THE DEVELOPMENT OF GENERAL-PURPOSE SIGNAL PROCESSING TOOLS [1]

Hualin Gao, Richard Duncan, Julie A. Baca, Joseph Picone

Center of Advanced Vehicular System, Mississippi State University
{gao, duncan, baca, picone}@isip.msstate.edu

ABSTRACT

This paper describes the design and development of a set of general-purpose signal processing software tools. A GUI-based configuration tool is presented that allows block diagrams to be created to represent the procedures of the signal data flow. A front-end tool is used to realize the signal processing by using the diagrams and raw speech data files without user's programming. We describe the design philosophy underlying the development of the tools as well as the key features that enable realization of our design goals of extensibility and easy usability. We also discuss results of tests to verify the correctness and usability of the tool set.

1. INTRODUCTION

Signal processing tools extract feature vectors from raw data, and play an important role in the development of pattern recognition systems. For example, a typical speech recognition system is shown in Figure 1. The block named as the Acoustic Front-end in Figure 1 encapsulates most of the signal processing portions of a recognition system. In this paper we describe the design and implementation of the general-purpose signal processing components in such a system, which provide a GUI-based environment to perform signal processing research and education.

Many signal processing toolkits are currently available including popular commercial products such as MATLAB [1]. These toolkits provide powerful computation and analysis capabilities, and sophisticated graphical interfaces. Nonetheless, users need to write some code or scripts to realize their design for the complex data flow procedures. Debugging those code and scripts is not a trivial task. Some other toolkits such as the popular speech recognition tool HTK[2], the front ends of them also provide signal processing abilities; users can change parameters of the algorithms in the data flow, but not the data flow itself. Adding new algorithms to those toolkits requires modifying the base code of the existing system, a potentially time-consuming and costly undertaking that can significantly impede many research efforts.

To address these problems, we developed a signal processing tool set, which allows users to design the procedure of the signal processing at their wills without any programming. In addition, users can easily create new algorithms and fit into this tool framework.

The feature for our software is to separate the design and processing of signal processing procedure. A graphic user interface (GUI) is used in design stage to draw data flow graph and configure each algorithm in the data flow graph freely by users. The GUI consists of roughly 18000 lines of Java code. The backend processing tool is designed in processing stage to process the data flow diagram designed by users using the GUI tool. It includes about 70000 lines of C++ code.

In this paper, we present our software design rationale and approach for maximizing extensibility and usability.

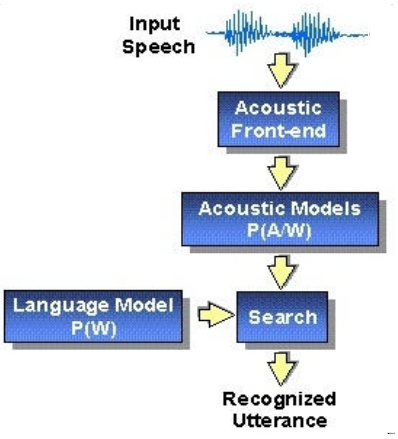

Figure 1: *A typical speech recognition system*.

1. This material is based upon work supported by the National Science Foundation under Grant No. EIA-9809300. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

2. CONFIGURATION TOOL

The magic ability of this signal process tool is to allow the users to realize their rapid prototyping design without any programming. Let's see how we use it and how it works through a simple example. Suppose we want to calculate the energy of a signal. The specification is the signal goes through a filter and window first, and then the energy is calculated. Let's see how easy it is to realize these by using our new tool. We can first start our general-purpose signal processing tool called transform_buider, which is developed using Java, and draw a block diagram as shown in Figure 2. There are four menus for this tool: file, edit, components and help menus. The file menu provides the abilities of saving and loading files. The edit menu provides editing ability for graph, such as adding, copying, cutting, deleting arc and removing blocks. The components include all the algorithms we support right now. Our current offerings in components can be sorted into two categories: basic DSP and support. The basic DSP components include commonly used algorithms, such as windows, filter, energy, cepstrum, Fourier transform and spectrogram etc. Support components allow high-level manipulation of data flow through block diagrams. Together, they provide a unique and powerful set of signal processing capabilities. The help menu gives users the help information. We can select algorithms from an inventory of predefined components menu, put them on the design working area and connect them by using the submenus in edit menu. Each block represents one algorithm for which the user can specify how to process data; each arc represents a data flow from one algorithm to another. Figure 2 shows the final data flow for our simple example. We can configure each component in the diagram by using a pop-up window as shown in Figure 3 after right-clicking the component. Users can configure

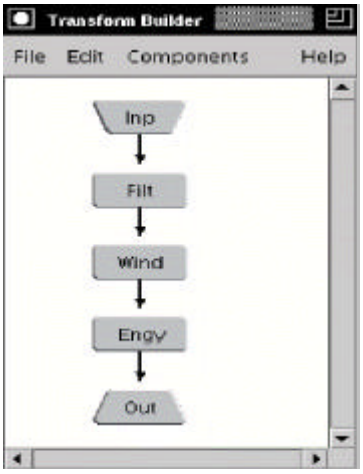

Figure 2: A configuration tool that allows users to create new front ends by drawing signal flow graphs.

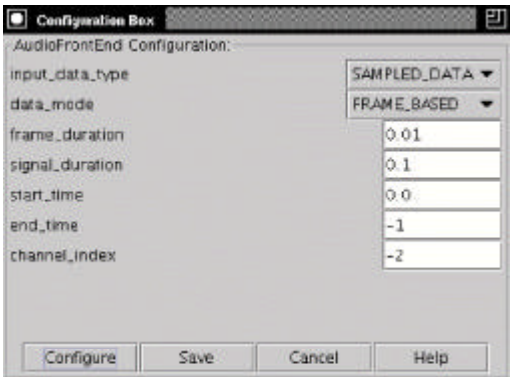

Figure 3: A pop-up window to configure each component.

parameters for each algorithm such as window type, window duration for window algorithm etc. The help button in Figure 3 provides help support for users. The configure button in Figure 3 directs users to next level configuration if it is necessary for this component. After the configuration, we can save all these to a file, which can be called recipe.sof here. This file records all the parameters for each algorithm and how the data flow goes. Finally we use the command line by starting our signal processing front-end tool isip_transform.exe (we will discuss it in section 3):

isip_transform.exe –p recipe.sof input.raw

The input.raw here is an input data file. The final result will be saved in a file specified in the output block.

The signal flow graphs are represented by a graph of component objects in the recipe file. Each component is a block in the graph, which wrappers one algorithm and its parameters. The design of this structure meets the requirements of the signal processing tool which will be discussed in next section.

The configuration tool can reload the recipe files and make modifications to them, and then save back to the same recipe files. This feature is very useful when the project is in the development stage which needs a lot of modifications or users want only make a small modification for an existing recipe file. This tool can also create a group of blocks and cluster them into a super block. The super block can be called just as a regular block with the same semantics. This feature can be used to build a large complex data flow graph.

Finally, to increase the extensibility of the tool, algorithms are presented in the interface through the components menu, populated from a resource file. All algorithms appearing in this menu are read from the resource file. Adding a new algorithm requires simply including a description into the resource file according to its format, such as algorithm name, algorithm parameters. No modifications to the source code of the signal

configuration and signal processing control tool itself are required. This has allowed users easily create new application specific algorithms, add them into this tool and take the advantage of this tool.

3. SIGNAL PROCESSING CONTROL TOOL

The signal processing control tool is a driver program and is called isip_transform.exe in our environment. It uses the signal processing library and algorithm libraries and output recipe files from signal configuration tool to fulfill the whole procedure of signal processing.

The signal processing library is a collection of specially designed modules, implemented as C++ classes, which serve as an interface between the block diagrams, created by the GUI configuration tool, and the computation algorithms, which will be described later. The components class in signal processing library wrapped all the algorithms inside it. All the algorithms look the same and work in the same way. The signal is processed according to the data flow graph in the recipe file. The algorithm object is created dynamically during the running time using the name and specification of each component. This process makes sure all the algorithms are looked the same in the signal configuration transform builder and can be configured in the same way. They signal processing library also using the graph theory to process the data flow graphs such as synchronization from different paths. It should be noted that the work of the signal processing library is hidden from the user by default. Its functions include: parsing the file containing the recipe created by the user with the configuration tool; synchronizing different paths along the block flow diagram contained in this file; preparing input/output data buffers for each algorithm, particularly for those requiring multiple frames of data, such as windows or calculus; scheduling the sequences of required signal processing operations; processing data through the flow defined by the recipe; and finally, managing conversational data. When users create their own new algorithms, it is necessary to register in the components class for the new algorithm, and no other modification is needed.

The algorithm library contains a collection of signal processing algorithms implemented as a hierarchy of C++ classes. The implementation of this hierarchy using an abstract base class, AlgorithmBase, and virtual functions or methods that comprise the interface contract, is the single most important feature, since it makes the library extensible. All algorithm classes are derived from this base class.

Because the design for algorithm class has the consideration of the extensible for the new algorithms, any new algorithm is easy to implement by just following our interface contract defined in AlgorithmBase class.

Expanding the collection of algorithms supported in our Algorithm library is the subject of on-going research.

It should be noted that the software described in this paper involves primarily the Algorithm and Signal Processing libraries [4] in the ISIP foundation classes (IFCs) hierarchy, which is part of a comprehensive public domain toolkit[3] for performing speech and signal processing research developed by the Institute for Signal and Information Processing (ISIP). The signal processing tools also take the advantage of the complex data structure and an abstract file I/O interface of the IFCs. They also inherited from the IFC some differentiating key features:

- Unicode support for multilingual applications;
- Memory management and tracking;
- System and I/O libraries that abstract users from details of the operating system;
- Math classes that provide basic linear algebra and efficient matrix manipulations;
- Data structures that include generic implementations of essential tools for speech recognition code.

4. EXPERIMENTAL RESULTS

We tested the quality of our toolkit along two dimensions, correctness and usability. The implementation of each algorithm is verified manually or by using other tools such as MATLAB. We have successfully built several complex front ends, including an industry standard front end based on Mel-frequency cepstrum coefficients (MFCCs) **[5]** as shown in Figure 4. The MFCC includes absolute energy, 12 MFCC's (often referred to as absolute MFCC's), and the first and second order derivatives of these absolute MFCC's. The cepstrum mean subtraction and maximum energy normalization are used in Figure 4. It is interesting to notice that the data path is very different from the left and right sides of the graph. Out software correctly process them. The processing results are checked manually step by step and prove they are correct.

We also assessed and enhanced the usability of our tools through extensive user testing conducted over the course of many workshops[6]. As part of this testing, we administered a user survey derived from the Questionnaire for User Interaction Satisfaction (QUIS), a measurement tool designed for assessing user subjective satisfaction with the human-computer interface[7]. Several features of the interface were modified or enhanced as a result of these tests. General examples include reductions in the number of menus, number of menu options, changes in wording of menu options, and modifications to the behavior of the drawing tool itself.

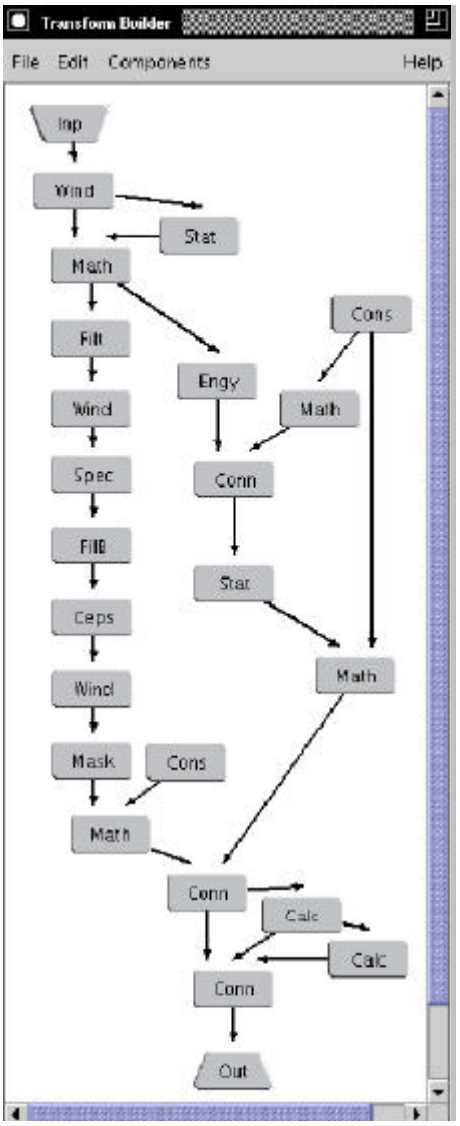

Figure 4: A MFCC block graph including cepstrum mean normalization and maximum energy subtraction.

5. CONCLUSIONS

This paper has presented the general-purpose signal processing components. These components were designed and developed in adherence to our philosophy of providing a flexible, extensible software environment for researchers. Our goal was to enable researchers to explore ideas freely, unencumbered by low-level programming issues. To achieve this goal, we implemented several critical features in our signal processing software tools, including a library of standard algorithms for basic DSP functions, the ability to add new algorithms to this library easily, and a GUI-based configuration tool for creating block diagrams to describe algorithms, allowing rapid prototyping without programming.

We have tested and verified these tools for both correctness and usability. We continue to monitor feedback from our user community in order to maintain the highest quality of the tools. These tools have been one of the most popular components of our toolkit, and are suitable for teaching basic concepts in digital signal processing. We also feel that visualization of the output results is very important and it is our future work.

6. REFERENCES

[1]. The MathWorks, Inc., Natick, MA, USA (see http://www.matlab.com/).

[2]. Hidden Markov Model Toolkit (HTK): http://htk.eng.cam.ac.uk/.

[3]. K. Huang and J. Picone, "Internet-Accessible Speech Recognition Technology," presented at *the IEEE Midwest Symposium on Circuits and Systems*, Tulsa, Oklahoma, USA, August 2002. (see http://www.isip.msstate.edu/projects/speech).

[4]. R. Duncan, H. Gao, J. Baca and J. Picone, "The Algorithm Classes," ISIP, Miss. State Univ., MS State, MS, USA, March 2003 (see http://www.isip.msstate.edu/projects/speech/software/documentation/class/algo/).

[5]. N. Parihar, J. Picone, "Performance Analysis of the Aurora Large Vocabulary Baseline System," *Proc. of Eurospeech '03*, Geneva, Switzerland, September 2003, 337-340.

[6]. J. Picone, Jon Hamaker, "Speech Recognition System Training Workshop," ISIP, Mississippi State University, MS State, MS, USA, May 2002 (see http://www.isip.msstate.edu/conferences/srstw/).

[7]. Chin, J. P., Diehl, V. A. and Norman, K. L., "Development of an Instrument Measuring User Satisfaction of the Human-Computer Interface," *Proceedings of SIGCHI'88*, New York, New York, USA, October 1988, 213-218. (see http://lap.umd.edu/q7/quis.html).