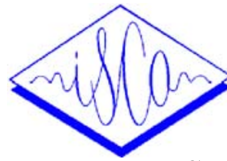
ISCA Archive
http://www.isca-speech.org/archive



6th European Conference on
Speech Communication and Technology
(EUROSPEECH'99)
Budapest, Hungary, September 5-9, 1999

C++ SOFTWARE ENVIRONMENT FOR SPEECH SIGNAL PROCESSING

Marcus M. Prätzas, Ulrich Balss, Herbert Reininger and Harald Wüst
Institut für Angewandte Physik, Johann Wolfgang Goethe-Universität, 60054 Frankfurt, Germany
praetzas@apx00.physik.uni-frankfurt.de

Abstract

Here we present the C++ library SPC (Speech Signal Processing Classes) as development tool for assembling of speech processing applications. SPC offers real-time processing, batch processing of large databases, visualization, and analysis of signals between processing steps. In SPC the data stream occurring in speech processing is partitioned in three different information flows: signal data, control information and visualization data.

Because hardware dependent program code is limited exclusively to some special methods, SPC can be adapted to different hardware environments easily. System specific code is encapsulated in low level parts of SPC and SPC user programs can be compiled on various platforms without any changes in source code. Up to now SPC supports Windows 95/98/NT, IBM AIX and LINUX.

Keywords
C++, software development environment, speech signal processing, SPC

10.21437/Eurospeech.1999-83

I. INTRODUCTION

Speech processing makes use of various sophisticated concepts and methods of information processing. For example low bit rate speech coding for mobile telecommunication relies on efficient data compression schemes. Pattern classification based on statistical models or artificial neural networks is applied for speech recognition and speaker recognition. Furthermore, the transformation of symbolic data in acoustic data plays a major role in speech synthesis. These complex applications of speech processing can be constructed from simple modules, which can be seen as universal building blocks for a variety of signal processing algorithms.

Due to this methodology, a software development tool was designed that is based on a modular concept. Here we present the software environment SPC (Speech Signal Processing Classes). SPC forms a hierarchically organized C++ class library. It is intend to be used for creation of speech processing applications offering as different features as real-time processing, batch processing of large databases, visualization or analysis of signals between processing steps. The paper is disposed as following. First we describe the modular structure of SPC. Thereafter, different concepts for the realization of signal processing are discussed. For an example application, its performance is compared with those of a traditional C program. At last, we give a conclusion and show some perspectives of further investigations.

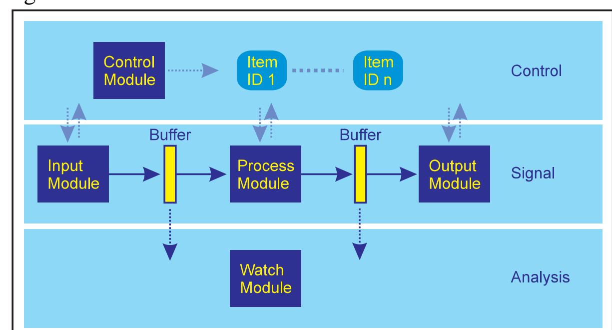
II. STRUCTURE

Base of SPC is a hierarchical structure and a highly modular realization. It consists of different modules for data storage and processing derived from a global base class which supplies access to a standard interface for data handling and error processing. In several lower layers more specialized features for the data buffers and processing modules are implemented. The data dependent interface between the processing modules allows an optimized data storage on one hand and on the other hand an efficient data transfer between several modules.

The data path is divided in three different components as shown in Fig. 1 which consist of

1. **Signal path**: This is the core of the signal processing where input data coming out of one or more input modules are processed by several processing modules and transferred to output modules within a processing chain. All of the modules are connected by signal buffers, which are the data storage elements within the chain.
2. **Control path**: Control and side information which is not part of the signal flow (block sizes information, sampling rate, filenames) could be distributed globally to the modules by a separate control path during an initialization phase.
3. **Analyzing path**: Signal data or derived values could be extracted between the processing modules for visualization, etc. The analysis layer has direct access to the data buffers.

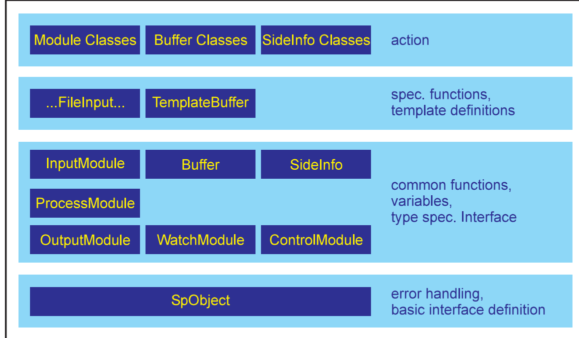
Figure 1: Partitioned information flow.



The objects of the SPC library are derived from an abstract class `SpObject`, as shown in Fig. 2. `SpObject` is responsible for error handling and contains some global variables and virtual methods as a first part of the interface between the processing modules and the data buffers. Several functionally different virtual classes are derived which define the structure of the library. They are divided in objects for input (`Input-`

Module), processing (ProcessModule), output (OutputModule), data buffers (Buffer), common data types (Common), visualization (WatchModule), control (ControlModule) and side information (SideInfo).

Figure 2: Hierarchies of the SPC library.



Using these structural classes, more detailed classes are derived in a third layer. E.g. based on the input module class, there exists a specialized class which supports file input functions. In this special layer general functions and classes are provided which could afterwards be used by the different processing modules in common.

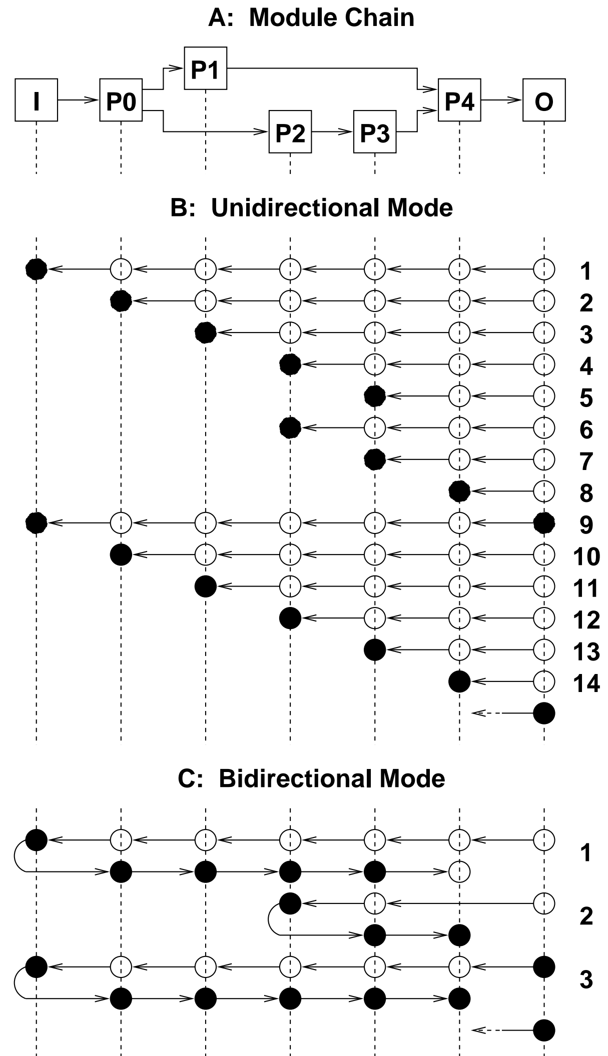
The last layer consist of the data and processing modules themselves. They are derived from their base class and if necessary from other objects which implement hardware dependent functionality.

III. Signal Processing

Signal processing within the modules is concentrated in their public method `short DoIt(void)`. In it the essential function of the module is realized. As the internal structure of DoIt is based on processing a standardized procedure, an autonomous controlling of the module is implemented. Nevertheless, already existing algorithms can be integrated at a well defined position of the source code of DoIt whenever a new module has to be designed. The return value of DoIt signals the main program whether data is processed (return value of 1) or not (0). The end of transmission (EOT) is advised to the following modules by setting a flag in the output buffers. In this way, the functions of the modules in the main program can be applied by repeated calls of their DoIt methods while the return value can be used for the decision of the sequence of the module calls, i.e. calls of their DoIt methods. Due to the autonomous control of the modules, this sequence results in an application-independent strategy. Three procedures were designed that are presented in the following.

First, because it seems to be most obvious, the modules can be called unidirectionally in the order, in which they occur in the signal flow (U-mode). Fig. 3B illustrates this methodology for the processing chain shown in Fig. 3A. But, since this permits a higher flexibility in the structure of the processing chain, the signal flow should be retraced iteratively, beginning with the output modules over the processing modules to the input modules. E.g. this is necessary for segmental processing of continuous signals or in case of feedback loops within the module concatenation. A cycle run is terminated as soon as one of the processing or input modules processed data. The program run ends if none of the modules operated.

Figure 3: Different operation modes of an SPC application. For the module chain shown in (A) the program behavior during some iterations of the U and B-mode is illustrated in (B) and (C) respectively. Successful ones of module calls are represented by filled sets, whereas empty sets unsuccessful calls are assigned.



Although this method operates reliably, its major weak point is situated in the fact that the processing chain is retraced often accurately up to the following module of that one that in the previous iteration operated. From this a high, unnecessary additional expenditure in the program management results. Therefore the second structure was sketched, which is more efficient. It moves on a bidirectional way through the processing chain (B-mode), as it is shown in Fig. 3C. As in the U-mode each iteration begins with the retracing of the processing chain, until one of the modules operated. In this way the high flexi-

bility of the U-mode is kept in the B-mode. But here the cycle run does not end with this module. Instead of this the processing direction is turned around and the received data are processed subsequently in all following processing modules. Like that, unsuccessfully calls of modules are distinctly less frequent.

In further experiments, instead of an iterative processing, also a thread based processing was investigated (M-mode). Here each module defines a separate thread, which can be processed parallel to the other modules [3]. As long as no data are available for processing, a thread is halted. Therefore, the controlling of the modules is extended by signals for thread control.

IV. Application

Any SPC application consist of five sections as shown for the simple example of an audio-monitor program in Listing 1. The audio signal is recorded and played back using the full-duplex mode of a soundcard. Depending on the hardware, which is selected via a compiler switch, the appropriate soundcard submodules are included:

1. **Creation**: The constructors of input and output modules are called. Parameters may be passed as the desired samplerate of 8000 Hz in the example. The buffers are created in the same manner. A ring-buffer of length 8192 and data-type `short` is used in Listing 1.
2. **Connection**: The modules are connected by assigning output buffer(s) for the input modules and input-buffer(s) for the output modules. Further processing modules have both: in- and output buffers. This assembles the processing chain.
3. **Initialization**: In the example the in- and output is started. If processing modules are used their parameters can be set at this time so far not passed through the constructor.
4. **Operation**: Calling the `Doit` routine of each module in an order as described in section III.
5. **Deletion**: Modules and buffers are deleted calling their destructors.

Listing 1: Audio-monitor program.

```
main() {
short sNEnde = 0;
AudioIn *pInput; AudioOut *pOutput;
Buffer*pBuffer;

pInput  = new AudioIn(8000);
pOutput = new AudioOut(8000);
pBuffer = new VShortRBuffer(8192);

pInput->SetOutputBuffer(pBuffer);
pOutput->SetInputBuffer(pBuffer);

pOutput->Start(); pInput->Start();

do {
    pOutput->DoIt();
    sNEnde = pInput->DoIt();
 } while (sNEnde);

delete pInput; delete pOutput;
delete pBuffer;
}
```

As one can see a simple audio in- and output is achieved within less than 20 lines of code. Using the same structure any SPC application can be implemented.

Even SPC modules may be used in conjunction with (existing) plain C or C++ program code the library is intended to cover all speech signal processing operation in an application. Therefore ca. 200 different modules are available up to now. Among them there are modules for linear prediction as well as modules for spectral analysis or hidden-markov modeling.

Two major quality criteria for signal processing libraries are their performance and their flexibility to use it in new applications. Performance is not only needed for nearly real-time applications but also for processing huge quantities of speech files. Flexibility is desirable in program development where buffers and modules can be combined nearly arbitrarily. In that case one might use a kind of buffer even if it has not the optimal structure. For this purpose SPC supports an optimized and a flexible mode (called release mode) which can be selected with a compiler switch[1]. The major properties of each mode are summarize in Tab. 1.

The performance of SPC have been tested for both operation modes. As an example the SPC implementation of an LPC vocoder [1] was compared to a traditional C-code implementation. The C-implementation uses standard algorithmic functions [4]. Data is stored in plain arrays. SPC uses 16 modules each representing an algorithmic function. The modules are connected with ring-buffers and are called in bidirectional order. The vocoder module chain is illustrated in Fig. 4.

Table 1: Optimization versus flexible use.

Property	Optimized (opt)	Release (rel)
buffer data-type	adjusted to the module	within the possibilities
buffer storage-type	adjusted to the module	arbitrary
buffer length	power of two	arbitrary
module and buffer error handling	none	read and write access

1. Of cause debugging is possible, which is a third possible mode.

Figure 4: SPC vocoder implementation. The buffers are represented by arrows.

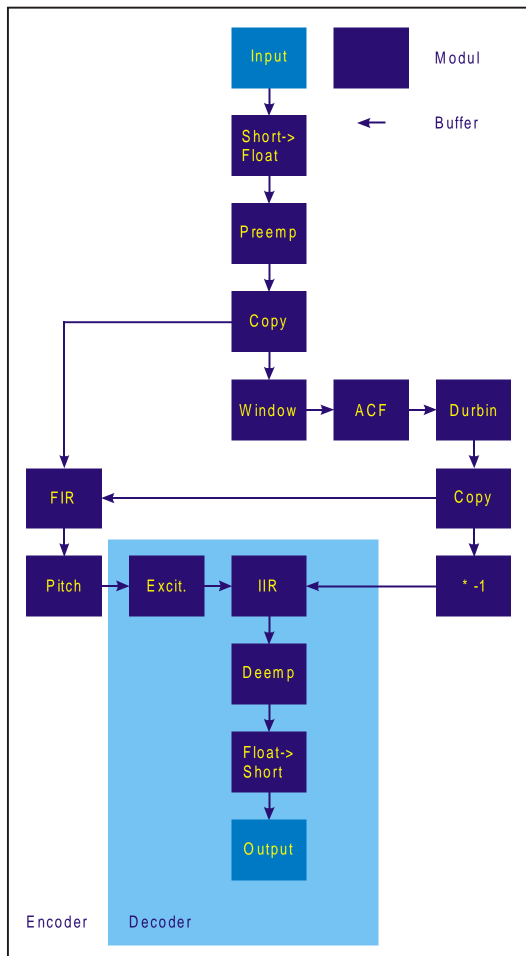
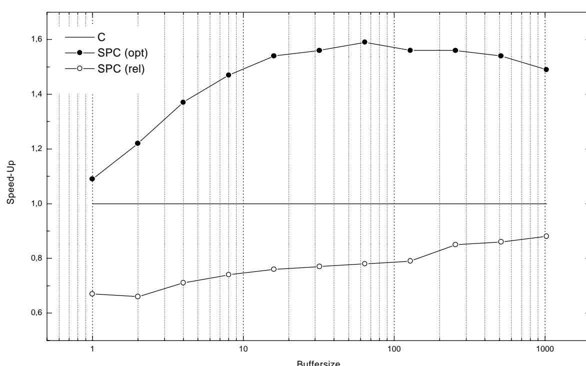


Figure 5: Performance of SPC implementation of vocoder measured as speed-up compared to a one-block processing implemented in plain C code.



The result of the performance test is shown in Fig. 5. The C program serves as baseline with a speed-up equals one. The buffersize of the C application is not increased, because it can not be changed without great effort. Starting with the same buffersize SPC is slightly faster because the memory access have been optimized e.g by in-line calls. Increasing the buffersize a maximum Speed-Up of about 1.6 is reached for the optimized version. The release version admits the use of nearly arbitrarily buffertypes. This feature decreases performance as it can be seen in Fig. 5 due to the use of virtual function which offer the flexibility of the release mode.

V. CONCLUSION

SPC is a C++ class library for the realization of speech applications. It consists of a large number of modules like modules for linear prediction, feature extraction, filtering, hidden-markov-models and neural networks. Together with modules for audio and file I/O and special modules for data visualization complex algorithms for speech processing can be realized in a flexible and efficient manner. The SPC library could be down-loaded for free together with a web-based documentation from our site [5]. Without any modifications user programs using SPC could be compiled for different platforms like UNIX, Windows 95/NT and Linux. The library could be simply extended by user modules. Every module, which is conform to the SPC standard, could be used in every SPC application having the same flexibility as a SPC standard module. The highly structured interfacing scheme between buffers and modules guarantees the application independent use of every module. This gives the possibility of a simple software reuse within larger projects.

Although not stated here SPC can be integrated in a Microsoft Windows based application using MFC [2]. Visualization modules to display signal data or calculated parameters are available. Currently those modules are extended trying to integrate multithreading support, which is not available for all system platforms yet.

VI. REFERENCES

[1] Deller, J. R., Proakis, G., Hansen, J. H. L.: Discrete-time processing of speech signals; Macmillan Publishing; New York; 1993.

[2] Microsoft Foundation Classes; Microsoft Visual C++ 5.0 Documentation.

[3] Prasad, S.: Multithreading Programming Techniques; McGraw-Hill; New York; 1997.

[4] Press, W. H., et al.: Numerical recipes in C: the art of scientific computing - 2nd. ed.; Cambridge University Press; New York; 1992.

[5] On-line manual and software for down-load: http://www.ap.physik.uni-frankfurt.de/spc