NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Canada

A CAD System for Acoustic Physical Modelling

Christopher Bruce Lea

A Thesis

in

The Department

of

Computer Science

Presented in Partial Fulfilment of the Requirements for

the Degree of Master of Computer Science at

Concordia University

Montréal, Québec, Canada

May 1992

ISBN 0-315-80938-8

Abstract

A CAD System for
Acoustic Physical Modelling

Christopher Bruce Lea

An Audio CAD system is presented that provides users with tools to graphically design sound generation objects. The objects synthesize the acoustical properties of real or imaginary audio instruments using physical modelling. The physical modelling synthesis method is based on inter-connectable resonator modules in hierarchical waveguide networks. Waveform samples are generated using parallel processing hardware to accelerate the design cycle. The system is implemented as a set of MAX objects on a Macintosh IIfx with a network of INMOS Transputers used to drive the waveguide network and a Sound Accelerator board used to audition the resulting waveforms.

In memory of my mother

This work was made possible due to the generous support and facilities provided by the people in the Electronic Music Studio at McGill University.

Table of Contents

List of Figures

List of Tables

1 Introduction

The work presented here discusses the issues related to the physical modelling of sound. It identifies the problems of creating and using models of sound generation systems whose purpose is to derive sounds that are considered acoustically interesting and predictable. A solution is proposed illustrating that it is possible to do this in a modular design environment that allows simple and intuitive constructs to be easily manipulated. Two main subject areas are covered that are considered to be the relevant points of interest to this topic. They are:

(1) the synthesis of digital sound by computer

(2) the conceptual representation of a general synthesis model.

A problem related to physical modelling of sound systems in a digital computer environment is that a user is often required to create new programs to test out a theoretical model. This is due, in part, to specific properties of the model that are not totally incorporated in any single sound generation design system. Another problem is that the majority of available sound generation design systems do not incorporate a user interface that allows easy and informative manipulation of the inherent ideas of the system being designed.

The requirements of a user of a design system to model sound can be stated as follows:

1. A means of visualizing (graphically representing) the sound that will be created.
2. A method of breaking the sound creation process into a series of sub-processes that are easily manipulated and can be viewed as complete entities on their own.
3. A method of testing partial or whole designs to see if the system behaves as expected. The designs are tested rather than verified because the available verification tools are weak and because the basis of evaluation is more often qualitative rather than quantitative.
4. A synthesis method that does not restrict the range of sounds possible. A general design system should not place a restriction on the models being created.
5. A language (set of input values, set of triggers to a model) that can be used to drive the models.

The work in the following chapters presents a proposal for a system that attempts to meet the requirements of a designer of sound generation models. The contribution by the author is focussed on two aspects: (1) the design of the

graphical computer aided design (CAD) environment, and (2) the parallel computation engine.

An overview and comparison of well-known sound synthesis models is provided in Chapter 2. The concept of a waveguide network is then defined and presented as a viable alternative to the other synthesis models.

A method of graphically representing and manipulating sound generation modules is presented in Chapter 3. In the discussion, the general design cycle is examined along with a proposal presenting what is needed by and satisfactory to a user during the design cycle. A software specification for a system to implement the two main tools for this method is then presented.

The implementation of a prototype of the generalized design system is presented in Chapter 4. The hardware platform is presented with the mapping of the appropriate modules from the software design presented in Chapter 3 to this hardware. This includes a discussion of the increase in throughput that is possible by employing parallel processing in the computation of the results (the digital representation of the sound wave).

2 Sound Synthesis

Sound synthesis by computer is the algorithmic generation of digital waveforms that represent the time-varying pressure of the air being moved by an acoustical wave. These digital waveforms can be converted to analog signals, and broadcast through an amplifier/speaker system to be auditioned.

The primary domains of application of sound synthesis by computer are music and speech. Some examples of the many disciplines using synthesized sounds are Composition, Cognition, Human-Machine Interaction, Psychoacoustics, Psychomusicology, and Telephony. These domains and disciplines require that sound generation be computationally efficient, and that the algorithm specification that describes the sounds be relatively easy to compose.

This chapter surveys several existing methods of sound synthesis by computer, and contrasts them by their computational and compositional requirements. The methods discussed are additive synthesis, non-linear models, and physical modelling ([8,23]).

2.1 Additive Synthesis

Additive synthesis is a process where basic sound waves are combined to create more complex waveforms. This method is derived from a theory by Fourier in which he describes complex periodic waveforms as combinations of harmonically related sinusoids. In additive sound synthesis, the complex waveforms desired are created from sets of harmonic components, or partials.

Fourier's theorem states that if **f**(t) is a waveform with a period of T seconds ($\forall$ t; **f**(t) = **f**(t + T)), then

$$\mathbf{f}(t) = \Sigma_{k\epsilon[0..\infty]} A_k \sin(k\mathbf{w}t+\phi_k)$$

where

- A_k is the amplitude of the k^{th} sinusoidal component of **f**(t).
- **w** is the fundamental frequency of the waveform (= $2\pi/T$).
- ϕ_k is the phase offset of the k^{th} sinusoidal component of **f**(t) (relative to sine phase).

If the only interest in sound synthesis were in the production of non-varying periodic waveforms, then a sound synthesis system could pre-generate one period of each of the desired waveforms through the specification of a set of input parameters. An efficient table look-up method could

then be utilized to produce any version of the waveform using frequency, amplitude, and phase as the parameters to the look-up method.

This method, while being able to produce qualitatively rich waveforms, does not support the independent temporal evolution of harmonics that appears in all naturally occurring sounds. Simple spectral changes, such as the rapid decay of higher frequency components relative to the lower frequency components observed in most naturally occurring sounds, for example, are lost in this method.

To permit time-varying waveforms, the spectral components can be represented by independent functions of amplitude, frequency, and phase, rather than simple value parameters. The output from these functions at time t would then be used as parameters to an oscillator (a simple sine table look-up method), and then the resulting sine components would be added together to produce the waveform at time t. The problem with this method is the need for a large data set for each function to describe the waveform accurately. Research in psychoacoustics, however, has shown that the number of components required to describe a waveform can be reduced while producing no objectionable effect on the resulting sound ([22]).

Another reduction which would have more effect involves the data sets of the functions defining the envelope of the spectral components. Here, a piece-wise linear approximation to a sound's time varying power spectrum is taken. These approximations are then stored as coordinates of the break points of the functions. Studies of orchestral instrument tones have shown that as few as five to seven line segments per envelope are needed to generate tones which are difficult to discriminate from tones generated using detailed amplitude and frequency functions ([44]).

In summary, additive synthesis is a powerful and general technique for producing waveforms, but it normally requires a large set of control parameters to specify a set of waveforms to be used in any controlled setting. Further, these parameters are not derived by simple observation, but are usually extracted through the analysis of sounds by short time spectrum measurement techniques. Thus, though powerful and general, additive synthesis requires much analysis and input from the composer, and is therefore somewhat limited in application as a good sound synthesis alternative.

2.2 Non-linear Synthesis

Non-linear (or distortion) synthesis uses aggregate non-linear functions with oscillators to derive output with many more components than the number of oscillators used. These non-linear functions are usually applied to only one of the inputs allowing for control of the sounds using only a few parameters. This enables the user to develop rich sounds while requiring few calculations for the parameters to be made. Two of the more widely accepted subclasses of Non-linear Synthesis are frequency modulation and non-linear waveshaping.

2.2.1 Frequency Modulation

Sound synthesis by Frequency Modulation (FM) is a method of producing complex audio spectra first introduced by John Chowning ([6]). In its simplest form, FM is achieved by adding the output from one oscillator to a carrier frequency to be used as the time-varying frequency parameter of another. The frequency **f**(t) at time t can be stated as

$$\mathbf{f}(t) = f_c + \delta\cos(2\pi t f_m/R)$$

where

f_c is the carrier frequency

δ is the peak frequency deviation from the carrier frequency

f_m is the modulating frequency

R is the sampling rate.

The waveform **y**(t) derived by applying such a function to the frequency parameter of an oscillator of peak amplitude A is given by

$$y(t) = A \sin(2\pi t f_c/R + \delta/f_m \sin(2\pi t f_m/R)).$$

It can also be stated for clarity that the components in such a waveform are given by the series $f_c \pm f_m$, $f_c \pm 2f_m$, $f_c \pm 3f_m$.... This is seen by examining the trigonometric identity that states that

$$\sin(\theta+\alpha\sin\beta) = J_0(\alpha)\sin\theta + \Sigma_{k\epsilon[1..\infty]} J_k[\sin(\theta+k\beta)+(-1)^k\sin(\theta-k\beta)].$$

where $J_n(x)$ is an order n Bessel function of the first kind evaluated at point x.

FM is not restricted to using a single oscillator as input to another. Many combinations of modulating frequencies are often used from cascades of oscillators (in series), to using a simple form of additive synthesis to produce a complicated waveform as the frequency parameter input to the oscillator. It is also worthwhile to note that if the value of the modulating frequency parameter is large

with respect to the carrier frequency, some of the resulting components will be negative and others will be at frequency rates higher than the Nyquist frequency; the Nyquist frequency is defined to be the frequency at one-half the sampling rate. Frequencies in waveforms that are above the Nyquist frequency are reflected around the Nyquist frequency such that their perceived frequency is indistinguishable from a lower unreflected frequency; this is known as aliasing. The perception of the negative and aliased frequency components will interfere with the other components in a manner that is difficult to predict, often producing interesting, if not unobjectionable, results.

The number of calculations required for producing FM is much lower than for that of Additive Synthesis, and the sounds derived from FM systems prove to be extremely rich. For these reasons, a number of FM based synthesizers have been marketed, and have enjoyed great commercial success. It must be stated, however, that to manipulate the structures necessary for FM and to predict exactly what sounds will be derived is not an obvious task.

2.2.2 Waveshaping

Waveshaping can be described as the class of sound synthesis derived by the post-manipulation of a waveform.

This is usually done using a non-linear function g that is composed with a waveform function f (representing the oscillator). The most popular form of waveshaping uses the output of oscillators as input to a function g composed of a set of Chebyshev polynomials. Their popularity might be seen by examining the relation

$$T_k(\cos\theta) = \cos(k\theta)$$

where $T_k(x)$ is the kth Chebyshev polynomial. Stated in simpler terms, this means that the composition of the kth Chebyshev polynomial with a cosine waveform produces the kth harmonic of that waveform. A waveform having predictable harmonic components can be derived by using a function g(f) composed of Chebyshev polynomials as follows:

$$g(f) = \Sigma_{i\epsilon[1..n]} w_i T_i(f)$$

where the w_i are the weights we wish to give to each harmonic and the f is the output of an oscillator having a frequency of the desired fundamental. If the desired harmonics are known beforehand a look-up table containing the weighted sums of the polynomials for each possible value of cosθ can be pre-produced. This will reduce the number of computations necessary. This method is not easily adaptable to producing time-varying sounds and has not enjoyed the commercial acceptance of FM.

2.3 Physical Modelling

Physical modelling is the process of examining the physical properties (frequency response, etc.) of known devices (instruments, resonance systems, etc.), and replicating their behaviour such that certain input will produce results that exhibit a similar set of output properties of a known device. This is usually accomplished by taking the output from a system modelling an excitation source, and passing it through a resonance system. The most common synthesis method used for physical modelling is subtractive synthesis. Another method, taking a radically different approach, uses waveguide networks. These two methods are discussed in the following sections.

2.3.1 Subtractive Synthesis

Subtractive synthesis is based upon the process of passing a harmonically rich signal through a time-varying digital filter. The purpose of the filter is to remove undesirable frequency components from the signal to produce the desired waveform. Generally, the source for the signal comes from either white noise or a pulse generator, both of which are rich in harmonics and are simple to produce.

Unfortunately, the computational costs of producing a time-varying digital filter are enormous. Even when implemented as infinite impulse response filters, they require continual recalculation as the twc main parameters, bandwidth and centre frequency, are varied, since the behaviour of a time-varying filter must be specified, and therefore interpolated, in the frequency domain and not the time domain. Subsequently, the mapping between coefficient variables of two time domain filters is not a simple task.

Another problem with time-varying digital filters is that their stability is not easily predicted during their migration from one frequency domain to another. Thus, in addition to the calculations necessary, checks must be performed on the parameters to ensure that the poles of the new filter do not wander outside of the unit circle.

These problems are not insurmountable, however. Digital filter theory has been well researched, and therefore, users of such subtractive synthesis systems will have access to a multitude of material to assist them in the design of their "instruments".

The standard use of computationally intensive digital filters for the resonance systems in Physical Modelling

detracts from this otherwise conceptually attractive methodology.

2.3.2 Waveguide Networks

A new theory using Waveguide Network Filtering put forth by Julius O. Smith ([33,35,37,38]) demonstrates a different approach to filtering, which is more efficient and easier to control. A waveguide can be thought of as a weighted undirected graph where the weights on the waveguides (bi-directional edges) represent the resistance and delay that a travelling signal will encounter p:ssing through a medium. The junctions of the graph are constructed so as to produce lossless signal scattering in a manner that would preserve the total stored signal energy in the network. A hierarchical waveguide network is a view of a waveguide network that allows the manipulation of sub-networks as if they were basic junctions ([10]). This allows the encapsulation of characteristics in a sub-network. To construct a full network with a desired effect from sub-networks is similar in theory to creating a fully connected graph out of a set of disjoint graphs. Of course this is more complicated than simply plugging an edge of one sub-network into a junction of another, but it can be shown to be almost that simple through an informal understanding of the functioning of waveguide networks.

Conceptually, waveguides represent the medium through which a sound travels, and the junctions of a waveguide network represent the points of interaction between these waveforms. In instrument modelling, junctions represent the resonance chambers of instruments, while the edges represent the delays a signal encounters while travelling through a medium. Furthermore, the edges represent the directed manner in which these signals travel from one resonance system to another.

More formally, if the signal travelling through a system is given as pressure, then the instantaneous power IP of a given sample is $IP_s = A_s^2/Z$; where A_s is the amplitude of the sample and Z is the impedance of the medium through which the sample is travelling. If the sum of the instantaneous power throughout the system is taken, then this represents the total stored signal energy of the network. A lossless network preserves total stored signal energy.

The main computations within a network take place within the junctions where the scattering of the signal occurs. This is as defined below:

Consider a parallel junction of N lossless waveguides of characteristic impedance Z_i (characteristic admittance $\Gamma_i = 1/Z_i$). If the

pressure waves entering the junction are given as P^+_i, $i \epsilon [1..N]$, the pressure waves leaving the junction are given as $P^-_i = P_j - P^+_i$, where P_j is defined as the resultant junction pressure given by $P_j = 2(\Sigma\Gamma_i P^+_i)/(\Sigma\Gamma_i)$ (see Figure 1).

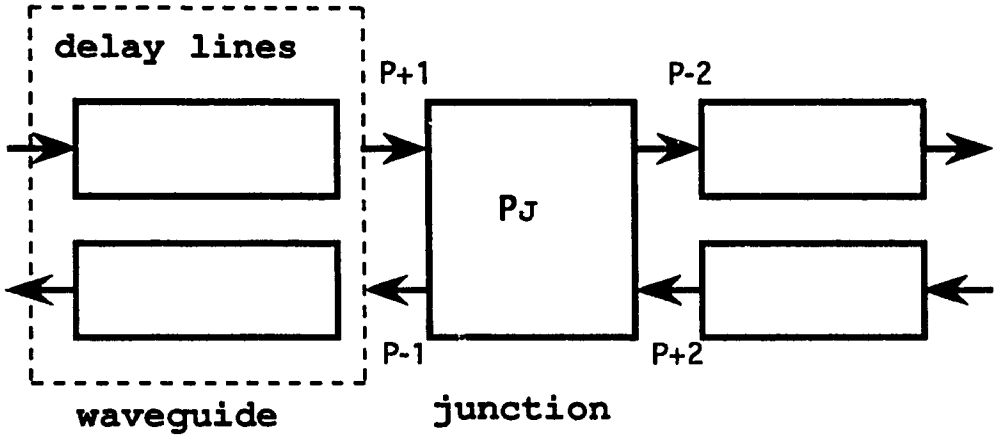

Figure 1 A junction with two waveguides.

This represents the calculation needed at each junction in the network. The time complexity C of the calculation at a junction is found to be

$$C = N\beta + (3N - 2)\alpha$$

where the elementary operations considered are

α floating point additions, and

β floating point multiplications.

The factor for the additions can be reduced when it is taken into account that the denominator for P_j does not have to be calculated for each computation cycle. This can save a

number of steps, but the total junction pressure will have to be recalculated if the admittance of an incoming waveguide's pressure value changes. The admittance will be changeable if time-varying changes to the network are to be included. Generally however, such changes are not performed often, and the computational price is not a heavy one to pay for the added bonus of simple control in a time-varying system.

The benefits of constructing recursive digital filters from lossless waveguide networks are described in Smith ([33]) and summarized here:

(1) the scattering junctions can be made time-varying without altering stored energy,

(2) an "erector set" for lossless networks is obtained, allowing any number of branches to be fitted together in any desired configuration (with changes allowed in real-time),

(3) limit cycles and overflow oscillations are easily eliminated, regardless of interconnection,

(4) an exact physical interpretation exists for all signals in the structure (this is difficult to achieve simply with standard digital filters), and

(5) the implementation is computationally efficient.

These closed structures of waveguides and junctions, by themselves, represent a lossless waveguide network. Losses are introduced into waveguide networks to properly model the effects of such things as I/O with the system and a medium's resistance to action within the real world. The points of I/O can be represented by edges where one of the directions on the edge is, in effect, turned off to produce sinks and sources of information. The calculations within a waveguide become ones which represent losses introduced by way of gain factors of less than 1 and/or low-pass filters with frequency responses strictly bounded by 1.

Waveguides must also be tunable to have specific harmonic properties. The frequency response **fr** of a waveguide is directly related to its delay length of **D** samples, which can only be an integral value, and the sampling rate **R** by **fr** $\propto$ **D/R**. To be able to "tune" the waveguides, small all-pass filters are introduced and applied to the data stream coming into each end of the waveguide. The parameters of these filters may be controlled by the user.

2.4 Summary

One of the main problems encountered by a user attempting to perform any form of sound synthesis is that

the causal relation between the specification of the system to produce the sound, and what the actual result will be is usually not obvious. The standard techniques for sound synthesis demand a good understanding of digital signal processing in order to produce intelligent and interesting results, but normally, the people who are interested in sound synthesis are not necessarily interested in the field of Electrical Engineering. Another more insidious problem is related to the actual time required for the computation of complex waveforms; often calculations take minutes, not seconds, to perform. This, coupled with the fact that much sound testing is required before a satisfactory result is obtained, naturally augments the level of frustration experienced by the user during the design cycle. Thus, a sound synthesis system requires that its design and output results be conceptually simple to understand and easy to manipulate.

Traditional physical modelling techniques are conceptually simple to understand, but most unfortunately require knowledge and use of computationally inefficient digital filters. Thus, to successfully model sounds with a natural quality under such systems, either a high speed computation engine, or a simpler, more efficient method of generating the time-varying waveforms is required.

Filtering using waveguide networks provides a simple, efficient sound synthesis technique, while affording a conceptually natural representation of sound sources.

3 Audio CAD

In this chapter a computer-aided design system for sound synthesis using waveguide networks is defined. The goal of this system is to simplify the design process in the creation of sound systems while not restricting the range of systems or sounds possible.

After defining the specification of a desired system a typical design cycle for a designer can be as follows. First the system's specification is translated into its basic elements and their interactions with each other. Next, the designer designs and tests these basic elements. Finally, these basic elements are tested within the context of the global design.

The testing of a system is done using a set of inputs set up by the designer either statically or interactively and then by examining the results. This examination can be through aural inspection or by using one of several static analysis tools available for digital waveforms. The cycle is repeated until the system as a whole meets with the designer's specification. The stage for the derivation of the basic elements from the specification is considered an integral part of the design cycle because the elements

needed to satisfy the specification may become clearer or change after a pass through the cycle.

The input needed to drive such a system must be easily interpreted by the system as well as easily manipulated by the user to allow the user correct utilization of the sound synthesis module that has been created. Furthermore, if the system is to be interactive as well as allow for stored values to be entered, the language that represents the input values should be easily translated and interpolated into the possible set of actions that can occur as a result of its use.

Typically, CAD systems for audio are used within the domain of audio engineering. They are created to aid the recording engineer with the processes of recording, editing, remixing, and remastering segments of audio. One general system that performs all of this is the software and hardware provided in a package entitled Soundtools created by DigiDesign. Other systems have uses that are more specific by having interfaces and tools that allow the manipulation of audio segments that are also to be included into film and video. Two examples of these systems include the Syntrex DPR44 system and the Sigma system by Digital Audio Research. The DPR44 system allows for graphical objects to be related to segments of audio. Not only do

these objects have shape but they also contain class information that aid in the task of relating the sound segments to other segments such as frames of film. The DAR SIGMA system also offers a graphic interface but this software is usable only with proprietary hardware.

CAD systems for sound synthesis have been slower to come to the markets. Most have been set up as language and input file based systems. Systems of this type can be traced to the MUSIC-N languages of sound synthesis such as csound from the Massachusetts Institute of Technology. There have been, however, some attempts to create graphical interfaces for these types of systems. Other more complete CAD systems for sound synthesis have been towards creating control languages for synthesis systems by representing the control constructs as graphical objects. One successful application of this has been with MAX (see section 4.2). This system has been applied and ported to at least three different hardware platforms. The original platform is for the musical workstation environment at the Institut de Recherche et Coordination Acoustique/Musique (IRCAM) in France. The other two platforms are for the more general Macintosh II and NeXT lines of computers.

In the following sections, the design of a graphical oriented audio CAD system based upon waveguide networks is

described. The discussion includes the fundamental points for achieving the goal of a design system that incorporates a "user-friendly" design cycle.

3.1 Module Definitions

Systems are best conceptualized if they are broken down into self-contained well-defined modules. This is just as true for sound systems as it is for any other system studied by humans. Therefore a desirable property of a design system for sound synthesis is the ability to encapsulate the concepts of a basic structure into a single module. Consider, for example, the modules that could be used to construct a model of a piano.

A piano has many strings, all of which are connected to a soundboard that exhibits a frequency response. To simplify the design a user could model a single string (Figure 2) and then attach the necessary strings to a module that represents the soundboard (Figure 3) ([9]). The string denoted in Figure 2 is represented by two waveguides one of which is a self-loop and an excitation source all attached to the same junction. The string is excited by a Hanning[1] pulse at a point in the string that represents where the

---

[1]Each entry w_j of a waveform in a lookup table of size N representing a Hanning pulse is defined by:

$$w_j = 1/2[1-\cos(2\pi j/(N-1))] \quad ; 0 \leq j < N$$

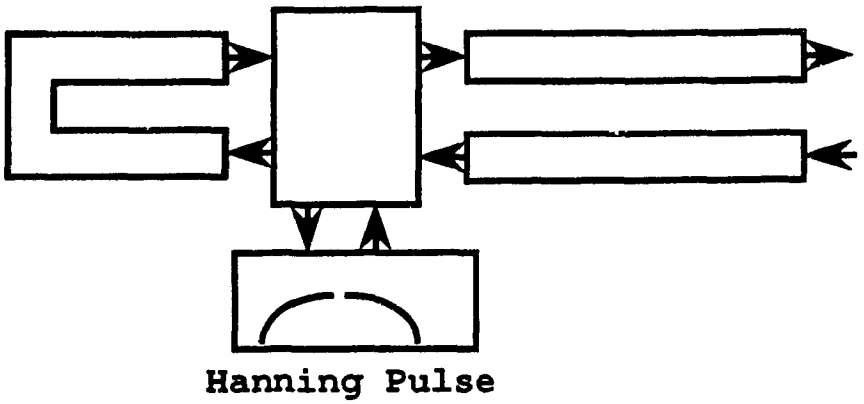

Figure 2 A waveguide subnetwork representing a string.

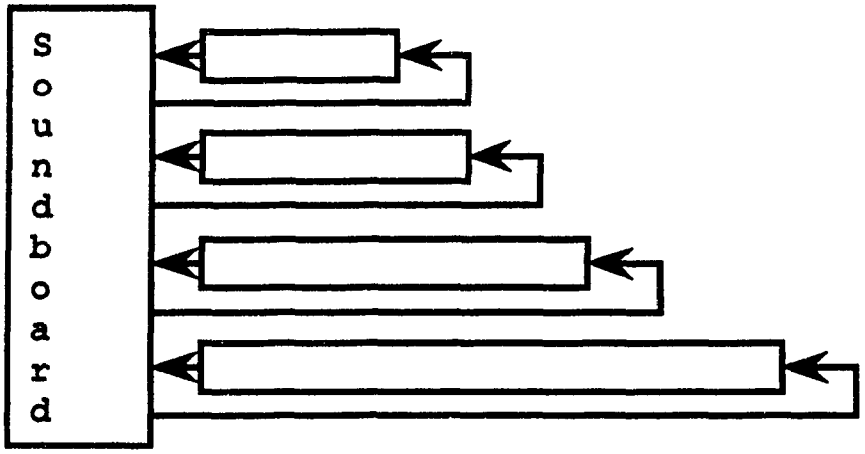

Figure 3 A waveguide subnetwork representing the soundboard of a piano.

string would be struck. This models the action of a string by allowing the returning pressure wave to interfere with the action of the excitation source while the action of

striking the string is taking place. The excitation would simply reflect the values of the pressure flowing through the junction when it was not in the process of exciting the string. The effect of this would be that the two waveguides would seem to be one longer waveguide.

An added benefit of modularity in such a system would be the possibility for reuse of the modules in more than one design; using the same strings in the model of a guitar, for example. To achieve proper modularity in a system the interface to any module must be well-defined and simple to understand. Even though waveguide networks seem to fulfil that purpose quite effectively, a more complete system should allow for a higher level of abstraction than the simple coupling of waveguides and junctions.

The basic elements of any sound system based upon physical modelling can be generalized as being a set of excitation elements and a set of resonance elements through which the sound pressure waves travel and interact. The control of the system, and therefore the point of entry for the input language, is thrcugh the excitation elements. Without loss of generality, the global resonance chamber can be conceptualized as a "room" and its excitation elements as a set of "players" performing upon "instruments". The input to the players can be conceptualized as a score. The "room"

will consist of a set of resonance chambers through which the resultant sound pressure waves will travel, two (for stereo) of which will be labelled as the points for the sound to leave the partially closed system. An instrument can be further broken down into having a set of excitation elements that will react to the same set of input values in a possibly distinct manner from other instruments and a set of resonance chambers. If this train of thought is continued, then an instrument can contain, within its set of excitation elements, a set of instruments. This recursive definition can be extended to resonance chambers as well, allowing a user to define a set of modules that have distinct and clearly defined rules and are easily conceptualized and manipulated.

For example, consider an instantiation of a concert hall as being composed of a set of players performing upon instruments within a room. The room and instruments within it all have a specific set of reverberation properties. An instrument in the hall, the piano from the previous example, is composed of a set of strings each distinctly reacting to a set of inputs caused by hammers and pedals each connected to a common well-defined resonance chamber. This resonance chamber, the soundboard of the piano with a specific set of frequency responses, is in turn sending its output into the concert hall to be mixed with the other instruments and the

returning signals (feedback) of the previous output. The feedback of the concert hall, along with the waveform resonating in the soundboard, will also have an effect on the strings of the piano through sympathetic vibrations. It is this natural feedback, which is easily and naturally defined in waveguide networks, and which is absent in most models of sound synthesis.

In summary, the definition of an instrument contains the definition of its manner of excitation and its resonance properties. The definition of a player refers to the section of the score to follow and the instrument to perform upon. The definition of a room contains its players and its resonance properties. Therefore, it can be stated that the specification of a room completely defines a system.

3.2 Conceptual Design

The conceptual design of the system is presented in the following sections. The presentation is given in a top-down manner starting with the Waveguide Editor module, which represents the entire system.

3.2.1 Waveguide Editor

The design system accessed directly by the user is called the Waveguide Editor. The Waveguide Editor consists of modules that perform the necessary operations for the user to create and edit sound synthesis networks, and to have them tested using input from a MIDI file (score) or from an interactive input device. A brief overview of these modules and their interactions is given in the following paragraphs. They are detailed more fully in the sections that follow.

As stated above, the specification of a room and the players within it completely defines a unique sound synthesis network. Thus, to specify a network, a room description is created. The module that is used to create a room is the Room Network Editor. Contained within a room is a system denoting its resonance characteristics and the definitions of the players who are to perform in the room.

The Instrument Network Editor is used to create the players' instruments and define their resonance properties. The Instrument Network Editor is accessed through the reference to it by the player. The definition of the resonance properties of the instrument may contain subnetworks. The Subnetwork Network Editor is used to create the subnetworks referred to in the instruments. The resulting definitions created by the Network Editors are high-level descriptions of room, instrument, subnetwork elements.

Once these elements and their interactions are defined, the resulting descriptions must then be transformed by a Loader module into a valid waveguide network consisting of waveguides, junctions, and excitation nodes.

Once loaded, a network may be performed. The module used to control the performance of a waveguide network is the Conductor. The Conductor dispatches control data from the score and/or any attached interactive input devices to the appropriate excitation nodes within the network.

While playing, the resulting waveform of the performance is "heard" by the Conductor and passed to a Sound Manager module. The Sound Manager may then either transfer the data to a Digital-to-Analog Converter so that

the results can be auditioned interactively, or write the data to a sound file for storage.

The requests that the Waveguide Editor accepts from the user and transforms into actions by the modules are given as follows:

Edit The edit command is a request by the user to edit a room. This command causes the Waveguide Editor to invoke a Room Network Editor with the name of the room as its argument. If the room already being edited is different from the one requested it is closed.

Load The load command is a request by the user to set up the referenced network to be played. If the user does not specify a room the current room being edited is used otherwise the specified room is loaded into the Room Network Editor.

Play The play command is a request by the user to generate sound using the currently loaded network. The user is prompted for a score file and the start and stop time limit for the performance.

Stop The stop command is a request by the user to discontinue the current performance.

Quit The quit command is a request by the user to exit the system. Any room currently being edited is closed before exiting.

3.2.2 Network Editor

Three Network Editors allow the user to edit the system at each of a network's conceptual levels: room, instrument, and subnetwork. The editors share facilities that manipulate the objects of the network as described in Table 1. The Network Editors are invoked by the Waveguide Editor at the request of the user.

Object\Editor	Room	Instrument	Subnetwork
Waveguide	✓	✓	✓
Junction	✓	✓	✓
Excitation		✓	
Player	✓		
Subnetwork	✓	✓	✓
I/O Port	✓	✓	✓

Table 1 Editable Network Objects.

The Waveguide, Junction, Excitation, and I/O Port objects are the base, non-divisible objects within the system. The modularization of the system is done throuqh the Player and Subnetwork objects.

The I/O Ports designate the specific entry and exit points for the data flow within a given Network Editor so

that the input and output to the given subsystem can be fully defined. The room editor has two output ports for the stereo output. The instrument editor has a single port designating its exit point. The instrument editor does not have a port for the entering data to it because the data to the instrument is from the Excitation objects defined within the editor. These Excitation objects receive their designated input from the section of the score that pertains to them, as defined by the Player object that refers to the instrument. The subnetwork editor is restricted to having a single input and a single output port to represent its data flow coupling.

The Player object references a Network Editor at the instrument level. The Player allows the association of a given instrument at a given location within a room with a specific set of instructions to react to. An analogy describing the use of a Player object can be to have two players who are both playing the same type of instrument, the violin, but each player is reading from a different section of the score. Another analogy could be to have two players who are each playing two distinct instruments but both players are playing the same section of the score.

The Subnetwork object references a Network Editor at the subnetwork level. The Subnetwork object allows the

encapsulation of a resonance system by associating itself with a network definition consisting solely of Waveguides, Junctions, and possibly further levels of subsystems using Subnetwork objects.

For each of these objects to be visually manipulated in a Network Editor, they need a descriptive shape and a means of describing the coupling to other objects within the same editor. They must also allow the user access to an object's variable set of parameters. The Network Editor then has four types of data managers to deal with each of these points of information as well as the global editing window. They are

Window Manager: The operations of this module are devoted to the manipulation of the editor's windowing environment, which is general to all editors. The operations include scrolling, resizing, opening, closing, saving of contents, etc...

Icon Manager: The operations of this module are devoted to the graphical manipulation of individual network objects within a window. This allows the association of an object's appearance to its meaning.

Port Manager: The operations of this module are devoted to the manipulation and control of the connections between graphical network objects within a window. It validates all attempted connections between

two objects. Table 2 lists the valid connection map that is used.

Parameter Manager: The operations of this module are devoted to the editing of the variable parameter data pertaining to each object within a window. The Junction and I/O Port objects do not need to be manipulated by this manager because they do not have any variable parameters other than their couplings, which are handled by the Port Manager.

	Plyr	Jnctn	Wave	Exctn	Sub	I/O
Plyr	⌀	✓	⌀	⌀	⌀	o
Jnctn	✓	⌀	✓	✓	o	i
Wave	⌀	✓	⌀	✓	i	o
Exctn	⌀	✓	✓	⌀	⌀	⌀
Sub	⌀	o	i	⌀	o→i	x=y
I/O	o	i	o	⌀	x=y	⌀

Legend::
- ⌀ ▲ illegal connection
- ✓ ▲ valid connection
- i ▲ input port of Subnetwork or I/O Port designated as input port
- o ▲ output port of Subnetwork or I/O Port designated as output port

Table 2 Rules for logical port connections in a Network Editor.

3.2.3 Loader

To produce waveforms, the separate conceptual specifications of the room and its players as defined by the user in the Network Editors, must be first transformed into a fully-connected waveguide network. The Loader, invoked through the use of the Load command, performs this transformation. (The Loader could be viewed as a parser of the conceptual specifications, and the specifications viewed as a language.)

The main operation of the Loader is to create executable instantiations of the nodes of a waveguide network through the substitution of network editor objects for their corresponding nodes. The Player and Subnetwork objects are repeatedly replaced by their constituent subnetworks, i.e., Player objects are replaced by their instrument definitions, and Subnetwork objects are replaced by their corresponding subnetwork definitions. The I/O Ports that are found in the corresponding instrument and subnetwork definitions are replaced by direct couplings to the specified network nodes. Care must be taken by the Loader such that an associated set of subnetwork definitions are not recursive and that the resulting network is fully connected. The latter restriction is not always necessary, when there are isolated networks not connected to the I/O

Ports, but would result in a network definition that contained unreachable nodes. Figure 4 demonstrates the substitutions for a room with a single Player. The associated instrument contains the two types of valid connections for an Excitation object plus a Subnetwork object that refers to a minimal subnetwork definition.

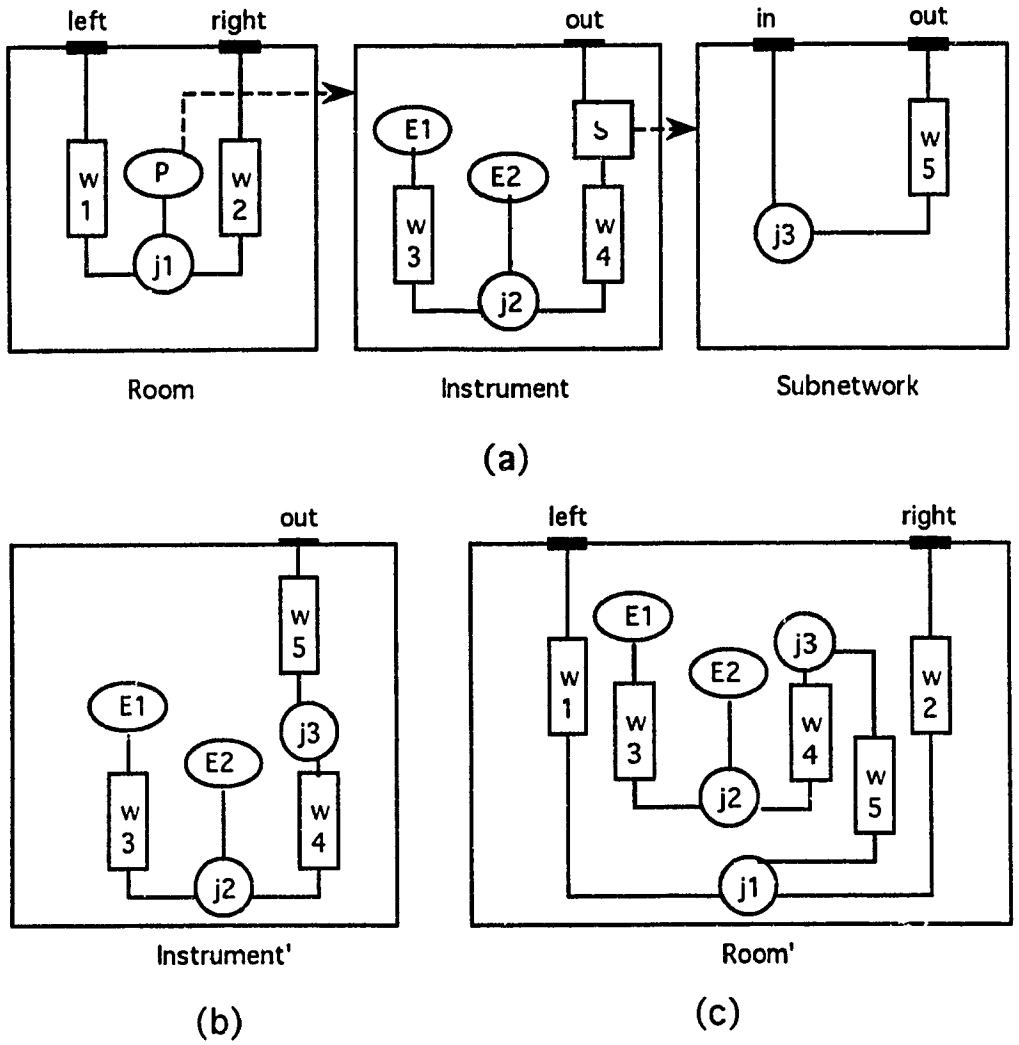

Figure 4 Example of a derived network through substitution of editor definitions. (a) original room, instrument, and subnetwork (b) derived instrument, (c) derived network.

3.2.4 Conductor

The purpose of the Conductor is to synchronize the actions of the waveguide, junction, and excitation nodes in a waveguide network, and the data being passed among them. A connection scheme is defined that is used to aid the Conductor to perform this synchronization. The two main data structures, which the Conductor manipulates, are the waveguide network and the score.

3.2.4.1 The Score

The score is given to the Conductor as a parameter to the Play command as issued by the user to the Waveguide Editor. The structure of the score is given as a set of Musical Instrument Digital Interface (MIDI) channels ([19,21]). Each channel represents a temporally ordered set of MIDI events. A MIDI event's structure consists of an activation time, given as the difference between its activation time and that of the preceding event, and a MIDI control command.

The advantages of using MIDI as the control language are:

(1) there are many software packages in existence that allow graphical editing of MIDI data,

(2) the input devices that communicate using MIDI data are numerous and diverse,

(3) it is flexible enough to allow meaning to be derived from a number of control statements that can be customized as the need arises,

(4) the language is well-defined by a standardization committee; included in the standardization is a format for file storage ([20]).

To be used as the control language for a waveguide network, MIDI commands must be interpretable by the excitation nodes within the network. The **Note On** command, for example, can be easily interpreted in the synthesis environment. The command consists of note-velocity pairs whose corresponding meaning is given as a fundamental frequency whose peak amplitude is relative to the velocity on a scale of 1-127. There exist, however, some desired actions for which there is no direct correspondence of MIDI commands to the action. Consider, for example, the actions required to emulate the implied phoneme actions required of a set of voiced words. MIDI allows for this sort of information to be coded through the use of user defined commands, but there does not exist a system that can automatically produce the necessary interpretations.

3.2.4.2 Time and Synchronization

The concept of time that is used at the waveform sample level is not based upon a real-time clock. The time given by the sample clock is based upon the number of samples that have been processed and the sampling rate to be used to play the waveform. For example, if the sampling rate R = 48000 samples/sec and there have been 1600 samples generated then the sample clock (relative to zero) should state that the current time t = 1600 samples / 48000 samples/sec = 1/30 sec.

To synchronize the dispatching of the score events, the Conductor must keep track of the sample-time within the execution by updating the sample-clock for every sample generated by the network. The events of each MIDI channel are queued in time-ascending order. A channel is considered active as sample-time reaches or exceeds the time scheduled for the MIDI events at the heads of the channel. There may be several events at the head of any active channel that may be considered active at the current sample-time. All of these active events are sent to the appropriate excitation nodes by the Conductor.

The synchronization of the nodes in a waveguide network occurs naturally by the data flow dependency between the

junctions and waveguides. This is due to the fact that a junction will wait for input from all of its connected waveguides before sending its output. The same is true for a waveguide; a waveguide will wait for the data from its associated junctions. In section 4.1, a protocol is defined that avoids a possible communication deadlock under such a scheme.

3.2.4.3 The Connection Scheme

To aid the Conductor in utilizing the natural synchronization of data flow between waveguides and junctions, while introducing excitation nodes and a means of extracting the output from the system, a connections scheme is defined.

A true closed waveguide network consists solely of waveguides and junctions. The connection scheme used is defined as:

> Waveguides can be connected to junctions, and junctions can be connected to waveguides. No other connections are valid.

To permit network input (i.e., to allow for a flexible inputting of excitation data to the system) connections are required to both waveguides and junctions. To limit the network information needed per node, and to keep the

connection scheme valid, a conceptual view of excitations is defined as:

> If an excitation node is connected to a junction, then it is viewed as a pseudo-waveguide. If it is connected to a waveguide, the excitation node is viewed as a pseudo-junction. These pseudo-nodes have only one connection to the network with an external connection used for the control data input.

Without loss of generality, the output, or "tapping", of the system can be taken from the waveguides only. Thus, for a stereo image to be produced, the system must be tapped from two waveguides. A conceptual view of a tap is defined as:

> A tap is a pseudo-junction with two connections. The two connecting waveguide nodes are the waveguides from where the output of the system is defined to originate. An external connection is used by the system to retrieve the stereo samples.

3.2.4.4 Conductor Actions

The set of commands received by the Conductor, their related data and actions are given as:

Load Network:

Load a network specification into the Conductor's network structure and initialize the modules necessary for the management of this network. This command must be the first command the Conductor receives. Afterwards it can be received at any time by the Conductor. The Load Network command causes the execution of any currently loaded network to halt and its specification to be cleared. The clearing of a network specification does not affect the Conductor's score structure except to reset the sample-clock to time 0.

Read Score:

Read a score into the Conductor's score structure and initialize the sample-clock. Because of the possibility for interactive input of control data, it must be assumed that any score that has been read is not necessarily complete. In fact a null score is not considered to be an error. This implies that the Conductor must make allowance for the asynchronous receipt of control commands during the execution of a network specification.

Play:

Play the currently loaded network using the current score. This command causes the Conductor to initiate the actions of the network by firing the appropriate waveguide network nodes. All of the network nodes valid for the current specification are set to be at their zero state. That is, the waveguide nodes all contain zero samples with their filters set to time 0, the junction nodes are awaiting their first set of samples with the associated admittances undefined, and the excitation nodes are not currently in a state of excitation.

Stop:

Stop the execution of the currently executing network specification. This command has no effect upon either the network structure or the score structure.

3.2.5 Sound Manager

The output from the system is a sequential list of integers representing the digital waveform that is playable through a digital-to-analog converter (DAC). Through the use of the Sound Manager module, the user has the option of either storing the samples in a file for future play, or playing them immediately while the samples are being generated.

Like the Conductor, this module also keeps track of the elapsed sample-time from the beginning of a session. The sample-clock here allows the user to predesignate a limit to the data in a meaningful unit of measure.

4 The Implementation

A prototype of the waveguide design system discussed in Chapter 3 has been implemented and tested. The hardware platform used for the prototype is comprised of a Macintosh IIfx, a board containing a set of INMOS T805 Transputers, and a board containing both a DSP chip and a DAC. The Macintosh is used for the basic editing, all user input (including interactive MIDI device input), and file management. The Transputers are used to perform the sample generation calculations and the DSP board is used for the output of the sample waveforms to an attached audio system.

The Macintosh was chosen for the implementation because of the availability of a large set of support software. This support software includes extensive development tools for the manipulation of files containing MIDI data and for the play and study of digital audio sound files.

In the following sections a description is given of the structure of the implemented design system, the operation of the modules, and other major details of the implementation. The discussion is divided into two major sections with the first section dealing with the two auxiliary boards that allow a certain amount of parallel processing to occur and

the second section dealing with the editing environment as implemented on the Macintosh.

4.1 Parallel Processing

If waveguides and junctions are viewed as processes then, following Hoare's model of communicating sequential processes ([13]), a connection between a junction and a waveguide represents two channels of communication: one channel for each direction of data transfer. The synchronization of the processes in a waveguide network occurs naturally by the data flow dependency between the junctions and waveguides. This is because a junction must wait to receive the pressure data from all of its associated waveguides before being able to perform its computation. The same is true for a waveguide so that the two sets of processes can be seen to work in tandem. To ensure that the model works without deadlock is a simple matter of having all of the waveguides, including the pseudo-waveguides (refer to section 3.2.4.3 for their definition), begin their execution by sending zero pressure values to their associated junctions before commencing their normal functioning.

4.1.1 Increase in Throughput

The increased throughput offered by utilizing a parallel architecture is measured by the differences between the parallel and sequential algorithms used. The elementary operations used for the analysis of parallel algorithms are the computational and routing steps ([4]). It is assumed, in this case though, that the computations of a node in the waveguide network will be the same whether they are implemented on a sequential or parallel architecture. This is also true for the other necessary operations of the controlling system. These operations include the updating of the sample-clock and the management of the stream of samples. Therefore, the main difference between the two algorithms is the steps needed for the routing of the data.

The parallel implementation of a network containing n nodes can be mapped to a set of m processors. If n is greater than m then multi-tasking is used where the scheduling is controlled by the synchronization of the communications. This means that the inter-process communication and the process swapping must be included in the overhead for the parallel system. The Transputer includes these steps as part of their set of basic commands. Therefore, it is assumed that the cost of the intra-processor communications will be minimal and is comparable

to the cost of a function call. Subsequently, the difference in time complexity between the sequential and parallel algorithms implemented per processor is negligible. It is further assumed that there is not a physical one-to-one mapping of connections between processes on different processors and physical channels of communication. This means that a separate inter-processor communication process is necessary on each processor to manage the resource of the physical channels. This inter-processor manager has access to the network configuration and is able to reroute data packets to the appropriate processes.

A problem can occur during communication with and within the Transputer network. There is a possibility for deadlock when two processes simultaneously attempt to perform the same write or read operation with each other. The problem does not exist between waveguide and junction processes linked directly because of the inherent alternation of the data flow that occurs, but it can show up between two processes on different processors. A solution is given such that there exists, on each Transputer, a process for receiving data externally for internal distribution and a process for sending data externally on behalf of the internal processes. This solves the problem because the data flow for these processes is in only one direction and while one process on any processor may be

blocked waiting for the receipt of data the other process is available for sending and vice versa.

The increase in throughput T offered by a parallel implementation can be shown as

$$T = TC_S / TC_P$$
$$= nC / ((nC + I(m,p))/p)$$

where

- TC_S is the time complexity of the sequential algorithm,
- TC_P is the time complexity of the parallel algorithm,
- n is the number of nodes,
- C is the average time complexity of the computation per node,
- p is the number of processors available,
- m is the number of nodes having connections to processes on different processors; $m \leq n$,
- I is the time complexity of the inter-processor communications which is a linear function of m and p.

The limiting factors in the increase in throughput possible are given as the number of processors available and I(m,p). This implies that a linear increase of n processors is possible if I(m,p) can be minimized. The means of minimizing I(m,p) is by reducing the amount of inter-

processor communication required by nodes within the waveguide network. The major criterion in the allocation of processes to processors is the minimizing of the number of connections that span processors. Another criterion is related to the load-balancing of processes to processors where an equal distribution of processes would mean that processors would be idle less often waiting for data from other processors.

4.1.2 The Transputer Network

The current implementation makes use of a board containing 3 INMOS T805 Transputer processors ([45]). These processors are high speed floating-point CPUs with multi-tasking, inherent in the machine language, based upon C.A.R Hoare's view of communicating sequential processes ([13]). The Transputers are limited to 4 external physical communication links each with the link map between them being arbitrarily set at load time.

The system implemented on the Transputer network represents the Conductor as discussed in Chapter 3. There is one processor, processor 0, that is allocated for the communication between the main system and the other two processors. The main process on processor 0 is viewed as being the Conductor and the main processes on the other

processors are viewed as being SubConductors. In addition to the passing of commands and pressure values, the Conductor handles the buffering of digital sample data and the transferring of the buffers to the main system. The two remaining processors are then free to handle the bulk of the processing performed by the nodes of the waveguide network. All three processors have their physical communication links set up so that they represent a fully-connected network. A diagram illustrating the communication links between processors, including the Macintosh and Sound Accelerator board, is presented in Figure 5.

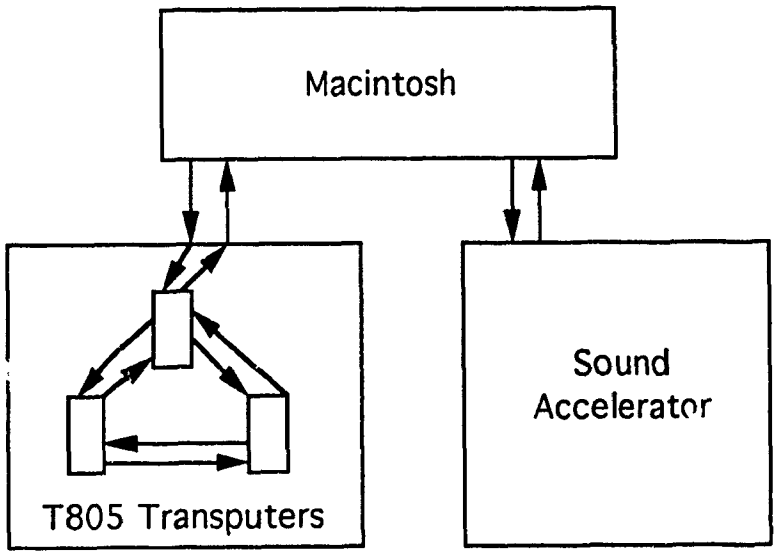

Figure 5 General configuration of the communication system.

The output generated by the Conductor will be the samples of the waveform resulting from the control language's actions being performed upon the network.

Reducing the overhead that is incurred by communication within any system implies that the samples be buffered at some point. This becomes even more evident when it is taken into account that almost all DACs require buffered data. Another argument for the buffering of the sample data is that file systems are known to have a higher throughput if the data is buffered before being written to disk. Therefore, to lower the overhead of communication between the Macintosh and the Transputers, the Conductor buffers the sample data before sending it to the Macintosh.

The Conductor sends the buffers upon demand due to the fact that the data is possibly being forwarded to a DAC. This is because the DAC generates an interrupt to request for a buffer of data. There are two cases to consider related to the rate of sample generation:

(1) The Conductor is generating samples at speeds greater than real-time. If this is the case, it is a simple matter to suspend execution of the network, after a certain number of buffers have been filled, until the demand for samples has caused the number of buffers held by the Conductor to drop below a predefined limit.

(2) The Conductor is generating samples at speeds less than real-time. If this is the case and the DAC is making demands for buffers that are not available, then

the resulting sound from the audio system will be unintelligible because the DAC must work in real-time. A valid response by the Conductor, therefore, to a demand to send buffers of samples is a response signifying that no samples are available. This allows the process on the Macintosh making the request to take an appropriate set of actions. The appropriate set of actions are discussed in section 4.2 on page 66.

The Conductor process receives commands and data from the system running on the Macintosh. The main purpose of the Conductor is to distribute the commands along with the necessary data to the rest of the network. The data flow with the Conductor is from the Macintosh and to the other Transputers in the network. This avoids deadlock with the other processes on the Transputer network by not allowing the Conductor to receive from the other Transputers. The Conductor also sends the buffers of samples to the Macintosh. The possibility for deadlock that exists with the Macintosh through both parties sending or receiving at the same time is avoided because a transfer from the Conductor to the Macintosh occurs only after receipt of a command specifying that the Conductor send any available buffers. This is the only occasion the Conductor has to send data to the Macintosh. The communication by the Conductor to the other Transputers is received by

the resulting sound from the audio system will be unintelligible because the DAC must work in real-time. A valid response by the Conductor, therefore, to a demand to send buffers of samples is a response signifying that no samples are available. This allows the process on the Macintosh making the request to take an appropriate set of actions. The appropriate set of actions are discussed in section 4.2 on page 66.

The Conductor process receives commands and data from the system running on the Macintosh. The main purpose of the Conductor is to distribute the commands along with the necessary data to the rest of the network. The data flow with the Conductor is from the Macintosh and to the other Transputers in the network. This avoids deadlock with the other processes on the Transputer network by not allowing the Conductor to receive from the other Transputers. The Conductor also sends the buffers of samples to the Macintosh. The possibility for deadlock that exists with the Macintosh through both parties sending or receiving at the same time is avoided because a transfer from the Conductor to the Macintosh occurs only after receipt of a command specifying that the Conductor send any available buffers. This is the only occasion the Conductor has to send data to the Macintosh. The communication by the Conductor to the other Transputers is received by

SubConductor processes, which reside on the other Transputers. There are two commands and one type of data packet left to consider where the Conductor plays a major role other than to simply reroute the command to the necessary parties. The commands are the Load Network Definition and the Load MIDI Tracks commands and the data packet is the Buffer Sample data packet.

The Load Network Definition command causes the Conductor, after retrieving the necessary data from the Macintosh, to stop all current activity within the network and to reinitialize data structures and send portions of the network to the appropriate Transputer.

The Load MIDI Tracks command causes the latest set of control input data to be deleted and the new data from the Macintosh to replace it. This data is then used when the Conductor receives a Buffer Sample data packet. The receipt of this data packet causes the Conductor to insert the stereo sample data into the current Sample Buffer and update the sample-clock. The time in the sample-clock is then checked against the time for the next set of MIDI events to occur. If there are events pending then they must be removed from the list and sent to the appropriate process within the network. A single event has the possibility of being sent to several processes within the network.

The process that implements the tapping of the stereo output for the network is the only other process that resides on processor 0. Its data flow is from the other Transputers to the Conductor and subsequently avoids the possibility for deadlock. The tapping process receives sample data packets from the rest of the network; a sample data packet contains the pressure value of the sample and the admittance that is associated with the sample. The data that the process sends to the Conductor is the resulting sample data packets that are being returned to the network and the Buffer Sample packets being buffered by the Conductor.

The functioning of the SubConductor is similar to that of the Conductor in that the SubConductor distributes commands and data from the system to the portion of the network it is servicing. The SubConductor consists of two processes: one for the receipt of data from an external source, and one for the sending of data to an external source. The process that receives from an external source is also be called upon to perform the system initialization and to enact the birth and death of processes for the portion of the network it is servicing. When the SubConductor processes are simply acting as distributors they can be viewed as if they were acting as proxies for the respective processes that wish to communicate. For

instance, if a waveguide node on processor 1 wishes to send a data packet to a junction node on the same processor it will simply send the data packet directly. However, if the junction resides on another processor then the waveguide must first send the packet to the outgoing process. This process will identify the Transputer on which the requested junction node resides by referencing the network definition and then will forward the packet to that Transputer. The incoming process on the subsequent Transputer will then reroute the packet to the appropriate junction node.

The processes are considered proxies because they act on behalf of the processes residing on the same Transputer. The details of deciding to send directly or not are hidden from the waveguide network nodes. This is done by giving the network nodes a set of anonymous communication channels through which they will communicate. The nodes cannot identify if the channel given to them is one directly connected to another network node or to a proxy process. The inter-processor communication is done by proxy because of the limit of physical channels between Transputers and because of the need to avoid deadlock. The remaining set of processes on the Transputers, other than Processor 0, is made up of the nodes of the variable sized waveguide network.

The junction process implements the definition of a junction in a waveguide network. When the junction receives a sample data packet, the junction stores the pressure value and adds it to the current total. When all of the data packets necessary to calculate the junction's total sample pressure have been received by the junction, it sends the resulting data packets that represent the outgoing pressure values from the junction to the associated waveguide processes. If the admittance value of any incoming sample has changed, a new junction admittance must be calculated, otherwise the previous value is used for the calculation of the outgoing pressure values.

The process representing a waveguide node sends the latest pressure values from its delay lines to the junctions to which it is associated. The waveguide process then waits for each associated junction to respond with the pressure value that is travelling in the opposite direction. The process will pass this pressure value through the filter associated with the delay line and add the result to the queue of pressure values that represents the given delay line. The cycle is then repeated. The waveguide may also respond to other commands sent asynchronously which alter the parameters of the waveguide. The variable parameters are the admittance, the gain factor, the filter parameters,

and the length of delay for each delay line in the waveguide.

In the discussion of the implementation of waveguides and junctions references were made only to communications with these two types of processes. This is consistent with the conceptual view of an excitation node as being a pseudo-junction or a pseudo-waveguide and the processes representing the excitation nodes maintain this view.

When an excitation process is first started up it is given a reference to a lookup table that is used for the creation of waveforms to be entered into the waveguide network. These waveforms are the manner in which the excitation processes excite the system. It is the responsibility of the SubConductor to initialize and to assign the appropriate tables needed by the excitations residing on a Transputer. The information to perform this task is found in the portion of the network definition sent to it by the Conductor.

The control an excitation process has on the network is through the waveguide or junction associated with the process. An excitation process can only be in one of three states: passive, exciting, or decaying. The state an excitation process is in determines the type of control that

is applied to the system. An excitation process begins its execution in the passive state. When an excitation process is in the passive state it simply returns the sample data received to the sending process.

An excitation process is instructed to excite the network, and thereby enter a state of excitation, by the receipt of a MIDI data packet from the SubConductor. While in a state of excitation, the process ignores the data contained within the sample data received from its associated process. Instead, the excitation process responds to the receipt of sample data packets by creating new admittance and pressure values that are to be returned in the data packets. The excitation uses the note value contained in the MIDI data received to establish the frequency of the waveform being entered into the system through the sample data. The velocity value in the MIDI data determines the gain factor applied to the samples of the waveform in the calculation of the individual pressure values. The velocity value also determines the admittance value that accompanies the pressure values in the sample data. If the excitation is connected to a waveguide, the excitation will also instruct the waveguide to alter its parameters. The resulting frequency response of the waveguide will then reflect the new values determined by the excitation. If the excitation is connected to a junction,

then the admittance values will automatically alter the network's total stored energy as a result of the calculations performed by the junction process. The excitation process remains in a state of excitation for a period of one cycle of the resulting waveform after which it will enter a passive state.

The excitation process enters a state of decay either after the number of cycles designated for it to begin the decay has been exceeded or after the receipt of a **Note On** command that contains a velocity value of zero. The excitation process causes the decay of the stored energy of the system by altering the gain factor that is applied to the pressure values it returns. If the excitation is connected to a junction it will also alter the admittance value in the sample data being received by the excitation process.

4.1.3 The Sound Accelerator Board

The current implementation makes use of a board called the Sound Accelerator containing a DSP56000 CPU chip and a DAC for its audio output. The Macintosh acts as a mediator between the Transputers which generate the data and the Sound Accelerator, enabling full parallelism to occur with a minimum of system interference.

The process implemented on the Sound Accelerator board contains 2 buffers of a fixed size containing sample data. When initialized it performs some necessary hardware initialization and then repeatedly fills the two buffers by interrupting the system with a request for a buffer of samples. The requests for buffers of samples are synchronized with the DAC so that a request is made every t seconds. t is related to the buffer size and the sampling rate by

$$t = N / R$$

where

N is the number of samples in a buffer

R is the sampling rate (samples/sec).

4.1.4 Summary

The benefits derived from a parallel implementation of a waveguide network are summarized as follows:

1. It is relatively simple to implement due to the inherent parallelism of the nodes of the network.
2. Changes in the topology of the system are easily affected by changing the destination channel in the sender process. The sender process need not know to which process the message is being sent. This allows a standard generalized view of the communication interface for a process

and implies that the system can be distributed with a minimum of centralized control.

3. The data-flow through the system regulates the synchronization of the processes. Care must be taken, however, to ensure that deadlock does not occur.

4. A linear increase in throughput over the sequential implementation is possible by allocating the processes to the processors in a manner that minimizes the amount of inter-processor communication.

4.2 The Waveguide Editor

The driver and editors for the design system were implemented as part of an interactive graphical object-oriented programming environment called MAX ([7]). MAX was developed originally to be used as a control language for sound synthesis and has now been developed for use on the Macintosh line of computers as a control language for MIDI applications. It is a data-flow language where the graphical images represent objects that can receive and send messages through their ports. MAX provides numerous objects that manipulate MIDI data. MAX can also be extended by users who wish to create their own customized objects ([46]).

The current implementation is developed as a set of MAX objects where the Waveguide Editor, the only object explicitly accessed by a user of the system, receives the MIDI input and user commands through its ports. The other objects implemented for the system represent the three types of Network Editor. The Network Editors inherit from a windowing class for graphical editing that MAX also provides.

The Waveguide Editor object handles the loading of and communication with the two boards as well as the maintenance of the Network Editors. The maintenance provided is for the referencing and dereferencing of unique instantiations of Network Editors; the dashed lines in Figure 4(a) denote examples of object to editor references. Lists are kept of the unique instantiations of Network Editors that are available in memory. The instantiation of a specific Network Editor occurs only when a reference to a definition of a module is made and it does not already exist in memory. The removal of a Network Editor and its contents occurs after a dereferencing operation brings the number of references to the specific editor to zero. The requests for referencing and dereferencing of a Network Editor can occur as a result of an action in another Network Editor. These requests, therefore, can come from a child of the Waveguide Editor. Because of this and also because the Network

Editors must take their requests from the user directly, the objects must be considered as active independent objects and not as passive objects that react only to requests from the Waveguide Editor. Care must be taken, therefore, to keep the data structures consistent. For instance, an editor may receive a request to die from the user directly and or indirectly through the Waveguide Editor and so the destructor for the object must take this into account.

The Loader module as discussed in Chapter 3 is implemented as a method within the Waveguide Editor. It makes requests for the pertinent data from the Network Editors starting with the room definition. If a definition is not in memory at the time of loading it will also cause the instantiation of the subsequent Network Editor needed to contain the definition. If at any time the resulting partial network does not constitute a valid network then the action of the Loader is halted and the user is informed of the error that caused the inconsistency. The partitioning of waveguide network nodes to Transputers is also the job of the Loader. The Loader transfers the resulting network definition to the Conductor residing on the Transputer board. The communications necessary for this and all other communications with the Conductor are managed by the Waveguide Editor.

The receipt of the Play command by the Waveguide Editor may require that a score (MIDI data) be transferred to the Conductor. Therefore, to retrieve the score from disk a module to read and transform Standard MIDI File ([20]) data is implemented in the Waveguide Editor. This module must also make use of the Waveguide Editor to Transputer communication subsystem. In addition, the Play command, depending upon whether the user wishes to listen to the results directly or store them into a file, may require the cooperation of the Sound Manager residing on the Sound Accelerator board. If the user has requested to listen to the results directly and the Conductor is not producing samples at a rate greater than or equal to real-time, the Waveguide Editor will keep a queue of buffers of samples and while accumulating new buffers, will continuously send the buffers in the order of entry into the queue. This queue has a fixed size determined through testing. The size of the queue was determined on the basis of allowing the sound produced to be long enough to give reasonable snapshots of the resulting sound while not using up all of the available memory that may be needed by other processes.

The Waveguide Editor keeps track of the real-time clock as well as updating the version of the sample clock kept in the Macintosh. The need to keep track of the sample-time is to aid in the estimation of when the next request for

buffers of samples from the Conductor can optimally take place. The Waveguide Editor attempts to minimize the number of requests for buffers when the results are sent to a file or when the Conductor is producing samples at rates less than real-time. This is to reduce the interference with the operation of the Conductor and the communication overhead between the Conductor and the Macintosh.

The Network Editor, as discussed previously, represents a class of editor objects. It can represent a subsystem of a waveguide network at the room level, the instrument level, or the subnetwork level. A reduction of the design was implemented such that the room level editor could not contain Subnetwork objects. An instantiation of a type of Network Editor results in a specification of a subsystem of a waveguide network being loaded from a file that contains the list of network object definitions for the specification. The definitions contain the type of object, the location of the object within a graphic editing window, and the object related parameters. Included in a specification is the map denoting the coupling of the objects.

The Network Editor class makes use of a class of object for a graphic editing window environment that is provided by MAX. This windowing environment provides the derived object

with a system where user input is translated into a predetermined set of methods being invoked with the appropriate parameters. When an object of the windowing class is instantiated the system will make a request from the derived object whose results will determine the subset of services that are to be provided for the windowing object. The services provided require that the resulting set of methods to be invoked for a service be defined within the derived object. This enables the designer of a graphic editor object in MAX to customize the environment to suit the object's needs for the style of editing required.

The Network Editor's window is divided into two regions; one for general editing, and one containing the icon frames for the available waveguide objects for a specific type of editor. Also contained in the icon frame window are the fixed set of I/O port objects for a specific Network Editor (see Figure 6).

When the user selects a point in the window by a mouse click, the type of action requested will depend upon whether the point selected is within the boundaries of an editable item. An editable item is defined graphically by an object's frame boundaries or a line connecting two objects' port icons. Either the Object Manager or the Port Manager can be invoked depending upon where in the selected object's

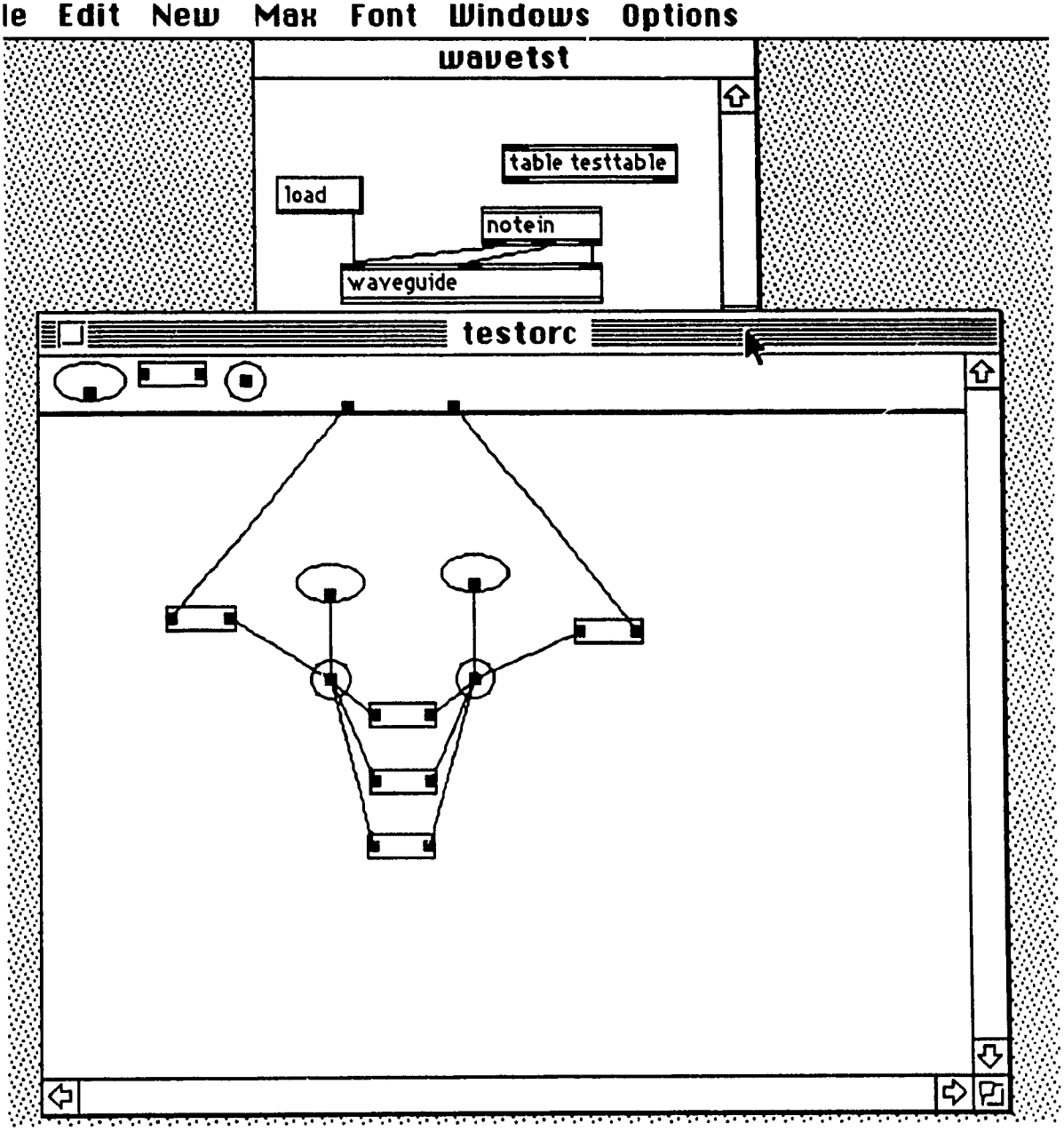

Figure 6 Example of a Room Network Editor's Window.

frame the selected point resides. If the selected point is not within the object's port icon then the Object Manager is invoked; otherwise the Port Manager is invoked. The last editable item to be selected is considered the default item for the availability of further actions to take place. The

default item, if one exists, is accentuated graphically within the general editing window.

To create a new object within the general editing window, the user selects a point within the boundaries of an icon frame. A new object with default values is then placed in the general window the position of which will depend upon where the object's frame is dragged to by the cursor movement controlled by the mouse. To alter the position of an object, the user selects a point within the object and drags the object's frame to the new location. To delete an object the delete command is invoked and the last selected object's icon along with the information related to that object is removed.

To create a connection between two existing objects, an object's port icon is selected and then the cursor is dragged to another object's port icon. If the connection scheme does not violate the rules for the coupling of objects then a line connecting the two objects' port icons is drawn. A connection between two objects is deleted by selecting a point that resides along the line connecting the two objects and then invoking the delete command.

The Macintosh's Dialog Box environment allows the user to alter the variable set of parameters of an object. A

specific keystroke is used to invoke the set of Dialogs for the last selected object (see Figure 7). These Dialogs are different for each type of object available but are not available for the Junction and I/O Port objects.

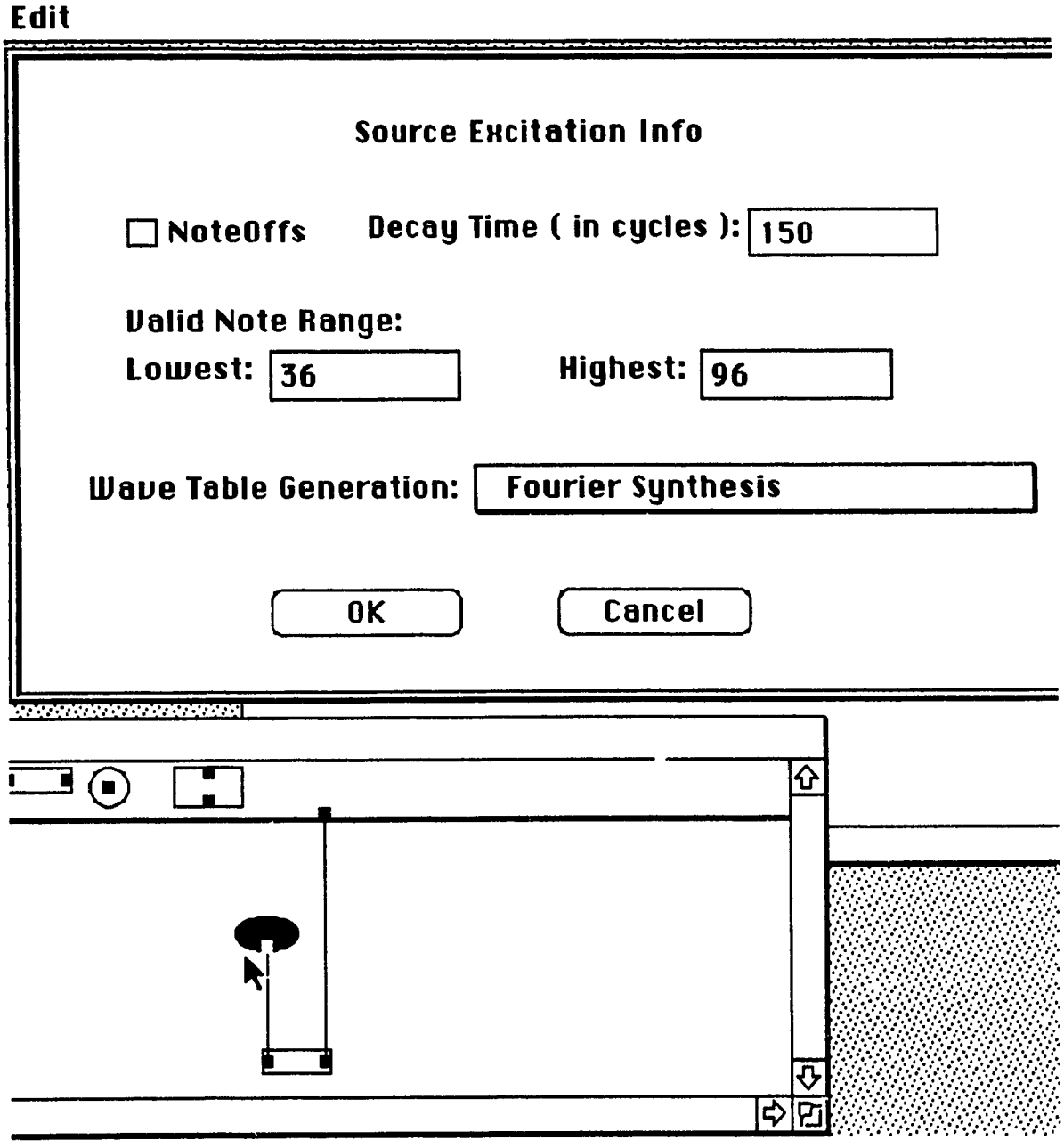

Figure 7 Example of an Excitation Object's Dialog.

The Dialog for a Player object contains editable entries for the name of the file containing its instrument definition and the track number for the MIDI data to be used to drive the instrument. The Dialog for a Subnetwork object contains only a single editable entry for the name of the file containing its subnetwork definition.

The Dialog for a Waveguide object contains editable entries for its delay and admittance with another Dialog, accessed within the first Dialog, being used for each of the filters that are related to each of the delay lines. The Dialog for a Waveguide's filters contains editable entries for the gain and a Dialog for its list of filter parameters.

The Dialog for an Excitation object contains editable entries for its table lookup creation function, the range of notes that are valid for the instrument, and a parameter denoting the number of cycles for the decay of the waveform, with another Dialog available for the list of parameters to be used for the table lookup function. The types of table lookup creation functions available are listed below:

Chebyshev

Cubic spline interpolation

Straight line interpolation

Exponential curve interpolation

Fourier synthesis

A MAX Table

This does not denote a function but rather a set of points that can be created graphically within MAX and therefore does not need a list of parameters but the name of the table to be used. The user can use a MAX table editor provided by MAX to create a fixed wave shape graphically.

Random

This does not require any parameters because the table to be used in the lookup method for this excitation will be filled with random values.

The Dialog for a list of parameters of any type will differ depending upon the view of the parameters as being a list of single, double, triple, or quadruple parameters. This is related to the function that uses them. For example, the Dialog for the filter parameters in a waveguide regards them as duples while the Dialog for the parameters of the Fourier Synthesis table lookup creation function regards them as triples (see Figure 8).

4.3 Summary

The prototype design system described above was able to produce results that proved satisfactory. One test was to produce one simple network and continually modify it with an

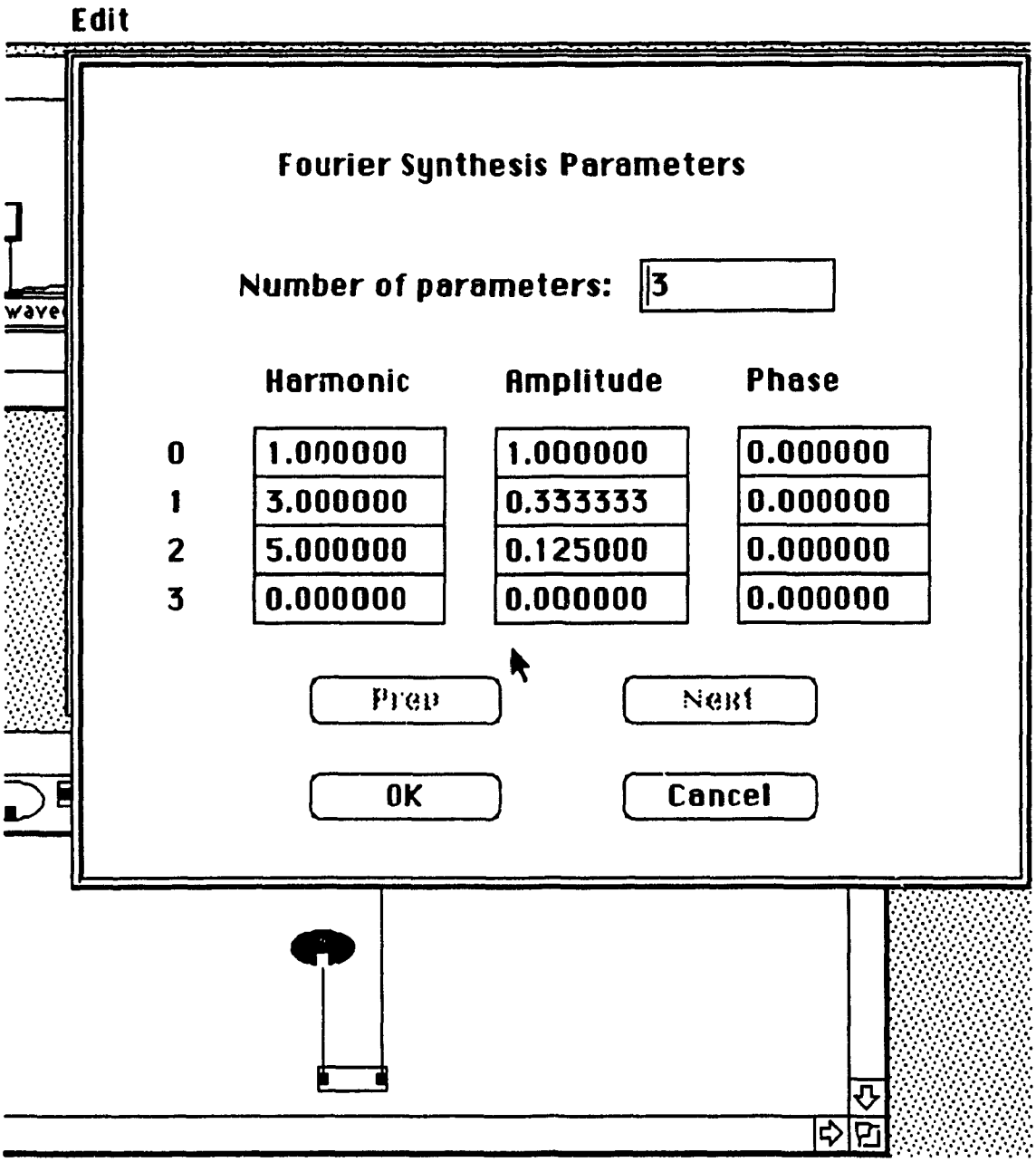

Figure 8 Example of a Parameter Dialog for Fourier Synthesis.

expected resulting waveform to be used as a guide for the modifications. Throughout this test the same MIDI input file was used so that the aural comparison would be easier to make. The ease of modification of the resonance of a room was as expected but the design and implementation of

instruments took more than simple intuitive manipulation to produce approximations to the desired sound. It became obvious though that the more experience that was gained manipulating the elements of a waveguide network the easier it would be to produce results according to specification.

It was not deemed necessary to implement the full design to be able to establish its legitimacy. The subset of the design did not allow the following concepts that could be considered an essential part of a full system:

(1) recursion at the instrument level

(2) variable number of ports across subnetworks

(3) cascaded waveguides

(4) a more flexible method of defining an excitations' set of actions

The prototype was not able to generate waveforms at rates greater than or equal to real-time. A test was made of the communication bandwidth across the Macintosh to Transputer link and this showed that this was not the bottle-neck of the system. This test was useful, however, in demonstrating a limit that could be achieved. Buffers were sent upon demand and the number of buffers sent per demand was a factor of the speed at which the process on the first Transputer could fill them with values representing a repeating sine wave. When a small change in the algorithm

was made the real cause of the slower processing rates was discovered. The change was to have a second process actually do the transfer and the first process' job was simply to fill the buffers. The request for buffers to be sent was received by the first process and passed on to the second which then enacted the transfer. This resulted in buffers being generated at rates that were less than real-time. The cause for the reduction in throughput was therefore seen to be the overhead of the inter-process communication and process rescheduling that is inherent in the Transputer command set.

The fact that waveforms were being generated at rates of less than real-time also had an effect on the waveforms that were produced during interactive input. The sounds were quite interesting but not what was projected.

5 Conclusion

Waveguide networks represent an efficient method for computing resonating systems for sound synthesis. They also lend themselves well to simple conceptual manipulation that eases the design process for creating general sound generation systems.

A system for the design of general sound production systems has been presented that incorporates waveguide networks. A higher level of conceptualization is presented to aid in the composition of modules consisting of waveguide networks. A prototype of the system has been implemented that further eases the process of design through efficient computation of the resulting waveforms. The prototype is implemented in an environment that allows a user flexible access of the system by incorporating it as an addition to an already existing system (MAX) designed specifically for the manipulation of elements for sound synthesis.

The inclusion of the system into the MAX environment entailed the creation of the editors and the creation of the system for including the code and the control of the added peripherals used by the Audio-CAD system. Even though MAX provides some basic tools for a user to create new objects that perform graphical editing, these tools are only a frame

for objects to be built upon. The code used for the Transputers and the Sound Accelerator was built in a different environment from that used for the MAX objects. This posed some problems when it was found that it was necessary to include the code for all three systems into the same object that controlled them. A library and set of tools had to be created simply to aid in the development of the prototype. These tools along with the library of functions for the communication interface between a MAX object and the Transputers proved to be successful not only for the work presented here but also for projects being explored by other people using the same platform.

Future possibilities for research in waveguide network theory could include the development of a formal theory of coupling and control of subnetworks. This would allow the mathematical prediction of the set of results of any given configuration. This is still an intuitive process which is enhanced through experience but it may also be that this is sufficient for its general use. It should also be possible to develop a specific system for speech synthesis based upon waveguide networks that would model the actions of a vocal tract during human speech. This would promise to be a more dynamic system than the present methods that simply concatenate phoneme parameters.

A question worth investigating is whether the physical modelling of sound systems using waveguide networks is limited to synthesis or is it possible to be used for analysis as well. For this to be true the data path of samples in a network would have to be invertible in order to be able to predict the set of actions a system would have to take to reproduce a sound.

6 Bibliography

[1] Jean-Marie Adrien and Xavier Rodet, "Physical Models of Instruments: A Modular Approach, Application to Strings", **Proceedings of the 1985 International Computer Music Conference**, Computer Music Association, 1985, 85-90.

[2] Jean-Marie Adrien and Rene Causse and Xavier Rodet, "Sound Synthesis by Physical Models: Application to Strings", **Proceedings of the 1987 International Computer Music Conference**, Computer Music Association, 1987, 264-269.

[3] Jean-Marie Adrien and Rene Causse and Eric Ducasse, "Dynamic Modelling of Stringed and Wind Instruments: Sound Synthesis by Physical Models", **Proceedings of the 1988 International Computer Music Conference**, Computer Music Association, 1988.

[4] Selim G. Akl, **The Design and Analysis of Parallel Algorithms**, Prentice Hall, 1989.

[5] K. Mani Chandy and Jayadev Misra, **Parallel Program Design - A Foundation**, Addison Wesley, 1988.

[6] John M. Chowning, "The Synthesis of Complex Audio Spectra by Means of Frequency Modulation", in **Foundations of Computer Music**, Curtis Roads and John Strawn eds., The MIT Press, 1985, 6-29.

[7] Christopher Dobrian and David Zicarelli, **MAX User's Manual**, Opcode Systems Inc., 1990.

[8] Charles Dodge and Thomas A. Jerse, **Computer Music: Synthesis, Composition, and Performance**, Shirmer Books, 1985.

[9] Guy E. Garnett, "Modelling Piano Sounds Using Waveguide Digital Filtering Techniques", **Proceedings of the 1987 International Computer Music Conference**, Computer Music Association, 1987, 89-95.

[10] Guy E. Garnett and Bernard M. Mont-Reynaud, "Heirarchical Waveguide Networks", **Proceedings of the 1988 International Computer Music Conference**, Computer Music Association, 1988, 297-312.

[11] John W. Gordon and John M. Grey, "Perception of Spectral Modifications on Orchestral Instrument Tones", **Computer Music Journal**, 1978, 2(1):24-31.

[12] J.W. Gordon and J. Strawn, "An Introduction to the Phase Vocoder", in **Digital Audio Signal Processing: An Anthology**, ed. J. Strawn, William Kaufman Inc., 1985.

[13] C.A.R. Hoare, **Communicating Sequential Processes**, Prentice-Hall International, 1985.

[14] David A. Jaffe and Julius O. Smith, "Extensions of the Karplus-Strong Plucked-String Algorithm", **Computer Music Journal**, 1983, 7(2):56-69.

[15] Kevin Karplus and Alex Strong, "Digital Synthesis of Plucked-String and Drum Timbres", **Computer Music Journal**, 1983, 7(2):43-55.

[16] Otto E. Laske, "Considering Human Memory in Designing User Interfaces for Computer Music", **Computer Music Journal**, 1978, 2(4):39-45.

[17] Joseph Marks and John Polito, "Modelling Piano Tones", **Proceedings of the 1986 International Computer Music Conference**, Computer Music Association, 1986, 263-268.

[18] Stephen McAdams and Albert Bergman, "Hearing Musical Streams", in **Foundations of Computer Music**, eds. Curtis Roads and John Strawn, "The MIT Press", 1985, 640-657.

[19] **MIDI 1.0 Detailed Specification**, International MIDI Association, 1988.

[20] **Standard MIDI Files 1.0**, International MIDI Association, 1988.

[21] B. Moog, "MIDI: Musical Instrument Digital Interface", **Journal of the Audio Engineering Society**, 1986, 34(5):394-404.

[22] F. Richard Moore, "Table Lookup Noise for Sinusoidal Digital Oscillators", in **Foundations of Computer Music**, eds. Curtis Roads and John Strawn, "The MIT Press", 1985, 326-334.

[23] F. Richard Moore, **Elements of Computer Music**, Prentice-Hall, 1990.

[24] James A. Moorer, "About this Reverberation Business", **Computer Music Journal**, 1979, 3(2):13-28.

[25] Dexter Morrill, "Trumpet Algorithms for Computer Composition", **Computer Music Journal**, 1977, 1(1):46-52.

[26] Michael J. Quinn, **Designing Efficient Algorithms for Parallel Computers**, McGraw Hill, 1987.

[27] Curtis Roads, **Composers and the Computer**, William Kaufmann Inc., 1985.

[28] J. Rodgers, "Digital Simulation of the Piano", **Proceedings of the 1982 International Computer Music Conference**, Computer Music Association, 1982, 358-366.

[29] Julius O. Smith, "Synthesis of Bowed Strings", **Proceedings of the 1982 International Computer Music Conference**, Computer Music Association, 1982, 308-340.

[30] Julius O. Smith, **Techniques for Digital Filter Design and System Identification with Application to the Violin**, Phd. Thesis, Stanford University, 1983.

[31] Julius O. Smith, "Spectral Pre-Processing for Audio Digital Filter Design", **Proceedings of the 1983 International Computer Music Conference**, Computer Music Association, 1983, 57-79.

[32] Julius O. Smith, "An Allpass Approach to Digital Phasing and Flanging", **Proceedings of the 1984 International Computer Music Conference**, Computer Music Association, 1984, 103-110.

[33] Julius O. Smith, "A New Approach to Digital Reverberation Using Closed Waveguide Networks", **Proceedings of the 1985 International Computer Music Conference**, Computer Music Association, 1985, 47-54.

[34] Julius O. Smith, "An Introduction to Digital Filter Theory", in **Digital Audio Signal Processing: An Anthology**, ed. J. Strawn, William Kaufman Inc., 198_.

[35] Julius O. Smith, **Waveguide Digital Filters**, Technical Report, Center for Computer Research in Music and Acoustics (CCRMA), Stanford University, 1985.

[36] Julius O. Smith, "Efficient Simulation of the Reed-Bore and Bow-String Mechanisms", **Proceedings of the 1986 International Computer Music Conference**, Computer Music Association, 1986.

[37] Julius O. Smith, **Elimination of Limit Cycles and Overflow Oscillations in Time-Varying Lattice and Ladder Digital Filters**, Tech. Report #STAN-M-35, Dept. of Music, Stanford University, 1986.

[38] Julius O. Smith, "Waveguide Filter Tutorial", **Proceedings of the 1987 International Computer Music Conference**, Computer Music Association, 1987, 9-16.

[39] John Strawn ed., **Digital Audio Engineering: An Anthology**, in The Computer Music and Digital Audio Series, William Kaufmann, Inc., 1985.

[40] John Strawn ed., **Digital Audio Signal Processing: An Anthology**, in The Computer Music and Digital Audio Series, William Kaufmann, Inc., 1985.

[41] John Strawn, "Orchestral Instruments: Analysis of Performed Transitions", **Journal of the Audio Engineering Society**, 1986, 34(11):867-880.

[42] John Strawn, "Editing Time-Varying Spectra", **Journal of the Audio Engineering Society**, 1987, 35(5):337-352.

[43] John Strawn, "Analysis and Synthesis of Musical Transitions Using the Discrete Short-Time Fourier Transform", **Journal of the Audio Engineering Society**, 1987, 35(1-2).

[44] David L. Wessel, "Timbre Space as a Musical Control Sequence", in **Foundations of Computer Music**, Curtis Roads and John Strawn eds., The MIT Press, 1985, 640-657.

[45] Colin Whitby-Stevens, "The Transputer", **The 12th Annual International Symposium on Computer Architecture**, IEEE, 1985, 292-300.

[46] David Zicarelli, **Writing External Objects for MAX**, Opcode Systems Inc., 1990.