# 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 simple 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 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 modularity, extensibility, and 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 a 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 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]. Such toolkits provide powerful computation and analysis capabilities, and sophisticated graphical interfaces. Nonetheless, they also contain serious deficiencies that limit their usefulness in a research environment. For example, run-time efficiency and file I/O are two common issues with such high-level tools.

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.

Adding new algorithms to such toolkits requires modifying the base code of the existing system, a potentially time-consuming and costly undertaking that can significantly impede many research efforts. Special problems for speech recognition front ends, such as synchronization and buffering of data along a data flow graph, are difficult to handle in a simple and easy to understand framework. Of even greater importance and difficulty, data preparation for algorithms that require multiple frames of data, such as windows and differentiation, can be problematic.

To address these issues, we have developed a modular, flexible environment for signal processing. The key differentiating characteristics of our system include:

- Competitive technology with maximum flexibility;
- Well-documented and simple APIs;
- An object-oriented software design in C++.

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

## 2. DESIGN AND IMPLEMENTATION

Research in the area of speech recognition requires the development of large applications in a relatively short

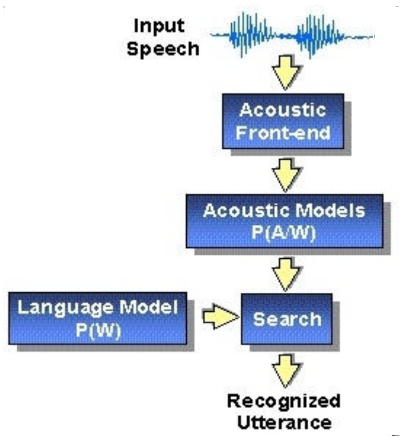


Figure 1: *A typical speech recognition system*.

period of time. To address these needs, the Institute for Signal and Information Processing (ISIP) provides a comprehensive public domain toolkit[2] for performing speech and signal processing research. This toolkit designed a large, hierarchical software environment to support advanced research in all areas of speech recognition, including signal processing. This environment contains ISIP foundation classes (IFCs) that provide features ranging from complex data structures to an abstract file I/O interface. IFCs are implemented as a set of C++ classes, organized as libraries in a hierarchical structure. Some key features include:

- 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.

We developed our signal processing toolkit to stringently adhere to the IFC design philosophy and framework. Although the signal processing tools discussed here are dedicated to the ISIP speech recognition tool kit, we pay a lot of attentions to our design and make it can be used in the general purpose signal processing. The software described in this paper involves primarily the Algorithm and Signal Processing libraries [3] in the IFC class hierarchy. At the outset, it was clear that the tools must not only allow a wide selection of algorithms, but also have the ability to vary every parameter of each algorithm easily and finally, provide users an efficient environment for evaluating new research ideas. Thus, the design requirements for these tools included:

- A library of standard algorithms to provide basic digital signal processing (DSP) functions;
- An ability to easily add new algorithm classes and functions without modifying existing classes;
- A block diagram approach to describing algorithms to realize rapid prototyping without programming.

Fulfilling the first requirement enabled users to directly realize a single algorithm such as a window with simple programming by building an algorithm object, and calling its functions. Meeting the second requirement allowed users to enhance system capabilities according to new requirements. This is described in Section 2.1. To meet the third requirement, we developed a signal processing control tool and a signal processing configuration tool. These are described in Section 2.2 and 2.3.

Our current offerings 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.

## 2.1 Algorithm Library

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. Users can directly use any of these algorithms in their application as shown in table 1 as long as users link their code with our library.

```
// define the filter properties
VectorFloat ma_coef(L"-0.97, 1.0");
VectorLong ma_lag(L"-1, 0");
// set the ma coefficients for filter
filter.setMACoeffs(ma_coef, ma_lag);
// filter the signal
filter.compute(filter_out, temp_window);
// take the hamming window for the signal
window.compute(out_data_window, filter_out);
// calculate the energy
 energy.compute(out_data, out_data_window);
```

*Table 1: An example for calculating energy by taking filter and window first for a signal.*

Because the design for algorithm class consider 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.

## 2.2 Signal Processing Configuration Tool

The magic ability of this signal process tool is to allow the user 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. It is very easy to realize these by using our new tool. We can first start our tool called transform_buider, which is developed using Java, and draw a block diagram as shown in Figure 2. We can select algorithms from an inventory of predefined components menu, 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 after right-clicking the component as shown in Figure 3. Users can configure parameters for each algorithm such as window type, window duration for window algorithm etc. Then 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 2.3), "isip_transform.exe –p recipe.sof input.raw". The final result will be saved in a file specified in the output block.

The 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 need 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. 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.

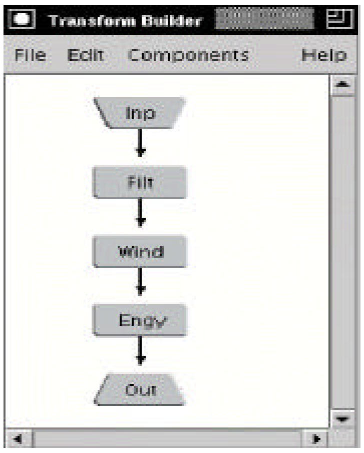
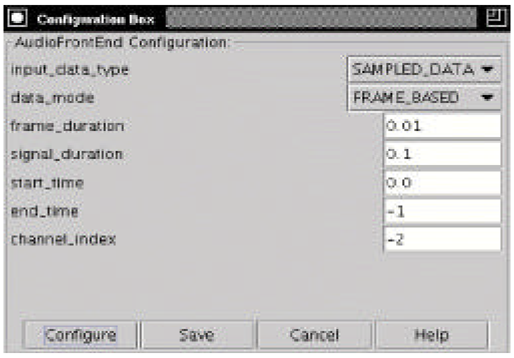


*Figure 2: A configuration tool that allows users to create new front ends by drawing signal flow graphs.*


Configuration Box

AudioFrontEnd Configuration:

| | |
|---|---|
| input_data_type | SAMPLED_DATA |
| data_mode | FRAME_BASED |
| frame_duration | 0.01 |
| signal_duration | 0.1 |
| start_time | 0.0 |
| end_time | -1 |
| channel_index | -2 |

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

## 2.3 Signal Processing Library and Control Tool

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, described in Section 2.1. 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.

The control tool is a utility 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.

## 3. 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) **[4]** 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. The processing

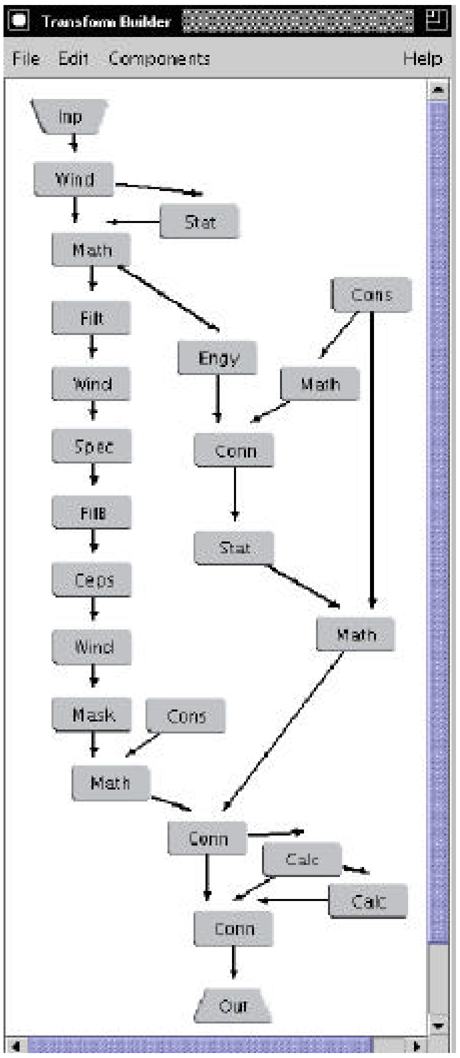


*Figure 4: A MFCC block graph including cepstrum mean normalization and maximum energy subtraction.*

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[5]. 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[6]. 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.

## 4. 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.

## 5. REFERENCES

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

[2]. 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).

[3]. 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/).

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

[5]. 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/).

[6]. 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).