulkarni. A reactive behavioral system for the intelligent room. M. eng. thesis, MIT, Cambridge, MA, 2002.

[16] Marcel Levy. Ringermute: Automated phone detection and response. Unpublished poster produced as part of coursework for CS790q at the University of Nevada, Reno, 2004.

[17] Matlab. http://www.mathworks.com/.

[18] Gnu octave. http://www.octave.org/.

[19] Tobias Oetiker. Rrdtool. http://oss.oetiker.ch/rrdtool/. Last Accessed 10/23/2009.

[20] J. P. Bello P. Brossier and M. D. Plumbley. Fast labelling of notes in music signals. In *Proceedings of the 5th International Conference on Music Information Retrieval (ISMIR 2004)*, Barcelona, Spain, October 2004.

[21] J. P. Bello P. Brossier and M. D. Plumbley. Real-time temporal segmentation of note objects in music signals. In *Proceedings of the International Computer Music Conference (ICMC 2004)*, Miami, Florida, USA, November 2004.

[22] Marcos Gridi Papp. Sound ruler acoustic analysis. http://soundruler.sourceforge.net/, August 2004.

[23] Havoc Pennington and Mark McLoughlin. System tray protocol specification. http://freedesktop.org/Standards/systemtray-spec, November 2004.

[24] Portaudio - portable cross-platform audio api. http://www.portaudio.com/.

[25] Lutz Prechelt. Technical report 2000-5: An empirical comparison of c, c++, java, perl, python, rexx, and tcl for a search/string-processing program. Technical report, University of Karlsruhe, 2000.

[26] Nitin Sawhney. Situational awareness from environmental sounds. Technical report, Speech Interface Group, MIT Media Lab, June 1997.

[27] B.N. Schilit, D.M. Hilbert, and J. Trevor. Context-aware communication. *IEEE Wireless Communications*, Volume 9, Issue 5:46–54, October 2002.

[28] Julian Smart, Robert Roebling, Vadim Zeitlin, Vaclav Slavik, Stefan Csomor, and Robin Dunn. The wxwidgets library. http://www.wxwidgets.org/.

[29] T. Tolonen and M. Karjalainen. A computationally efficient multi-pitch analysis model. *IEEE Transactions on Speech and Audio Processing*, Vol. 8(No. 6):708–716, November 2000.

[30] George Tzanetakis and Perry Cook. Marsyas: a framework for audio analysis. *Organised Sound*, 4(3):169–175, 1999.

[31] Rodrigo Vivanco and Nicolino Pizzi. Computational performance of java and c++ in processing fmri datasets. In *OOPSLA '02: Companion of the 17th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*, pages 100–101, New York, NY, USA, 2002. ACM Press.

[32] Rodrigo A. Vivanco and Nicolino J. Pizzi. Scientific computing with java and c++: a case study using functional magnetic resonance neuroimages. *Software: Practice and Experience*, 35(3):237–254, 2004.

[33] Weka machine learning project. http://www.cs.waikato.ac.nz/ ml/index.html.

[34] Brian Westphal and Jim King. A genetic algorithms based automatic phone-ring detection system. Unpublished paper produced as part of coursework for CS790k at the University of Nevada, Reno, 2003.

[35] Ian H. Witten and Eibe Frank. *Data Mining*. Morgan Kaufmann, 2000.