MARF for PureData for MARF

Serguei A. Mokhov
SGW, EV7.139-2, 1455 De Maisonneuve Blvd. W.
Department of Computer Science and Software Engineering
Concordia University
Montreal, Quebec, Canada
H3G 1M8
mokhov@encs.concordia.ca

ABSTRACT

In this paper, we present the use and integration of the Modular Audio Recognition Framework (MARF) and PureData (PD) open-source projects to achieve multiple goals for audio (but not limited to audio-only) processing. The goal of this work is to provide MARF's capabilities as a collection of externals to the PD project via JNI/CNI interfaces as well as for PD to provide an easy-to-use integrated graphical environment for common MARF tasks and applications. The PD patches may later influence development of the MARFL language for similar tasks (which is an independent intensional language for distributed/parallel demand-driven context array-based evaluations).

Keywords

MARF, PureData, audio processing, signal processing, pattern recognition, algorithms, programming languages, JNI

1. INTRODUCTION

This section presents a brief introduction to the MARF and PD projects for the unaware reader.

1.1 MARF

MARF [1, 2] is an open-source research platform and a collection of voice/sound/text and natural language processing (NLP) algorithms written in Java and arranged into a modular and extensible framework facilitating addition of new algorithms. MARF can run distributively [3] over the network or may just act as a library in applications or be used as a source for learning and extension. A few example applications are provided to show how to use the framework. There is also a reasonably detailed manual and the API references in the javadoc format as the project generally tends to be well-documented. MARF and its applications are open-source and hosted at `SourceForge.net`, where the project's page can be found at `http://marf.sf.net`. Serguei Mokhov, the author of this paper, is the founder and a core active developer of the MARF project.

1.1.1 Algorithms

MARF has a number of algorithms implemented for various pattern recognition and some signal processing tasks. These include Fast Fourier Transform (used in FFT-based filtering as well as feature extraction [4]), Linear Predictive Coding (used in feature extraction), neural network (used in classification), various distance classifiers (Chebyshev, Manhattan a.k.a city-block [5, 6], Euclidean [7], Mahalanobis [8], Diff[1], Hamming [9], and Minkowski [10]), cosine similarity measure [11], the Zipf's Law-based classifier [12], general probability classifier, and CFE-based 2D filters [13].

In general, MARF's pipeline of algorithms is presented in Figure 1, where the implemented algorithms are in white boxes, and the stubs or in progress algorithms are in gray. The pipeline consists of four basic stages: sample loading, preprocessing, feature extraction, and training/classification. Being plug-in oriented, any new implementations of those algorithms/stages can be easily added or replaced. One of the initial goals of the MARF project was to have a framework to test and compare different algorithm for the pipeline stages and how well they work together to allow researchers picking the better combination for their needs or to test their newer inventions against the performance of the existing implementations. For the broad collection of results, comparisons of the runs and algorithms, etc. please refer to the MARF's manual [1].

1.1.2 Applications

There are a number of applications that test MARF's functionality and serve as examples of how to use MARF's

[1]internally developed within the project, similar in behavior to the Unix `diff` utility

Pd Convention 2007. Montréal, Québec, Canada.

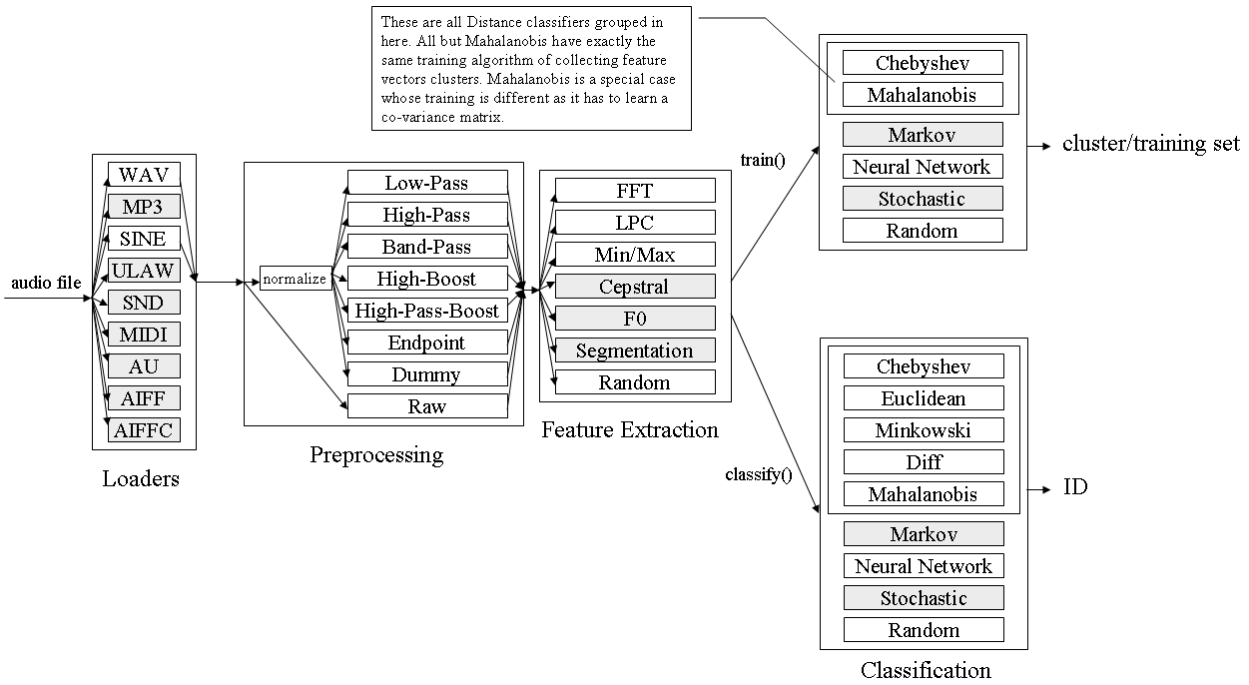

Figure 1: MARF's Pattern Recognition Pipeline.

modules. Most prominent applications are the following:

- `SpeakerIdentApp` – Text-Independent Speaker Identification (who, gender, accent, spoken language)
- `LangIdentApp` – Language Identification (written language, based on the NLP techniques)
- `ProbabilisticParsingApp` – Probabilistic Parsing of Natural Language (e.g. English)
- `ZipfLawApp` – Zipf's Law Statistical Study (see [12] for more details).

Smaller and module-specific testing applications for unit, regression, and functionality testing are the following: TestFFT, TestFilters, TestLPC, TestLoaders, TestMath, TestNN, TestPlugin, TestSampleRecorder, TestWaveLoader, and Regression.

As of this writing, natively in MARF the application are command-line based, and there only exist a rudimentary GUI prototype.

1.2 PureData

Certainly, everybody knows what PureData (PD) [14] is at the PureData Convention, but a small more formal description in the paper is still applicable. PureData is an open-source real-time graphical programming environment for audio, video, and graphical processing. It is the third major branch of the family of patcher programming languages known as Max (Max/FTS, ISPW Max, Max/MSP, jMax, and others) originally developed by Miller Puckette and company at IRCAM. The core of PD is written in C and maintained by Miller Puckette and includes the work of many developers, making the whole project very much a community effort. PD has a number of "externals" (PD plug-ins) written for it that extend the core PD functionality for artistic performance and real-time video and audio effects. PureData is also a language of the data-flow programming languages family naturally suited for visual programming and prototyping of signal and multimedia processing.

2. PURPOSE

Here we detail the aspects of the two projects that can be shared between MARF and PD for mutual benefit.

2.1 MARF for PureData

In Section 1.1.1 we listed a some of the algorithms MARF implements as of this writing. These implementations are made readily available for PureData as externals for use in its patches allowing quick comparison and picking the best algorithm for the patchers' work (depending on the task at hand, of course) in various sound or video signal processing needs. While some of the algorithms are already or soon will be implemented in PD natively (e.g. FFT), availability of the alternative implementations will allow choosing the best one after performance comparison. Additionally, MARF's implementations can be just adapted to C as the complete source code is available.

2.2 PureData for MARF

PD's graphical interface can serve as a decent GUI tool for MARF users and developers alike for testing and, even, rapid application development. Modeling and execution of MARF's applications, for example, for speaker, musical instrument, emotion, language identification, and voice authentication as a set of PD patches can be done with PD in a simpler way. Additionally, some algorithms (source code implementation) can be ported from PureData to MARF as well as the other way around at the source code level.

Additionally, the MARFL language is being developed (a separate topic for a separate project and publication) to "script" MARF's tasks to allow quick MARF application development. An ongoing evaluation and research is being done if and how PD can influence the MARFL's design.

3. MARF-PROVIDED EXTERNALS

This section briefs on the number of the PD externals originated from the mentioned (and some omitted) MARF's algorithms.

In order to support a pipeline style execution of MARF as in Figure 1 in a PD patch e.g. corresponding to a speaker identification application (the primary goal of this work for MARF), the sample loading (`marf_sl~`), preprocessing (`marf_pr~`), feature extraction (`marf_fe~`), and training and classification (`marf_cl~`) externals are created with the corresponding inlets and outlets to accept MARF-style input of arrays and preceding in the pipeline modules, as well as a signal type (hence the ~). Since MARF does some signal processing, it is natural for its code to provide signal classes and externals. As such, the `dsp` and `perform` methods are implemented for nearly all the modules where appropriate. The outlets provide the array, signal, and reference to self outputs to allow further feed of data down the pipeline or MARF-unrelated PD objects. Moreover, the entire `MARF` class that hosts the pipeline implementation is made as an external `marf~` as a shortcut for the entire pipeline, but it looses flexibility where one may want some of the modules separately.

While constructing the pipeline in PD, one can have the generic pipeline stages (e.g. a the mentioned objects `marf_sl~`, `marf_pr~`, `marf_fe~`, and `marf_cl~`), with an extra parameter inlet for a symbolic name of the specific algorithm (e.g. "fft", "lpc", "euclidean", 'nn', etc.). One can also have concrete module objects, e.g. `marf_fe_fft~`, or just a reference to the concrete algorithm (e.g. `marf_fft~`, `marf_lpc~`, `marf_mahalanobis~`, etc.) independent from the pipeline for the general-purpose use.

A complete list of the produced externals and convenience objects will be accompanied by the detailed documentation of the contributed code to the PD community. Along with the externals an example application patch will be provided.

4. INTERFACING MARF AND PD

A particularly interesting aspect is making the two projects work with each other, primarily because MARF is mostly written in Java, whereas PD is implemented in C. There are two possible approaches to do that. First, in order to interface the two, we use Java Native Interface (JNI, [15], or its GNU equivalent, CNI [16]) that allows Java programs to "talk" to programs written in other languages, including C, and vice versa. Thus, we create own JNI-based mechanism for invocation of the JVM and `marf.jar`'s classes from C. Alternatively, we write externals directly in Java and tie them in to PD through `pdj` [17] to manipulate PD objects. The following is the description of the two approaches.

4.1 JNI

MARF PD implementation provides a library of externals for the mentioned algorithms, called `marf.pd_linux`[2] according to the PD library naming conventions.

The process involves the following steps: creation of the corresponding C-language wrappers of the MARF modules destined for the inclusion as a PD externals library. The wrappers adhere to the PD coding standards and the API, and internally, call the Java methods of the MARF classes. From the JNI point of view this involves starting up a JVM (if not started yet), looking up class reference, getting the needed method identifier, and actually perform the call with appropriate parameters and gather the results, and pass them on to PD outlets.

4.2 pdj

There is an external by Pascal Gauthier called `pdj` that allows one to write Java code to interact with PD objects [17]. Its API is entirely based on Cycling74 Max/MSP `mxj` object implementation. This helps Java `mxj` objects to run on PD with `pdj`, but introduces an additional dependency on the Cycling74 API (the appropriate jar files should be present) and advocates writing Java-based externals to manipulate PD objects. In this paper, the approach and direction are slightly different: the Java objects to be "externals" are already prewritten not as such elsewhere (MARF, which acts like a blackbox), and the wrapper PD externals are written in C and manipulate Java objects instead, and map the results back to the PD objects with the MARF Java objects being unaware that they are invoked from PD. To summarize, the difference is Java externals manipulate PD objects in `pdj` vs. C wrapper externals manipulate `marf` Java (via JNI) and PD (classical, via C) objects. The latter approach allows calling the MARF classes an objects regardless their number and extension, symbolically, without having created wrapper Java classes. Thus, the actual `marf` externals are technically written C, just at some point of execution they invoke Java objects from `marf.jar` for the actual business logic.

[2]Linux is a start, but it is meant to be portable.

5. FUTURE WORK

The future work in this area will focus on polishing and perfecting the MARF's externals and their inclusion into PD along with other available externals there, the MARFL language and UI design for MARF applications and design of the new, innovative applications in both projects based on the presented contributions.

Integration and cooperation with the authors of `pdj` as well as PDMTL Abstractions [18] is planned to optimize and enrich the contribution efforts. Specifically for the former, while there exist the dependency on Cycling74's API, is still worth investigating a version of MARF patches that rely on `pdj` in order to compare its performance with the current implementation and possibly make it interoperable with Max/MSP.

6. CONCLUSION

We believe this contribution to the PD and open-source communities helps various multimedia and reseach projects in the field of audio processing, pattern recognition, and beyond. There are not enough audio recognition externals for PD, so MARF can contribute its potential at no cost. Likewise, the graphical environment of PD can help prototyping MARF applications.

7. ACKNOWLEDGMENTS

Here we would like to acknowledge the contributions from the original MARF co-founders, namely Stephen Sinclair, Ian Clement, Dimitrios Nicolacopoulos as well as all subsequent contributors to the project.

This research and development work was funded in part by the Faculty of Engineering and Computer Science of Concordia University, Montreal, Canada.

Finally, we would like to salute to the PD open-source developers, community, and the PDCon07 organizers for the outstanding work.

8. REFERENCES

[1] MARF Research & Development Group. *Modular Audio Recognition Framework and Applications*. SourceForge.net, 2002-2007. `http://marf.sf.net`.

[2] Serguei Mokhov, Ian Clement, Stephen Sinclair, and Dimitrios Nicolacopoulos. *Modular Audio Recognition Framework*. Department of Computer Science and Software Engineering, Concordia University, 2002-2003. `http://marf.sf.net`.

[3] Serguei Mokhov. *On Design and Implementation of Distributed Modular Audio Recognition Framework: Requirements and Specification Design Document*. Department of Computer Science and Software Engineering, Concordia University, 2006. `http://marf.sf.net`.

[4] Stefan M. Bernsee. *The DFT "à pied": Mastering The Fourier Transform in One Day*. DSPdimension.com, 1999-2005. `http://www.dspdimension.com/data/html/dftapied.html`.

[5] H. Abdi. Distance. In N.J. Salkind (Ed.): Encyclopedia of Measurement and Statistics. Thousand Oaks (CA): Sage, 2007. `http://en.wikipedia.org/wiki/Chebyshev_distance`.

[6] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall. New Jersey, US., 1995. ISBN 0-13-103805-2.

[7] H. Abdi. Distance. In N.J. Salkind (Ed.): Encyclopedia of Measurement and Statistics. Thousand Oaks (CA): Sage, 2007. `http://en.wikipedia.org/wiki/Euclidean_distance`.

[8] P.C. Mahalanobis. On the generalised distance in statistics. Proceedings of the National Institute of Science of India 12 (1936) 49-55, 1936. `http://en.wikipedia.org/wiki/Mahalanobis_distance`.

[9] Richard W. Hamming. Error Detecting and Error Correcting Codes. Bell System Technical Journal 26(2):147-160, 1950. `http://en.wikipedia.org/wiki/Hamming_distance`.

[10] H. Abdi. Distance. In N.J. Salkind (Ed.): Encyclopedia of Measurement and Statistics. Thousand Oaks (CA): Sage, 2007. `http://en.wikipedia.org/wiki/Distance#Distance_in_Euclidean_space`.

[11] E. Garcia. Cosine similarity and term weight tutorial, 2006. `http://www.miislita.com/information-retrieval-tutorial/cosine-similarity-tutorial.html`.

[12] George Kingsley Zipf. *The Psychobiology of Language*. Houghton-Mifflin, New York, NY, 1935. `http://en.wikipedia.org/wiki/Zipf%27s_law`.

[13] Shivani Haridas. Generation of 2-d digital filters with variable magnitude characteristics starting from a particular type of 2-variable continued fraction expansion. Master's thesis, Concordia University, Montréal, Canada, July 2006.

[14] Miller Puckette and PD Community. *Pure Data*. puredata.org, 2007. `http://puredata.org`.

[15] Beth Sterns. *The Java Native Interface (JNI)*. Sun Microsystems, Inc., 2001-2005. `http://java.sun.com/developer/onlineTraining/Programming/JDCBook/jni.html`.

[16] GCJ Contributors. *The Compiled Native Interface (CNI)*. GNU, 2007. `http://gcc.gnu.org/onlinedocs/gcj/About-CNI.html`.

[17] Pascal Gauthier. *java external plugin for pure-data*. http://www.le-son666.com, 2006. `http://www.le-son666.com/software/pdj/`.

[18] Pure Data Montréal Users Group. *PDMTL Abstractions v.2*. http://wiki.dataflow.ws, 2007. `http://wiki.dataflow.ws/PdMtlAbstractions`.