



THE SPAR SPEECH FILING SYSTEM

M.A. Huckvale¹, D.M. Brookes², L.T. Dworkin³, M.E. Johnson¹,
D.J. Pearce³, L. Whitaker³

ABSTRACT

SPAR (Speech-Pattern Algorithms and Representations) is the name given to the consortium of University College London, Imperial College, GEC Hirst Research Centre, Plessey Research and Leeds University funded under Alvey Project MMI-09. The project is concerned with advanced speech analysis algorithms, and from the outset saw the need for a system for speech data management. The SPAR Speech Filing System (SFS) was developed to support the design and comparison of analysis algorithms, and to manage many different parametric representations of the speech signal.

This paper describes the motivation behind SFS, the structure of SFS speech files, the support it provides for both the programmer and the user, and also an associated statistical tool that analyses SFS files. The paper concludes with a short justification for some of the design decisions.

INTRODUCTION

Speech research involves the transformation and interaction of a number of different representations of the speech signal. For algorithm development it is necessary to compare the results of different methods and to correlate signal representations at different acoustic-phonetic levels. There is also a need to perform statistical analyses over linguistic contexts. In addition a collaborative project such as SPAR requires a mechanism for the communication of data and algorithms.

The SPAR Speech Filing System (SFS) was developed to meet these requirements. We have found that the SFS methodology provides a unifying framework for software development and a consistent user interface. SPAR project members have used SFS as a common tool for the development of 'SPARbase': a speech processing system implemented in 'C' to run under Unix System V.

SFS DATA STRUCTURES

Using SFS, many different representations of a speech token can be stored in a single speech file. The software currently supports the following representations, and a speech file may contain an unlimited combination and repetition of types:

1. (SP) Speech pressure waveform.
2. (LX) Laryngograph waveform.
3. (TX) Points of voiced excitation.
4. (FX) Fundamental frequency.
5. (AN) Annotations.
6. (LP) Phonetic Feature description.

¹ Department of Phonetics and Linguistics, University College London, Gower Street, London WC1.

² Department of Electrical Engineering, Imperial College of Science and Technology, Prince Consort Road, London SW7.

³ GEC Hirst Research Centre, East Lane, Wembley, Middlesex.

7. (SY) Synthesizer control data.
8. (WD) Chart-structured text.
9. (DI) Spectrogram.
10. (VU) Voicing information.
11. (CO) Spectral coefficients.
12. (FM) Spectral peaks.
13. (EN) Speech energy.
14. (PC) LPC coefficients.
15. (MM) Markov model.
16. (TR) Speech parameter tracks.

The internal structure of a file has been kept very simple: new data sets are just appended, and traversal is performed by following a set of forward links. There is, however, the capability for extending the design with hierarchical structures.

At the top of a speech file is a "main header": a data structure containing information about the source of the primary data in the file. Since this is usually a database recording, the main header is organised to hold such information as the speaker name, token, recording session, recording conditions, etc. There are also file codes to identify SFS versions, to differentiate speech files from other Unix files, and to tie back a file to the user and machine which created it.

Users see the internal contents of a speech file as sequence of "items". Each item is actually a data set preceded by an "item header": a data structure initialised by processing programs that records information about the type, format and size of the data set. One important aspect of the item headers is the recording of a 'processing history' which ties each data set to the program that created it, the 'parent' data sets from which it was created and details of the set of processing parameters used. Since the processing history is vital to maintaining a record of the data sets within a file, SFS routines (see below) protect files from pairs of data sets with identical processing histories.

The SFS file architecture imposes a 'transformational' model onto speech processing programs: each takes a set of 'input items' (i.e. input data sets) and produces a set (possibly empty) of 'output items'. Thus a larynx-synchronous LPC analysis program might take a speech waveform (SP) and a set of voice excitation markers (TX) to produce an output LPC (PC) item. This transformational model encourages programmers to write programs that treat input items in generic types, so that for example, it is as simple to perform an analysis on synthetic speech as it is on natural speech. The transformational model is also easier for the user, since the documentation of a program shows which inputs each program requires, and which it produces. SFS also specifies standard program mechanisms for the selection of input items by the user.

The contents of a speech file can be displayed in a 'tree' form on the basis of the processing histories. At the root are the primary data sets: natural speech, laryngograph waveform, hand annotations. Nodes and leaves represent processed data sets and branches in the tree represent processing links between items.

SFS SOFTWARE SUPPORT

SFS provides programmers with a library of data-set manipulation routines designed to protect them from details of the low-level implementation. In addition SFS provides a set of graphics routines that allow displays

of speech data on a number of different graphics and hardcopy terminals. The SFS provision of routines for reading, writing and displaying data sets results in programs that may be ported to other Unix systems. We limit our continuing improvement of the filing system to within SFS routines, thereby protecting a large investment in existing programs and data. As an example, there is only one routine for adding items to speech files - and all processing programs use it. This has allowed us to build in protection against the corruption of speech files.

SFS encourages programmers to provide a consistent user interface and standard program documentation. This has been extremely useful in enabling programs written at one site to be used at another. The consistent command-line interface has allowed us to write a menu-driven interface to the SPARbase suite of programs for end-users of the software.

SFS also provides a number of utility programs for the manipulation of speech files themselves. There are utilities for the examination of the contents of files, for the editing of the main header and for the deletion of unwanted data sets (preserving processing histories where required). There are utilities for copying data to and from foreign data formats, such as binary, ILS and the Alvey Speech Club format.

SFS SPEECH MANIPULATION LANGUAGE

One of the benefits of constraining speech data into generic types is that it is possible to write data analysis programs that assess the results of different algorithms. In addition the SFS preserves the relationship in time between different representations of the signal, so that analyses and comparisons can be made across acoustic-phonetic levels.

The SFS speech manipulation language ('sml' for short) takes the process of generic data analysis a stage further by providing a *programming environment* in which measurements may be made within files, and statistics gathered across files. Thus it is possible to assess the acoustic characteristics of a phonetic context both across processing programs and across instances of the context. 'sml' is able to establish contexts by filename, by main header information and by annotations within files. 'sml' contains simple facilities for statistical analysis and for graph plotting.

We see 'sml' as an important tool in furthering our research into relationships between speech patterns and their phonetic interpretation.

DISCUSSION

We have made critical decisions in the design of the SPAR Speech Filing System:

1. All data for a single token is stored in a single file.
2. Data sets are kept small enough to load into memory.
3. Programs only communicate through information stored in files.
4. Programs are designed to operate at the command line level.

We have chosen single files because this makes it simple for the user to generate and keep track of different representations of the speech data and because it allows tight control over the recording of processing

history.

We have chosen to limit the size of data sets to the memory capacity of the computer because this makes the writing of speech processing programs much more straightforward. Programs assume that the input data set can be loaded into memory and they halt if the data is too large. We have not placed artificial size constraints within programs, so as larger virtual memory systems become available the upper data set size automatically increases. To deal with long database recordings, the user must either chop the recording up into many smaller files, or use a "link" mechanism in which pointers to sections of the recording are kept within speech files, rather than the speech data itself. We hope that this latter approach will allow us to extend the speech file concept to optical disk storage and to data access over a local-area network.

Program communication through files is necessary because our software development effort is spread over many sites. We have chosen to develop a large number of small programs for processing files in what we hope is the Unix tradition. Although the format of a data set must be constrained by its generic type to be accessible by other programs, the developer has the possibility of recording special information in the item header for unusual circumstances.

Finally, designing our system with a simple, consistent command line interface has had a number of benefits: it makes programs created at other sites predictable to use, it makes possible a menu-driven interface, which in turn gives a coherence to SPARbase that makes it accessible to naive computer users.

SOFTWARE AVAILABILITY

To determine the availability of the SFS software components please contact the first author at the address shown.

ACKNOWLEDGEMENTS

The work reported here has been supported through grants from Alvey project MMI-09, the Science and Engineering Research Council and the RSRE Speech Research Unit.