CENTER FOR COMPUTER RESEARCH IN MUSIC AND ACOUSTICS MAY 1988

Department of Music
Report No. STAN-M-52

AN ENVIRONMENT FOR THE ANALYSIS, TRANSFORMATION AND RESYNTHES OF MUSIC SOUNDS

Xavier Serra

Research sponsored in part by
The System Development Foundation

CCRMA
DEPARTMENT OF MUSIC
Stanford University
Stanford, California 94305

An Environment for the Analysis, Transformation and Resynthesis of Musical Sounds

Xavier Serra

Center for Computer Research in Music and Acoustics (CCRMA)
Department of Music, Stanford University
Stanford, California 94305

This paper describes an environment developed at CCRMA for the analysis, transformation, and resynthesis of sounds. It has been written on a Lisp Machine workstation, using an Array Processor to speed up the signal processing operations. The program is designed as a research tool and a sound manipulation workbench for music composition.

Introduction

One of the traditional computer applications in music is the manipulation of prerecorded sounds (digitized sounds). These manipulations can be simple, like looping (a section of a sound repeated over and over) and mixing (adding sounds), or they may involve soffisticated signal processing techniques such as the Short-Time Fourier Transform (STFT) or Linear Predictive Coding (LPC) (Moorer 1979, Schafer and Rabiner 1975). There are already many commercial products for music applications that use the somewhat simpler techniques.

The environment presented in this paper integrates in a single program the more advanced techniques, most of which are still in an experimental stage. With this program the user has access to a variety of signal processing tools used to analyze sounds, transform the analysis data and resynthesize the sounds from the transformations.

The motivation for developing the environment was to make possible the research for a Ph.D. Dissertation. This research project involves the analysis, transformation and resynthesis of inharmonic musical sounds by means of a particular implementation of the Short-Time Fourier Transform called Parshl (Smith and Serra 1987) in conjunction with LPC (Linear Predictive Coding). The hope is that such a programming effort will not only be used to complete the dissertation work but that it can be further used for other applications. It may especially be useful to the composer who whishes to create new sounds with the techniques available on the program to later integrate them in a composition.

The next section describes the software and hardware environment on which the program has been developed. Then follows a description of the actual program.

Description of the System

The program has been developed on a Lisp Machine workstation (Symbolics LM-2) and makes use of

an array processor (FPS AP-120B). For sound conversion a set of custom made 16-bit A/D and D/A converters are being used. The software uses tools borrowed from SPIRE (Speech and Phonetics Interactive Research Environment).

The Lisp Machine

The Lisp Machine is conceptually unlike any other computer (Weinreb and Moon 1981). It was originally developed at the M.I.T. Artificial Intelligence Laboratory as a means of effectively writing, using, and maintaining large interactive Lisp programs. The LM-2 was the first commercially available Lisp Machine, introduced by Symbolics in 1981.

The system software of the LM-2 constitutes a large-scale programming environment, with over half a million lines of system code accessible to the user. Object-oriented programming techniques are used throughout the system to provide a reliable and extensible integrated environment without the usual division between an operating system and programming languages. Zetalisp is the Lisp dialect used on the LM-2, which is closely related to the Maclisp developed in the 1970s, and to the Common Lisp specification.

The main characteristics of the LM-2 hardware are:

- 36-bit processor
- virtual memory
- high resolution black and white display
- color display
- mouse
- dedicated 300 Mbyte disc drive
- Chaos network
- Unibus

At CCRMA there are four LM-2s on the Chaos network. They share a tape drive for permanent storage and are connected to the main-frame computer of the center (Foonly F4) via Ethernet. Through the F4 the LM-2s have access to several printing devices and the D/A and A/D converters. Fig. 1. shows the hardware configuration of the system.

Flavors

Our program, like all the LM-2 software, makes extensive use of objects, a programing style which was first used in the Smalltalk and Actor families of languages.

Object-oriented programming deals with objects, which are instances of types, and generic operations defined on those types. The definition of a type is done by defining the data known to the type and the operations that are valid for those data. Then an instance of that type can be created. Each instance maintains a local state and has an interface to the world through the defined operations. Thus, in object-oriented programming, data and procedures are encapsulated within an instance of the type.

The support of object-oriented programming on the LM-2 is done through a collection of language features known as the Flavor System. *Flavors* are the abstract types; *methods* are the generic operators.

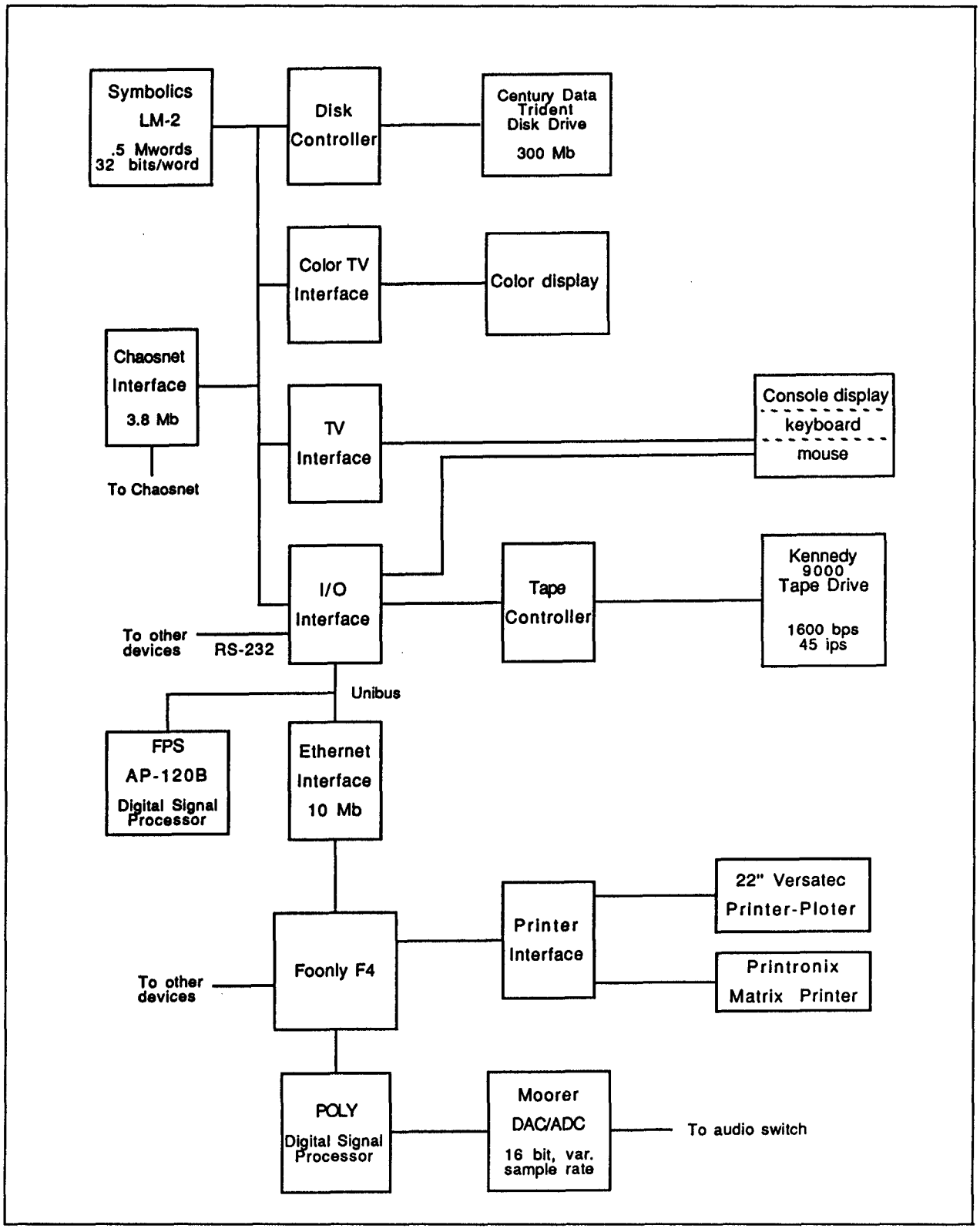

Figure 1. Hardware configuration of the overall system.

The objects are *flavor instances* that are manipulated by sending *messages*, which are requests for specific operations.

The flavor dependencies form a graph structure; they are not constrained to be hierarchical as in some languages that support an object-oriented style. Fig. 2 shows an example of flavor dependencies.

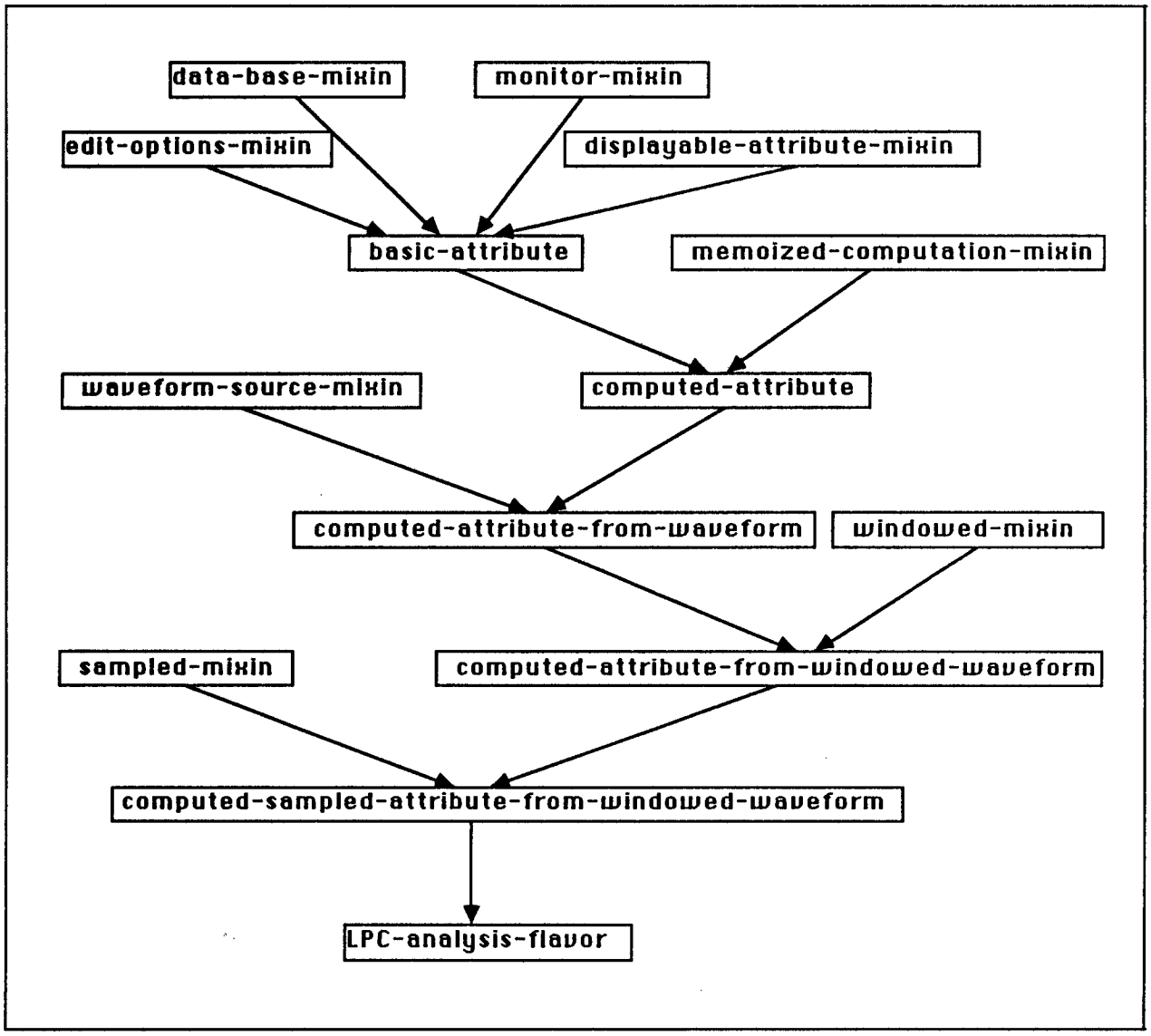

Figure 2. Example of flavor dependencies.

The Array Processor

Connected to the Unibus of one of the Lisp Machines is the array processor (AP-120B). The AP-120B (from Floating Point Systems, Inc.) is a high-speed (167-ns cycle time) peripheral floating-point Array

Processor, which works in parallel with the host computer. Its internal organization is particularly well suited to performing the large numbers of reiterative multiplications and additions required in digital signal processing. The highly parallel structure of the AP-120B allows the "overhead" of array indexing, loop counting, and data fetching from memory to be performed simultaneously with arithmetic operations on the data. This allows much faster execution than on a typical general-purpose computer, where each of the above operations must occur sequentially.

The AP-120B comes with a Math Library which includes over 350 routines covering a wide range of array processing needs. These routines, written in AP Assembly Language, can be called by functions on the Lisp Machine or other programs written in AP Assembly Language. The AP performs arithmetic operations using a 38-bit floating-point format: one exponent sign bit, nine exponent bits, one mantissa sign bit, and 27 mantissa bits. The binary point is always located between the mantissa sign bit and the most significant bit of the mantissa.

The combination of the Lisp Machine and the Array Processor allows one to maintain a high level of both numeric and symbolic processing power, which is very appropiate for our application.

SPIRE

SPIRE is the Speech and Phonetics Interactive Research Environment, which runs on Symbolics Lisp Machines (Cyphers 1985, Kassel 1986, Roads 1983, Shipman 1982). It is a program for manipulating speech signals and computations on those signals interactively. In addition, it can be used as a basis for developing other speech processing systems and can be extended and customized by the user to perform specific tasks.

SPIRE was implemented by David Shipman at MIT in 1982. Since then the program has been modified by many members of the Speech Communication Group of MIT and runs on the last models of Lisp Machines built by Symbolics. The original SPIRE was designed for collecting speech data and looking at transformations of it, but the recent versions have become more general and allow other applications. On the LM-2 the last version of the SPIRE that can be run is 1984 version. There has been a few changes on the Zetalisp from the time of the LM-2s, and SPIRE is only supported for the last releases of the Lisp Machine software.

SPIRE's basic tools can be used on a system for the analysis, transformation and resynthesis of musical sounds. However some tools are very specific to Speech and have to be changed or completely rewritten, and some others that we would like to have are not available in SPIRE. Therefore our program is a combination of parts of SPIRE that have not been altered, others that have been rewritten, plus code written from scratch.

Description of the Program

The program is entirely written in Zetalisp and it makes extensive use of the Flavor System available on the LM-2. The user interface is based on the display system from SPIRE. The signal processing computations use the AP-120B and the collection of array processing utilities that come with it. The control of the AP-120B has also been borrowed from SPIRE.

From SPIRE are taken the concept and part of the implementation of the three basic entities that make up the program. These are: *utterance,* Computation System, and Display System.

Utterance

An *utterance* (a term borrowed from speech and not very appropriate for music) is the basic data structure of the program. The utterance is implemented as an object and usually groups together information related to a single sound, including:

- a pathname where the utterance is stored
- a digital representation of the sound
- a set of computations based on the digitized waveform, called attributes

The acoustic signal is digitized with the A/D converters from the Foonly F4 and then it is stored as the *original waveform* of an utterance on the Lisp Machine. All the analysis, transformations, documentations, or any kind of data that we may think of can be stored on the utterance in the form of attributes.

After a few attributes have been stored in a single utterance, the size of the utterance becomes very big and difficult to handle. It is best to save only the attributes that are not computed (documentations) and the ones that are hard to recompute (pitch trajectories or elaborate transformations). Once we have a synthesized waveform that we want to save, we may store it as the *original waveform* of a newly created utterance.

Computation System

The computation system is responsible for analyzing, transforming, resynthesizing, and generally manipulating utterances. This includes the control of the array processor, and all the signal-processing tools required for the computations.

The control of the array processor has been borrowed from SPIRE and includes an assembler, a debugger, a library of canned AP routines supplied by the manufacturer, and software tools for loading the programs into the AP and transferring data to and from the AP.

Routines for the Array Processor can be written in AP-120B assembly language or in FPS-Lisp. FPS-Lisp is a highly-constrained Lisp subset which compiles into AP assembly language. The FPS-Lisp facility is primarily used to chain together or iterate through sequences of precoded routines which are available in the Math Library. For most of our purposes the FPS-Lisp facility is sufficient and there is no need to write in assembly language.

Signal processing tools are built on top of the low level AP routines and include: Spectrums, STFT (Short-Time Fourier Transform) and LPC (Linear Predictive Coding) for analysis and synthesis, pitch detection, filtering, spectrograms, and a modification of the traditional STFT called Parshl (Smith and Serra 1987). Other tools are available for editing waveforms and their analysis results, and for mixing sounds. Since the program keeps the basic SPIRE characteristics, it is easy to add new computations or modify the existing ones.

The computation results are called *attributes*, which are objects or flavor-instances, and they are computed by having messages sent to them. For example, an FFT would be an instance of the flavor

called "FFT-flavor". The instance would be first created and then it would be computed by sending messages to it with the input waveform and the values of its control variables. An attribute may receive a computing message from the display system, from another attribute which requires its data, or from a specific function call from outside the display system.

Display System

The display system allows a complete and interactive control over the computation system and utterances. It is based on the window system of the Lisp Machine, which is a very powerfull tool for dealing with displays.

The system has three different window levels. The higher one is called the *layout* , the middle one the *display* and the bottom one the *overlay* . The data of the attributes is displayed on the overlays. Fig. 3. shows a typical organization of the display system.

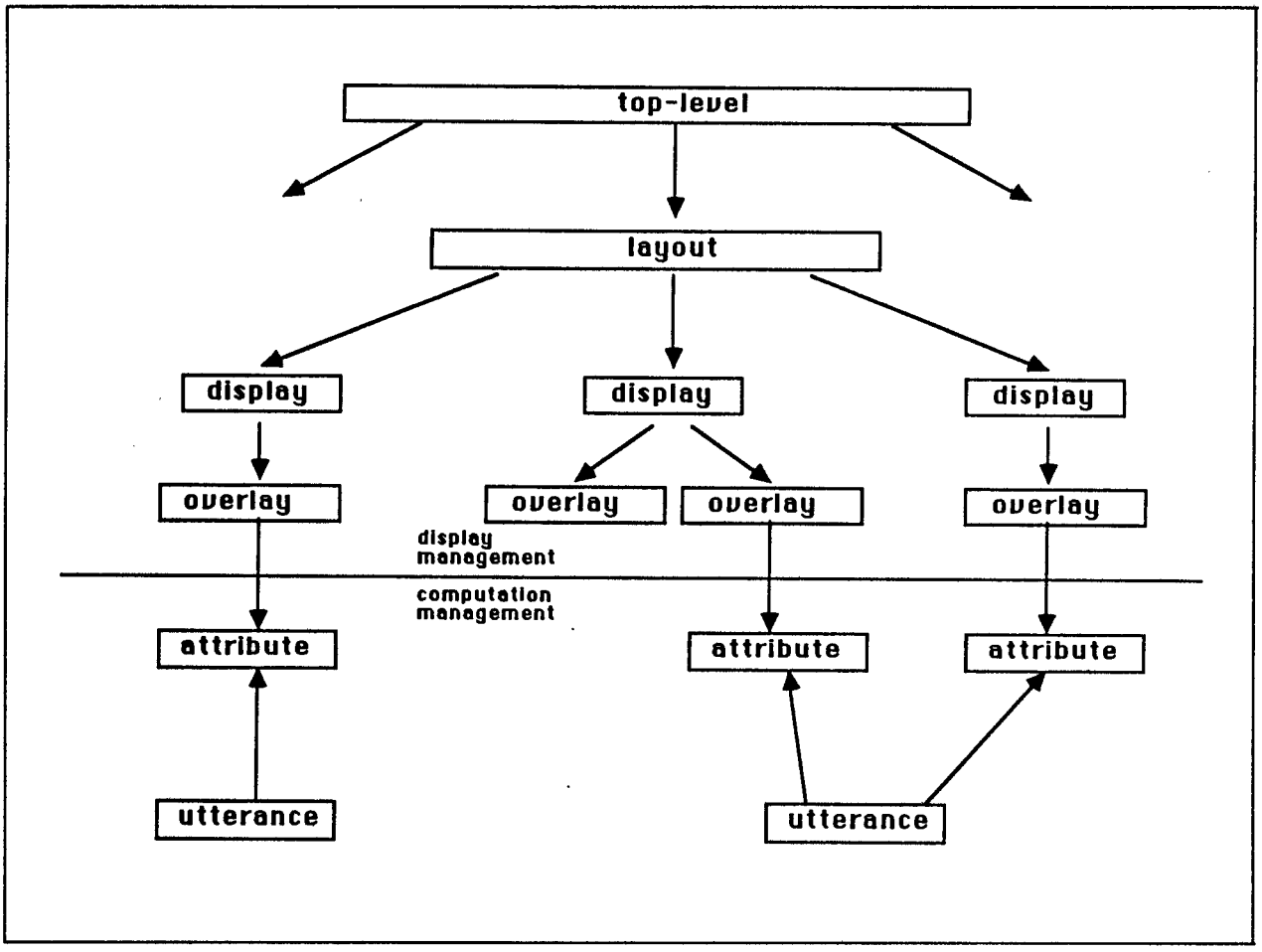

Figure 3. Example of a Display System.

The overlays are the simplest display objects that the program manipulates, and they describe how the values are drawn on the screen. They come in two varieties:

- attribute overlays, which draw the values of an attribute
- background overlays, which draw utterance-independent annotations

The background overlays are mainly used to display the two different kinds of cursors that are available and also for background grids or axis. The overlays are tranparent. Two overlays can occupy the same area of the screen and both will be drawn.

A display is a rectangular area of the screen containing one or more overlays. Unlike overlays, displays are not transparent. Partially covered displays will be hidden from view.

A display contains information which can be accessed by its overlays. For example, we may want to draw two overlays on a common axis. The scale and position of each overlay's axis default to the values stored in the containing display.

The overall screen is managed through layouts. Each layout specifies a collection of displays and their positions. There can be any number of layouts, but only one is displayed at any time. Some layouts have been designed in advance and come with a set of displays and overlays to be used for a specific task. For example there is one layout to study the problem of time-domain windowing of waveforms. Another one is designed to look at different aspects of the analysis results of Parshl. But the normal layouts are called "blank" layouts. On these, the user defines the structure of displays and overlays interactively during every particular session. Fig. 4. shows an example of a layout.

Conclusion

A program has been presented that integrates a set of tools for the purpose of analyzing, transforming and resynthesizing musical sounds. Its front end is an interactive display system from which the user can control the different tools available. Waveforms can be played and displayed in a variety of ways. The results of analyzing these waveforms with different signal procesing techniques may also be displayed. The analysis data is manipulated visually and new waveforms are generated by using the transformed analysis data. These synthesized waveforms can even be manipulated with the same set of tools from which they where obtained.

Work is underway to assert the possibilities that such a set of tools can offer to create new sounds. In particular we want to come up with indications of how to use Parshl and LPC-related techniques to manipulate inharmonic sounds (Serra 1988, Serra 1985, Smith and Serra 1987).

The specific issues that are being adressed in the context mentioned are:

- time domain windowing
- preprocessing techniques
- time varying analysis
- separation of deterministic from noisy part of a sound
- combined use of LPC-Parshl
- sound transformations

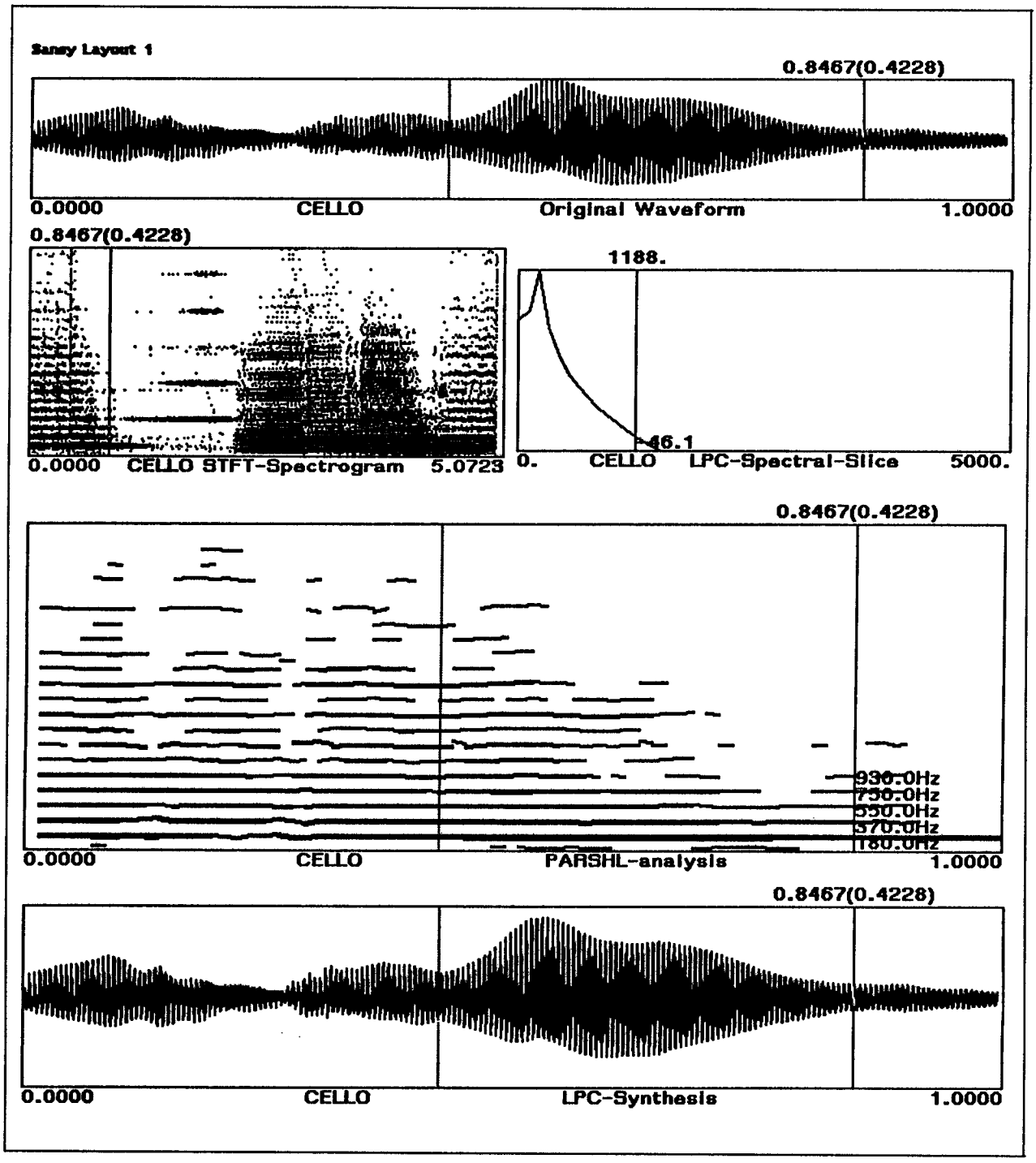

Figure 4. Example of a layout.

Acknowlegments

I wish to thank Tovar, programmer/analyst at CCRMA, for the incredible job of maintaining the LM-2s running in perfect condition and creating very useful tools for them.

References

Cyphers, David S. (1985) *SPIRE: A Speech Research Tool,* Master's Thesis, Dept. of Elect. Eng. MIT. May 1985.

Kassel, Robert H. (1986) *A User's Guide to SPIRE,* Speech Communication Group. MIT. 1986.

Moorer, James A. (1979) "The Use of the Phase Vocoder in Computer Music Applications," *J. Acoust. Soc. Amer.*, vol. 26, no. 3/2, pp. 42–45.

Moorer, James A. (1979) "The Use of Linear Prediction of Speech in Computer Music Applications," *J. Acoust. Soc. Amer.*, vol. 27, no. 3, pp. 134–140.

Roads, Curtis (1983) "A Report on SPIRE: An Interactive Audio Processing Environment," *Computer Music J.*, vol. 7, no. 2, pp. 70–74.

Schafer, Ronald W. and Lawrence R. Rabiner (1975) "Digital Representations of Speech Signals," *Proc. IEEE*, vol. 63, pp. 662–677.

Serra, Xavier (1986) "A Computer Model for Bar Percussion Instruments," *International Computer Music Conference,* 1986, pp. 257–262.

Serra, Xavier (1988) "Analysis, Transformation and Resynthesis of Inharmonic Sounds for Computer Music Applications," in *Overview, Center for Computer Research in Music and Acoustics (Recent Work),* ed. X. Serra and P. Wood, Department of Music Technical Report STAN-M-44, March 1988.

Shipman, David W. (1982) "Development of Speech Research Software on the MIT Lisp Machine," The Journal of the Acoustic Society of America, 103rd Meeting, Chicago, Illinois, 26-30 April 1982.

Shipman, David W. (1982) "The use of the FPS-100 from the MIT Lisp Machine," Research Laboratory of Electronics, MIT, 14 December, 1982.

Smith, Julius and Xavier Serra (1987) "PARSHL: An Analysis/Synthesis program for Non-Harmonic Sounds Based on a Sinusoidal Representation," *International Computer Music Conference,* 1987, pp. 290–297.

Weinreb, Daniel and David Moon (1981) *Lisp Machine Manual,* Symbolics, Inc.