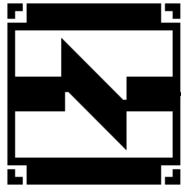

Center for Research in Electronic Art Technology
University of California, Santa Barbara

FASTLab Inc.
4764 Calle Camarada
Santa Barbara, California, 93110, USA

Expert Mastering Assistant (EMA)
Version 2.0

Technical Documentation

Stephen Travis Pope
Alex Kouznetsov
September, 2004

TABLE OF CONTENTS

Overview 2
The EMA Executable 4
EMA Step-by-step Operation 6
Design Overview 6
Appendix A. User Customization: THE CLIPS Expert System 9

Copyright (C) 2004. University of California, Santa Barbara. All Rights Reserved.

Overview

This document describes the design, and implementation of the “Expert Mastering Assistant” (EMA) tool version 2.0 developed by UCSB Center for Research in Electronic Art Technology (CREATE), and FastLAB Inc. for Panasonic Spin-Up Fund.

System Architecture

The “expert mastering assistant” (EMA) is a prototype artificial-intelligence-based software tool that “listens” to a set of musical selections and gives expert advice to a mastering engineer, suggesting parameters for signal processing modules that perform the signal processing: equalization, compression, reverberation, etc.

EMA suite consists of two major components: the interactive EMA application that analyses and processes individual songs with real-time interactivity, and a number of development applications that are required as a part of the expert system training process (Figure 1).

Training Tools

Mastering recorded music is a complex process involving human expertise and audiophile-grade processing equipment. We wish to assist mastering engineers with a software system that can analyze music (both at the signal level and at the musical level), suggest what processing should be applied during the mastering process, and provide a real-time interactive interface allowing user to experiment and apply specific mastering settings.

EMA relies on a number of expert systems to both classify the training data and to suggest optimal mastering parameters. On one hand, a large quantity of source material must be analyzed to improve the quality of analysis data, yet a much smaller set of parameters (or representatives) is needed in order to incorporate heuristic knowledge base that corresponds to mastering engineer preferences. The training process involves analyzing a large number of songs in order to produce a reduced‘ set of style representatives that form the basis for parameter computation for the rule-based mapping system.

The batch analysis driver program runs the FMAK analyzer on an unlabeled training set of approximately 3000 songs. The input data (assumed to be a CD or digital transfer of the musical content) is stored in the system (on a hard disk) for use by the analysis engine. The first-stage signal analysis derives a set of approximately 40 features; these include both time-domain and spectral-domain features. The first-level features are then used by the subsequent analysis phases, which derive higher level musical and recording/production features. These analysis processes can also request additional signal-level features to be derived. The derived signal-level, musical and production features are stored into the Analysis Results SQL database (PostgreSQL). EMA uses the FMAK (FASTLab Music Analysis Kernel) library. For more details on FMAK Library please refer to “FASTLab Music Analysis Kernel Library: Technical Documentation”, for more details on the analysis driver please refer to “FASTLab Music Analysis Kernel Utilities: Technical Documentation”.

Even though the analysis engine performs a considerable amount of data reduction internally, the total amount of information in the database is quite large (over 100 MB at present). The next stage of data reduction is performed by the clusterer that reduces the entire data set to a small number of representative members (songs), typically about 30 or

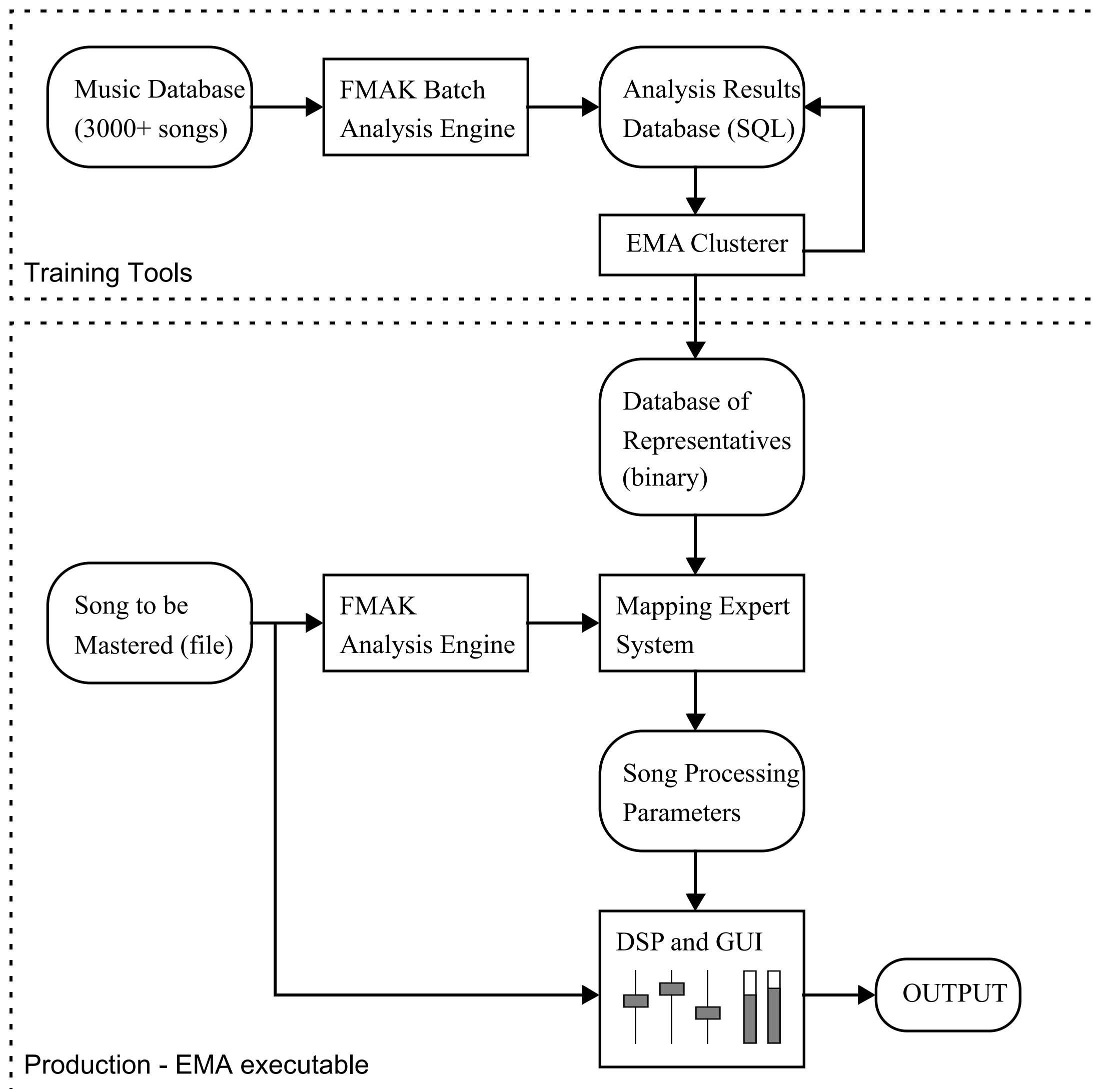

Figure 1: EMA Architecture Overview

less. These representative are then stored in a compact binary file to be used by the mapping expert system to determine optimal processing parameters for a given song. The clusterer operates as a stand-alone executable and accesses the data in the Analysis Results SQL database. The clusterer produces a cluster id label for each song that is stored back in the database table. For more details on the clusterer please refer to "FASTLab Music Analysis Kernel Utilities: Technical Documentation".

The clusterer is a full-featured multi-stage clustering algorithm implementation optimized for the specific requirements of EMA application:

- Optimized for large databases (uses a pre-clustering stage)
- Good performance with irregularly-shaped clusters and wide cluster size variations
- Low sensitivity to outliers

The EMA clusterer is almost entirely based on the CURE clustering algorithm as described in S. Guha, R. Rastogi, and K. Shim. "CURE: An efficient clustering algorithm

for large databases." In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 73--84, New York, 1998.

A separate program called the DB extractor is used to create the run-time system's pruned and flattened binary database file.

The EMA Executable

The main EMA application is a standalone mastering tool for stereo and surround (6 channel) content. In EMA, the musical content is analyzed by a process very similar to that performed by the FMAK analyzer, and the Mapping Expert System makes decisions about what processing to recommend for the mastering process. This data is presented to the user, who can change the parameters of the signal processing software (with interactive feedback).

The expert system evaluates the features of the musical selection, comparing it to data for other music of similar genre (stored the Database of Representatives). The system uses rule-based heuristic reasoning to determine what processing is appropriate for a given musical selection. This is then used to compute suggested parameters for the signal processing software.

The proposed processing choices, as well as the system's reasons for choosing them, are presented to the mastering engineer via the graphical user interface (GUI). This will allow the user to see why the system is choosing certain processing parameters, and to tune or override them, all with real-time interactive audio feedback from the processing system. The user's interaction with the processing parameters will be recorded and can be played back in the style of automated mixing consoles or software-based digital audio workstations.

The EMA system's GUI is composed out of several subpanes (Fig. 2, 3). The system informs the user about the analysis and reasoning processes and basic genre and property decisions in the logging view. The metering section includes precise level meters as well as stereo correlation and other kinds of displays. The user can tune the system interactively with the controls for the digital signal processing engine. The user interface is configurable, meaning that the user can expand or collapse any of the panes to concentrate on specific tasks.

The actual mastering processing is done by the signal processing block, which consists of a real-time digital filter/parametric equalizer, a dynamic-range compressor/expander, a stereo reverberator, and multi-channel separation enhancement. For more advanced applications, other signal processing functions such as MP3 or AAC encoders, or a stereo-to-5.1-channel up-mix processor may be added.

The output of the mastering processing stage is played in real-time for auditioning (and interactive control via the GUI), or stored on disk.

EMA runs under MacOS X 1.0.3.5 or newer; it requires a fast processor (1.25 GHz or faster recommended) and large RAM memory (1 GB or more recommended).

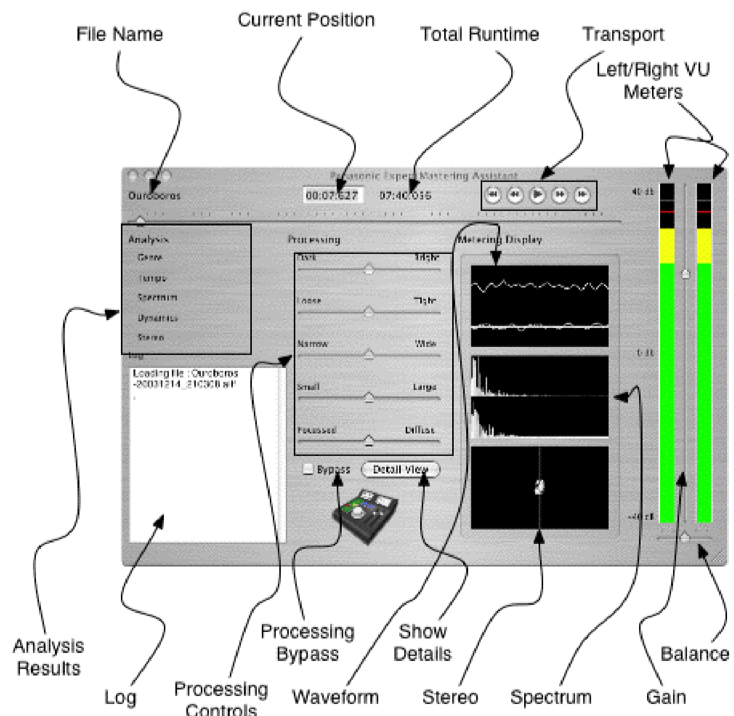

Figure 2: EMA GUI: stereo version

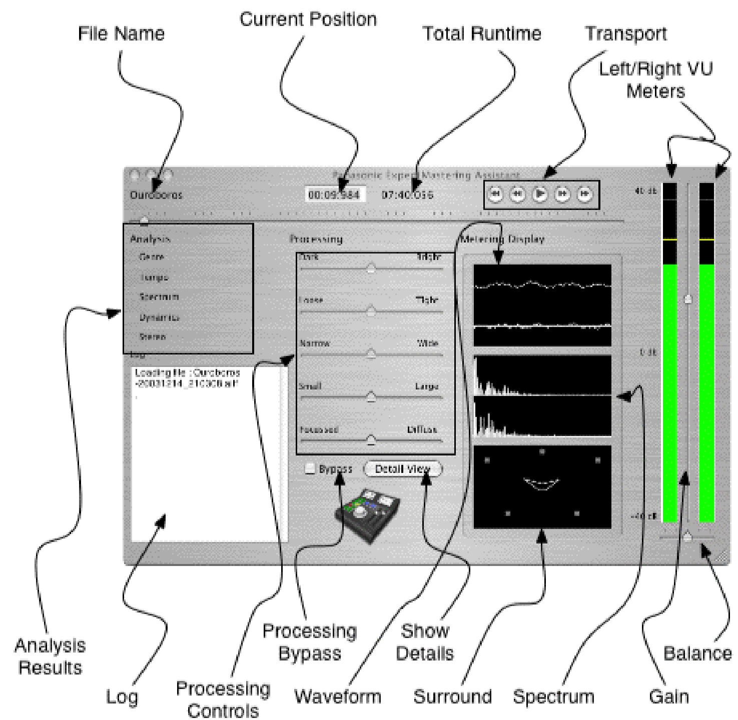

Figure 3: EMA GUI: surround version

EMA Step-by-step Operation

EMA is a GUI-driven application, most processing takes place in response to user action; once running, all EMA functionality is accessed through the application's GUI. There are no background "daemon" processes.

EMA is delivered as a stand-alone "double-clickable" Macintosh OS X application. Installation of AudioUnit components (beyond the EMA application) may be required to run EMA on a new computer.

To load a file, select "File->Open" menu; AIFF and RIFF/wav files are supported. Once a file is loaded, the analysis will start automatically. Once analysis is complete, the EMA application will suggest optimal processing parameters and set the on-screen GUI. Signal processing is performed in real time with live user interaction with the GUI sliders and low-latency feedback via the sound output. The GUI allows the user to bypass the processing as a whole, or any single component of the processing. The output of the signal processing is played over any CoreAudio output device.

The EMA application is available in currently shipped as a stereo version, although a surround-sound version has been developed. The only difference between the two is the ability to process multichannel audio and the surround field view meter.

Design Overview

Object Design

Given the scope of the prototype development effort, and our experience with similar systems (e.g., CSL, Siren, and CRAM), we determined that a formal object modeling process was not necessary for the EMA system prototype.

The EMA implementation in C++ uses several core classes and a variety of helper classes. An instance of the EMA class is the application manager; it is composed out of a number of other objects that "do the work" of EMA as listed below.

```
Analyzer _analyzer;                    // the signal analyzer
FeatureList _features;                 // the current song's feature vector
Expert _expert;                        // the parameter mapping expert
Processor _processor;                  // the real-time signal processor
ConsoleAutomation _recorder;           // the GUI interaction recorder
Logger _logger;                        // the GUI message logger
Transport _transport;                  // the GUI transport controller
EMAState _state;                       // the current operational state
Buffer _song;                          // the current song as a CSL buffer object
Genre _genre;                          // analysis output enumerations
Tempo _tempo;
Reverb _reverb;
Stereo _stereo;
```

Most of these components are straightforward and well-defined operational blocks.

File Loader and Data Analysis

The file loader is triggered by the user selecting a file with the "File Open" menu item.

The file loader reads sound files in a number of formats (AIFF and RIFF/wav formats supported at present) and loads them into memory (in their entirety, no streaming is assumed). The buffer manager allocates the sample storage for integer and floating-point versions of the musical material. The integer data is used by the analysis engine, and the floating-point version is used for the signal processing.

The analysis engine (FMAK Library) is normally triggered automatically by the operation of loading a sound file. After its creation, the analysis engine object is passed a buffer of 16-bit interleaved stereo samples, and performs all the standard analysis steps. Once the processing is completed, the analysis engine can be queried to get the song feature vector, approximately 40 parameters that describe the song content according to the feature grouping introduced above. Please refer to “FASTLab Music Analysis Kernel Library: Technical Documentation” for further details.

Mapping Expert System

The property-to-parameter mapping system takes the feature vector and compares it to stored data for a sampling of popular songs (the so-called genre database). One result of this is an identifier of the data cluster family, or “genre” of the song. Given this cluster identifier, the mapping system can get the feature vector for the “typical” song in the cluster. The differences between the feature vector of the current song and that of the “template” for the genre are used by the rule-based expert system to suggest a set of processing parameters. In effect, each song is “tuned” to be more consistent with its genre.

The output of the mapping system is a collection of high-level and low-level signal processing parameters. The elements in this data collection correspond to the controls in the high-level and detail views of the GUI. There are no “hidden” parameters.

The mapping may optionally also tune the formulae that map the high-level song properties onto lower-level processing parameters.

When done, the mapping system supplies the suggested processing parameters to the GUI and the signal processing engine.

GUI and Processing

EMA GUI and DSP provides an interface for fine-tuning audio mastering parameters. The signal processing is performed in real time with live user interaction with the GUI sliders and low-latency feedback via the sound output.

Metering

The GUI provides a number of signal displays and output log views. The high-level analysis attributed are displayed in a series of pull-down menus showing style, tempo, spectrum, reverb, and stereo-width. The logging display is a scrolling text view in which the system can inform the users of system activity, mainly during the analysis phase. The following audio signal displays are present:

- VU meters: fast-response RMS/peak-reading color meters
- Stereo oscilloscope signal display
- Spectrum display (derived from FFT of the processed data)
- Stereo/surround field view with inter-channel difference signal.

Signal Processing

The signal processing consists of the processes of:

- Volume control,
- Equalization,
- Dynamic-range processing (compression/expansion),
- Reverberation, and
- Stereo-field processing.

The GUI allows the user to bypass the processing as a whole, or any single component of the processing. The output of the signal processing is played over any CoreAudio output device while playing.

Control and Interaction

The GUI allows the users to start/stop sound playback, to "scrub" through the song at random, and to audition any part of the song.

The high-level property sliders allow the user to change several low-level processing parameters at once. The current definitions of the high-level parameters are as follows.

High-level parameters (range 0.0 - 1.0)	
dull - bright	spectral slope, EQ, reverb
	bright = hi-pass, dull = lo-pass
	bright > 0.75 = add slight bright reverb
loose - tight	expansion, reverb
	tight = dynamic expansion
	loose = add slight short reverb
narrow - wide	stereo-width, reverb
	wide > 0.75 = add slight long reverb
small - large	expansion, reverb, EQ
	small = slight compression
	larger = longer reverb decay time, dynamic expansion, lo-pass EQ
diffuse - focussed	stereo-width, reverb, EQ
	focussed = narrow stereo-width;
	diffuse < 0.2 = add slight short reverb, lo-pass EQ

The GUI also provides a low-level detailed view that allows the user to fine-tune the signal processing.

Appendix A. User Customization: THE CLIPS Expert System

The kernel of the EMA parameter mapping system is a rule-based expert system implemented using the CLIPS (C-Language Integrated Production System) expert system shell (see http://www.ghg.net/clips/CLIPS.html).

What is CLIPS?

CLIPS is a productive development and delivery expert system tool which provides a complete environment for the construction of rule and/or object based expert systems. Created in 1985, CLIPS is now widely used throughout the government, industry, and academia. Its key features are:

- Knowledge Representation: CLIPS provides a cohesive tool for handling a wide variety of knowledge with support for three different programming paradigms: rule-based, object-oriented and procedural. Rule-based programming allows knowledge to be represented as heuristics, or “rules of thumb,” which specify a set of actions to be performed for a given situation. Object-oriented programming allows complex systems to be modeled as modular components (which can be easily reused to model other systems or to create new components). The procedural programming capabilities provided by CLIPS are similar to capabilities found in languages such as C, Java, Ada, and LISP.
- Portability: CLIPS is written in C for portability and speed and has been installed on many different operating systems without code changes. CLIPS comes with all source code which can be modified or tailored to meet a user's specific needs.
- Integration/Extensibility: CLIPS can be embedded within procedural code, called as a subroutine, and integrated with languages such as C, Java, FORTRAN and ADA.
- Interactive Development: The standard version of CLIPS provides an interactive, text oriented development environment, including debugging aids, on-line help, and an integrated editor.
- Fully Documented: CLIPS comes with extensive documentation including a Reference Manual and a User's Guide.
- Low Cost: CLIPS is maintained as public domain software.

How does EMA use CLIPS?

When the EMA system loads a new song, it uses the FMAK analysis library to analyze it. EMA then searches the genre database to classify the song according to its proximity to a number of selected musical genre representatives. Given the song’s feature data, and that of the selected genre representative, EMA passes this information to the CLIPS-based expert system, which uses a set of rules to determine the high-level processing parameters.

The code examples below given the basic format of the CLIPS template (object) definitions (which are stored in the file defs.clp, a part of the EMA application bundle), and an example annotated rule (the rules are in the file rules.clp). (Surprise! The CLIPS description language looks just like LISP!)

```
;; The main CLIPS song source/target template

;; This obviously corresponds to an EMA FeatureCollectionStruct,
;; which itself corresponds to an FMAK FeatureCollection object.

(deftemplate song

                        ;; song-global features
 (slot duration (type FLOAT) (default 0.0))
 (slot num-segments (type FLOAT) (default 0.0))
 (slot segment-weight (type FLOAT) (default 0.0))
 (slot quiet-sections (type FLOAT) (default 0.0))
 (slot loud-sections (type FLOAT) (default 0.0))
 (slot fade-in (type FLOAT) (default 0.0))
 (slot fade-out (type FLOAT) (default 0.0))

                        ;; peak window statistics
 (slot rms-amp (type FLOAT) (default 0.0))
 (slot peak-amp (type FLOAT) (default 0.0))
 (slot lo-rms (type FLOAT) (default 0.0))
 (slot hi-rms (type FLOAT) (default 0.0))
 (slot dyn-range (type FLOAT) (default 0.0))
 (slot zero-crossings (type FLOAT) (default 0.0))
 (slot stereo-width (type FLOAT) (default 0.0))
 (slot spect-centroid (type FLOAT) (default 0.0))
 (slot spect-slope (type FLOAT) (default 0.0))
 (slot spect-variety (type FLOAT) (default 0.0))
 (slot spect-tracks (type FLOAT) (default 0.0))
 (slot lpc-tracks (type FLOAT) (default 0.0))
 (slot lpc-noise (type FLOAT) (default 0.0))
)

;; The target genre

(deftemplate genre
   (slot name (type SYMBOL) (default unknown))
)

;; The processing parameters template
;; This represents the high-level processing features controlled by EMA.

(deftemplate parameters
 (slot width (type FLOAT) (default 0.0))
 (slot smallness (type FLOAT) (default 0.0))
 (slot tightness (type FLOAT) (default 0.0))
 (slot focus (type FLOAT) (default 0.0))
 (slot brightness (type FLOAT) (default 0.0))
)
```

When EMA runs, it copies the current song's data, as well as that of the selected genre-specific target song, into the CLIPS expert system's memory in the form of song templates (called song and target in the expert system rules). The parameters data structure represents the high-level processing parameters; these are modified by the expert system rules.

CLIPS rules consist of a condition clause (the "if" part) and an action clause (the "then" part); these are separated by the token "=>". In the condition clause, one can check for any number of conditions such as the target song's genre, and the relationships between the derived features of the target and current song. The action clause generally alters some aspect of the parameters structure, as shown in the example rule below.

```
;; Example dynamics rule -- make metal tighter

(defrule match-target-dynamics          ;; IF clause
    (genre (name Metal))                ;; the song is heavy metal
    (target (dyn-range ?tdr))           ;; set variable ?tdr to the target's dynamic range
    (song (dyn-range ?sdr))             ;; set variable ?sdr to the song's dynamic range
    (test (> ?tdr ?sdr))                ;; if tdr > srd
    (test ( > ?sdr 0.0))                ;; and sdr > 0
    (not (match-dynamics))              ;; and this rule has never been used
    (parameters (tightness ?tig))       ;; set variable ?tig to the tighthess parameter
    ?par <- (parameters)                ;; set variable ?par to parameter structure

 =>                                     ;; THEN clause

    (assert (match-dynamics))           ;; assert a symbol to signify that this rule has fired

    (modify ?par (tightness             ;; change the global high-level parameters
        (+ ?tig                         ;; and add a bit to the tightness parameter
            ( -                         ;; get the 5th root of the ratio of the dynamic ranges
                    (** ( / ?tdr ?sdr) 0.2)
                    1.0)))))
```