RAPID PROTOTYPING OF SIGNAL PROCESSING ALGORITHMS

Issac J. Alphonso

Institute for Signal and Information Processing
Department of Electrical and Computer Engineering
Mississippi State University, Mississippi State, Mississippi 39762
alphonso@isip.msstate.edu

ABSTRACT

A common need in the area of signal processing, particularly in speech research, is the development of large applications in a relatively short period of time. Research in signal processing has been largely limited by the relatively long turn around time from the prototyping phase to the production phase. In this paper, I would like to describe a large hierarchal software development environment, based on C++, developed to for the rapid prototyping of signal processing algorithms. The environment allows the programmer to easily link together a large number of signal processing classes, and this in turn facilitates the rapid production of signal processing applications. There are three building blocks that help in creating the algorithmic prototypes: the signal object file, the concurrent development and multilingual data support. This environment relies heavily on object orientated programing philosophies and structured programming techniques.

THE SIGNAL OBJECT FILE

The primary goal of an environment that allows for rapid prototyping of algorithms is a flexible data file format. In order for such an environment to be successful, provide platform- independence, it should provide support for a machine-independent binary file system. The Signal Object File (Sof) provides three fundamentally important features: transparent support of ASCII and binary files; the ability to store multiple instances of arbitrary user defined objects in a file; and the ability to retrieve pieces of complicated objects.

In an object-orientated system, it is important to have the ability to store multiple instance of an object. For example, a class called Versions has been put under a revision control system like CVS (Concurrent Version System). Most Sof files will contain multiple instances of this object describing changes that have been made to the file. This feature can save a lot of time in prototyping algorithms as one can retrieve a previous revisions of a class. This feature is also used to support large text databases in which sentence are written to file as text objects. Each file can contain thousands of sentences.

One distinct advantage that Sof has over other object-orientated I/O schemes is that each piece of data does not need to be tagged with identification bits. This ends up proving to be extremely costly for vector-orientated data types such as speech signals. Hence, a storage mechanism is needed that does not introduce any overhead pre signal sample. Such a mechanism should also support the storage of vector data in machine-dependent sizes, such as short integers or one-byte unsigned characters.

The Sof call is primarily an index manager that maintains a list of objects and their locations in the file. The actual I/O

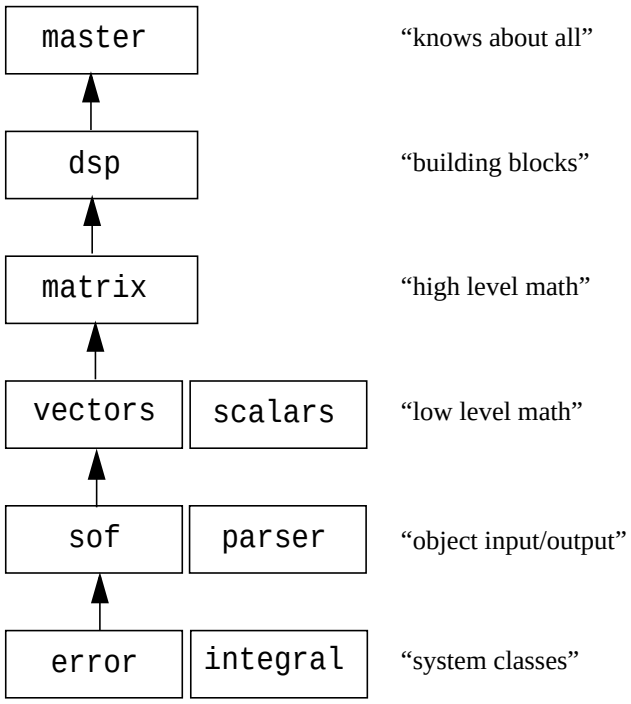

Figure 1. A structured programming implementation of a signal processing class hierarchy. All objects can read/write themselves and all mathematics is performed using well developed set of mathematical operators that support all the

methods are member functions of each class - they are not generic member functions of the Sof class. Since each class is a hierarchy of the other, lower-level classes, implementing these methods for each class is straightforward - execute the I/O methods for each method for each object in the class to be stored externally in a file. A the very bottom of the hierarchy the scalar classes - these know how to read/write themselves to ASCII or binary in a machine-independent manner.

ASCII files are quiet simple. They simply consist of data prefaced by an object tag. An index is constructed on the fly when the file is opened. Binary files are more complicated that ASCII files. The file header contains a binary string which is nothing more a pointer to a data object. The pointer indicates the position of the last data object in the file, upto which data should be read. The file footer, which is also a binary string, contains an index that of the starting positions and sizes of all objects in the file. Aa example of both files types in is figure 2. After the file is opened, both the ASCII and the binary files are processed in a similar manner.

ASCII files are provided so that editable and readable files can be supported. All data files in the environment weather signal files, text databases, or recognition models, are Sof files. Users need not write any special parsing algorithm for parameter files - something that usually takes a large amount of time. Sof provides a single, uniform interface to all files.

CONCURRENT DEVELOPMENT

The development time of signal processing algorithms, has been growing exponentially in recent years. An environment, that allows for rapid prototyping of signal processing algorithms, that is supported by a revision system like RCS or SCCS, can be made even more effective if it could allow more than one person to manipulate the same file simultaneously. The ability to have multiple developers working on the same project, without any restrictions like file locking or reserved checkouts, can produce a significant decrease in the time during the prototyping phase. This Concurrent Versions System (CVS), while incorporating all the version control features of RCS, by default provides unreserved checkouts. In this model, the developers can edit their own working copy of a file simultaneously.

The ability to allow more than one person to edit the same file simultaneously is not without its drawbacks. The first person to add his changes to the current version has no way of knowing that another person has started to edit it. When the other person goes to add his changes he will get a error and must bring his working copy upto date with the current version of the file. However, CVS does support mechanisms which facilitate various kinds of communication, without actually enforcing rules like file locking or reserved checkouts.

Apart from concurrent development, CVS provides a flexible modules database that provides a symbolic mapping of names to collections of directories and files. This facility allows developers to check out an entire directory trees and not just single files. Another feature that CVS supports is branches, which allows several lines of development to occur in parallel, and provides mechanisms for merging branches back together when desired. CVS can also tag the state of the directory tree at a given point and recreate that state in the future. These facilities are in keeping with some of the most basic features object orientated programing.

The cilent server facilities that of CVS enables developers scattered by geography or slow modems to function as a single team. The version history is stored on a single central server and the client machines have a copy of all the files that the developers are working on. However, the network between the client and the server must be up to perform CVS operations but need not be up to edit or manipulate the current versions of the files. Clients can perform all the same operations which are available locally.

```
# Sof v1.0 #     (ASCII file)
# Signal 0 #
sample_frequency = 8000.00 hz;
sample_type = linear;
sample_precision = 16 bits;
sample_compression = none;
sample_data = {
{255,-367,1024,-699,4321},
{27,-266,-555,1024}};
# Float 23 #
value = 2342;


# Sof V1.0 #     (binary file)
[binary string]
[signal object data in binary]
[float object data in binary]
[symbol table]
[binary index]
```

Figure 2. An ASCII Sof file followed by a binary Sof file. Both files contain two objects: a signal object and a float object. After the file opened both the ASCII and the binary