1

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS
STANDARD REFERENCE MATERIAL 1010a
(ANSI and ISO TEST CHART No. 2)

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

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.

**Recording Studio Automation
- Towards the Virtual Studio**

by
Joseph L. W. May

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of
Master of Computer Science

School of Computer Science
Carleton University, Ottawa, Ontario

July 9, 1990

© copyright 1990, Joseph L. W. May

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-68864-5

The undersigned recommend to the Faculty of Graduated Studies
and Research acceptance of the thesis

**Recording Studio Automation
- Towards the Virtual Studio**

submitted by Joseph L. W. May, B.C.Sc.
in partial fulfillment of the requirements for
the degree of Master Of Computer Science

A handwritten signature in dark ink, appearing to read "N. La Pino".

Thesis Supervisor

A handwritten signature in dark ink, appearing to read "F. Fale".

Director, School of Computer Science

Carleton University

August 1, 1990

ABSTRACT

This thesis discusses a recording studio automation and composition tool for smaller, home-based type recording studios. The system provides a centralized, consistent user interface to all of the hardware in the recording studio. This interface is seen as the first step towards the virtual studio concept. The system also provides a method for mixdown type automation for all of the studio devices. Finally, the system offers an integrated compositional tool to capture performance and mixdown sequences into named entities - sound objects. Sound objects can be combined and temporally arranged for playback. They also offer a hierarchical approach to the composition process through the grouping of sets of sound objects into larger, more complex sound objects.

This system is designed to offer a high level of control over the production of audio through the support of hierarchical sound objects and the virtual hardware concept.

**I would like to dedicate this thesis to Karl Gradt, my High School
computer science teacher and my friend.**

ACKNOWLEDGEMENTS

I wish to thank my supervisors Professor Nicola Santoro, and Professor Dave Thomas for their support and encouragement throughout my thesis.

I would like to thank Professor Jorg Sack for his encouragement throughout my Masters program.

I would also like to thank Lou Nel for his comments on and support of my work, and for providing many thought provoking discussions related to music and music performance.

I would like to thank Daryl Graf for making me feel my work was important and for introducing me to the wonderful world of Smalltalk.

I would also like to acknowledge the contributions of Randy Toni to the working system described in this thesis. In particular, Randy was responsible for the implementation of the MIDI sequencer editor window, the add multiple channel dialog box, the MIDI files support in the sequencer window, and various editor features.

I would like to thank my fiancée, Della MacKay, for her patience throughout the last two and a half years, and for her encouragement and love of my music.

Most of all I would like to thank my good friend Thong Nguyen whom with I spent a great many hours discussing ideas for the design of my thesis. I particularly wish to thank Thong for his undaunted help during the early stages of the implementation and for teaching me a great deal of what I know about object-oriented programming.

TABLE OF CONTENTS

Table of Contents	vi
List of Figures	ix
List of Tables	xi
1..... Introduction	1
1.1..... Music Recording	2
1.1.1..... The Recording Process	2
1.1.2..... Reasons for Multi-track Recording	3
1.1.3..... The Recording Studio	4
1.1.4..... The Studio Personnel	4
1.1.4.1..... The Producer	5
1.1.4.2..... The Recording Engineer	5
1.1.4.3..... The Musician	5
1.1.4.4..... The Composer	5
1.1.5..... The Small Personal Studio	6
1.1.6..... The Studio as the Instrument	6
1.2..... The Recording Console	6
1.2.1..... The Stereo Console	7
1.2.2..... Multiple Buss Console	10
1.3..... Special Effects Processing	11
1.3.1..... Equalization	12
1.3.1.1..... Using Equalization	13
1.3.2..... Reverberation	14
1.3.3..... Compression/expansion	15
1.3.4..... Other types of processing	16
1.4..... MIDI	16
1.4.1..... MIDI Controllers and Receivers	18
1.4.2..... Other MIDI Devices	19
1.5..... Sequencers	19
1.6..... Summary	20
2..... Survey of Current Systems	21
2.1..... Current Automation Techniques	21
2.1.1..... Automated Analog Mixing Consoles	21
2.1.2..... Automated Digital Recording Consoles	22
2.1.3..... MIDI Equipped Digital Consoles	23
2.1.4..... MIDI controlled VCAs	27
2.1.5..... Summary	27
2.2..... Sound Objects	28
3..... The Automation System	29
3.1..... The Computer Aided Audio Production System	29
3.1.1..... Device Representation	29
3.1.2..... Introduction to the System	31
3.1.3..... The System Screens	31
3.1.4..... The System Libraries and Editors	33
3.2..... The Device Editor Screen	34

3.2.1.....	Channel Editing	35
3.2.2.....	Device Editing	39
3.2.3.....	A Complete Example	42
3.3.....	The Sequencing Screen	45
3.3.1.....	The Sequencing Screen Menu	45
3.3.2.....	The Device Controller Window	46
3.3.3.....	Formalizations - What is a Device?	51
3.3.4.....	Formalizations - Definition of a Channel	52
3.3.5.....	Advantages of the System	53
3.3.6.....	The Sequencer	53
3.3.7.....	Sequencing Control Movements - Mixdown Automation	57
3.3.8.....	Punch-in Recording	60
3.3.9.....	Control from an External Device	60
3.3.10.....	The Master System Window	61
3.4.....	The Cue List Editor	62
3.5.....	Summary	64
4.....	The Hierarchical Sound System - Sound Objects	66
4.1.....	The System's Sound Objects	67
4.1.1.....	Simple Sound Objects	67
4.1.2.....	Composite Sound Objects	68
4.1.3.....	The Composite Sound Object Editor	69
4.1.3.1.....	The Attach Operations	73
4.1.4.....	Cue Points and the Sound Object Editor	75
4.1.5.....	Playing Sound Objects	77
4.2.....	A Sound Object Approach to Composition - an Example of Sound Object Use	77
4.2.1.....	Example 1 - Sound Effect for Film	77
4.2.2.....	Example 2 - Development of a Song	79
4.3.....	Summary	81
5.....	The Implementation	83
5.1.....	The Implementation Platform	83
5.2.....	The Classes	84
5.3.....	The Device Structures	86
5.3.1.....	The Device Editing Structures	86
5.3.2.....	The Device Controller Structures	88
5.3.3.....	Sequencing	90
5.3.4.....	Inheritance in the Design of Controls	94
5.4.....	The Sequencer	96
5.5.....	The Sound Object Structures	96
5.5.1.....	The Structure	96
5.5.2.....	Playing Sound Objects	97
5.6.....	The MIDI Management Classes	98
5.7.....	The Screen Switcher	99
5.8.....	Suggestions for Future Improvements	100
6.....	Extensions to the System	101
6.1.....	Extensions	101
6.1.1.....	General Editor Features	101
6.1.2.....	Sound Object Editor	102

6.1.3.....	Control Movement Editor	103
6.1.4.....	Music Editors	104
6.1.5.....	Control Shape Editor	106
6.1.6.....	The Sequencer	107
6.1.7.....	Device Controller	107
6.1.8.....	Sound Samples as Sound Objects	108
6.1.9.....	The Routing Screen	109
6.2.....	The Virtual Studio	110
6.2.1.....	Dynamic Routing of Audio Signals	110
6.2.2.....	DSP	111
6.2.3.....	The Desktop Studio	111
7.....	Conclusion	113
7.1.....	Chapter summaries	113
7.1.1.....	Chapter 1	113
7.1.2.....	Chapter 2	113
7.1.3.....	Chapter 3	114
7.1.4.....	Chapter 4	114
7.1.5.....	Chapter 5	115
7.1.6.....	Chapter 6	115
7.2.....	Summary	115
Appendix 1.....	Class Descriptions	117
Appendix 2.....	Definitions	121
	References	123

LIST OF FIGURES

Figure	Description	Page
1-1	An eight input stereo mixing console	8
1-2	The signal flow and buss usage of a simple mixing console	10
1-3	The change in frequency response when using EQ	13
2-1	The flow of MIDI information when using a MIDI console with a standard sequencer	24
2-2	The improper handling of punch-in/out	26
2-3	The problem encountered because of a sequencer's inability to chase the control movement data	26
3-1	The graphical representation of a recording console	30
3-2	The screen menu	33
3-3	The device screen menu	35
3-4	The channel editor menu	36
3-5	The various control types	37
3-6	The move operation of the channel editor	37
3-7	The resize operation of the channel editor	38
3-8	The duplicate operation of the channel editor	38
3-9	Changing a font in the channel editor	39
3-10	Scaling the channel editing window	39
3-11	A device editing window	40
3-12	The device editor window menu	41
3-13	The add multiple channel dialog box	42
3-14	The creation of a hypothetical device	44
3-15	The final device created from figure 3-14	45
3-16	The sequencing screen menu	46
3-17	Dragging the needle of a knob and clicking on a push button	47
3-18	The device controller menu	48
3-19	The control movement MIDI message dialog box	49
3-20	Using a device	50

3-21	An example sequencer window	55
3-22	The sequencer window pull down menu	56
3-23	The MIDI sequence editor window and its pull down menu	57
3-24	The current mix pop-up menu	59
3-25	The list of channels affected by the current mix	59
3-26	The Master System Window	62
3-27	The Cue List Editor Window and its pull-down menu	64
4-1	The sound object screen menu	69
4-2	The sound object editor	70
4-3	The sound object editor's pull-down menu	72
4-4	Attaching components in a sound object editor	74
4-5	Cue points in a sound object editor	76
4-6	The sound object 'Car Passing and Crashing'	79
4-7	The 'Chorus' composite sound object.	80
4-8	The 'Song' sound object	81
5-1	The device editing structures	88
5-2	The device controller structures	90
6-1	The control movement editor	104
6-2	A graphical score editor	105
6-3	A piano-roll type score editor	106

LIST OF TABLES

Table	Description	Page
5-1	The list of system classes	84

1. INTRODUCTION

This thesis discusses a recording studio automation system designed to fill the mixdown automation needs of the recording engineer as well as provide a way for the composer/musician to approach the creation of music. The automation system itself uses extensions to current automation concepts to achieve a hardware independent, generic automation system that is seen as the first step toward the virtual studio concept. The key to the compositional aspect of the mixdown automation system is that it incorporates a sound composition tool which allows a composer to integrate his production work into the earliest stages of the composition and performance.

The system described in this thesis is aimed at the combination composer/performer/engineer. The rationale behind this is based on the fact that home-based recording studios are becoming extremely popular. In these types of studios, the owner often assumes the roles of composer, performer, and recording engineer. This system exploits this fact by offering a system which allows much of the production work to be integrated into the composition and performance stages.

This production work is integrated into the composition process through the support of sound objects. Sound objects allow the capturing of both musical information and production (mixdown) information and allow the composer/engineer to treat this information as if it were sound samples. These sound objects can be arranged on time lines to create arbitrary playback sequences and sets of sound objects can be encapsulated into larger more complex sound objects. Sound objects offer a hierarchical approach to the composition of music and sound.

In summary, the system described in this thesis provides a centralized, integrated system for composition and production with a consistent user interface to all of the hardware in the recording studio.

This chapter will look at the recording studio and the recording process. We will look at the equipment used in a recording studio, and discuss the personnel involved in a studio session. Chapter 2 will go into the details of recording automation. Here we will look at the

type, of automation desirable and the advances in technology which make this automation possible. Chapter 3 will look at a solution to the automation problem. This chapter will present the first part of the system, the mixdown automation system. Chapter 4 presents the final part of this thesis, the concept of sound objects. Sound objects are introduced as a method for the encapsulation and naming of performance data (both performed sequence data and “performed” console control movements). Once the performances are captured into sound objects, we can arrange and combine the performances in any way desirable. Chapter 5 discusses the implementation details of the system. The implementation platform, the structure of the code, and various design aspects will be discussed here. Chapter 6 looks at extensions to the system and looks at the feasibility of the virtual recording studio. Chapter 7, the conclusion, summarizes this thesis.

1.1. MUSIC RECORDING

This section will describe the standard studio recording process [MILL] [NAZAR]. It will follow the process from the performance of the music to the production of the final master recording. It will introduce and explain the roles of the studio personnel as well as talk about the differences between small and large studios.

1.1.1. The Recording Process

The recording process starts with the musicians performing the musical material, either simultaneously or separately, while the performance is recorded onto multi-track tape¹ [BARTb] [BARTc]. In general, each instrument is recorded onto a separate track of tape. Note that each instrument may be recorded one at a time, and it is not necessary for the entire group of musicians to perform simultaneously. In fact, one musician can perform more than one part by recording a second part on a separate track while listening to the first part. Using this overdubbing recording technique, it is possible for one single musician to perform a song that it would take many musicians to perform live. Note also that it is not necessary that all the performances be recorded on the same day or even in the same place. An initial set of performances could be recorded in one place, and then the multi-track tape

¹Multitrack tape is magnetic recording tape with many (4, 8, 16, 24) tracks running in parallel. Each track can be recorded on independently of the others, and any group of tracks can be played back at the same time as recording on another group of tracks. Multi-track tape is similar to standard cassette tape, except it is usually wider in order to accommodate a greater number of tracks.

sent to another city for the rest of the performances to be recorded a day or even a month later.

Once all of the tracks have been recorded, it is time to mix the multi-track recording down to the stereo master recording². This process involves playing back each track of the multi-track tape through a separate channel on the mixing console and recording the result on the final stereo mastering device. As the signal passes through the channel on the mixer, we can use the mixer's controls to manipulate the sound (volume, EQ, stereo placement, effects). During the mixdown process, the recording engineer will balance the relative volume levels of the tracks, adjust the EQ appropriately, use the panning control to get the correct stereo placement of each track, and possibly send some of the signals to outboard effects devices for additional processing such as the addition of echo or reverb [BARTd].

1.1.2. Reasons for Multi-track Recording

It may appear that the intermediate process of recording the music onto the multi-track tape is an unnecessary step. After all, why not just ask the musicians to play live and make the final master recording immediately? This way our master recording would be a first generation recording of higher quality because we do not have this intermediate step of recording onto another tape recorder. This method, although having the capability to yield a higher quality product, is used very seldom in the recording industry. There are many reasons for the intermediate recording on multi-track tape. For one, during the initial recording process, the recording engineer can concentrate on getting a technically good sound from each of the instruments. The mixdown process can then be used to try to achieve an artistically good sound and the engineer does not have to worry about the technical aspects as much. If the final master were recorded initially, then the recording engineer must try to achieve both technical and artistic perfection simultaneously.

A second reason for the use of multi-track is that we can provide a second, different mix long after the final product is completed. For instance, we can provide an instrumental version of a song which originally had vocals by just remixing without the vocal tracks. We can also provide different mixes for different applications. For instance it is common practice in the recording industry, for technical reasons, to provide a different mix for release on CD than on LP. Also, a mix designed for play on small cheap portable cassette players may be different from a mix intended for play on a good quality CD system. It is

²Although most master recordings are in stereo, they may also be mono (1-track), or in other formats such as 4 or 6-track for use in movie soundtracks, for example.

also quite common to provide an edited remixed version of some material for use in dance clubs.

Probably the most important reason for the use of multi-track recording is that if one of the performing musicians makes a mistake during recording, it is only necessary to rerecord that musician's performance, and not the entire band's performance. This way we cut down on the number of retakes, since the larger the number of musicians playing simultaneously, the greater the chance that a retake must be done. Other reasons for the use of multi-track recording include the ability to record several takes of many of the musicians' performances and compare them to see which combination of takes sounds better.

1.1.3. The Recording Studio

While electronic instruments, such as synthesizers, drum machines and electric guitars, may be plugged directly into the recording console to be recorded on the tape, many instruments, such as acoustic guitars, pianos, acoustic drums and vocals, must be recorded through a microphone. The recording of these acoustic instruments usually takes place in a sound proof, acoustically dead room to avoid the recording of background noise. Because of this need for acoustic isolation, many recording studios are designed such that the musicians perform in a large, sound proof room (with heavy drapes or sound absorbing materials on the walls) and all the recording equipment is in a separate room. The two rooms are usually separated by a window so that the recording personnel and the musicians may communicate visually. While this type of recording studio design is still the norm, a new type of design is now becoming quite popular for the recording of electronically generated music.

Much of today's music is recorded with only electronic instruments such as synthesizers, samplers and drum machines. Because no acoustic instruments are used, there is no need for an acoustically silent recording studio. This type of music has led to the popularity of a new type of recording studio design with only one room which holds the musicians, the instruments and all the recording equipment [KEY86] [KEY89Oct].

1.1.4. The Studio Personnel

This section will explain the roles of all of the personnel involved in the studio recording process. It will explain the job that each person performs and the interactions between the personnel [NAZAR].

1.1.4.1. The Producer

The producer deals with the artistic aspect of the sound. His job is to direct the recording engineer in the kind of sound he wants to achieve. He will suggest the type of effect desired and the engineer will try to create that effect. While the producer is familiar with the basic concepts of all of the equipment being used, he does not necessarily have a lot of technical knowledge about them. The producer also functions as the director of all activities during a studio session. He provides direction to the performing musicians as well as all the other studio personnel.

1.1.4.2. The Recording Engineer

The recording engineer is the liaison between the producer and the equipment in the recording studio. The recording engineer's main job is to deal with the technical aspects of the sound. He does not really deal with the artistic aspects of the sound. He concentrates on getting as clean (best quality) a signal as possible onto the tape. As the producer's interface to the recording console, the engineer will get requests such as: "make that snare drum more punchy", "make that vocal sound more distant", and the engineer will have to manipulate the EQ and effects in order to achieve the desired effect.

1.1.4.3. The Musician

The musician's main concern is with the performance of the music. He concentrates on getting as good a performance as possible. Each musician must also concentrate on his interactions with the other musicians (or taped tracks) during performance.

1.1.4.4. The Composer

While the composer is not really studio personnel, the role of the composer is covered here for completeness, and to emphasize the distinct roles of the musician and the composer. The composer is the person who creates the music. He is not necessarily the person who plays the music. He usually writes the music in some notational form that can be read by the musicians who will perform the piece.

Note that the abilities of more than one of these roles may be found in more than one person. For instance, it is very common to see the composer of the music do his own performance. Also, it is not uncommon to see a member of a performing band act as musician and producer. In fact, if we take this combining of roles to the extreme, we will

have the same person doing the composing, performing, producing and engineering. This relates to our next subject, the personal studio.

1.1.5. The Small Personal Studio

The past few years have seen a reduction in the cost of music and recording equipment while the quality has increased. This allows enthusiasts to construct small, low cost recording studios of relatively good quality. These small "personal" studios allow someone who does their own composing, performing, producing and engineering to produce recordings without the need for expensive, professional recording studio time. In fact, there has been a tendency in the past few years towards these types of home based studios. Many of today's recordings are started in a home based studio, and only brought in to a larger studio for special equipment usage or final mastering [KEY86].

The small personal studio is an important concept in this thesis since the system presented attains its greatest advantage when used by someone in the combination role of musician/engineer.

1.1.6. The Studio as the Instrument

These small personal home studios have led to a new kind of composition process. In electronic music composition the music is no longer considered a sequence of notes playing on a particular instrument. Instead, the composer wishes to create music through the manipulation of sounds. This type of composition relies heavily on effects created with the entire studio: the instruments, effects units and the mixing console. In essence, the entire studio becomes the instrument.

This area of composition is mentioned here since this is one of the areas which can benefit most from the type of automation described in this thesis. In particular, the concept of sound objects allows the creation of a sound by combining sounds generated via the control of many devices.

1.2. THE RECORDING CONSOLE

This section will explain how the recording console works with examples of its use. It will explain all of the controls usually found on the console as well as the concepts of bussing and monitoring. Many of the concepts such as processing, EQ, and reverb will be mentioned here but will be covered in more detail in the following sections.

The recording console is the heart of the recording studio and not because it is usually the largest piece of equipment in the studio. Every audio signal in the studio is routed through the console. All the microphones, electronic instruments (synthesizers), and effects devices are plugged into the console inputs and can be mixed together and then routed to any combination of the monitor speakers, the musicians monitoring headphones, the multi-track recording tape, and the final stereo mastering tape. The multi-track tape can also be brought into the console inputs for the final mixdown and mastering.

The recording console's main job is to mix the audio signals together and route them to the monitoring system and the multi-track tape. Most consoles also provide some degree of sound modification in the form of parametric EQ (discussed in section 1.3.1) and allow the input sounds to be routed through outboard effects devices (such as reverb or compression).

1.2.1. The Stereo Console

For our initial discussion on recording consoles, let us consider an eight input console designed to mix the inputs to a stereo (two output) signal. A typical console of this configuration is shown in figure 1-1. Notice the main part of the console is divided into eight vertical strips - the input channels. Each channel contains controls which manipulate the sound passing through the channel. The very right end of the console is the output section. This area contains the controls pertaining to the global operation of the console. We will first discuss the controls found on the input channels. Keep in mind that the sound from the input of each channel passes through each of the control's electronics and eventually is mixed in with the sound from all the other inputs on the stereo buss³ and finally appears at the stereo outputs.

The control at the bottom of the console is the fader. This controls the volume of the sound. This is perhaps the most used control and is therefore located in the most convenient spot: closest to the recording engineer.

³A buss is an electronic circuit which sums all the electrical signals put onto it. In essence, it is the entity actually performing the mixing of the sounds.

Figure 1-1 - An eight input stereo mixing console.

Another important property of a sound during mixdown, besides its volume, is the stereo placement of the signal. The pan knob controls this stereo placement. With the pan knob turned all the way counter clockwise, the sound will appear only in the left speaker. With the pan knob turned all the way clockwise, the sound will appear only in the right speaker. With the pan knob in the center of its rotation, the sound from that channel will appear in the center of the stereo field (i.e. the sound will come equally from both speakers). The sound can also be placed elsewhere in the stereo field by turning the knob to an intermediate position not fully clockwise or counter clockwise.

We can alter (color) a sound using the EQ section of the console. Most consoles offer some sort of EQ, either fully parametric, semi-parametric, or fixed frequency EQ (more details on EQ can be found in section 1.3.1). The most common type is semi-parametric EQ (as shown in the figure) which allows control over the center frequency and the gain. A console will usually contain two to four bands of semi-parametric EQ where the frequency ranges covered by the bands are mostly disjoint but usually overlap a little. The sound will pass through the EQ section before it is put onto the stereo buss. Many consoles have a switch to completely by-pass the EQ section to prevent any coloration of the sound by the EQ circuitry.

Finally, the control which facilitates the most creativity is the auxiliary (or effects) send. This allows the input signal for each channel to be mixed together and sent to an external

device. This external device processes the sound and returns a new sound which is brought into the console through the auxiliary returns or through an unused channel on the console. External (or outboard) effects devices can be used to perform additional processing from what the console offers. Examples of these types of processing are: graphic equalization, reverb, echo, and compression. Processing will be covered later in section 1.3. Many consoles offer a switch which allows the auxiliary send to be pre or post fader. That is, the signal sent to the effects device will either be attenuated by the fader section (post-fader), or just left at its full level (pre-fader)⁴.

The other controls on the console include: the trim knob which controls the gain of the input preamplifier and is used to match the level of the input signal to the level required by the electronics of the console; the solo button which, when pressed, will turn off all the other channels' sounds so that only the current channel will be heard (this is done temporarily in order to isolate the sound of one particular channel for quick editing without interference from other channels); and, the mute button which will remove the current channel's sound from the stereo buss.

Most consoles are equipped with some kind of metering device (VU meters, or LED peek meters) so that the engineer can set the levels of the signals to ensure that the electronics are operating at an optimum level. The meters are either on a per channel basis or just for the final stereo output.

The output section of the console contains the controls which affect the overall output level, the auxiliary return levels, and the stereo placement of each of the auxiliary returns. The master fader(s) controls the overall output level. This is the volume of the stereo buss which is output at the master stereo outputs. The auxiliary return controls the level of the signal from the outboard effects device which is put onto the stereo buss. There may also be a panning control for each of the auxiliary returns which controls the stereo placement of the auxiliary return signal. Figure 1-2 shows the signal flow in a simple mixing console along with the busses used.

⁴In the post fader position, the signal sent to the auxiliary device is derived from the output of the fader circuitry from that channel. In this case the signal will be attenuated by the fader. In the pre fader position, the auxiliary signal is derived before the effect of the fader circuitry and the resultant signal is therefore not attenuated by the fader and is left at full volume.

Figure 1-2 - The signal flow and buss usage of a simple mixing console. Note that the input section will be repeated many times (once for each input channel). Also, there will be an auxiliary buss for each auxiliary output channel.

1.2.2. Multiple Buss Console

While it may be common to see stereo consoles in a live performance situation, they are usually not found in the professional studio control room. Their use is usually limited to the few applications where only one stereo monitoring system is required or as use as a submixer⁵ in a live performance or studio situation. It is quite uncommon to see a stereo only console in a professional studio environment. In order to facilitate the multi-track tape recorder and the monitoring needs of the engineer and the musicians, it is necessary to have

⁵The submixer mixes together one section of the instruments. This submix is then sent to the main console where it will be routed to the main speakers. An example of a submix situation would be when a keyboard player with many synthesizers wants to mix all of his instruments together so that he can control the balance and processing of the sounds. He can then provide the main console with a stereo mix of all of the keyboards. He can also provide himself with a stereo monitor of this submix.

a console with a multiple buss architecture⁶. The busses are used for routing the output of a channel to various places. The channel can be routed to one or more tracks on the multi-track tape recorder, it can be routed to the musicians monitoring headphones, it can be routed to the studio monitoring speakers, or any combination of these. A typical professional console will have one buss for every track of tape, a stereo monitor buss, and possibly additional monitoring busses for the musicians. Each channel will be equipped with push button switches (one for each buss) to add the output of that channel onto the selected busses. The output section may also contain gain controls which will control the volume of the signal being output at the buss outputs. There may also be gain controls which control the amount of signal from the tape busses which is put onto the stereo monitoring buss. This way the recording engineer can monitor what is being recorded on the tape.

One last difference between the stereo console described above and the professional studio console will be a set of inputs on the studio console to mix and monitor the output of the tracks from the multi-track tape recorder. Most professional consoles will have a separate set of inputs (similar to ordinary channel inputs) for this purpose. The semi-professional consoles may cheat a bit by offering a switch on the regular input channels which switches the input of that channel between the instrument connected there and the corresponding track on the multi-track tape recorder. Consoles equipped with this ability can get by with half the number of channels as their professional counterparts, but lack the ability to mix (and process) all of the tape tracks and all of the console inputs simultaneously.

1.3. SPECIAL EFFECTS PROCESSING

This section will explain the common types of processing which may be applied to signals during the recording or final mixdown processes [EARG]. The two reasons to apply processing to a sound are technical and artistic. Technical processing would include processing a sound in order to increase the quality of that sound. For example, we may want to reduce some unwanted portion of the signal such as noise. Artistic processing is

⁶Monitoring refers to the musician listening to what he is playing (usually through headphones). The recording engineer will give the musician a rough mix of what is currently on tape with the musician's instrument also in this mix. The monitor mix usually differs from the final mix. For instance, the engineer may have to increase the level of the musician's instrument in the monitor mix in order for the musician to hear himself clearly.

done to change a sound into some other type of sound or to modify the sounds in a mix so that they sound better together.

The most common types of processing are: equalization, reverb, and compression/expansion. These will be explained in detail in the following sections.

1.3.1. Equalization

Equalization, or EQ, is the increasing or decreasing of some of the frequencies of the original signal [BARTa]. EQ is probably the most common type of processing, and as such, some type of EQ is usually found built in to every recording console. The most common type of EQ is parametric. This is the type of EQ usually found on consoles. Fully parametric EQ consists of three parameters: frequency, gain and bandwidth. The frequency parameter indicates the center of the frequency range affected by the equalization. The gain parameter indicates how much to cut (decrease) or boost (increase) the given frequency in the source sound. The gain parameter is measured in decibels. The bandwidth parameter, also called Q or quality factor, indicates how big a frequency range, around the center frequency, will be affected. It is common for many consoles to fix the value of the bandwidth parameter, and allow only the two parameters frequency and gain. This type of EQ is called semi-parametric EQ. Still other consoles offer only a fixed frequency EQ which only allows control over the gain parameter. Figure 1-3 shows an example of the change in frequency response of a signal when using EQ.

Another type of EQ is graphic equalization. A graphic equalizer divides the audible frequency spectrum into many frequency ranges and provides a control to change the gain of each of these frequency bands. Note that if a particular frequency must be cut/boosted with a graphic equalizer, it will affect all the frequencies falling within that frequency band. Graphic equalizers are useful if many frequency bands must be changed over the entire spectrum. They are useful for moulding a frequency response contour. Whereas, parametric equalization is better for jobs requiring the modification of a specific frequency.

One other type of EQ is shelving. Shelving EQ provides two parameters: frequency and gain. With shelving EQ, the gain will be applied to all frequencies above or below the given frequency.

Figure 1-3 - The change in frequency response when boosting at a center frequency of 1 kHz.

1.3.1.1. Using Equalization

We will now discuss some of the reasons for using EQ along with some practical examples. As was discussed earlier, there are two reasons to use EQ: technical and artistic. We will first discuss some of the technical uses for EQ. The technical uses of EQ involve trying to fix up some problem with the sound. For example, it is quite common for sampler manufacturers to try to reduce the cost of samplers by providing cheaper output filters in the digital to analog conversion circuitry. This often results in the unwanted appearance of extra high frequencies caused by the aliasing of the lower frequencies during this conversion process. We can remove these unwanted high frequencies by passing the output of the sampler through a shelving EQ set to eliminate the unwanted high frequencies.

Another use for EQ might be to eliminate an unwanted 60 Hertz buzz from a synthesizer signal, which was introduced due to faulty power supply conditioning or a faulty grounding circuit. In this case we could use parametric EQ to cut the 60 Hertz signal enough to make it inaudible. A third use for EQ might be to eliminate high frequency hiss from the playback of a low grade tape. In this case, too, we can just use shelving EQ to eliminate the high frequencies above a certain point. A final example for technical uses of EQ involves the recording of a voice through a microphone. Many microphones exhibit an effect called proximity effect, which is the boosting of low frequency signals when the microphone is very close to the source [BARTe]. In many cases this can be a desirable effect. But if it is not desired, and we have a recorded vocal track on tape which was

recorded too close to the microphone and are not in a position to rerecord the vocal track, we can use EQ to fix it. We can use a graphic equalizer to reduce the general bass content of the sound, thereby reducing the appearance of the proximity effect. We can also achieve a similar effect with parametric EQ using a large bandwidth and set to decrease the low frequencies.

Some other technical reasons for the use of EQ would be to compensate for the poor frequency response of an audio system, or to compensate for room acoustics.

An artistic use for EQ would be as a special effect. We can use a graphic equalizer or two bands of shelving equalizers to cut the high and low frequencies of a piano to make it seem like a cheap or toy piano. We can use parametric EQ to boost the mid frequencies of a snare drum to make it sound "punchier". And finally, we can use EQ when combining many sounds during final mixdown to achieve a good blending of the sounds.

Finally, an example of a use for EQ that falls between technical and artistic occurs when mixing sounds together. Consider the case of mixing a vocal track and a piano track. It is possible to achieve a case where the vocals seem to be loud but yet hard to understand. One fix to this problem is just to increase the level of the vocal track. While making the vocal track understandable, this can make the vocal track seem "too loud". Another approach to this problem is to use EQ. We can use parametric EQ to cut the frequencies from the piano track in the same area where the main frequencies reside in the vocal track. In practice, this process will not have much affect on the piano sound but will now cause the vocals to become much more understandable without affecting the general balance between the levels of the two sounds.

1.3.2. Reverberation

One very important type of processing is the addition of artificial reverberation to a sound. Reverberation (or reverb) refers to the ambient characteristics of a sound within a room (the echoes). It is a common practice to add "life" to a sound by adding artificial room ambience.

When instruments are recorded from a microphone, it is desirable to record with the microphone as close as possible to the instrument. This way we can eliminate the recording of background noise, musicians shuffling their feet, the turning pages of sheet music, and other extraneous noise. But when instruments are recorded in this manner, we do not record the actual room ambience of the instrument. In this case, it is often desirable to introduce some artificial ambience to the sound. In fact, it is often the case where we

specifically record an instrument with no ambient sound so that we can create, artificially, our own ambient environment.

Electronic instruments, such as synthesizers, samplers and drum machines, generally plug directly into the recording console and they are therefore recorded with no room ambience. This often leads to a very dead, lifeless sound. Here, too, is another case where artificial ambience must be added.

To add artificial ambience to a sound we use reverberation devices. The most popular types of reverb devices are DSP⁷ based devices which can create an artificial acoustic environment and place any sound in that environment. With these devices, we can create a complete acoustic environment to place our sound in. We can create a room of a certain size, with walls of a certain type of material (sound absorbing or reflecting), and we can place our sound source as well as our listener at any point in the room. The device will then generate, in realtime, the sound which the listener would hear if he were in this acoustic environment.

1.3.3. Compression/expansion

Compression refers to the reduction of the dynamic range of a sound. Dynamic range is the range from the loudest to the softest sound (usually measured in decibels). Many instruments, such as guitars, vocals, and drums, have a very wide dynamic range and contain many transient spikes. A singer's loudness may vary over a large range. A compressor is used to control the amplitude of extremely dynamic sources by dynamic attenuation of the signal.

One typical use of a compressor is during the recording of vocals. Vocals typically vary greatly in volume. A singer may emphasize the emotion of a sung phrase by singing louder. Variations in the distance between the singer and the microphone (which is on the order of inches) can cause wide variations in the recorded amplitude. Common practice is to record the vocals through a compressor to reduce the variation in the singer's amplitude.

Another example is in the recording of drums. Drums tend to have very high transient levels for the first few milliseconds of the sound and then level off to a constant level for a few hundred more milliseconds. These initial transients can be very damaging to amplifiers

⁷DSP chips are RISC architecture Digital Signal Processors specially designed to solve the problems typically found in high quality digital audio processing. High quality audio applications require the high speed manipulation of high precision (16-48 bit) numbers. A typical DSP chip can, in one instruction, fetch two 24 bit numbers from a table indexed by two index registers, multiply the two numbers together, add the result to an accumulator, and increment the two index registers. Typical speeds for these processors allow execution of 10 million of these types of instructions per second.

and speakers. We can use a compressor to reduce the attack transients on these drum sounds, thereby increasing the average output level of the sounds without overloading the amplification system.

LP records have a very limited dynamic range. Therefore, recordings destined for reproduction on LP records must be heavily compressed. In this way the compressor can help put a higher overall level on the tape by controlling the dynamics of the input signal. This increases the level at which the signal can be recorded which in turn reduces the amount of noise inherent in the playback of the track.

Expansion is the reverse process of compression. An expander will increase the dynamic range of a signal. It can be used to exaggerate the dynamics of instruments. It can also be used to restore the dynamic range of a signal from tape which was originally compressed in order to record the signal at a higher level. In fact, this is the basic principle behind many tape noise reduction systems such as dbx® [KEY89Sep].

1.3.4. Other types of processing

There are many other types of processing besides EQ, reverb, and compression/expansion. Some of the more common effects are: delay and echo, pitch shifting (actually changing the pitch of a sound without changing the speed of playback), chorusing and flanging (a very fast echo effect which gives a moving quality to a sound), excitation (the process of dynamically manipulating the phase of a signal to give the appearance of the addition of high frequencies), and auto-panning (the panning of the signal continuously from the left to the right speaker and back). These are only a few of the many types of effects available. This discussion was intended to give an appreciation for the complexity of production work.

Due to the introduction of general purpose DSP chips, a whole range of DSP based musical processing devices have appeared on the market. It is common to see an affordable digital effects device capable of performing one or more of the effects described above.

1.4. MIDI

MIDI is a unidirectional serial communications specification designed to pass information between electronic music equipment. MIDI stands for Musical Instrument Digital Interface. The MIDI specification [IMA] is a document agreed upon by the music equipment manufacturers: the MMA (the MIDI Manufacturers Association) and the JMSC (Japan MIDI Standards Committee). The MIDI specification first appeared in 1981

[MOOG] and additions are constantly being added to it as manufacturers find more and more uses for it [ANDER] [BOOM] [MOORE].

The communication of information on MIDI is carried out by the transmission of one, two or three byte messages, as well as some special purpose arbitrarily long messages. It communicates information at a rate of 31,250 bits-per-second. The MIDI specification details the use of three connectors: MIDI In, MIDI Out, and MIDI Thru. The MIDI In connector is used for receiving MIDI information. The MIDI Out connector is used for transmitting information to another device's MIDI In port. The MIDI Thru port is used to transmit the information, received at the MIDI In port, on to other devices. This is used for daisy chaining many instruments together.

Note that the intent of MIDI is not for the real-time transmission of digital audio information, but for the transmission of musical events. For instance, the most common type of MIDI message is the **note on** command. This is sent from the MIDI Out port of the keyboard when a key is pressed. It is sent to the MIDI In port of a synthesizer which, upon reception, will start playing the note. When the key is released, it will transmit a **note off** command which will cause the receiving synthesizer to stop playing the note. Other information which can be transmitted over MIDI, besides note information, includes: key pressure (how hard you hold the keys down on the keyboard), pitch bend wheel position, program or preset (which sound is being used) changes, sustain pedal position, timing and synchronization messages, and messages to re-initialise the equipment or set the equipment to a certain state. In addition to the above standard messages, each manufacturer can define their own specific messages for the communication of information between their own devices.

The MIDI specification defines sixteen logical channels over which the messages are transmitted. A particular synthesizer can be set to receive on any or all of the sixteen MIDI channels. As well, a master MIDI keyboard can be set to transmit on any of the sixteen MIDI channels (usually only one channel at a time, but in some cases more than one at a time). Suppose there are a group of synthesizers all daisy chained through their MIDI Thru ports and each is set to receive on a separate MIDI channel. By changing the channel which the MIDI keyboard is transmitting on, the MIDI information can be directed to any one of the synthesizers in the chain. During a live performance, this technique can be used to access any one of a group of synthesizers from a master keyboard. If the keyboard is replaced with a MIDI equipped computer, a multipart score can be played by sending each part to a separate synthesizer. Using this technique one person can use the computer to

achieve performances that would normally not be possible for one musician to perform. This is discussed in further detail in section 1.5.

1.4.1. MIDI Controllers and Receivers

MIDI Controllers are devices which transmit MIDI information. The most common type of MIDI controller is a standard synthesizer (or piano) type keyboard. This was the easiest and obvious instrument to adapt as a MIDI controller and hence was the first type of MIDI controller to appear on the market.

Another type of MIDI controller is MIDI drum pads. Drum pads are specially designed, drum-like pads, with velocity and position sensing hardware built into them. They sense how hard and where on the surface a drummer hits them, and send out a programmable MIDI message each time they are struck.

Another type of MIDI transmitting device is the pitch to MIDI converter. This device will convert standard audio information from any monophonic instrument, such as a human voice or a wind instrument, into MIDI information. Special pitch to MIDI converters also exist for guitars. These devices have a pickup mounted near each string of the guitar, and function as six monophonic pitch to MIDI converters.

There are many other types of MIDI controllers such as saxophone-like and trumpet-like MIDI controllers. These controllers are specially designed devices that work like the traditional instrument but instead of actually producing sound, produce MIDI information. These devices open up the infinite world of synthesizer sounds to musicians trained in the performance of traditional instruments.

There are many types of devices which receive MIDI information for the purpose of producing sound. The most obvious is the synthesizer. Other devices include samplers, drum machines, and digital piano modules.

Combining the different types of controllers and receivers, it is possible for a full MIDI equipped band to give an untraditional performance. The keyboard player would have his MIDI output connected to the MIDI input of a drum machine and could play all the drum parts from the keyboard. The drummer would have his MIDI drum pads driving a sampler programmed for special sound effects and orchestral stabs. The MIDI equipped wind player would be playing a bass patch from the synthesizer. And finally, the guitar player would use his pitch to MIDI equipped guitar to play flute sounds on a second synthesizer.

1.4.2. Other MIDI Devices

Many other types of devices, besides musical instruments, send and receive MIDI information. Effects devices (devices that perform reverb, echo, etc.) are often equipped with a MIDI input to accept program or parameter changes. These parameter changes can be used to modify the effect of the device in realtime. There are also MIDI equipped lighting controllers that allow a keyboard or computer to control the sequencing of lights and visual effects during a live concert.

Perhaps the most interesting and unexploited MIDI device is the MIDI controlled audio mixing console. This is generally a specially designed mixing console with a MIDI interface that allows the transmission of MIDI information whenever the controls on the console are moved. It will also allow the remote control of the console through the reception of MIDI commands. The received commands can cause the console's controls to actually change position (if the knobs are motorized). If the knobs are not motorized, then only the value of the control is changed (not the position). In either case, the sound going through the console will be affected. Through these types of MIDI equipped mixing consoles, it is possible to create a relatively inexpensive automated production system. As we will see later, these types of consoles (and MIDI controlled effects devices) are the basic technological innovations which make the system described in this thesis possible.

1.5. SEQUENCERS

Sequencers are software products for personal computers (or special purpose hardware devices augmented with built-in software) for recording and playing back MIDI information in realtime [AIKIN]. They are generally designed to simulate the multi-track recorder concept. The user is allowed to record a MIDI performance on any one of the sequencer's tracks. The user can then play back that track on one of the sixteen MIDI channels while recording another part on another track. The user can build up a performance consisting of many parts, and have the entire multi-track sequence play back. Basically sequencers are just multi-track recorders for MIDI information.

Many sequencers contain additional features (or utility programs) for editing and arranging the tracks. For instance, Vision™ (Opcode Systems, Inc.) includes an editor for editing the actual MIDI events contained in a track. This editor allows the user to add and delete events as well as change the value or time of an event. Another sequencing package, Music-X™ (Microillusions), lets the user edit sequences using a piano roll type editor. Notes can be added or removed as well as moved around in time. The length of notes can

be changed by stretching or shrinking the note on the screen. A wrong note can be changed into the proper note by just shifting the note vertically to the proper position.

1.6. SUMMARY

In this chapter we have looked at the recording studio and the recording process. We have seen the devices used during the recording process: the instruments, recording console, multi-track recorder, and effects devices. We have also discussed the studio personnel and their roles in the recording process. Finally, we have looked at MIDI and sequencers: the key technologies behind current automation systems.

From the description of the recording process, it should be obvious that there is a great need for some kind of automation. The recording engineer must manage a great deal of information during recording. The recording engineer, and his assistants, must control all of the devices in the studio in real-time. As the number of devices to be controlled increases, the probability of errors increases exponentially. Therefore, it is necessary to introduce some type of automation into the recording process which decreases the probability of error. This can be accomplished by making the perfected portions of the recording process easily reproducible.

In the following chapters, we will look at studio automation in more detail. We will see current solutions to the automation problem, and then present another solution to the problem. This solution is introduced as automated control of the mixing console only, and is then extended to the control of the entire studio.

2. SURVEY OF CURRENT SYSTEMS

2.1. CURRENT AUTOMATION TECHNIQUES

It is clear, from the discussion in the previous chapter, that there exists a great necessity for some kind of automated control (record/playback of movements) of the mixing console. In fact, many pieces of equipment (tape recorders, synthesizers, effects devices) in a recording situation could benefit from this type of automated control.

This thesis discusses a system which solves these automation needs. It also introduces the concept of sound objects: an encapsulation of sequenced control movements and musical data into a named entity.

One particular motivation for recording studio automation is described in [AES] overview of the 2nd AES International Conference:

“...B. Morgan Martin presented “The Digital Mixing Console” in which he reviewed the operational constraints of the analog console, such as the time and effort needed to preserve the system settings from one session to the next. In particularly complex situations, it may be necessary to resort to the “lockout” - closing the studio to other business - in order to preserve the settings while personnel take a break between sessions. If this is impossible, the sea of knobs might be photographed, so that the engineer can recall the settings later on.

By contrast, the entire status of a digital console may be stored on floppy disk and reset hours, days, or weeks later.”

2.1.1. Automated Analog Mixing Consoles

Before we begin our discussion of the automation system, let us look at recent developments in products which make this automation possible. Many standard analog consoles can be retrofitted with an option to motorize the faders. The fader positions can be read or changed by a microprocessor based controller unit. Automation is achieved during

mixdown by synchronizing the controller unit with the playback of the tape. Any movements of the faders by the recording engineer will be encoded digitally on a track of the tape (or on a floppy/hard disk). When the tape is played back, the controller unit again synchronizes with the tape and performs the movements on the faders which were previously recorded. On this second pass, the recording engineer can make additional changes to the settings of the faders. These changes are merged with the previously recorded movements and the result is stored on a second track of tape (or a second file on the floppy/hard disk). This process can be repeated indefinitely producing a mix which would require many engineer's assistants to achieve in realtime. Quite often, after many hours in the studio trying to produce a final mixdown, the engineer can lose perspective of the original goals. With an automated mixdown system, the system can keep track of each stage of the mixdown, so comparisons can be made to an earlier mix. If the earlier mix is found to be better, the engineer can go back to that mix.

Note that equipping the faders with motors adds a great deal to the usefulness of the system at the expense of increased cost. Many automated consoles exist which reduce cost by removing the motorized option from the faders. In these consoles, the controller unit can still read the position of the fader, but when the controller changes the value of the fader, the fader does not move to reflect its new value.

Even though the fader is the most obvious control to automate, we do not need to restrict ourselves to only fader automation. Indeed, many consoles offer both motorized effects (auxiliary) send controls in addition to motorized faders. It is very rare to see a console which has been completely automated with every control being motorized, as this would be very expensive to produce and maintain.

2.1.2. Automated Digital Recording Consoles

The newly developed digital recording consoles are similar to their analog counterparts, except they perform all the operations digitally⁸. Another difference between digital and analog consoles is that the digital consoles often have very few knobs. Digital consoles are designed to allow the user access to its parameters (virtual knobs) through a series of menus or by mapping the small number of knobs that do exist to a subset of these parameters. Similar to analog consoles, digital consoles may be equipped with either motorized or non-motorized knobs. Some digital consoles are even designed with no knobs

⁸Digital consoles perform the mixing and EQ operations digitally (internally). All external connections are still analog, although some consoles offer digital (such as AES/EBU, S/PDIF) inputs and outputs for interfacing to digital instruments and tape recorders.

at all and are intended to be "programmed" via MIDI from an external MIDI keyboard or computer. The fact that there are very few knobs on these digital consoles makes them very hard to use in real-time. The usual method of using this type of console involves pre-programming many "scenes" (a setting of the parameters which can be stored in an internal memory for later recall) which can then be recalled from front panel buttons or via MIDI commands. Using this type of console, a group of performers can achieve automated mixdown of a live performance by having someone change scenes at the appropriate time. These consoles can also be used in a studio mixdown situation by pre-programming the scenes and having the recording engineer switch the scenes at the appropriate time. Since this type of console are scene (snap-shot) oriented, it makes continuous control of the parameters virtually impossible without the intervention of an external computer.

2.1.3. MIDI Equipped Digital Consoles

Most digital consoles are equipped with a MIDI interface which allows realtime access to the parameters in the console via MIDI codes. The intent of the MIDI interface is so that the console can be used with a MIDI sequencer (on an external computer). The sequencer can be used to record continuously changing values of the parameters and thus allow a higher degree of automation than the scene concept. The normal method of automated mixdown is to synchronize the sequencer with the source tape recorder (if mixing down from tape). The sequencer is set to record incoming data from the MIDI mixing console. When the tape starts playing, any movement of the knobs on the console is transmitted out the MIDI port as a note. Since sequencers deal with note data, each parameter in the console is mapped to one of the 128 MIDI note numbers (or one of the 128 control changes). If we wish to add additional mixing changes we need to route the MIDI output of the sequencer track just recorded into the input of the mixing console. The MIDI output of the mixing console is routed into the input of the sequencer. When the tape is started, the previously recorded control movements are transmitted (as MIDI note data) from the sequencer to the mixing console. Any new control movements on the console are recorded on a second track of the sequencer. One thing to note, is that the mixing console must also transmit the incoming MIDI data (from the sequencer) out its MIDI out port, in addition to any new data created by control movements. This allows the single recorded track on the sequencer to contain the MIDI data from both overdubs of the mix. The result is a merging of the two data streams: the MIDI input, and the MIDI data created by moving the controls on the console. Figure 2-1 graphically depicts the operations just described.

a) Movements made to fader 1 are transmitted out the console's MIDI out port and recorded in the sequencer on track 1. Note that this diagram only shows the MIDI data paths. It does not show the audio connections from the multitrack tape recorder to the console or the audio connections between the tape recorder and the sequencer (necessary for synchronization between the tape and the sequencer).

b) The previously recorded movements on track 1 are sent out the sequencer's MIDI out port. They are received at the console's MIDI in port. The MIDI stream created by moving fader 2 is merged with the stream coming in the MIDI in port. The result is sent to the sequencer and recorded on a second track.

Figure 2-1 - The flow of MIDI information when using a MIDI console with a standard sequencer.

Since these MIDIED consoles are designed to run independently of the sequencing package and the sequencer and the console are limited to communications in the MIDI protocol, the console only knows a limited amount of information about the sequencing process. This leads to two problems. The first is the inability to handle punch-in and punch-out properly. Punch-in is when you wish to record something new during some

time subset of the original material. The newly recorded material will take the place of the old material. Punch-out occurs when you finish recording the new material and you return to the action of the old material⁹.

The first problem with punch-in is that many automated consoles make no attempt to handle it. The biggest problem, though, is that MIDI does not support any kind of communication between the console and the sequencer which will accommodate punch-in/punch-out. Therefore, the console must have all the intelligence relating to punch-in/punch-out and must communicate the result back to the sequencer. The disappointing part is that most consoles don't have this intelligence. Figure 2-2 is used to clarify this concept. Figure 2-2a shows the initial recording of fader 1. During the second recording (Figure 2-2b), we punch-in at time t_1 , move fader 1 higher and then punch-out at time t_2 . At the punch-out point, the fader should return to the previously recorded lower level. This requires the console to have a separate memory location for each parameter on the console so that when a control is punched out, it can return the last position which was sent from the external controller. Most consoles do not have this capability.

The second problem is the inability of most sequencing packages to chase the control movements of a track when starting playback of a sequence somewhere in the middle of the track. MIDI commands are transmitted from the console only when the controls are moved. These commands are recorded in the sequencers memory as is. When a sequencer is started somewhere in the middle of a track, the controls on the console are not updated to their proper position until the next event has been reached for each control. Standard sequencers are not capable of knowing that the data recorded is positional data from a console since the data was transmitted as MIDI note data. For example, in figure 2-3, the sequencer is started at time t_1 . With a standard sequencer, fader 1 will not change position until time t_2 . From time t_1 to t_2 , the position of the fader remains at whatever position it had been when the sequencer was started.

⁹Punch-in comes from the world of tape recording. Suppose there was a musician's performance recorded on tape, and most of the performance was fine except for some small piece in the middle. Rather than rerecord the whole performance the section which is in error is rerecorded. To do this, the tape is played back, and at the beginning of the section to be rerecorded, we will punch-in the record mode. At the end of the segment, we will punch-out of the record mode

Figure 2-2 - The improper handling of punch-in/out.

If a sequencer does not chase the fader movement commands, starting the sequencer in the middle of a track may leave the controls at improper positions

Figure 2-3 - The problem encountered because of a sequencer's inability to chase the control movement data.

Using automated consoles with sequencers poses many problems. These problems are the result of the fact that sequencers are designed to record mostly note data. They are usually not well equipped to handle information from a MIDI console. In addition to the

problems discussed above, there is the fact that sequencers usually don't offer any graphical editing of these recorded control movements.

2.1.4. MIDI controlled VCAs

No discussion of mixdown automation would be complete without mentioning the cheapest solution to mixdown automation: MIDI controlled VCAs (voltage controlled attenuators). These units have an audio input, an audio output and a MIDI input. Their purpose is to pass the audio input to the audio output after attenuating the signal by an amount which can be set via MIDI commands. Using these units, some degree of automation can be achieved in a standard analog recording studio by placing these units in any analog signal path which needs to be controlled by an external computer. For instance, fader automation can be achieved by placing a bank of MIDI controlled VCAs in the inputs of the console and taking the signal that would normally be put into the console and rerouting it into the corresponding MIDI controlled VCA. The faders on the console would be left at maximum, and each VCA would take the place of the fader. The VCAs would be controlled by an external MIDI device and the changes could be recorded in a sequencer and played back at a later time as described above. MIDI controlled VCAs can also be used to automate any control where the console offers the ability to place an outboard unit in the direct signal path. MIDI controlled VCAs offer the small scale studio an opportunity to provide mixdown automation without the expense of a fully automated console with motorized controls. Of course for this scheme to work elegantly, a good computer interface is essential.

2.1.5. Summary

This section has discussed recent developments in products which make mixdown automation possible. These include ordinary analog consoles retrofitted with motorized faders and MIDI equipped digital consoles with and without motorized faders. These also include MIDI controlled VCAs, devices without any user input mechanism at all. The two modes of operation of these consoles was also discussed. The first mode is the scene or snap-shot concept, where the user stores the setting of all the controls in an internal memory of the unit and can recall the whole snap-shot at the touch of a button. The second mode is when they are used in conjunction with a standard sequencer. The sequencer can be used to record the control movements and play them back at a later time. Also discussed were the many problems of using automated consoles with sequencers, since standard

sequencers don't have the intelligent data handling required by the consoles, and can't provide all the operations required by a user of this type of automation system.

This thesis presents a system which solves the problems inherent in current automation systems. The system will handle all of the console types described above, and is especially useful for consoles (and MIDI controlled VCAs) which have no external controls.

2.2. SOUND OBJECTS

Sound objects facilitate the integration of the automated production work into the compositional process. Sound objects allow the capturing of both musical (MIDI) information and production information and allow the composer/engineer to treat this information as if it were sound samples. These sound objects can be arranged on time lines to create arbitrary playback sequences. Groups of sound objects can be encapsulated into larger more complex sound objects. Sound objects offer a hierarchical approach to the composition of music.

The notion of sound objects introduced in this thesis is similar to that developed in [SCAL]. [SCAL] describes Kyma, an object-oriented environment for music composition. Kyma's sound objects allow the combining of other sound objects into a named entity. [SCAL] describes Kyma as:

"Everything in Kyma, from a single timbre to the structure of an entire composition, is a Sound Object. These Sound objects can be manipulated, transformed and combined into new Sound Objects. Furthermore, Sound objects can be continuously redefined as work on a composition progresses; objects which were once encapsulated in other objects can be brought to the top level, top level objects can be combined and hidden within a yet higher level object."

While the encapsulation notion of Kyma's sound objects is similar to the one presented in this thesis, Kyma differs from our system in that its sound objects represent a computational (specified numerically or algorithmically) method for the creation of sound. Sound objects, as presented in this thesis, represent a capturing of actual performance information.

3. THE AUTOMATION SYSTEM

3.1. THE COMPUTER AIDED AUDIO PRODUCTION SYSTEM

As described in the previous chapters, this thesis presents a software system to handle the automation needs of the recording engineer. In addition, this thesis explores a new way of looking at and dealing with the creation of segments of music (sound).

This chapter will look at a solution to the automation problem. It will discuss an automation system designed to fill the mixdown automation needs of the recording engineer as well as providing a base for sound object creation. Sound objects themselves will not be discussed until the next chapter.

3.1.1. Device Representation

The first part of the automation system is the graphical representation of the console. As discussed previously, many consoles have very few controls. Some offer user access to the parameters via a system of front panel controls and menus. Some only offer control of their parameters via MIDI commands. Because of this poor user interface, the first thing we will do is present a system which will replace the user interface of the consoles with a graphical interface on a computer.

We wish to emulate a standard recording console on the screen of the computer. To do this we will offer the user a graphical representation of the console (similar to that shown in figure 3-1) and allow the user to manipulate the controls using the mouse.

In order to create a graphical representation of the console, it is necessary to offer the user (or the device interface designer) an editing environment in which to design the representation of the console and a way to associate each graphical knob with the actual parameters of the console. One approach to this is to offer the user a graphical entry system where the user can place objects (controls) on a window. An editor similar to MacDraw II (© 1988 Claris Corporation) would be an appropriate type of interface. The editor would

offer a series of primitives: sliders, rotary knobs, and push buttons. The user could select a primitive, and place it on the console window. The control primitives could be moved around and once they were on the window. The editor could also offer primitives for text and additional graphical objects such as lines and rectangles.

Figure 3-1 - The graphical representation of a recording console.

At this point we should make note of the fact that all consoles consist of many duplicated entities: the channels. For example, in figure 3-1 above, there are eight identical input channels and one output channel. Using the editor just described, we would need to offer some kind of duplicate facility where we could select one channel, once it had been created, and duplicate it many times. This approach has two basic problems. The first problem is that if we wish to correct a mistake made on the original channel, such as changing the position of a control, we have to make this change to all the channels which we created during the duplicate operation. The second problem will be made more clear later on in this thesis, but is basically that we do not have the ability to reference the channels as a whole (as an abstract entity from software). These problems are a result of the fact that during the duplicate operation, we lose the ties to the original object we had

duplicated. The second approach to user creation of the console attempts to solve this problem.

The approach taken in this thesis is similar to the editor described above with an additional abstraction. The approach taken here is to offer an editor, as described above, to lay the controls down on a channel. Once the channel is created, the user can then put the channels onto the final device using another editor. The remainder of this chapter will be devoted to a discussion of an actual implementation of a system which uses this approach to designing devices.

3.1.2. Introduction to the System

The automation system is built using Smalltalk/V on a Macintosh II computer. A great deal of effort has gone into making the system look and feel like any other Macintosh application. This is so that users accustomed to the Macintosh interface can learn to use the system with relative ease. When learning to use the system, new users can concentrate on learning the system specific operations and do not have to concentrate on learning mechanical details of mouse and window operations. For details about the underlying Smalltalk implementation of the system, refer to chapter five.

This chapter will discuss the general operation of the system from the user's view. It will describe each of the components (windows) of the system. Where ever possible, actual screen snapshots of windows and pull-down menus have been included to give the reader a good feel for what the actual system works like without having to actually use it. Great care has been taken so that these screen snapshots convey as much information as possible. They have also been designed to be continuous. That is, the screen snapshots themselves explain the complete operation of the system.

3.1.3. The System Screens

The first thing to notice about the system is that it is composed of many screens. This screen concept was introduced because, depending on the application, a user of this system may require that he have many windows open simultaneously. It is not unreasonable to foresee an application where a user would have twenty or thirty windows open all at the same time.

Also, the system allows the user to operate on many abstract levels. The lowest level of operation allows the system user to design his own device interfaces. The second level

the device interface created at the lowest level and to create sequences and events. The third and highest level allows the user to operate on these sequences and events. Because a user may wish to switch between each of these levels and each level requires a different mind set, the system has been designed to present these levels as separate screens. This parallel between the screens and the abstract concept of levels reinforces this abstract concept of levels.

There are five screen types: the Smalltalk screen, the device editor screen, the sequencing screen, the sound object screen, and the utility screen. Each type of screen has its own set of functions. The Smalltalk screen is the standard Smalltalk/V Mac environment. This screen is part of the automation system but is included while the system is in under development. In a final application, this screen would be removed. The device editor screen allows the user to design the graphical representation of his console. The sequencing screen is where the user will use the graphical representation of the console to create a sequence of events for a real hardware console. This sequencing screen also allows the user to control movements on these consoles. The sound object screen allows the user to create the sequences, created on the sequencing screen, as sound objects.

To switch between these various screens, the system offers the screen menu. The screen menu, shown in figure 3-2, is always the last item on the menu bar. The screen menu lists the screen types with a check mark beside the screen which is currently active. To select a different screen, the system will position to that screen and place that screen in the screen pull-down menu.

Another feature of the screen facility we have not yet discussed is that the system allows multiple screens to work on. The intent of this feature is for when a user has many windows open on one screen, he can create an additional screen of the same type and move some of the windows on that screen. The screen pull-down menu allows the user to add and remove screens. When the Add Another Screen menu item is selected, a screen of the same type that the system is positioned on is created. A menu item to remove the screen can also be added to the screen menu. Figure 3-2 shows an example of the screen menu and the result on the screen menu.

The Remove Screen menu item will remove the screen that the system is currently positioned on. If there are any windows open on that screen, they will be closed. Note that the system allows the removal of the last screen of a given type, so that there is always at least one screen of each type available.

The menu bar contains various menus and changes depending on what screen is active and what window is active. The last pull-down menu item is always the Screen menu. The second last menu will be the screen specific menu for the screen the system is currently positioned to. This menu will have the same name as the screen's name. Any menus before this menu (apart from the file menu) are window specific menus and depend on what type of window is currently active. This menu will be discussed when each window type in the system is discussed. The file menu is the standard Macintosh application file menu and contains items to open and close windows and files, as well as an item to quit from this application.

Figure 3-2 - The screen menu. This figure demonstrates the operation of adding a new screen to the system.

3.1.4. The System Libraries and Editors

Throughout the system, we offer editors to create and modify various objects (channels, devices, sound objects). The system maintains a library of all of the objects created by the user. There is a separate list for each category of object. The categories which the system library consists of are: channels, device templates, device controllers, sequencers, and sound objects. The editors which operate on these objects will be discussed in more detail in the appropriate sections of this chapter.

To access the editor for an object, the user must position to the appropriate screen for that type of object. Each screen's menu will offer items to operate on objects appropriate to that screen. The menu will usually contain an item to create a new type of object. Upon creation, the object will be added to the system library under the appropriate category along with the object's name (which the system requests from the user). An editing window will

also be created and opened. The user can then make changes to the object using this editor. When the editing window is closed, the object still remains in the system library. The editing window for the object can be opened again by using the open window entry (found directly below the creation entry for that object on the screen's menu). To delete an object from the system library, there is an item in the screen's menu directly below the open window entry which will prompt the user for the name of the object to be deleted from the library.

One final thing to note is that all of the graphically based windows (channel editor, device editor, device controller, sound object editor) scale all of the objects contained within them when they are resized. This allows a window to be used at any size. An editing window can be sized very large in order to perform very delicate editing operations. Several windows can be sized smaller in order to tile the screen with many non-overlapping windows. A window such as a device controller window can even be resized very small and used for display only during sequencing operations. Note that this ability for the user to use the various windows at any size is quite important since, as mentioned above, he may require a high number of windows open at the same time.

3.2. THE DEVICE EDITOR SCREEN

The device editor screen is the editing environment which allows the user to define the appearance of the console he wishes to manipulate. At this point we will start referring to the hardware to be controlled as a device. The reason for this will become more apparent later when we look at the use of the system to control devices other than just recording consoles.

In the introduction to this chapter we introduced the concept of a channel. The channel is an abstraction which will allow us to refer to a group of controls as its own entity. The device editor screen offers an editor to lay controls (rotary knobs, slider, push buttons) down onto a channel. Once the channel has been defined in this way, we have a second editor which will allow us to lay channels down onto a device.

The Device Editor Screen Menu

The device editor screen menu (shown in figure 3-3) contains an entry to create (and open) a new window for editing a new channel. The system will prompt the user to give this channel a name and it will add this new channel to the system library of channels under this name. If the window is closed by the user, it can be re-opened at any time using a second menu entry from the device editor screen menu which allows the user to open a channel window for a channel in the system library.

The device editor screen menu also contains entries to create a window for editing a new device in a similar fashion to the creation of channels. These devices are also given names and are added in the system library of devices. The menu also contains entries to re-open a device window, and to delete devices and channels from the system libraries.

Figure 3-3 - The device screen menu.

3.2.1. Channel Editing

The first step in the creation of a device is the creation of the channels which make up the device. Once we have created a channel editing window (using the device screen menu entry for creating new channels), we can start laying controls on the window. You will note that when a channel editing window is active, a channel editor menu will appear in the menu bar at the top of the screen. The menu (shown in figure 3-4) contains items to add controls, add display (graphical) objects, remove and duplicate items on the window, change the attributes of items (type or size of font for instance), and to arrange overlapping items on the window. The types of controls currently supported are shown in figure 3-5

and are: rotary knobs, double (embedded) rotary knobs, vertical sliders, horizontal sliders, and push buttons. The double rotary knob is actually two concentric knobs in one control. The double rotary knob functions as if it were two separate controls. There is an additional entry in the control type list, channel name, which is a control, when selected, will allow the user to type some arbitrary text which will be displayed in the box (see section 3.3.2 for a more detailed explanation of its use). Although channel name is not strictly a control, it behaves very similar to a control and is placed in the control list instead of in the display object list. Its intent is to provide the user with a place to label channels on a device, possibly with the actual piece of hardware which is actually connected to that channel. The types of display objects which can be added are general text and channel numbers. Channel number is an item which will display the channel number of that particular channel when there are many channels of that type on a device (this will become clearer when we look at the device editor). Note that all text items (channel numbers, channel names, text) on a channel window allow the manipulation of their font type and size.

Note the distinction between controls and display objects. Controls are designed to do something when the user clicks the mouse on it, whereas display objects don't do anything. Display objects are typically used for labelling controls with text.

Figure 3-4 - The channel editor menu.

Figure 3-5 - This diagram shows the various control types: vertical slider, horizontal slider, rotary knob, double rotary knob, and channel name block.

Any item on the channel editor window can be moved or resized. Moving is accomplished by clicking the mouse button while the cursor is inside the item to be moved (see figure 3-6). The item can be moved around as long as the mouse button is held down.

Figure 3-6 - Before, during and after a move operation.

Resizing is accomplished in a similar fashion by clicking the mouse button while the cursor is inside the item near the bottom right corner (see figure 3-7).

Figure 3-7 - Before, during and after a resize operation.

Items can be selected for various editing operations by clicking the mouse button quickly (down then up) while the cursor is within that item. When an item is selected it will be shown highlighted (its interior region will be shown inverted). The editing operations are shown in the channel editor menu when an item has been selected. The various editing operations change for each type of object selected, but the most common ones are: duplicate the item (figure 3-8), remove the item, change the font of a text item (either style or size) (figure 3-9), or change the actual text content of a text item.

Figure 3-8 - A knob is selected and then duplicated.

Figure 3-9 - A text item is selected and then the font is changed from Geneva to Monaco.

As discussed in the introduction to this section, editing windows can be used in any size. If an editing window is resized, all of the items within it will be scaled so that they all fit within the window. Figure 3-10 shows a sample channel editing window drawn in three different sizes.

Figure 3-10 - Scaling example showing the same channel editing window drawn with three different sizes.

3.2.2. Device Editing

The second step in the creation of a device is the placing of channels onto a device. We do this using a device editor. The device editor is similar in operation to the channel editor described above. We create a device by placing channels onto the device. The channels are

selected by name and must have already been created and exist in the system library. The device editor allows the creation of a device by placing rectangles, which represent the channel boundary, onto a window representing the device. The final look of the device will not be shown here, it can be viewed on the sequencer screen by opening up a device window for this type of device.

An example of a device editor window is shown in figure 3-11. Channels are drawn as outlined rectangles. The name of the channel name ("Chnl 1" and "Chnl 2" in the figure) appears inside the rectangle along with the channel number. The editor works in a similar fashion to the channel editor. It allows moving and resizing of items as described for the channel editor.

Figure 3-11 - A device editing window.

The menu for the device editor window (shown in figure 3-12) contains items to add a channel to the device, add many parallel channels to the device, and remove and label items on the device editor window.

Figure 3-12 - The device editor window menu.

When the user selects the menu entry to add a channel, the system will prompt the user to select the channel from the system library of channel names, and then prompt the user for the location of the channel. The menu entry to add multiple channels is used to add a number of parallel channels to a device. When selected, it will prompt the user to select the channel from the library, the number of channels to add, the channel number of the first channel (the other channels' numbers follow this one sequentially), and whether the channels are arranged horizontally or vertically parallel. Figure 3-13 shows the dialog box used to enter this information. The user will then be prompted to enter the boundary for the group of channels, and then all the equal sized channels will be added to the device. This is merely an editing convenience and is included since many devices consist of many parallel channels. Figure 3-11 above shows the result of using the add multiple channel command. In this example, the channels were added with two invocations of the add multiple channels menu item: one to add the six horizontally spaced "Chnl 1" channels, and one to add the four vertically spaced "Chnl 2" channels.

There is an additional menu entry to give each channel a channel number. In order for this channel number to appear on the final device, the channels used must have a channel number box in them. (Recall, from section 3.2.1, that the channel number is one of the display object types which can be put onto a channel). The channel number is shown in the device editing window as a number preceded by a '#' (see figure 3-11 above).

Figure 3-13 - The add multiple channel dialog box.

3.2.3. A Complete Example

We will now give a complete example of the creation of a device. This section is included here to clear up any misunderstandings with the creation of devices and the editors used to create the channels and devices. This example will demonstrate every type of control and display object type. This section only describes the creation of the device and

shows what the final device looks like. It will not cover the use of the device itself. This will be covered in the section following this one.

We will design an interface to an imaginary console with five input channels and one output section. The input channels each have a fader, a pan control, an auxiliary effects send, two bands of semi-parametric EQ and a mute button. The output section has a master output level control, an auxiliary effects return level, and a headphones level control. We will start the creation of the device by creating two channel editing windows: one for the input channels, and one for the output channel. We will then lay appropriate controls on the editing windows and include some text labels next to each of the controls. We will then create a device editing window and add five of the input channels to it and one of the output channels. Figure 3-14 shows the results of the editing operations so far. Note that the box containing the '1' on the input channel is a channel number box. On the final device this box will be replaced by the channel number of each channel. The use of the box containing 'Channel Name' will be demonstrated in section 3.3.2 below.

Figure 3-14 - The creation of a hypothetical device with five input channels and one output channel.

We have now finished the creation of the device. The device is shown in figure 3-15. To see what the actual device looks like, you must switch to the sequencing screen and create a new instantiation of the device just created (This process will be explained in detail in the next section). Note how the channel number box which was placed on each channel has now been replaced by the channel number of that channel.

Figure 3-15 - The final device.

3.3. THE SEQUENCING SCREEN

The sequencing screen is the screen where the user will make use of the devices created on the device editor screen.

3.3.1. The Sequencing Screen Menu

In order to use a device, the user must create a new instantiation of the device from the template which was defined using the device editor. This new instantiation, called a device controller, is created using the "Create New Device Window" entry from the sequencing screen menu (shown in figure 3-16). This menu entry will first prompt the user for which

template to use, prompt the user to give it a name, then it will create a new device controller from the template. It will then add this new device to the system library of device controllers and open up a window for this device. If the window is closed by the user, it can be opened again at any time by using the "Open Device Window" menu entry to open up a device controller window for a device stored in the system library. Device controllers can be removed from the system library using the "Delete Device From System" menu entry.

The sequencing screen menu also contains entries to create, open and delete sequencer windows. Sequencer windows will be discussed more fully in section 3.3.6.

Figure 3-16 - The sequencing screen menu.

3.3.2. The Device Controller Window

In this section we will see how to actually use the interface to the device which we have designed. We will see how to use the device we have created and how to map our on screen controls to the controls of the actual hardware.

Figure 3-15 above showed the result of creating a new device controller for the example discussed in section 3.2.3. We will now discuss the device controller window operations. The first thing to note about the device controller window is that the controls on the device actually function. A simple click of the mouse button within the boundary of a control will cause the control's needle to jump to the position of the mouse. The user can also hold the mouse button down and drag the needle around (figure 3-17). Push button controls are the

only controls without a needle since they only have two states: on (the interior of the button is black) and off (the interior of the button is white). Clicking the mouse on a push button control will reverse its state.

Figure 3-17 - Dragging the needle of a rotary knob and clicking on a push button.

We saw earlier how MIDI controlled consoles use MIDI control change or note messages to receive parameter (virtual knob) changes. Therefore, we need a way to map our control movements onto these MIDI control change and note messages. When a new device controller is created, its controls do not control anything (actual hardware). To map a control to an actual hardware knob, we must use the "Assign MIDI Command to Control" menu entry from the Device Controller menu (figure 3-18). When this menu item is selected, the system will prompt the user to enter the information about how this control operates using the dialog box shown in figure 3-19. The dialog box allows the user to select the minimum and maximum values for that control (transmitted as the value of the MIDI message), the type of MIDI message which is sent when this control is moved (control change or note-on message), the control or note number which corresponds to this control, whether the scaling of the control is linear or logarithmic, and whether the scaling is reversed or not. Horizontal sliders and rotary knobs produce their lowest value when they are furthest to the left. The reverse scaling selection will cause these controls to have their minimum setting at the far right position instead. Vertical sliders produce their lowest setting when they are at the bottom of their travel, except when reverse scaling is selected.

Figure 3-18 - The device controller menu. This figure also shows the scaling down of the controls when the window is resized smaller. Compare with figure 3-20 which is a larger sizing of this window.

Once you have assigned a MIDI command (message) to a control, you can use it to manipulate an external device's control. Every time the on-screen control is moved, it will send out a MIDI message of the appropriate type (control change or note-on) containing the position of the control (a value ranging from the minimum specified value to the maximum specified value). The MIDI message is transmitted on the channel specified for this device (see the section on assigning the MIDI channel of a device below). Note that the MIDI assignments of the controls must match the MIDI assignments of the control on the device being controlled. Most MIDI controllable devices offer a changeable mapping between the incoming MIDI messages and the parameters which these messages control. Care must be taken so that the mapping table in the device matches the mapping in our system.

Enter Midi Parameters For Message

Control	Value
32	32
Min	Max
0	127

☒ **Reverse Scaling**

☐ **Linear Scaling**

CANCEL **OK**

Figure 3-19 - The dialog box used to specify the MIDI message to transmit when a control is moved.

The last type of control to talk about is the channel name control. We discussed this briefly earlier but did not go into too much detail. The channel name control is a control which can be added to a channel exactly like other control types. It does not actually control any hardware, but it does do something when you click the mouse inside it. When clicked on, the system will prompt the user to enter some arbitrary text. This text will then be displayed in the channel name box. The intent of the channel name box is to provide the user a method to label channels on a device, possibly with the actual piece of hardware which is actually connected to that channel. For instance, if the vocals are connected to channel 3 of a console, then the user can label channel 3 with the name "vocals" and therefore does not have to remember that the vocals are connected to channel 3. This allows the user the ability to label the channels on the virtual (screen) console in a similar fashion to the way he would label the channels of an actual console with a grease pencil. Figure 3-20 shows a device which has its channel name fields filled in.

Figure 3-20 - Using the device. Note the filled in channel name fields.

The device controller contains an entry, "Assign MIDI Channel of Device" which will allow the user to set which one of the sixteen MIDI channels the controls on this device will transmit on. This channel must match the receiving channel on the actual hardware device.

The last two menu items on the device controller menu deal with the use of the device controller during sequencing. "Record Starting Setting of Control" will record the current setting of the last moved control as the position that this control should be set at when the sequencer is started. This position is recorded with the currently active sequencer. Each sequencer can have its own set of starting settings for a device's controls. The "Set All Controls To Starting Settings" menu item will move all the controls on this device to their recorded starting settings so that the user can see what the settings are. Sequencers will be discussed in more detail in section 3.3.6.

The final thing to mention about the device controller window are the two buttons at the bottom of the window. These two buttons, "Punch Out Last Control" and "Punch Out All Controls", are used during sequencing when in punch in mode to take controls out of recording mode and will return the controls to their previously recorded movements. The first button is used to just punch out the last moved control, whereas the second button is used to punch out all buttons. Punch in recording mode will be discussed in detail in section 3.3.8.

3.3.3. Formalizations - What is a Device?

Up until now we have had a fairly intuitive introduction to the system. We will now formalize the descriptions of the components of the system.

We have seen how a device in the system is an interface to a piece of hardware. It was assumed that there was a one to one correspondence between our device window and the actual hardware device. This is not necessarily so, though. Our device window is merely the user's abstraction of some hardware. You can think of it as being a virtual hardware device. For instance, suppose our studio contained two 8 channel consoles chained together. The two consoles really act like one 16 channel console. When we turn to the computer control of these devices, there is no reason that we can't think of the two consoles as being one 16 channel console. We can then design an interface to this 16 channel console and offer the user this virtual piece of hardware. The user does not need to concern himself with the configuration of the actual hardware. Another example would be one piece of hardware which had many controllable components in it which might be better visualized as separate devices. For instance, a multitimbral synthesizer may have 8 separately controllable synthesizer sections. We can create 8 separate virtual devices, each with their own window, to match the user's model of 8 independent devices.

As the system has been presented so far, it appears that the only type of device which can be controlled by the system is MIDI controlled recording consoles. This is not the case at all. The system can be used to control any MIDI device which offers real time control of its parameters. We just need to create a device controller interface to the device, and map the controls to the parameters of the hardware device. For instance, we can create an interface to a synthesizer which offers realtime control of its envelope eight envelope parameters by creating a device with eight sliders on it. These sliders will then give us on screen control of the eight envelope parameters. Digital effects processors often have realtime controllable parameters which can be accessed via MIDI. Using our system it is

very easy to create a simple knob/slider interface to such a device. This can be applied to almost every device in the studio: synthesizers, effects processors, drum machines, and recording consoles. Most of these devices benefit from this type of interface since the only other way to change their parameters is usually very poor, for example via a system of nested menus to find parameters and then using parameter increment and decrement buttons to change their value. It thus becomes easier to use the devices since each parameter can be changed by just moving a knob and the current state of all the parameters can be viewed at once.

The system also allows us to hide the actual hardware from the user. We can design hardware independent interfaces so that as the hardware changes (either through upgrades or because the user changes studios) the user can remain with the same graphical representation of the hardware. The user can use two completely different recording consoles, with a very different layout of the controls but having the same functionality, with no difficulty switching between the two. We can also use this technique to offer the beginning user a simplified view of a device. A certain type of user may not be interested in using all of the functionality of a device, therefore we can create an interface to the device with only the needed controls thereby simplifying the use of the device.

3.3.4. Formalizations - Definition of a Channel

We have introduced the notion of a channel in our system to make dealing with consoles easier. The channel abstraction allows us to easily create and edit a console type device consisting of many similar channels. When we create an interface to a device which does not explicit channels, such as a synthesizer, we must first create one channel containing all the controls of the device, and then place this one channel on the device. We could also approach the design of the interface to this synthesizer by creating a channel containing half of the controls and place two of these channels on the device. We could even create one channel per control and place one channel for every parameter onto the device. The latter approaches would be incorrect, though.

A channel is defined to be the smallest cluster of hardware which the user can cause, through recabling, sound to pass through. For example, the user can plug an instrument into any one of the inputs of a console. The sound passes through the channel of the console and is then sent to the output buss. At no place in the channel can the user break the signal chain and decide to pass the signal into a different piece of hardware. Therefore, each channel on the recording console shall be emulated by a channel in our system. Synthesizers and effects processors shall only contain one channel unless they actually

have separately addressable signal paths, in which case each signal path would be its own channel.

The reason for the channel distinction will become more apparent when we look at sound objects (in section 4.1.5), but briefly, has to do with the validation of control movements when layering sound objects.

3.3.5. Advantages of the System

Before we look at the automation aspects of the system, we should look at the benefits of the system as presented so far. As discussed above, we can hide the hardware from the user of the system. As the hardware in the studio changes, the graphical on-screen representation of the hardware can remain the same.

Another benefit of this system is the consistent user interface to every device in the studio. Each hardware device will usually have its own distinct interface. In order for a user of the studio to operate all of the devices, he must learn each of the interfaces. Our system offers one consistent interface to every device in the recording studio thus decreasing the learning time to become accustomed to a new studio.

The final benefit of this system is the centralized control of the entire studio. Every device in the studio can be controlled from the one computer. We therefore no longer have to have each device within reach of the user. In fact, most of the hardware devices can be removed from the studio control room altogether. This is an important concept which we will see again when we discuss the virtual recording studio in section 6.2. This remoting of the hardware is also important to remove the clutter of devices in front of the user. As more and more devices are required in studios, it becomes increasingly difficult to arrange the hardware in such a fashion as to make all of it easily accessible. Also, depending on the application, a user may only require use of a small percentage of the hardware offered by the studio. Using our system, the user can simply open up the interfaces to the devices he wishes to use during his recording session, and therefore does not have to be faced with the clutter of unneeded hardware devices. We can also conserve the users visual real-estate by the use of many staggered overlapping windows which can take up much less space than the real hardware itself.

3.3.6. The Sequencer

Now we move on to the reason for the name of the sequencer screen: the sequencer window. The sequencer window is a standard multi-track MIDI sequencer with some additional properties to allow the sequencing of not only incoming MIDI information, but

control movements as well. We will start our discussion by looking at the standard sequencer features which are found on the sequencer window.

A sequencing window is created using the "Create New Sequencer Window" menu entry from the sequencing screen menu. The system prompts the user to give the sequencer a name, adds this sequencer to the system library of sequencers, and opens up a window for this new sequencer. The sequencing window menu also contains entries to delete sequencers from the system library of sequencers, and to open up a window for a sequencer stored in the library.

A sample sequencer window is shown in figure 3-21. The top half of the window is a simple standard eight track sequencer. The sequencer allows the user to change the mode, channel assignment and text description of each channel. A non-empty track can be in one of three modes: record, playback, or off. When the sequencer is turned on, any incoming MIDI messages will be recorded on the tracks which are in record mode. The MIDI messages from all the tracks which are in playback mode will be merged together and sent out the MIDI out port. The channel designation is used as the MIDI output channel for the events played back on each track regardless of what MIDI channel it was recorded from. The text description on sequencer tracks is provided for the convenience of the user and is used only for labelling the tracks.

Recall that a MIDI sequencer is just a multi-track tape recorder for MIDI information. To carry the analogy further, we will also emulate a tape recorder's stop, play, record, and pause buttons. These sequence control buttons are located just under the track section on the sequencer window. The play button will initiate playing of the tracks, which are in playback mode, from the beginning of the sequence. The playback tempo (speed) can be set in the master system window (see section 3.3.10). The record button is similar to the play button in that it initiates playback from the beginning of the sequencer, but has an additional function to engage record mode which allows the recording of the incoming MIDI information. When in record mode, any MIDI information received at the MIDI in port will be recorded on the tracks which are in record mode. The pause button will temporarily stop the playing of the sequence until it is re-engaged with the play (or record) button. The stop button will terminate the playing or recording of the sequence.

Figure 3-21 - An example sequencer window. The first four tracks have information recorded on them. Any recorded MIDI messages will be put on track five.

Note that the sequencer is not intended to be a full function sequencer. Its purpose is to provide a place to record MIDI information intended to control external synthesizers and other equipment. This control is required for the operation of sound objects which will be described later in chapter 4. The sequencer will interface well with other sequencers so that more powerful sequencers can be used for the creation of sequences which can then be transferred into this system. Individual tracks (or the entire sequence) can be filed out in standard MIDI file format so that sequences can be transferred to and from other sequencing packages. The "File Out Track", "File In Track", "File Out Sequence", and "File In Sequence" menu entries on the sequencer window pull down menu (shown in figure 3-22) allow the filing in and out of tracks and sequences in the standard MIDI file format.

The Make Sound Object menu entry of the Sequencer Window pull-down menu will make a new simple sound object from the current sequence (both the sequenced track data and the sequenced control data). This operation and sound objects will be discussed in more detail in section 4.1.1.

Figure 3-22 - The sequencer window pull down menu.

The "Edit Track" menu entry allows the editing of the MIDI information on an individual track. Figure 3-23 shows, the MIDI sequence editor, the result of executing this menu entry and selecting one of the tracks for editing. The window gives a list of the MIDI events which are stored in the track in the order in which they were recorded. The time of the event is shown at the very left of each entry and is in the format of beat, bar, and sub-beat (1/24 of a beat - a MIDI standard defined time). The event type follows the time and is one of: note on, note off, polyphonic aftertouch, channel aftertouch, pitch wheel change, program change, control change, or a call to a sound object. Following this are the parameters for this type of event: channel (if appropriate), and the values of the event. The sequence menu contains entries to add and remove events. The parameters of a MIDI message can be changed by clicking the mouse on that event. The boxes at the bottom of the MIDI editor window will then show the current value of all the parameters (including the time of the event). Any of these parameters can be changed by clicking on the box containing the parameter. The system will then prompt the user for the new value of the parameter.

4.1.3.1. The Attach Operations

The editing operations as discussed so far will allow the placement of components at specific times within the composite sound object. This fixed placement of components is not entirely adequate, though. For instance, it is obviously desirable to be able to have one component follow directly after another component. This is simply accomplished with the system by placing the second component directly after the end of the first component. A problem arises, though, if the length of the first component were changed. This would require the user to re-edit the sound object and move the second component manually.

The attaching operations have been designed to avoid this re-editing procedure when sound objects change length. These operations allow the attachment of one component to another so that the second component will directly follow the first component no matter what the length of the first component is. Attachment of components is done using the Attach Front of Component menu item. This menu item allows the user to attach the front of the currently selected component to the end of any of the components which are visible in the editing area. The selection of which component to attach to is done via a pop-up menu of the names of all the visible components and cue points (to be discussed in the following section) in the editing area. Once a component is attached to something, that component cannot be moved in time using the editor. It will only move if the component it is attached to changes length or moves. When you attach a component to something, that component becomes a slave to that thing. The editor will allow attached components to be moved vertically, though. An attached component can be unattached using the Detach Component menu item.

Figure 4-4 shows an example of attached components. The attachments are indicated by the dotted line between components. This figure also demonstrates the use of the null sound components for spacing. The null sounds allow attached components to be spaced an exact amount of time apart. In the figure, D will occur at precisely two beats after the end of A due to the two beat null sound. C will occur directly after D. B and E will occur precisely one beat after the end of A. The entire length of the example sound object is 8 beats or 2 bars (as shown by the ending time of B in the status display window). Note that the editor will not allow the user to move B,C,D,E and the null sounds in time. Their vertical position can be changed, though. The editor will allow the user to move A in time. If A is moved, all of the attached components will also move.

times in the sequence when they were recorded. The on-screen controls will move and any hardware controls that they are mapped to will also move. This process allows us to create a repeatable mixing process that will play back the same every time. Now, in order to facilitate a more complicated mix, the system allows the user to overdub on top of the first mix. This is accomplished by again putting the sequencer into record mode and recording additional control movements. These movements are added to the original mix as if they were performed originally simultaneously with the first mix. Of course this process can be repeated indefinitely, each pass adding something new to the mix. Using the computer in this fashion, one person can record a very complicated mixing process which would normally require many recording engineer's assistants to achieve.

It is obvious that it is very desirable to be able to create a mix in this overdubbing fashion. But perhaps there are additional high level features which can be added to increase the power of the system. For instance, it would probably be very useful to be able to save previous versions of a mix during an overdub so that the user can compare the two mixes to see if it got better and, if not, possibly return to that old mix. It might also be desirable to be able to compare completely different mixes of the same sequence. The system supports these ideas using the "Current Mix" pop-up menu located under the sequence control buttons. The sequencer can store an indefinite number of mixes within itself. The "Current Mix" pop-up menu will, when clicked on, show all of the mixes currently defined in this sequencer. When a sequencer is initially created, there is only one mix defined, called the Initial Mix. If any controls are moved by the user while the sequencer is in record mode, when the sequencer is stopped, the system will prompt the user for the new name of the mix. If the user enters a new name, a new mix will be created containing both the old mix and the new movements and will be put into the Current Mix pop-up menu under the given name. If a new name is not given, a new mix will not be created. Instead, the old mix will have the new movements merged into it. Figure 3-24 shows the use of the Current Mix pop-up menu.

A previous mix can be auditioned at any time by just selecting the mix using the pop-up menu and putting the sequencer into play mode. Any new mixes created from overdubbed control movements are created using the currently selected mix as its base. The initial mix is an empty mix and can be used to create a new mix from one that has no control movements at all.

Figure 3-24 - The current mix pop-up menu.

The bottom most part of the sequencer window, the affected channel list, is a list of all of the channels affected by the currently selected mix. If any control has been moved at all in that mix, its channel name will appear in this part of the window. Any control which has a starting setting recorded for it in this sequencer (see figure 3-25) will have its channel name displayed here also. The use of this list will become more apparent when we look at the conversion of sequences into sound objects. This list is used for display only so that the user can tell at a glance which channels are affected by the current mix.

Figure 3-25 - The list of channels affected by the current mix: 'Mix 2'.

3.3.8. Punch-in Recording

Up until now we have seen how to record control movements into the sequencer. We have also mentioned that this type of automation system should have the ability to edit movements after they have been recorded. This is what the punch-in record mode does. Punch-in record mode is invoked, on an individual control basis, when the sequencer is running in record mode and the user manipulates a control which already has movements programmed into it. The control will follow its programmed actions up until the user grabs it. After that, the control will be under the control of the user just as if it had not had any movements programmed at all. If the user lets the control go, it will remain stationary. When the user grabs a control which has movements programmed into it, it is said to be in punch-in mode. To return the control to its preprogrammed movements the user can press the Punch Out Last Control button on the device controller window. This button will punch out the last control moved by the user. When the control is punched out, it will immediately jump to the position it would have been at had the user not entered punch-in mode and the control will resume its preprogrammed movements. The device controller window also contains a button, Punch Out All Controls which will punch out all controls on the device controller window which are currently in punch-in mode.

Punch-in recording mode is named for the similar operation which occurs when operating multi-track recording tape. When using tape we can punch into recording mode in the middle of a track, re-record a segment of that track, and then punch out of recording mode. The net effect is the erasure and re-recording of a segment of that track.

3.3.9. Control from an External Device

We have presented the on-screen devices as a replacement interface for the actual interfaces of the hardware devices. This is an advantage for those types of devices with a poor interface, such as consoles with no knobs where the parameters are accessed through a menu system. But for devices which actually have a good interface, or devices where the user is quite familiar with the interface, this may be quite a disadvantage. For this reason the system will accept MIDI input from external devices and will update the on-screen controls appropriately. In this fashion the system does not impose the graphical mouse based method of operation on the user if the hardware has the capability to send as well as receive MIDI messages.

When using an external device to control an on-screen device, the Record from mixer mode must be selected in the Master System Window. See section 3.3.10 for more details on the Master System Window.

The Currently Active Sequencer

Note that the system allows only one sequencer window to be open at a time. This is to avoid confusion by the user about the context he is working in, since recording of control movements on a device controller window is done in the context of this sequencer window. We call the open sequencer window the currently active sequencer.

Besides storing control movements, the currently active sequencer is also used to store the starting (time zero) positions of each of the controls. It is often desirable to have controls originate at a given position at the start of a sequence. The system allows this ability with the Record Starting Setting of Control menu entry in the device controller pull down menu. This menu item will record the current setting of the last moved control as the position that this control should be set at when the sequencer is started. This position is recorded with the currently active sequencer. Each sequencer will have its own set of starting settings for a device's controls just as each sequencer has its own set of sequenced control movements.

3.3.10. The Master System Window

The master system window, shown in figure 3-26, resides on the utility screen. This window cannot be closed and there is only one in the entire system. This window contains parameters which affect the whole system globally. Currently there are three parameters capable of being set through this window. These three parameters all affect the recording and playback of the sequencers. The first parameter, tempo, is the speed of playback of the currently active sequencer in bpm (beats-per-minute). The tempo facility as implemented is merely for conversion between internal and external clock sources and is not intended to be a full featured tempo facility. A fully functional tempo facility should allow tempo maps which allow the tempo to vary over time. Also, each sequencer should be able to store its own tempo map.

The second parameter in the master system window is the origin of the system clock. When internal clock is selected, the system will generate its own clock at the rate set by the tempo. When external clock is selected, the system will use the clock received at the MIDI input port. This mode is used for the synchronization of the system with external sequencers and other MIDI clock generating devices. In particular, this mode is used to obtain a higher resolution clock than is currently available using the internal clock (since the MIDI clock is higher resolution than the system's internal clock). When the external clock is set, the tempo set in the master system window will be used to convert the received MIDI

clock into an absolute timing reference used for absolute times like those used in the cue list editor window and the sound object editor window.

The third parameter indicates where the MIDI data received at the MIDI input port originated: keyboard or MIDI mixing console. When recording keyboard (or any other MIDI device) performances into a sequencer, this parameter must be set to Record from keyboard. When the incoming MIDI data is to be used to control an on-screen device, this parameter must be set to Record from mixer. This parameter is required to distinguish the type of sending device since the MIDI messages sent out by the recording console are the same type of (note and control) data sent out by a keyboard.

Figure 3-26 - The Master System Window.

3.4. THE CUE LIST EDITOR

Just before we get into the section on sound objects we should cover a little related material by first finishing up the discussion of the utility screen. We saw earlier how the utility screen holds the one and only copy of the Master System Window. The utility screen currently has one other purpose, it contains the Cue List Editing window. The cue list is a system list which allows the user to map text names (or descriptions) onto times. It offers a facility to enter name/time pairs (cues) and store these in a list. Entries can be added to or removed from the list, and the time associated with any name can be changed to another time. This facility has been included to support sound effects and music work for video and

film. This list makes a convenient place to store all the time references which may change due to the re-editing of a film¹⁰. We will see later (in section 4.1.4) how sound objects can be synchronized with the time cues entered in the editor. In this section we will just discuss the operation of the Cue List Editor itself.

The Cue List Editor, shown in figure 3-27, consists of one selection field and two entry fields. The selection field (the left most field) contains a the list of the currently entered text names. This scrollable list is used to select an entry for editing. Clicking the mouse on any entry will cause its description to appear in the editing field.

The editing field, on the bottom right of the window, is used to enter or change the time associated with a cue. After the time has been entered, the user must select the Save entry from the file menu.

The last field, the type field, contains two buttons which are used to select the format of the time that the user is entering. The two formats are song time and SMPTE time. Song time is used for timing references within songs. It uses a standard bar/beat timing method with the addition of a sub-beat count. The time consists of three numbers: bar, beat, and sub-beat. There is 4 beats per bar, numbered 0 to 3 (the system uses a 4/4 time signature), and 24 sub-beats per beat, numbered 0 to 23. With this definition, a sub-beat is equivalent to a MIDI clock unit, which is where the maximum clock resolution for the system is derived from. A song time is entered in the format 'bar:beat:sub-beat'. Note that the actual time which a song time corresponds to depends on the tempo of the song. The tempo is entered in the Master System Window as discussed earlier, and represents the number of beats per minute.

SMPTE time is used for absolute timing references. SMPTE times are not affected by the tempo setting. SMPTE timing is the standard timing reference for use in the film and video industries. It consists of four numbers: hour, minute, second, and frame. There are 24 frames per second, numbered 0 to 23. The format for entry of a SMPTE time is 'hour:minute:second:frame'. There are four types of SMPTE timing: 30-Frame, 30 Drop-Frame, 25-Frame, 24-Frame [DAVIS]. For simplicity, this system implements only the 24-Frame format. Future extensions include the support of the other three timing systems. Along with sound objects, the inclusion of the SMPTE timing ability makes this an ideal system for video and film sound effects work.

¹⁰Most often when music and sound effects are created for a film, it is done on an early pre-released version. When the final version of the film is completed, quite often portions have been cut out and moved around causing the time cues to move.

Figure 3-27 - The Cue List Editor Window and its pull-down menu.

Figure 3-27 also shows the cue list editor's pull-down menu. This menu allows the creation of a new cue template in the editing window which the user can change and save as his cue. The menu also allows the removal of the cue which is currently selected in the selection field.

3.5. SUMMARY

This chapter has presented our mixdown automation system. The system supports the use of standard MIDI controllable consoles with knobs as well as consoles and MIDI controlled VCAs which have no knobs through the graphical on-screen representation of the console. The system offers an environment which allows the user to design the look of this on-screen console, and can even design different representations for different applications. This on-screen console's controls can then be used to manipulate the consoles controls. The system supports mixdown automation through the use of sequencers which record control movements and allow the user to create a complicated mix through indefinite

overdubs. The sequencer allows the saving of previous mixes during an overdub so that the user can compare a new mix to an older one and even return to that older mix.

The advantages to this mixing automation are the repeatability of the mixing process. As the number of devices (or channels) to be controlled increases, we don't have to have an exponential increase in the possibility of an error. After a mix has been completed, we can make minor changes relatively easy and do not have to perform the entire mix again. The computer provides a natural environment for the saving of mixes. Instead of writing the position of every control down on paper, the system saves all of the control settings in the mix file. This means that the mixing process can be done over several sessions, and one long mixing session is no longer required to produce an elaborate mix. The saving of old mixes allows us to compare a new mix to an older version to see if the changes being made are good ones¹¹.

It should be noted at this time that while the system was designed so that the end-user (recording engineer) could design the graphical representation of his console, this is not the only way to use the system. A specialized device designer can be used to design the graphical representations of the hardware in the studio, thus freeing the user from having to learn to use the device and channel editors. This was the main reason for the placing of the editing tools on the editing screen and the sequencing tools on the sequencing screen. A simple user of the system should not have to deal with the editing screen at all.

¹¹Quite often after many hours of mixing, the engineer/producer can lose sight of the original goals. Ear fatigue, after hours of mixing, can also contribute to a degradation in the mix.

4. THE HIERARCHICAL SOUND SYSTEM - SOUND OBJECTS

We will now present the remaining major portion of the system: sound objects. Sound objects are an encapsulation of a sequencer's MIDI and control sequencing data into a named entity. We can build up a library of these sound objects, and can play any one by just giving its name. We can also encapsulate (group) several sound objects into a new composite sound object which is a temporal ordering of these sound objects. This composite sound object can be used as a component of a still larger composite sound object. We can play any sound object by specifying its name, or we can use any sound object as an atomic element of the system and place it as an event in one of the sequencer's tracks.

Sound objects were designed to allow a higher level of control over the sound creation and production process. Current systems, both automated and non-automated, only allow control of their lowest level entity: the controls. Sound objects allow us the ability to capture this type of control information into an abstract entity which can be triggered by the user. This allows a different approach to composition. It allows the user to create sounds which require complex control movements. Once the sound is created and captured in a sound object, the user can use this sound object in place of the complex control process required to make that sound. A composer can make use of this process by first designing the sounds as sound objects, and then, after removing ones mind set from the low level tasks, placing these sound objects into the composition at the appropriate points.

The sound object concept offers a parallel to a typical compositional approach. Initially the composer does not wish to deal with the low level details of how each sound will actually be produced. Composition is conceived in terms of placing abstract sounds at certain places in time. It is not until later that all these details will be worked out.

Using the ability of the system to allow the placement of sound objects into larger sound objects, the composer can create increasingly more complex sounds. This hierarchical model of composition allows the composer to create sounds which would normally be very difficult to achieve otherwise.

The notion of sound objects allows the user to conceptualize the object as being an actual sound (a digitized sound sample, say). But sound objects are only a container for the information required to create the sound (given the hardware contained in the studio). In

particular, they contain only control information, not digitized sound. This allows us to abstract the control information into entities which appear to be actual digitized sound.

One may ask that if sound objects are just a way of representing this control information as digitized sound samples, why not actually use digitized representations of the sound and remove the whole control process altogether. In other words, once we have created a sound, why is there a need for the initial control information which was used to create that sound? The reason for wanting to keep the initial control information is that we may want to go back at a later time and edit the sound. For example, we may want to change the movement of one of the knobs which was used in the creation of the sound object. This would be impossible if we did not retain the control information which was used to create this sound. This is the major reason behind the need for the flexibility which sound objects offer.

4.1. THE SYSTEM'S SOUND OBJECTS

Now that we have discussed the motivation for sound objects, we will see how the system implements the concept. The system has two kinds of sound objects: simple and composite. A simple sound object is one which is created directly from a sequencer. It is a subset of the sequencer data. Simple sound objects are the most primitive type of sound object.

A composite sound object is a sound object which is made up of a temporal ordering of other (component) sound objects. This ordering is specified using a composite sound object editor where the user can place the component sound objects onto a time line representing the playback lifetime of the composite sound object.

4.1.1. Simple Sound Objects

The creation of a simple sound object is done on the sequencer screen with the sequencer pull-down menu entry Make Sound Object (as introduced in section 3.3.6). This will prompt the user for a name and will then create a new sound object with the given name and place it in the system library. The user will then be asked to specify the subset of the tracks and the recorded control movement data that should be included in the sound object. The user can specify any subset of the non-empty sequencer tracks to be included in the sound object. He can also specify which mix is to be used, and which channels from that mix should be included in the sound object. The affected channel list, on the bottom of the sequencer window, shows the channels affected by the currently selected mix. This

display is included so that the user is made aware of the need to distinguish the affected channels so that when it comes time to create a sound object, he has an understanding of which channels have been used.

The need for the flexibility of being able to select a subset of a sequencer's data for inclusion in a sound object, as opposed to all of the data, is not immediately obvious. The reason for this ability is the anticipation that many users, during the creation of sound objects, will require the ability to create these sound objects in the context of other musical material. A user may require some background music (or sound effects) in order to create the sequenced tracks which will become the sound object. With the ability to select a subset of the sequencer's information, the user can use unused tracks of the sequencer to record this background or support music, and when creating the sound object, just leave out these tracks during the track selection process.

The last piece of information the system will prompt for when creating a new simple sound object is the length of the sound object. The system will present, as the default length, the time of the last event (MIDI track event or control movement event) in the sequencer. The user has the option of lengthening or shortening this time. If the time is lengthened, the sound object will be extended with silence. If the time is shortened, the sequence will be shortened by removing all events (except for note off events) which occur after the newly specified end point. This length setting feature is useful in a musical composition context where the sound objects represent actual musical entities, such as bars or phrases. In this case, we would require the ability to set the length of the sound object if the length must be a whole number multiple of bars but the recorded data did not work out to be a multiple of bars.

4.1.2. Composite Sound Objects

A composite sound object is a grouping or encapsulation of other (simple or composite) sound objects into a single named entity. The sound objects which are contained within the composite sound object are called the component sound objects. The component sound objects are added to the composite sound object using a sound object editor which allows the placing of the components onto a time line which represents the playback lifetime of the composite sound object.

All sound object manipulations are done on the sound object screen. A new composite sound object is created using the New Sound Object menu item of the sound object screen's menu (shown in figure 4-1). This menu entry will prompt the user for a name, create a new empty composite sound object of this name, place it in the system library, and

then open an editing window for this new composite sound object. The sound object screen's menu also contains an entry to open up an editing window for a composite sound object stored in the system library, and an entry to delete a sound object from the system library.

Figure 4-1 - The sound object screen menu.

4.1.3. The Composite Sound Object Editor

We will now discuss the operation of the composite sound object editor. Figure 4-2 shows an example of a sound object editor. The main portion of the window, the editing area, is a four direction scrollable window. This area represents the time span of the composite sound object being edited. The component sound objects are shown as rectangular boxes, containing a name, within the editing area. The top portion of the editing area is the actual time line for this sound object. The times are shown in song time format (bar:beat.sub-beat). Note that these times are relative to the beginning of the sound object.

The top portion of the sound object editing window is the status display. On the very right of the display is the window size. This number indicates the width of the editing area in bars. This number can be changed via the sound object editor menu. The other two displays indicate the start and end times, in song time format, of the currently selected component sound object. If there is no selected component, these displays will be blank.

The sound object editor works in a similar fashion to the other graphically based editors in the system. A component sound object can be moved around in the editing area by clicking the mouse within its boundary and dragging it to its new position. Clicking and releasing the mouse button on a component will select that component for editing. The component is referred to as the currently selected component and is indicated as such by being highlighted (blackened internally). The component 'Sound Object 2' in the figure is

the currently selected component. Its start and end times are shown in the status display. A component's horizontal position is used to indicate the time at which it begins relative to the beginning of the composite sound object. Its vertical position is merely for the use of the user. Typically, the vertical positioning would be used to separate components which overlap in time. For instance, the vertical spacing of the three components in the figure makes it possible to see each one clearly even though 'Sound Object 2' overlaps the other two components in time.

Figure 4-2 - The sound object editor.

The sound object editor menu, shown in figure 4-3, contains many items to manipulate the components and to change the parameters of the editor. The first item in the menu is the Add Component item. This prompts the user with a list of all the sound objects currently stored in the system library and, once selected, the new component will be placed on the editing area allowing the user to move it to its proper position. Components can be

removed from the window using the Remove Component menu item. This item will remove the currently selected component.

The Add Null Component menu item allows the user to add a component filled with silence to the window. The system will prompt the user for the length of the component. Null components are used for separating sound objects by a specific amount of time when using the attach operations (discussed below).

The Align at Time menu item is used to move the currently selected component to a specific point in time. The system will prompt the user for the time at which the start of the component will be placed. This menu item is used as an alternative to moving the component with the mouse when the time placement of the component must be very accurate.

The Set Window Size menu item will prompt the user for the new window size. The number entered must be a whole number. The window size represents the number of bars which appear in the editing area of the window. The menu items Double Size and Halve Size are included to quickly double and halve the window size.

Every sound object must have an explicit length. The length for a composite sound object can be specified explicitly, or can be set to vary depending on where its component items are placed. The composite sound object editor indicates the length of the sound object with the End of Sound Object marker (shown in the figure). The Floating End and Set End menu items are used to affect this length. The Floating End menu item is used to turn the floating end feature on and off (a check mark beside the item indicates the feature is on). When the floating end feature is on, the end of the composite sound object being edited will be at the end of the last most component (as shown in the figure). If a component is moved past the end during editing, the end point will be adjusted accordingly. The floating end feature causes the end point to "float" with the end of the last component. When the floating end feature is off, the end point will remain fixed. In this case, the Set End menu item can be used to explicitly set the end point of the composite sound object. If a component is moved past the current end point though, the end point will be moved to the end of this component.

The floating end feature is more than just a mere editing convenience. The floating end feature is also used to indicate whether the sound object is variable or fixed length. With the floating end feature off, the end point of the sound object is always in the same place and will therefore remain the same length. With the floating end feature on, the end point of the sound object is always at the end of the last most component. This implies that the length of the sound object will vary as the length of this last most component varies. For example,

using the sound object created in figure 4-2, if we were to open up an editor for 'Sound Object 3' and shorten its length, the length of the example sound object will automatically change. Note that this change does not require any editing to be performed to change the length of the example sound object. The length will automatically be changed where ever the example sound object is used. If it were used as part of another sound object, for instance, that sound object's length may vary also.

The floating end feature allows us to model sounds with and without rigid time structures. It allows the user to model fixed length musical entities, such as bars and phrases, using fixed length sound objects. It also allows the modelling of more abstract entities, such as sound effects, using sound objects whose length depends on the length of other sound objects.

Figure 4-3 - The sound object editor's pull-down menu.

4.1.3.1. The Attach Operations

The editing operations as discussed so far will allow the placement of components at specific times within the composite sound object. This fixed placement of components is not entirely adequate, though. For instance, it is obviously desirable to be able to have one component follow directly after another component. This is simply accomplished with the system by placing the second component directly after the end of the first component. A problem arises, though, if the length of the first component were changed. This would require the user to re-edit the sound object and move the second component manually.

The attaching operations have been designed to avoid this re-editing procedure when sound objects change length. These operations allow the attachment of one component to another so that the second component will directly follow the first component no matter what the length of the first component is. Attachment of components is done using the Attach Front of Component menu item. This menu item allows the user to attach the front of the currently selected component to the end of any of the components which are visible in the editing area. The selection of which component to attach to is done via a pop-up menu of the names of all the visible components and cue points (to be discussed in the following section) in the editing area. Once a component is attached to something, that component cannot be moved in time using the editor. It will only move if the component it is attached to changes length or moves. When you attach a component to something, that component becomes a slave to that thing. The editor will allow attached components to be moved vertically, though. An attached component can be unattached using the Detach Component menu item.

Figure 4-4 shows an example of attached components. The attachments are indicated by the dotted line between components. This figure also demonstrates the use of the null sound components for spacing. The null sounds allow attached components to be spaced an exact amount of time apart. In the figure, D will occur at precisely two beats after the end of A due to the two beat null sound. C will occur directly after D. B and E will occur precisely one beat after the end of A. The entire length of the example sound object is 8 beats or 2 bars (as shown by the ending time of B in the status display window). Note that the editor will not allow the user to move B,C,D,E and the null sounds in time. Their vertical position can be changed, though. The editor will allow the user to move A in time. If A is moved, all of the attached components will also move.

Note that while the attach operations were designed to achieve an affect similar to the grouping operation of MacDraw¹², the attach operations offer a more powerful facility than grouping. That is, the attach operations are more than the mere editing convenience of having one handle on a group of sound objects. The attach operations allow the user to specify a temporal relationship between component sound objects. If one of the component sound objects were to change length, the components which were attached to it may change position. The temporal relationship between the components remains the same, even if the components change length.

Figure 4-4 - Attaching components.

¹²MacDraw allows the grouping of a group of objects into an entity which can be treated as if it were one object.

4.1.4. Cue Points and the Sound Object Editor

Section 3.4 discussed the operation of the cue list editor. In that section we saw how the system supported the ability to maintain a cue list (a list of name/time pairs). It was also stated that the purpose of these cues was for the synchronization of sound objects to these time cues. This synchronization is specified using the sound object editor. The sound object editor will automatically display all time cues which fall within the time period spanned by the window. The time cue is displayed as a vertical dotted line the entire height of the editing area, with the name of the cue displayed in a box at the top of this line. The line is aligned directly at the point in time where the cue falls.

The sound object editor allows the user to synchronize sound objects to these cues by allowing the attachment of components to these cue points. This attach operation is similar to the attachment of one component to another except that the system allows a component's beginning or end point to be attached to a cue point. Attaching the beginning of a component to a cue point allows the start of playback of the component to begin at the cue time. Attaching the end point to the cue causes the component to start at such a time that it will complete at the cue time.

The Attach Front of Component menu item of the sound object editor was discussed above. It was mentioned that the user can select one of the visible cue points to attach the component to. If a cue point is selected, the component will be attached to that cue point. As with attachment to other components, the system will not allow a component attached to a cue point to be moved in time. The component can only be moved vertically.

The system provides an additional menu entry, Attach Rear of Component, to attach the end point of a component to a cue point. The operation is similar to the attach front operation except that the pop-up menu displays only the visible cue points and not the visible components. It should be noted that the system will only allow one front or rear attachment for a component. Attaching the front of a component to something will undo the rear attachment, and attaching the rear will undo the front attachment.

Once a component is attached to a cue point, it will not move in time unless that cue point moves. As soon as a cue point is changed, though, any attached sound objects will move immediately. Note that, if this operation moves a component which affects the end point of the sound object, the length will change. This, in turn, can affect the lengths of other sound objects. As discussed above, this change will be taken care of automatically by the system, the user is not required to update the sound objects manually.

Figure 4-5 shows an example sound object editor showing four cue points. Component A has its end attached to 'Cue 2'. Since 'Cue 2' represents the time 1:3.05, and since A is

two beats long, A must start at time 1:1.05. Component B has its start attached to 'Cue 1' and therefore starts precisely at 1:0.00. D and C are attached to B and therefore follow directly after B.

Figure 4-5 - The sound object editor showing cue points. Cue 1 = 1:0.00, Cue 2 = 1:3.05, Cue 3 = 2:3.12, Cue 4 = 3:3.00.

It should be noted that the cue point times used within a composite sound object are relative to the beginning of the sound object, even if they are specified in SMPTE time format. The only case in which these times represent absolute times, is when they are used in the top most sound object in a hierarchy (of sound objects). For instance, in the figure above, when the example sound object is played (via the Play menu item), the 'Cue 1' point will always occur at time 1:0.00. If the example sound object were placed at time 1:2.00 in another sound object, and this other sound object were played, the 'Cue 1' point in the example sound object will occur at time 2:2.00.

4.1.5. Playing Sound Objects

The last item on the sound object editor menu to mention is the Play item. This item, when selected, will cause the composite sound object represented by the current sound object editor, to play itself. The sound object will playback using the tempo set on the Master System Window. The user will not be allowed to use the system while playing a sound object. The user may terminate playback of the sound object by pressing the mouse button.

There is one thing to note when creating and playing back composite sound objects. It is possible to create a composite sound object which is impossible to play. This is caused by creating a composite sound object with overlapping components that use the same channels. Recall that the creation of a simple sound object from a sequencer involves picking which channels, from the affected channel list, are to be included. The new simple sound object requires the use of those channels (usually because the sound generated by its sequencer is processed by the actual hardware represented by the channels). Therefore, if two sound objects have at least one channel in common, it will be impossible to play them simultaneously. In other words, you cannot use the same piece of hardware for two different things (if that piece of hardware can only perform one function at a time).

If the system is asked to play a composite sound object which cannot be played, due to overlapping components whose channel lists intersect, the system will not play the sound object but will respond with a message indicating the intersecting components.

4.2. A SOUND OBJECT APPROACH TO COMPOSITION - AN EXAMPLE OF SOUND OBJECT USE

In order to better explain the functionality of sound objects, this section will present two real examples. The first example is the design of a sound effect for film work. The second is the creation of a standard pop song.

4.2.1. Example 1 - Sound Effect for Film

This example will present the design of a simple sound effect for a film piece. The piece of film contains a car which enters the viewers vision from the left, drives by the viewer off to the distance on the right and then crashes into a wall. Our sound effect will be the sound of the car approaching from the left (the left speaker), passing the viewer (the sounds pans to the right), going off to the right (the sound becomes more reverberant to sound far away), and crashing. The equipment to be used consists of a MIDI keyboard/synthesizer, a

MIDI controlled recording console, a digital reverberation device, a stereo monitoring system, and, of course, our system.

The first part of the sound design is to create the sound of a car on the synthesizer in such a way that pushing down a key of the synthesizer's keyboard gives us the car sound. We then record that key press into a new sequencer (call it 'Car Sequence'). We can even use the pitch bend wheel of the synthesizer to change the pitch of the sound as the car passes the viewer to simulate the Doppler shift in the car's sound. Recall from our discussion in chapter one that the panning knob controls the stereo placement (left/right speaker balance) of a sound. We will then use the sequencer to record the movement of the panning knob, for the car sound, which will move from full left placement to full right placement as the car passes the viewer. We now wish to make the sound of the car seem farther away. To do this we will decrease the volume of the car sound and increase the amount of reverberation. Recall that we can control the amount of reverberation on a sound via the effects (or auxiliary) send control. We now record the control movements of the volume and the effects send controls into the 'Car Sequencer'.

We have now completed the sequencing for the sound effect for the car. We will now create the sound object 'Car Passing' from the 'Car Sequencer' sequencer, remembering to include the most recent mix and the channel which was controlled.

We now wish to create the sound of the crash. For this we will use a crash sound which was previously created for another part of the same film. The crash sound is stored in sound object 'Crash'. So all that's left is to put the two sound objects together into one sound object. To do this we create a new composite sound object called 'Car Passing and Crashing'. We put the two sound objects 'Car Passing' and 'Crash' onto the composite sound object at the appropriate points. Perhaps the 'Crash' component could be attached to the 'Car Passing' component. This would guaranty that no matter where the sound effect was placed in time, the 'Crash' would always occur immediately after the 'Car Passing' sound. Figure 4-6 shows the sound object editor for 'Car Passing and Crashing'.

Now all that's left is to place our sound object 'Car Passing and Crashing' onto a sound object editor for the entire film. We would place the sound object at the appropriate point in the film or we could attach it to a cue point representing the starting point for the sound effect.

Figure 4-6 - The sound object 'Car Passing and Crashing'.

4.2.2. Example 2 - Development of a Song

This example is an outline of how to use the system to develop a simple song using sound objects. This example is only one way to approach the creation of a song. It is definitely not the only way to approach the creation of a song with this system.

Keep in mind that while we present only the recording of the musical material, the production aspects are recorded right from the beginning. This is interesting since the production occurs along with the song creation, as opposed to completely after the fact.

The first step in the creation of our song is the creation of a simple drum track which we will use to synchronize to when we record our musical material. This drum track will be replaced once all the music is recorded. We then wish to create the verses for our song. We will start this process by creating a new sequencer and placing our drum pattern onto one of the tracks. We then record two bar piano and bass parts onto two of the other tracks of the sequencer. We then create the 'Verse Riff' sound object from this sequencer. The next step is the creation of the 'Chorus Riff' sound object in a similar fashion to the creation of the verse.

The two bar verse and chorus riffs which were created will be used to create the final verse and chorus parts by repeating the riffs a number of times. We do this by creating the 'Verse' composite sound object which contains 'Verse Riff' repeated twice. We also create 'Chorus' which contains 'Chorus Riff' repeated four times (see figure 4-7).

Figure 4-7 - The 'Chorus' composite sound object made up of the 'Chorus Riff' repeated four times.

The next step is to create 'Intro', 'Ending', and 'Bridge' sound objects. Finally we create the 'Song' sound object by placing all the components onto it in the order: Intro, Verse, Chorus, Verse, Chorus, Bridge, Verse, Ending.

One final step to finish up our song involves removing the temporary drum track which we used initially to synchronize the performance of the parts to. We then have a drummer perform the entire song drum parts in one long take. This is captured into a sound object and placed onto the final song. Figure 4-8 shows the final 'Song' sound object with all of its components.

Note that this has been a bottom-up approach to composition (more or less). The lowest level entities (the chorus and verse riffs) are created first and then captured into higher level composite sound objects. We could have used the system to take a top-down approach to composition. The song sound object could have been created first. Then the verse, chorus, bridge, intro, and ending sound objects would be placed onto the song. Finally, each of these components would then be defined (i.e. the musical material recorded into them).

Figure 4-8 - The final 'Song' sound object.

4.3. SUMMARY

In this chapter we have looked at the system's implementation of sound objects. We have seen that sound objects are a method of encapsulating musical material into a named entity. Simple sound objects contain only musical material (MIDI and control sequencing information), while composite sound objects allow the encapsulation of other sound objects (simple or composite). The composite sound object is used to specify a temporal ordering of its component sound objects.

Sound objects provide a good tool for the development of musical material. Their use in sound effect creation is obvious. Their use in musical composition is not as clear a benefit. They allow a very unstructured approach to composition. That is, they support top-down and bottom-up design simultaneously, which is a great benefit in an artistic environment.

Using sound objects, small, simple sound objects can be encapsulated into larger, more complex sound objects. This hierarchical approach to composition allows the composer the ability to create sounds which would be very difficult to create otherwise.

The simplicity of sound objects comes from the fact that they can be conceptualized by the user as being simple sound samples. The sound object editor reinforces this conceptualization by representing component sound objects as "black boxes", giving the user no information about them other than their name and length.

The real advantage of sound objects is that they allow the editing of the data which they represent very easily. The musical material which was used to create the sound object can be edited as easily as it was created. In addition, the sound object editor allows the creation of sound objects which can vary depending on the length of its component sound objects. The attach operations provided by the sound object editor allow specific orderings to be given to component sound objects which are independent of the lengths of the components.

5. THE IMPLEMENTATION

This chapter talks about the implementation details of the system. We will look at the implementation platform, the classes created to implement the system, and various aspects of the system design including the device structures, the sound object structures and the real-time management.

5.1. THE IMPLEMENTATION PLATFORM

The system was implemented on a Macintosh II using the Smalltalk/V Mac programming environment. This environment expects the programmer to code in Smalltalk, an object-oriented programming language, but includes the support of calls to procedures which can be written in any language for applications which must interface with low level machine routines or speed and timing sensitive routines. The Smalltalk/V Mac programming system consists of more than just a Smalltalk interpreter. It is a programming environment which allows the development of moderately large applications. It consists of a class library of approximately 160 classes which feature extensive bitmap graphics operations, very extensive list handling capabilities, and the support of user defined windows and general user interface design. The environment also includes extensive debugging and run-time variable inspection facilities, a variable cross-referencing utility, and a class hierarchy browser for browsing through the system's class library as well as for the creation and editing of new classes.

The Macintosh is an excellent platform for this system because of its standardized user interface techniques. If one develops an application which follows the Macintosh guidelines, the user will be familiar with the basic operation of the program immediately. The learning curve is substantially decreased. The system could readily be used by someone relatively new to computers. This benefits us since we can test the system on professional audio engineers to get feedback on the use of the system, and they do not have the same level of difficulty using the system as they would on another platform. There are no cryptic commands to learn. Everything is graphical and fairly obvious.

Smalltalk is an object-oriented programming language. The object oriented paradigm is a natural way to model complex objects such as the controls, channels and devices used in

the system. In addition it offers a very intuitive way to model this type of real world entity since we view the actual hardware device as an object consisting of many other objects (controls). Smalltalk seemed a natural choice for the system because of its extensive class library and development and debugging tools. For instance, the Smalltalk class library includes an extensive array of collection (list) types and operations, which allow a convenient way to manage the system libraries of channels, devices and sequencers. The inheritance which Smalltalk provides allows the creation of an abstract control type which contains the common behavior of all controls. This makes the addition of new controls simpler as they can just inherit most of their behavior from the abstract control.

The version of Smalltalk distributed by Digitalk, Smalltalk/V, is particularly well suited to this application because it allows the creation of custom designed, graphically oriented windows with great ease. It also allows the design of standardized user interface environments with great ease. Smalltalk/V Mac is particularly well suited for this application since it allows the programmer to build applications which conform to the Macintosh standards. Smalltalk/V Mac allows the simple creation of Mac-like pull down menus and windows. V Mac itself looks like a Macintosh application.

5.2. THE CLASSES

The following table (Table 5-1) is a list of all the major classes created to implement the system. The indentation of a class name indicates that it is a subclass of the class above. A short description of each class can be found in Appendix 1. The following sections will describe the more important design aspects of the system in greater detail. Note that class names with a suffix of 'Classes' do not contain any code. They are used to group related classes together in one place.

```
Thesis
  DeviceEditingClasses
    Controls
      DisplayControl
        ChannelName
        DoubleRotary
        HorizontalSlider
        PushButton
        Rotary
        Slider
      NonControlDisplay
        TextDisplay
        TrackNumberText
      SequencingControl
```

- ChannelNameSeq
- DoubleRotarySeq
- HorizontalSliderSeq
- PushButtonSeq
- RotarySeq
- SliderSeq
- SequencingControlID
- DeviceController
- DeviceEditor
- DeviceTemplate
- ChannelEditor
- ChannelOutline
- ChannelSeq
- ChannelTemplate
- EventClasses
 - Event
 - EventEditor
- MidiClasses
 - MidiPort
 - MidiMessage
 - MidiStream
- ScreenClasses
 - DeviceScreen
 - ScreenSwitch
 - SequencingScreen
 - SoundObjectScreen
 - UtilityScreen
- SequencerClasses
 - SequenceController
 - SequenceEditor
 - SequencerWindow
- SoundObjectClasses
 - SoundObject
 - CompositeSoundObject
 - SimpleSoundObject
 - SoundObjectEditor
 - SoundObjectPlayer
- UtilityClasses
 - AbstractTime
 - SmpTime
 - SongTime
 - Clock
 - Queue
 - Startup
 - SystemWindow

Table 5-1 - The list of system classes.

- ChannelNameSeq
- DoubleRotarySeq
- HorizontalSliderSeq
- PushButtonSeq
- RotarySeq
- SliderSeq
- SequencingControlID
- DeviceController
- DeviceEditor
- DeviceTemplate
- ChannelEditor
- ChannelOutline
- ChannelSeq
- ChannelTemplate
- EventClasses
 - Event
 - EventEditor
- MidiClasses
 - MidiPort
 - MidiMessage
 - MidiStream
- ScreenClasses
 - DeviceScreen
 - ScreenSwitch
 - SequencingScreen
 - SoundObjectScreen
 - UtilityScreen
- SequencerClasses
 - SequenceController
 - SequenceEditor
 - SequencerWindow
- SoundObjectClasses
 - SoundObject
 - CompositeSoundObject
 - SimpleSoundObject
 - SoundObjectEditor
 - SoundObjectPlayer
- UtilityClasses
 - AbstractTime
 - SmpteTime
 - SongTime
 - Clock
 - Queue
 - Startup
 - SystemWindow

Table 5-1 - The list of system classes.

5.3. THE DEVICE STRUCTURES

This section will look at the underlying structures used for the creation and manipulation of devices. In particular, we will look at the internal structures created during the creation of a device. We will also look at the method for sequencing and animating the controls.

5.3.1. The Device Editing Structures

We have seen, in chapter 3, how a device is created. In summary, a channel editor is used to place controls onto a channel. A device editor is then used to place channel templates onto a device. To use the device just created, we would go to the sequencing screen and create a new instance of that device. This new instance is called a device controller. This is an actual working copy of the device - the controls actually operate. In order to use the device controller window to control the actual hardware device, appropriate MIDI messages must be mapped to each of the controls. Once this mapping has been performed, whenever a control is moved, it will transmit the corresponding MIDI message, which will be received by the actual hardware device causing its control (or parameter) to change.

We will now discuss how the operation of creating a device is represented internally. The relevant classes are: DeviceController, DeviceEditor, DeviceTemplate, ChannelEditor, ChannelOutline, ChannelSeq, ChannelTemplate, and the subclasses of DisplayControl, NonControlDisplay, and SequencingControl.

The class ChannelEditor implements the channel editor window. This editor operates on an instance of ChannelTemplate. That is, when a new channel is created, an instance of ChannelTemplate is created and placed in the system library. When the editing window is opened for this channel template, a ChannelEditor is created to edit this template. The channel template contains the channel related information (the list of controls on this channel), the channel editor contains the editing related information (the channel template being edited, the currently selected control). When a new control is added to the channel, it is added to the list in the channel template.

The six subclasses of DisplayControl (ChannelName, DoubleRotary, HorizontalSlider, PushButton, Rotary, Slider) correspond to the six control types which can be added to a channel. When a new control is added to a channel (using the channel editor), a new instance of one of these six classes is created and added to the control list stored in the channel template. This new control instance contains its coordinates within the channel.

Note that these coordinates (values between 0 and 1) represent the fractional position within the channel bounding rectangle, and not the absolute pixel (integer) position. This coordinate system is used to facilitate the easy conversion (scaling) of the positions to absolute pixel coordinates. This is necessary since, when editing windows are used at various sizes, the elements they contain must be scaled to the appropriate position and size.

The device editing structures are analogous to the channel editing structures. The class `DeviceEditor` implements the device editing window. The editing operations are performed on an instance of `DeviceTemplate`. A channel added to the device template is a new instance of `ChannelOutline`. The class `ChannelOutline` is used to represent the channel outlines which are placed onto the device template. The channel outline instance will contain its position within the device bounding box, and a pointer to the channel template which it represents. The device template instance contains a list of all the channel outlines contained within it.

Note the differentiation between channel templates and channel outlines. While they both are used to represent a channel, channel templates are containers for the controls on a channel, and channel outlines hold the position of a channel on a device. The purpose of the two cannot be combined into one class since a device may contain many channels of the same type. In this case, the device template would contain one instance of `ChannelOutline` for each of the channels on the device. There would still only be one instance of the channel template, though.

Figure 5-1 shows the structures of the device editing operations discussed so far. In the figure, the device template contains many of one type of channel outline, and one of another type of channel outline¹³. The device editor is used to edit this device template. The many duplicated channel outlines all point to the same channel template. The remaining channel outline points to a second template. Each template has its own channel editor which is used to add the controls to the template.

¹³This is a common example of the design of the interface to a small recording console where there would be many input channels and one output channel.

Figure 5-1 - The device editing structures.

Note that the code is designed in such a way as to allow new control types to be added very easily. The system makes use of a general mechanism to request control specific information (attributes) from the user. Using attributes, we can generalize the editor/control model so that new control types can be added easily.

5.3.2. The Device Controller Structures

Now that we have looked at the structures created during the editing of a device, we will look at the structures created when we create a device controller window. In order to use a device, a new instantiation of the device must be created. This instantiation process creates a device controller window for the device. This instantiation process uses the device editing structures to build a structure which will allow efficient sequencing. This system tries to retain the channel abstraction by creating a device controller which contains only channels. Each channel will be a container for the controls which appear on that channel.

The controls which appear on the device controller window are instances of the subclasses of `SequencingControl` (`ChannelNameSeq`, `DoubleRotarySeq`, `HorizontalSliderSeq`, `PushButtonSeq`, `RotarySeq`, `SliderSeq`). These classes implement the sequencing version of the controls in `DisplayControl`. Each one of these classes corresponds to one of the subclasses of `DisplayControl`. During the building of the device controller structures, a new instance of one of these sequencing controls must be created for each control which appears on the final device controller window. The purpose of the sequencing control is to serve as a container for the control specific information such as the MIDI message corresponding to this control, the current value of the control, whether the control has been punched in (during sequencing), the actual pixel coordinates of the control's bounding box on the screen, and a pointer to the display control which this control corresponds to.

The class `ChannelSeq` is used to represent channels on a device controller window. A new instance of this class will be created for every channel which appears on the device controller window. This class serves as a container for the sequencing controls on that channel, as well as its bounding box on the device controller window.

The class `DeviceController` implements the device controller window. This window displays the device and allows the user to operate the controls on the window. The device controller instance serves as a container for the device specific information such as the device's MIDI channel number, as well as the list of the channel sequencers (the instances of `ChannelSeq`).

Figure 5-2 shows the structures created when a new device instantiation (device controller) is made. The device controller contains a list of the channel sequencers. Each channel sequencer has a list of the control sequencers associated with that channel. Each control sequencer represents one of the controls which appear on the device controller window. Note the pointers from the device controller, channel sequencer, and control sequencer back to the corresponding entities in the original device structure. These pointers allow the channels and controls to know their location without storing it explicitly in themselves. This allows simple editing operations (moving and resizing controls and channels) to be immediately reflected in the final device without requiring the creation of a new device controller.

Figure 5-2 - The device controller structures.

5.3.3. Sequencing

This section will look at the method for sequencing and animating the controls. We will look at the prioritization of the display operations, handling of the punch-in and punch-out operations, and the interfacing of the device controller with the sequencer. The system has been designed so that the real-time operations are as efficient as possible. In particular, a queued display system has been incorporated so that the display is updated only when no other time critical events need servicing.

The class `SequenceController` is the real-time interface between the system and Smalltalk. The sequence controller is called periodically by Smalltalk's main I/O polling loop. When called, the sequence controller will allow the currently active sequencer to process any pending events. The events which the sequencer has to process include incoming MIDI information (to be recorded onto tracks of the sequencer) and display queue

processing. The display queue is a queue, kept by the sequencer, of pending display operations. When the value of a control is changed by the sequencer (during sequence playback), that control will not update the display with its new position immediately. Instead, it will place itself on the sequencer's display queue. When the sequencer has no other events pending, it will process the display queue by telling each control contained within it to update the display with its new position. This postponing of the display updating allows the system to respond immediately to more important realtime events such as incoming MIDI messages.

The handling of the punch-in and punch-out of controls requires special attention. We have seen, in the last section, that the controls on the device controller window are represented by instances of the subclasses of `SequencingControl`. We also learned that the control's current position (value) is stored within itself. In order to implement punch-in and punch-out, the controls must also store a second value which is the value which the control should return to when punched out. When the user grabs a control during sequencing, that control is put into punch-in mode. The user moves the control around and then returns to the previously programmed sequence for the control by hitting the punch-out button. At this point the control must return to the position at which it would be at this point in time had the user not punched-in this control. In order to implement this, it is necessary to store a second value within the control. This value would be used during punch-in mode, and would hold the last position of the control set by the sequencer.

Almost all of the logic for the handling of the events associated with controls is contained within the control classes. We will now look at how these events are handled (using an informal pseudo-code style to show the methods which handle these events). Before we begin with the operations, we must first look at the information contained within the control itself which aids in the processing of events. The following fields are contained in each control:

- value - the current value of the control
- display pending flag - a flag indicating if the control is on the display queue
- punch-in flag - a flag indicating if the control is punched-in
- punch-out value - the value the control should return to when punched-out
- midi message - the MIDI message which the control will output when changed

Another important structure is the list of recorded control events kept by the sequencer. This list contains a time stamped event for each case that a control is moved. During sequence playback, this list is scanned by the sequencer. For each event in the list, at the

appropriate time, the sequencer will tell the control to move to its new position. When the controls are moved manually by the user (punch-in mode), an event will be placed onto this list each time the control changes position.

We will begin our discussion of the event processing methods by looking at the sequencing initialization method for controls. The sequencing initialization method of each control is called when sequence playback (or recording) is initiated. This method initializes the flags which are used during sequencing: the display pending flag, and the punch-in flag.

```
sequencing initialization
  clear the display pending flag
  clear the punch-in flag
```

The select method is called when the mouse is pressed down within a control's boundary. This will engage punch-in mode and move the control to the position of the mouse (cursor). Note that the on-screen position of this control is updated immediately (and is not placed on the display queue). This is necessary since the user must get immediate feedback when moving controls. This method will also inform the sequencer that the control has been changed by the user. The sequencer will time stamp this event and store it in its list of recorded control events.

```
select
  store value into the punch-out value
  set the punch-in flag
  set the value to the new position
  update the display immediately
  if the sequencer is on then record this change with the sequencer
  output the MIDI message with the value
```

The animate method is called when the mouse is dragged (movement with the mouse button down) within a control's boundary. This method is called by the system each time the mouse changes position while the mouse button is held down. When the mouse button is pressed down, the system calls the select method, and while the mouse is dragged it calls the animate method repeatedly. This method perform a similar operation to the select method except there is no need to engage punch-in mode.

```
animate
  set the value to the new position
  update the display immediately
  if the sequencer is on then record this change with the sequencer
  output the MIDI message with the new value
```

The change value during sequencing method is called by the sequencer each time it wishes to change the value of a control. This occurs when the sequencer scans through its recorded control events list. If the control is punch-in mode, the new position which just be stored in the punch-out value. If the control is not in punch-in mode, then the control's value will be updated, its corresponding MIDI message sent out, and the control will be placed on the display queue.

```
change value during sequencing
  if punch-in flag set then
    store the new value as the punch-out value
  else
    if the window is active and display pending flag not set then
      set display pending flag
      add this control to the display list
    store new value into value
    output the MIDI message with the new value
```

The activate window method is called when the window which the control is on becomes the active window (that is, the window is moved to the front). Since control displays are not updated when a window is behind another window, this method will redisplay the control on the screen.

```
activate window
  if sequencer is on then
    if display pending flag not set then
      set display pending flag
      add this control to the display list
    end if
  otherwise
    update the display immediately
```

The display method is called during the processing of the sequencer's display queue. This method will display the control on the screen.

```
display
  if the window is active then update the display immediately
  clear the display pending flag
```

The punch-out method is called when this control is punched-out (because of selection of the punch-out last control, or punch-out all controls buttons on the sequencer window). This method will change the control's value to its last programmed value.

```
punch-out
  store punch-out value into value
  if the window is active and display pending flag not set then
    set display pending flag
    add this control to the display list
  record this new value with the sequencer
  clear the punch-in flag
  output the MIDI message with the new value
```

5.3.4. Inheritance in the Design of Controls

The inheritance which Smalltalk provides has been a very useful feature in the design of this system. In particular, inheritance has been used in the design of the control types to create a set of six control types with very little difficulty. In addition, inheritance allows the addition of new control types to the system with a minimal amount of programming effort.

As was discussed earlier, the control implementation is split into two parts: the part which implements the editing of the controls, and the part which implements the sequencing of controls. The class `DisplayControl` provides the abstract behavior for the controls used for editing. In particular, this class provides the behavior for a rectangularly shaped control which can be placed anywhere on a channel editing window. To add a new control type, a new subclass of this class would be created. As long as this subclass were implementing a rectangular control, this subclass would just have to implement two methods. The first method would be an instance method which would draw the representation of this control in a rectangularly shaped area. The second method would be a class method which would simply return the text name of this type of control ('Horizontal Slider' for instance).

If the programmer wished to create a type of control which did not have the same behavior as the abstract control type, the programmer could still use inheritance to his advantage. He could still create a subclass of the abstract class, but, in this case, would

have to implement more than just the two methods described above. He would have to override the methods which would normally be inherited from the superclass, with methods which provide the appropriate behavior for that new type of control. The methods which implement the common behavior between the abstract control and the new type of control would not have to be re-implemented, however.

As an example, suppose the programmer wanted to add a control type which was similar to the rotary control, but instead, did not have a rectangular bounding box around it. During the editing of the control, the user would resize the control by grabbing the inside portion of the control near the rim (instead of the bottom right corner as in the case for rectangular controls). We can implement this new type of control very simply by noting that a rectangular bounding box could still be used to represent the coordinates of the control internally. The only difference in behavior from the abstract control, is that we will not draw a rectangular box around the control and to resize the control, the mouse must be near the rim of the circle instead of the rim of the rectangle. Therefore, this new control can be implemented by creating a new subclass of `DisplayControl` with only four methods:

- the class method to return its text description
- the method to draw itself
- the method which determines if the cursor is near the resize point of the control
- the method which determines if the cursor is within the control (for moving the control)

The power of inheritance is again demonstrated in the design of the class `SequencingControl`. This class provides the abstract behavior for controls used during sequencing. In particular, it assumes the control has a value, has a moveable indicator which indicates this value, and contains a MIDI message which is sent out when the indicator changes position. A new type of control can be created by adding a new subclass to this class. If the new control type has the same basic behavior as the abstract control, then only a few methods need be implemented, and all the methods which implement the sequencing behavior can be inherited from the abstract class. The methods which must be implemented in the subclasses are:

- the method to draw the background of the control
- the method to draw the indicator (at a specific value)
- the method to undraw the indicator (the undraw operation erases the indicator without erasing what was underneath the indicator)
- the method which will return the new value of the control given the position of

a mouse click within the control's boundary

As in the case for the `DisplayControl` class, a control with a behavior different from that of the abstract behavior of `SequencingControl` can be created by overriding the methods in the superclass which contain the behavior that must be changed.

5.4. THE SEQUENCER

The details of sequencing have been described earlier in section 5.3.3. The sequencer is mentioned here for completeness. In summary, the things to note about the sequencer are:

- the sequencer holds the tracks, where each track is a list of time stamped MIDI events.
- the sequencer holds the mix list. Each mix is a collection of time stamped control movement events.
- playing the sequencer processes the lists in order.
- recording saves the incoming MIDI data into a new track.

5.5. THE SOUND OBJECT STRUCTURES

This section discusses the internal structure of sound objects and the method for playing sound objects.

5.5.1. The Structure

Recall that there are two types of sound objects: simple and composite. Simple sound objects are a subset of the information contained in a sequencer. This subset consists of a subset of the eight tracks of the sequencer, and a subset of the affected channels of one of the mixes. Composite sound objects are sound objects which contain other (simple or composite) sound objects. These component sound objects occur at specific times within the composite sound objects.

The class `SoundObject` is an abstract class which implements the common behavior of all sound objects. Its two subclasses, `SimpleSoundObject` and `CompositeSoundObject`, implement the two types of sound objects. This common behavior includes the ability to store and retrieve the name of the sound object, the ability to retrieve the length, and the ability to play the sound object.

The internal structure of a simple sound object is relatively simple. The simple sound object contains a pointer to the sequencer which it uses, and a list of the tracks, within that sequencer, which are included in the sound object. The sound object also contains a pointer to which one of the mixes from the mix list are included in this sound object, and a list of the channels from within that mix which are to be used.

The internal structure of composite sound objects is equally simple. The composite sound object contains a simple unordered list of all the sound objects contained within it. Within this list is the time at which each sound object occurs. Note that the reason the sound objects are unordered, is that the actual times at which the sound objects occur cannot be determined without knowing the tempo at which the sound object will be played at. Recall that SMPTE time references must be converted to song time references using a tempo factor. As well, song time references must be converted to SMPTE time references using a tempo. Therefore, component sound objects placed at SMPTE and song time marks within a composite sound object cannot be ordered in time without first knowing the tempo at which the sound object will be played back at. This tempo is only available when the sound object is played.

5.5.2. Playing Sound Objects

The class `SoundObjectPlayer` implements the timing mechanism for the playing of sound objects. The actual logic for the playing of the sound objects is contained within the sound object classes. The `SoundObjectPlayer` class is basically the interface between the system's timing utilities, and the sound object playing routines. It implements the driver for the calling of the sound object's `play` and `playNext` methods.

Most of the logic for playing sound objects resides in the sound object classes. The sound object classes implement two methods for their playback: `play` and `playNext`. The `play` method is called once to initiate playback of the sound object. The sound object will initialize any variables it needs to play itself, and will return the time (relative to now) at which the `playNext` method should be called next. The `playNext` method will then be called repeatedly to play the sound object. After the method has played the events which occur at the current time, it will return the time at which the `playNext` method should be called again. The `playNext` method for the sound object will not be called again until that time.

The playback method for simple sound objects is relatively simple. It basically processes the events on the track lists and the selected mix list in order (since they are already ordered by time). The `play` method for simple sound objects basically initializes internal pointers to point at the beginning of these lists. It then returns with the time of the

first event. The `playNext` method just plays all the events which occur at the current time. When all these events have been processed, it returns with the time of the next event to be processed.

The playback method for composite sound objects is a little more involved. Before the sound object is played, the components must be sorted in order of starting time. To do this, all the components which start at times relative to SMPTE timing references must have their start times converted to song time format. This is the major job of the `play` method for composite sound objects. Once the sorting procedure is done, the `play` method will initialize some internal pointers, and then return with the time of the first sound object. The `playNext` method of composite sound objects serves a similar purpose that the `SoundObjectPlayer` class does. It serves as a driver for the component sound objects' `play` and `playNext` methods. That is, since the playing of the component sound objects is handled by the components themselves, the composite sound object's `playNext` method must call the component's `play` and `playNext` methods. The `play` and `playNext` methods of the component sound objects cannot tell if they are being called by a composite sound object player or by the `SoundObjectPlayer`'s driver. The job of the composite sound object's `playNext` method is to manage the calls to the components at the appropriate times. As usual, when this method returns, it will return the time when it should be called next.

5.6. THE MIDI MANAGEMENT CLASSES

This section will discuss the classes designed to implement the interface to the MIDI interface. The three classes are: `MidiPort`, `MidiMessage`, and `MidiStream`. The `MidiPort` class is the system's interface to the MIDI port. This class is responsible for the interfacing with the C primitives¹⁴ which open the MIDI port, and read and write strings of bytes to the MIDI port.

The standard Macintosh is capable of supporting two MIDI ports: one connected to the modem interface, and one connected to the printer interface. Once the `MidiPort` class has been initialized, the two class methods `MidiPortA` and `MidiPortB` will each return an instance of the class which will interface with the modem MIDI port and the printer MIDI port respectively. The two important instance methods in this class are: `read` and `write`. The

¹⁴The C primitives are short routines, written in C, which allow the Smalltalk code to interface with the Macintosh ROM routines. The particular ROM routines involved in this case allow the opening of the MIDI port, and the reading and writing of strings of bytes. The C primitives are necessary since the Smalltalk language offers no support to call the toolbox routines directly.

read method will return a byte array containing the bytes which were sent to the MIDI input port since the last read. The write: method will send an array of bytes out through the MIDI output port.

The class `MidiStream` provides a higher level interface to the MIDI port than does the class `MidiPort`. The `MidiStream` class allows the reading and writing of MIDI messages (instances of class `MidiMessage`) to the MIDI port. Since the read and write operations this class provides deal only with MIDI messages, this class is responsible for the parsing of the bytes in the MIDI stream into MIDI messages. The two important methods implemented in this class are: `read` and `write:`. The `read` method returns the next MIDI message in the MIDI stream (or `nil` if there is no messages in the buffer). The `write:` method will send the MIDI message out the MIDI output port.

The class `MidiMessage` is used to represent MIDI messages. This representation is used as a higher level representation than provided by simple byte arrays. Instances of this class can be created by an application and then written to the MIDI port using class `MidiStream`. Instances of this class are also created and returned by the `MidiStream` `read` method during the parsing of the incoming MIDI bytes.

5.7. THE SCREEN SWITCHER

The screen switcher classes are the last major part of the system to discuss. The screen switcher classes implement the virtual screen facility. This facility was developed because a typical user of this system would usually have many windows open at the same time. As well, the screen concept reinforces the abstract level concept (described in chapter 3), since the screens have been designed to incorporate related operations onto the same screen.

The class `ScreenSwitch` implements the logic for the screen switching utility. This class manages the Screen pull-down menu, and allows the addition and deletion of screens. The screen switcher contains a list of all the windows on each screen. The screen switcher creates the illusion of a screen switch by closing all the windows on the current, and opening all the windows which correspond to the screen to be switched to.

The classes `DeviceScreen`, `SequencingScreen`, `SoundObjectScreen`, and `UtilityScreen` implement the screen specific operations. Each class is responsible for the creation of the menu used on that screen and the handling of all the menu operations for that menu.

5.8. SUGGESTIONS FOR FUTURE IMPROVEMENTS

Currently the system consists of approximately 60 classes and approximately 15,000 lines of Smalltalk code (500K bytes of Smalltalk source code). We can reduce this code bulk somewhat by a redesign of parts of the system. In particular, we could create an abstract class for the graphical editors (the channel editor, the device editor, and the sound object editor). This abstract class would contain the common behavior to these types of editors. This abstract behavior would consist of an editing window which allowed the addition of rectangular components. The editor would allow the user to select, drag and resize these components. When the window was resized, all of these components would be scaled appropriately. This abstract class would support the addition of subclasses (ChannelEditor, DeviceEditor, and SoundObjectEditor) which completely defined the behavior of the editor.

All of the graphical windows (the device editor, the channel editor, the sound object editor, and the device controller window) in this system have very similar behavior. For instance, all of the windows understand mouse down and drag commands. All of the windows know how to draw themselves. Again, code bulk could be reduced by creating an abstract class which handled a generic window model. This class would contain the common behavior of the management of the bitmaps which the windows draw their contents on, the handling of the mouse events, and the handling of window events, such as resizing and scrolling. All of the windows classes would then be made subclasses of this abstract class.

The changes discussed above would, in addition to decreasing the amount of code, increase the understandability of the system. The abstract classes discussed introduce a logical relationship among the various types of windows/editors.

6. EXTENSIONS TO THE SYSTEM

This chapter will look at extensions and generalizations of the system. The first part of this chapter deals with short term extensions to the system. These extensions can be implemented with a small or moderate amount of effort and are described in a fair amount of detail. The second part of this chapter looks at the virtual (or desk top) recording studio. This concept can be considered as a projection of the ideas presented in this thesis into the near future.

6.1. EXTENSIONS

This section of the chapter looks at short term extensions to the system. The extensions range in complexity from simple to moderately complex. These extensions include expanding the ability of current system features, such as the addition of new features to the existing editors, as well as creating completely new features, such as new types of editors. Finally, this section will discuss general changes to the system to increase its usefulness. These changes and extensions are indeed implementable and are described in enough detail to aid someone wishing to extend the system with these features.

6.1.1. General Editor Features

There are several general editing features which are relevant to all of the graphically based editors (the channel editor, the device editor, and the sound object editor). The first feature is the addition of cut, copy, paste, and duplicate abilities to the editors. The cut, copy, and paste operations should allow the transfer of the edited components between different windows of the same type. In addition, some sort of group selection feature should be incorporated to allow the copying of groups of components between windows. For instance, it should be possible to select a group of controls from one channel editing window and copy them into a channel editing window for another channel.

The editors could also benefit from some kind of alignment facility where components can be aligned with one another or centered vertically or horizontally within the window boundaries.

Another feature which all of the editors could benefit from would be an undo operation. The undo command would allow the nullification of the last operation performed by the user. A redo command, to undo the affect of an undo command, would also be useful.

Currently, all the editors perform their editing operations on the actual copy of what they are editing. That is, all editing operations are destructive. Perhaps a better approach would be to perform all editing operations on a copy of the data, and when the user is finished editing, he would have to save his changes back to the original object being edited. This would allow a safer editing environment, where an entire editing session of a particular object can be undone.

6.1.2. Sound Object Editor

There are several fairly simple features which can be added to the sound object editor. The first feature involves error checking. As was mentioned in section 4.1.5, "Playing Sound Objects", the system cannot play composite sound objects which contain overlapping components whose channel lists intersect. When an attempt is made to play such a sound object, an error message will be generated by the system. Currently, this error checking mechanism is done at play back time only. This error checking could be extended to perform the intersection checking during the editing of the sound object. Note that this type of error checking is a little more involved than it first seems. The obvious method to implement this checking mechanism would be to perform checking for overlap each time a composite sound object's components were modified. But this checking must be extended to include more than just the composite sound object currently being edited. Any modifications to a composite sound object which is a component of a second composite sound object, may have an affect on this second composite sound object. Indeed, all composite sound objects which contain the edited composite sound object somewhere in their hierarchy must be rechecked for intersection.

The second feature which can be added to the sound object editor is the ability to repeat component sound objects. There would be two types of repeats: indefinite repeat and repeat a sound object a fixed number of times. Note that this repeat function is different from and in addition to the duplicate ability described above for general editor features. The repeat function allows a number of copies of a component sound object to repeat one after the other. In effect, all the copies are attached to one another, so that if the length of the component sound object changes, these components will still follow each other directly. The same effect can be achieved with the duplicate facility if all of the copies were attached

to one another in sequence. Note that the indefinite repeat allows the creation of a sound object with no specific end point. It will play indefinitely if not terminated by the user.

6.1.3. Control Movement Editor

The control movement editor is a new editor which would allow the editing of the control movement information which is recorded in a mix. In the current system, the only method of entering control movement information is to perform it in real time. This certainly has its disadvantages, especially in the case where a very fine grained accurate control is required over the movement of a control. The control movement editor would allow the user to enter very accurately placed (both in time and position) control movements.

When an editing window is opened, the user will be asked for which control he wishes to open the editing window on. The system would then search the current mix for all cases where the specified control was moved. Once collected, these movements would be displayed graphically on a position versus time graph.

The editor would present the user with an editing window of position versus time. The time axis would be scrollable, and both axes could be scaled independently. The movements would be displayed as a series of connected control points. These control points could be grabbed by the mouse and moved on either axis. As in the sound object editor, control points could be attached to the system's events (which appear at the top of the window). Additional abilities could be given to allow the user to fit a smooth curve (spline or Bezier curve) to the control points. Figure 6-1 shows an example of a control movement editor window. The point with the circle around it indicates a control point which is attached to a system event, 'Cue 1' in this case.

Figure 6-1 - The control movement editor.

6.1.4. Music Editors

The current method for editing the sequenced information contained in the tracks of the sequencer is through the MIDI track editor. This editor allows a very primitive editing mechanism since it only offers the ability to edit the actual MIDI event information which is received. If this system were being used in an actual music creation context, a higher level music editor would be desirable. For instance, a standard score editor would be an appropriate editor for many composers. This editor would allow the user to place notes on a graphical representation of a music staff. Figure 6-2 shows one approach to a score editor. The panel at the left of the window is used to select the particular object (note or rest) to be placed in the score. The mouse can be used to select sequences of notes which can be copied or moved to other parts of the score.

Figure 6-2 - A graphical score editor¹⁵.

Another type of editor which could be useful to a composer is a piano-roll display type editor. This editor displays notes on a vertically and horizontally scrollable window. The vertical axis would represent the pitch of the note. The horizontal axis would represent time. At the time at which a note is played, a rectangular box will appear. The length of the box represents the length of the note. Figure 6-3 shows an example of this type of editor.

¹⁵The graphical icons for this figure have been taken from the Deluxe Music Construction Set v2.5 ©Electronic Arts.

Figure 6-3 - A piano-roll type score editor.

6.1.5. Control Shape Editor

The system currently supports four types of controls which can be placed on devices for the purpose of controlling external hardware. These control types are: horizontal slider, vertical slider, rotary knob, and double rotary knob. A generic control shape editor would allow the user to create his own control types. This editor would allow the user to create the graphics for the background of the control and of the movable needle and to describe the animation process for the needle.

The background would be created by placing primitive graphics objects (rectangles, lines, circles) in a box representing the outline of the control. The graphics primitives used here must be scaleable so that when the control is scaled to different sizes, the control looks the same. Therefore, bitmap pictures cannot be used for control backgrounds, since bitmaps cannot be scaled to arbitrary sizes.

The control's needle graphic would be created in a similar way to the background. The user could then specify how the needle moves over the background: up/down, left/right, rotate around a point, or slide between two arbitrary points. The user could also specify the resolution (the number of values) of the control: two for push buttons, 127 for general

slider type controls, and 10 for controls like panning knobs (since only a limited resolution is required for panning settings).

This type of control shape editor would allow the user access to control types which were not included with the system. Using this ability, the user could create device interfaces which more closely match their mental model of the device.

6.1.6. The Sequencer

The sequencer is the heart of the automation system. Therefore, we can enhance the automation system with the addition of new features to the sequencer. One particular feature involves the interpretation of the mixes. The sequencer stores the old mix each time a new mix is created. While this new mix was based on the older mix, the new mix becomes a completely independent mix of its own. The system does not retain any dependency ties between successive mixes. If the older mix is edited, the mixes based on that older mix are not affected.

In some cases it may be desirable to retain a dependency to an older mix. When the user creates a new mix, the system could offer the user the choice of either creating a new independent mix, or creating a mix which contains only the changes from the previous mix. In the latter case, the user could go back and edit that previous mix and have the changes reflected in the new mix.

A second feature, which would greatly increase the usefulness of the automation system, is the capturing of a set of controls' positions and placing these into the current mix over a subset of time. This feature allows the user to define a set of control settings to be used for a period of time, after which the controls return to their previous programmed positions. In effect, this feature allows the user to do a simultaneous punch-in and punch-out on a set of controls.

The user would use this feature by specifying the controls (and their settings) to be used. The user would then specify the period of time at which to use these settings. The system would then place the corresponding control movements into the current mix. This feature allows the user to manipulate the controls in an off-line fashion (when the sequencer is off) until he gets the particular effect he wants. Then he can place the control movements into the current mix at the specified times (like a subroutine call).

6.1.7. Device Controller

The device controller window allows the editing of the MIDI messages assigned to the controls. In many instances where there are many similar channels on a device it is the case

that the MIDI messages for corresponding controls on each of these channels is very similar. This window could benefit from the ability to propagate the MIDI message assignments of one channel onto the other similar channels of the device. This propagation would involve the incrementing of the controller (or note) number mapping of the message of each successive channel.

The device controller also allows the setting of the initial positions which the controls should be in when the sequencer is started. A great benefit would be achieved if this window allowed the user to copy these settings from one channel to the other similar channels.

6.1.8. Sound Samples as Sound Objects

In the introduction to chapter 4 we saw that sound objects are a way of representing musical control information as if it were an actual sound sample. The obvious question to ask then is, can we extend the sound object paradigm to represent actual sound samples? The answer, of course, is yes.

Currently, the system only deals with the control of external synthesizers, samplers, consoles, and effects devices. In its current state, there is no way to handle digital sound samples internally. With the addition of audio analog-to-digital and digital-to-analog converters into our system, the system could be extended to handle digital sound samples. In particular, the system would offer a subsystem which would allow the capturing of audio information and allow the digital editing of the created sound samples. These sound samples could then be used as simple sound objects in the system. These sound samples could then be placed as components in composite sound objects in parallel with other sound samples or other simple sound objects which control external synthesizers.

Note that the direct support of sound samples by the system does not offer any more functionality than can be achieved by the system's triggering of samples on an external sampler. The integration of the support of sound samples into the system, though, does offer a more elegant solution to the handling of sound objects. In particular, if the system stores the sound samples internally, it can compute the length of the sound object automatically¹⁶. In addition, higher level operations such as looping a sample, or playing a

¹⁶A sound object can be used to represent a sample which is on an external sampler. This sound object would contain one note which would trigger the sample on the external sampler. The user would have to set the length of the sound object manually to represent the actual length of the sample, since the system has no way of knowing the length of the sample, or even knowing that the note in the sound object is used to trigger a sample.

sample backwards could be added to the composite sound object editor to allow the use of these effects during the composition stage (as opposed to the sound design stage).

6.1.9. The Routing Screen

When doing a mixdown, it is often desirable to be able to control a group of controls together. The group of controls would be increased and decreased together. For instance, take the case of recording a group of four background vocalists. Each vocalist's microphone would be brought into a separate channel on the mixing console. Once the relative level of each vocalist was set, we would only wish to deal with the level of the group as a whole. We might, for instance, want to increase the level of the group during the chorus of the song, and decrease it during the verses. The relative levels of the vocalists remains the same throughout.

We can achieve this type of grouping operation by introducing the ability of a control which can be used to manipulate other controls. To completely generalize this concept, we will say that every control has both an input and an output. A control's output can be used to control an actual hardware control or it can be mapped onto the input of another control. An arbitrary function can be used to specify the mapping of the output of one control to the input of another. This mapping can also be an absolute mapping, or it can specify a relative position from the current setting of the control.

The output of a control is not limited to the control of only one other input. Each output could be connected to an indefinite number of other control's inputs. Using this concept, the user could create a virtual device interface whose controls were used to manipulate groups of controls on other devices. Each time one of these controls were moved, every control that it was mapped to would also move. The relative (as opposed to absolute) mapping scheme could be used so that the relative levels of the controls in the group could be set independently of one another, but the group of controls would still increase or decrease by the same amount when the master control was manipulated.

The mapping of the outputs of controls to the inputs of other controls would be performed on a new screen: the routing screen. This mapping could be done either graphically or textually. A graphical representation of this mapping would offer a box with an input and output for each control involved in the operation. The user could connect the output of one box to the input of another box with a wiring tool that allowed the creation of a path between the two boxes.

We could also generalize the routing concept completely by using the routing screen to specify the mapping between controls (which are connected to hardware) and the actual hardware which they control.

6.2. THE VIRTUAL STUDIO

The remaining portion of this chapter is devoted to a discussion of the virtual studio concept. The virtual studio is the computer simulation of studio components which do not really exist. The system described in this thesis can be used to implement this concept to some degree. We saw how this system can be used to create "virtual" interfaces to devices in the studio. That is, the user can create his own interface to each device in the studio. This interface does not necessarily correspond exactly to the actual functionality of the device being controlled. For instance, we saw how a virtual 16 channel console interface could be used to control two 8 channel consoles. Also, novice users could use a simplified version of an interface to a device. The actual devices which these interfaces reflect do not actually exist.

The virtual device concept supported in this system is only a fraction of the actual functionality which could be achieved with a full-blown virtual studio concept. A complete virtual studio system would allow the user to create an arbitrary device, possibly in a similar fashion to the way this system allows the user to create interfaces to devices. The system would allow the user to place controls on a device and to define what each control does (how it manipulates the sound passing through it). The user could create channels of controls where the sound would pass through each control in the channel. Each control, in turn, would modify the sound in some way. Once the user completely described the "virtual" device, the computer would then use the hardware in the studio to simulate the device. The computer would route the audio signal through the devices required to perform the operations defined by the virtual device. For instance, a recording console with EQ can be simulated by a console with no EQ and a separate EQ device. The computer would route the audio signal through the EQ device followed by sending the EQ output to the console.

6.2.1. Dynamic Routing of Audio Signals

One benefit which can be achieved through this kind of intelligent system is that the computer could allow better use of the hardware in the studio. The computer could allocate

resources, in real-time, which would be needed to create a segment of sound¹⁷. For instance, suppose that we have two mixing consoles: one with parametric EQ and a simpler one without EQ. Now if we have a sound which requires EQ during only a small portion of the time that it is played, we could normally route the signal to the simpler console. When the time occurs that the sound requires EQ, the sound can be switched to the console with EQ. When the EQ is no longer needed, the sound can be switched back to the other console, thus freeing up a channel on the console which can then be used to process another signal requiring EQ. This intelligent dynamic routing of audio signals allows us to build a lower cost studio. It is not necessary to equip a studio with a large sophisticated recording console when only twenty percent of the functionality of the console will ever be used at one time. The same functionality can be possibly achieved with a small complete console and a large simpler console.

6.2.2. DSP

The introduction of a DSP chip into our system would allow us to extend the virtual studio concept even farther. We could use the DSP to emulate any device in the studio. The DSP can perform mixing, EQ, sound processing, sound synthesis, or sampling. We could use the DSP to fill in for any piece of hardware which the user required that the studio did not contain. In fact, it would be possible to simulate an entire studio with only DSP chips.

6.2.3. The Desktop Studio

As DSP chips become faster and more powerful, we can emulate increasingly more complex devices with only one chip. Indeed, there will come a time when the entire functionality of today's recording studios will be contained in a personal computer. This view of the future might seem to invalidate everything discussed in this thesis. After all, the computer has direct access to the DSP chips and is no longer limited to communications in the MIDI protocol. But one of the more important contributions of this thesis has been the definition of an environment in which the user can create his own "virtual" devices. The need for this kind of an environment becomes even more important when we look to the desktop studio. The desktop studio's DSP chips will be able to emulate any device, so the ability for the user to create his own devices becomes easier. More importantly, if the user

¹⁷This real-time allocation of studio resources is only possible through the ability to change the routing of audio signals through the system in real-time. This is made possible by the new crop of MIDI controllable audio patchbays which are appearing on the market.

is allowed to create his own devices, the need for the kind of environment described in this thesis becomes essential.

7. CONCLUSION

This thesis has discussed a recording studio automation and composition system for the small recording studio. This system provides a centralized, consistent user interface to all of the hardware in the studio. This interface is seen as the first step towards the virtual studio concept. The system also provides a method for mixdown type automation for all of the studio devices. Finally, the system offers an integrated compositional tool to capture performance and mixdown sequences into named entities - sound objects. Sound objects can be combined and temporally arranged for playback. They also offer a hierarchical approach to the composition process through the grouping of sets of sound objects into larger, more complex sound objects.

7.1. CHAPTER SUMMARIES

This following sections summarize the contents of each of the chapters in this thesis.

7.1.1. Chapter 1

Chapter 1 introduced the recording studio, its components and the recording process. The major studio personnel were introduced and their roles in the recording process were discussed. This chapter also introduced MIDI and sequencers. The main purpose of this chapter was to introduce the reader to the terminology and concepts to be used in the thesis, as well as to demonstrate that there is a great need for automation in the studio.

7.1.2. Chapter 2

Chapter 2 discussed the types of automation which is currently available for recording studios. It also discussed the technology which makes this automation possible, and the problems with the current systems. The problem with using an automated console with a sequencer is that standard sequencers don't have the intelligent data handling required by the console, and can't provide all the operations required by a user of this type of automation system.

Finally, this chapter introduced the notion of sound objects as implemented in Kyma [SCAL]. Kyma's sound objects are containers for descriptions of sound. They can also be used to combine other sound objects into one named entity. While this encapsulation notion

is similar to the one presented in this thesis. Kyma differs from our system in that its sound objects represent a computational method for the creation of sound. The sound objects described in this thesis represent a capturing of actual performance information.

7.1.3. Chapter 3

Chapter 3 discussed the base portion of this thesis: a solution to the automation problem. The automation system uses currently available, MIDI controllable devices along with a graphical representation of these devices to implement a scaled down version of the virtual studio.

The system supports the use of standard MIDI controllable devices through its on-screen representation of these devices. The system offers an environment which allows the user to design the look of this on-screen representation, and can even design different representations for different applications.

The system supports mixdown automation through the use of sequencers which record control movements and allow the user to create a complicated mix through indefinite overdubs. The sequencer allows the saving of previous mixes during an overdub so that the user can compare a new mix to an older one and even return to that older mix. The advantages to this mixing automation are the repeatability of the mixing process, and the ability to make minor changes to the mix after it has been recorded without having to perform the entire mix again.

7.1.4. Chapter 4

Chapter 4 discussed the sound object portion of this thesis. Sound objects allow the capturing of music and mixdown information into named entities. They also offer a hierarchical method of composition. The key to the compositional aspect of this system is that the sound composition tool allows the composer to integrate the production work into the earliest stages of the composition and performance.

Sound objects are an encapsulation of musical material. Simple sound objects contain only MIDI and control sequencing information. Composite sound objects allow the encapsulation and temporal ordering of other sound objects. Using this ability to place sound objects into larger sound objects, the composer can create increasingly more complex sounds. This hierarchical model of composition allows the composer to create sounds which would normally be very difficult to create otherwise.

Sound objects also allow a very unstructured approach to composition. That is, they support simultaneous top-down and bottom-up sound design, which is a great benefit in an artistic environment.

The simplicity of sound objects comes from the fact that they can be perceived as simple sound samples where the user has the additional ability to change the method which originally created the sound. This ability to edit the sound object after its creation is the real advantage of sound objects. The musical material which was used to create the sound object can be edited as easily as it was created. In addition, the sound object editor allows the creation of sound objects which can vary depending on the length of its component sound objects. The attach operations provided by the sound object editor allow specific orderings to be given to component sound objects which are independent of the lengths of its components.

7.1.5. Chapter 5

Chapter 5 discussed the implementation details of the system. Improvements for future versions of the code were also suggested.

7.1.6. Chapter 6

Chapter 6 looked at short term and long term extensions to the system. The short term extensions included simple editor enhancements as well as new types of editors. The long term view of our system brings us to the virtual studio concept.

The virtual studio system allows the user to graphically create any type of device he wishes. The system then uses the available resources to simulate the user created device. With the introduction of a MIDI controlled audio router into our system, this virtual studio concept is technically possible. We can also introduce DSP devices into our system to allow the simulation of any piece of hardware. We can even replace everything in the studio with only DSP devices, thereby realizing the desk top studio.

7.2. SUMMARY

This thesis has discussed a recording studio automation system designed to fill the mixdown automation needs of the recording engineer as well as provide a method for composers to approach the creation of music. The automation system itself uses extensions to current automation concepts to achieve a hardware independent, generic automation system that has been demonstrated to be the first step towards the virtual studio. The key to

the compositional aspect of the system is that it incorporates a sound compositional tool which allows the composer to integrate the production work into the earliest stages of the composition creation.

In summary, the system described in this thesis provides a centralized, integrated automation system for composition and production with a consistent user interface to all of the studio hardware.

APPENDIX 1 - CLASS DESCRIPTIONS

This appendix contains a short description of each of the classes created for the system. The classes are ordered in the same order in which they appear in the class hierarchy.

Thesis - This class is used as a container for all the new classes created for the system. It also contains the global variables used by the system (in its class variables). Its class methods are used to initialize the system (and the global variables). This class also stores the system's libraries in its class variables.

DeviceEditingClasses - This class is used as a container for all the classes which have to do with devices.

Controls - This class is used as a container for the control classes. Controls are the switches/knobs/buttons which are used to build a device. This class also contains classes which are used to draw extra things on a device such as text/track number/lines/boxes. This class contains three abstract subclasses:

DisplayControl - the visible version of the control used for editing and displaying the control during creation of a device

SequencingControl - the version of the control used when a sequencing window is opened.

NonControlDisplay - this class is used for display only objects on a device's window.

DisplayControl - This class defines the abstract behavior of the controls which are placed on channels. The actual object which is placed on a channel will be one of this class's subclasses. The subclasses are: ChannelName, DoubleRotary, HorizontalSlider, PushButton, Rotary, and Slider, which implement each of the control types. This class contains only the editing and display information for the control, the actual use and animation of the control is implemented in the SequencingControl subclasses.

NonControlDisplay - This class defines the abstract behavior of the display objects. Display objects cannot be operated like the controls, but they can be manipulated by the track editor.

The actual object which is placed on the channel will be one of this class's subclasses. The subclasses are: TextDisplay and TrackNumberText.

SequencingControl - This class defines the abstract behavior of the controls which are used on a device. This is a distinct class from the DisplayControl class because the types of things that you want to do when you sequence a control are completely different from the things you want to do when you are manipulating a control during the creation of a device. The actual object which is on the device will be one of this class's subclasses. The subclasses are: ChannelNameSeq, DoubleRotarySeq, HorizontalSliderSeq, PushButtonSeq, RotarySeq, and SliderSeq. These subclasses perform the control and animation of their respective controls in the DisplayControl class.

SequencingControlID - This class is used by the sequencer to refer to particular controls on a device. In particular, it is used to distinguish the components of a multiple component control (such as the double rotary control).

DeviceController - This class manages a device controller window.

DeviceEditor - This class manages a device editing window, an editor for a device template.

DeviceTemplate - This class is used to represent the device which channels are placed onto during editing.

ChannelEditor - This class manages a channel editing window, an editor for a channel template.

ChannelOutline - This class is used by the channel editor when laying channels down onto a device. This class represents the channel outlines which are placed onto the device.

ChannelSeq - This class is used to represent a channel of controls on a device controller window.

ChannelTemplate - This class is used to represent a channel with controls on it.

EventClasses - This class is a container for the event classes.

Event - This class represents the events which can be entered in the cue list editor (name/time pairs).

EventEditor - This class manages the cue list editor.

MidiClasses - This class is a container for the MIDI related classes.

MidiPort - This class provides an interface to the C based MIDI primitives. This class provides basic byte array read and write operations to the MIDI port.

MidiMessage - This class is used to represent a MIDI message such as note on, aftertouch, or system exclusive messages. This representation is intended as a higher level representation than the simple byte arrays.

MidiStream - This class is used to read/write MidiMessages from/to the MIDI port. This class's main job is to convert the stream of bytes coming in over the MIDI port into a sensible stream of messages. It takes care of running status so that the calling application can just receive MIDI messages. It also supports running status for the output of MIDI messages as well.

The available operations to the user are

read - which will return the next pending MIDI message (or nil if none)

write: aMidiMessage - which will write the MIDI message to the MIDI port

ScreenClasses - This class is used as a container for the screen classes.

DeviceScreen - This class manages the device screen.

ScreenSwitch - This class implements the virtual screen facility. This class is responsible for the system's screen pull-down menu which allows the user to switch between these virtual screens, as well as create and remove screens. The classes: DeviceScreen, SequencingScreen, SoundObjectScreen, and UtilityScreen implement the screen specific operations.

SequencingScreen - This class manages the sequencing screen.

SoundObjectScreen - This class manages the sound object screen.

UtilityScreen - This class manages the utility screen.

SequencerClasses - This class is a container for the sequencer related classes.

SequenceController - This class is the interface between the realtime events and the currently active sequencer. When a sequencer is started, it informs this class, and when any events happen in the system (such as control movements, or clock events generated from the polling loop in TerminalStream read) this class will inform the sequencer.

SequenceEditor - This class manages the MIDI sequence track editor.

SequencerWindow - This class manages the sequencer window. It also handles the sequencing operation by interpreting event messages received from the sequence controller, and by handling the output of the MIDI information stored in the sequencer tracks.

SoundObjectClasses - This class is a container for the sound object related classes.

SoundObject - This class is the abstract sound object class. This class defines the common behavior for all types of sound objects. Examples of operations which are common to all types of sound objects are 'Return your name', 'Return your length', and 'Play yourself'. Since this is an abstract class, sound objects must actually be an instance of one of the subclasses. This class has two subclasses: CompositeSoundObject, and SimpleSoundObject.

CompositeSoundObject - This class implements composite sound objects. A composite sound object is a sound object which contains other sound objects. This class allows the placement of other sound objects into a composite sound object at specified points in time (either song time or SMPTE time).

SimpleSoundObject - This class implements simple sound objects. A simple sound object is a selection of some of the information (a subset of the tracks and a subset of the affected channels) from a sequencer.

SoundObjectEditor - This class implements the composite sound object editor.

SoundObjectPlayer - This class implements the sound object player. A sound object player is used to play a sound object (either simple or composite).

UtilityClasses - This class is used as a container for the various utility classes.

AbstractTime - This class is the abstract class for the times used within the system. This class provides the common behavior for all types of time. Currently there are only two time type supported: song time and SMPTE time.

SmppteTime - This class provides the SMPTE time type. SMPTE time is used for absolute timing references within a sound object. Its format is hours:minutes:seconds:frames, where there are 24 frames per second.

SongTime - This class provides the song time type. Song time is used for tempo dependent timing references within sequencer tracks and sound objects. The format of a song time is bar:beat.sub-beat, where there are 4 beats per bar, and 24 sub-beats per beat. The actual absolute time of a song time is dependent on the tempo set in the master system window.

Clock - This class implements a clock facility to be used during sequencing. It manages a time count which can be returned in any time type.

Queue - This class implements a queue data type. It is used by the SequencerWindow class to keep a list of the controls which need to be updated on the display.

Startup - This class performs the system start-up initialization.

SystemWindow - This class manages the master system window.

APPENDIX 2 - DEFINITIONS

This appendix contains definitions of various terms used throughout this thesis.

aliasing/imaging - occur during the conversion processes between continuous and discrete signals. Are the appearance of unwanted signals due to improper analog filtering.

attenuation - the decreasing of the level of a sound.

compression - the reduction of the dynamic range of a sound.

decibel (dB) - ten times the logarithmic ratio of two power quantities.

DSP (digital signal processor) - RISC architecture processors specially designed to solve the problems typically found in high quality digital audio processing.

dynamic range - the range from the loudest to the softest sound.

EQ (equalization) - the increasing or decreasing of certain frequencies in a sound source. The standard treble and bass controls found on home stereos are simple forms of equalization.

expansion - the reverse process of compression. Expansion increases the dynamic range of a sound.

fader - the volume level controls on a mixing console (either the channel controls or the master console controls).

JMSC - Japan MIDI Standards Committee.

MIDI - Musical Instrument Digital Interface.

mixdown - The process of mixing many channels of sound (from a multitrack tape recorder or a multichannel MIDI sequencer) to the master (usually stereo) recording.

MMA - MIDI Manufacturers Association.

mono (monophonic) - one channel sound.

multitrack tape - magnetic recording tape with many (4, 8, 16, 24) tracks running in parallel. Each track can be recorded on independently of the others, and any group of tracks can be played back at the same time as recording on another group of tracks. Multi-track tape is similar to standard cassette tape, except it is usually wider in order to accommodate a greater number of tracks.

overdub - to add additional tracks to a multitrack performance while monitoring the already recorded tracks.

panning - the stereo placement of a sound

reverb (reverberation) - the ambient characteristics (the echoes) of a sound within an acoustical environment (a room).

sampler - a synthesizer like device which allows the digital recording and playback of sounds. Most samplers offer some sort of manipulation of the sound during the playback process.

sequencer - Records and plays back real time MIDI performances. Basically a multi-track recorder for MIDI information.

stereo (stereophonic) - two channel sound. Usually designed to simulate a one dimensional (left to right) sound stage.

synthesizer - an electronic device which can create sound.

REFERENCES

- [AES] "The Art and Technology of Recording", 2nd AES International Conference Report, J. Audio Eng. Soc., Vol. 32, No. 9, 1984 September, pp 698-703.
- [AIKIN] Jim Aikin, "MIDI Sequencing for Musicians", May/June 1988, GPI Publications, Cupertino, CA.
- [ANDER] Craig Anderton, "MIDI for Musicians", 1986, Amsco Publications, New York, NY.
- [BARTa] Bruce Bartlett, "Recording Techniques: Equalization", db, The Sound Engineering Magazine, March/April 1987, Sagamore Publishing Company Inc., Plainview, NY, pp 10-15.
- [BARTb] Bruce Bartlett, "Recording Techniques: Tape Recording, Part 1", db, The Sound Engineering Magazine, May/June 1987, Sagamore Publishing Company Inc., Plainview, NY, pp 8-14.
- [BARTc] Bruce Bartlett, "Recording Techniques: Tape Recording, Part 2", db, The Sound Engineering Magazine, July/August 1987, Sagamore Publishing Company Inc., Plainview, NY, pp 30-36.
- [BARTd] Bruce Bartlett, "Recording Techniques: Doing a Mixdown", db, The Sound Engineering Magazine, September/October 1987, Sagamore Publishing Company Inc., Plainview, NY, pp 12-15.
- [BARTe] Bruce Bartlett, "Recording Techniques: A Short Course in Microphone Training", db, The Sound Engineering Magazine, November/December 1987, Sagamore Publishing Company Inc., Plainview, NY, pp 14-17.
- [BOOM] Michael Boom, "Music through MIDI", 1987, Microsoft Press, Redmond, WA.
- [DAVIS] Gary Davis and Ralph Jones, "Sound Reinforcement Handbook", June 1989, Hal Leonard Publishing Corporation, Milwaukee, WI.
- [EARG] John M. Eargle, "An Overview of Stereo Recording Techniques for Popular Music", J. Audio Eng. Soc., Vol. 34, No. 6, June 1986, pp 490-503.
- [IMA] "MIDI 1.0 Detailed Specification, Document Version 4.1", January, 1989, International MIDI Association (IMA), Los Angeles, CA.
- [KEY86] Ted Greenwald, "Keyboards in the Professional Studio", Keyboard Magazine, June 1986, Miller Freeman Publications, Cupertino, CA, pp 64-86, 167-168.
- [KEY89Oct] Dominic Milano, "Interview with Peter Gabriel", Keyboard Magazine, October 1989, Miller Freeman Publications, Cupertino, CA, pp 32-48.
- [KEY89Sep] Brent Hurtig, "Making Tracks", Keyboard Magazine, September 1989, Miller Freeman Publications, Cupertino, CA, pp 94-96.
- [MILL] Fred Miller, "Studio Recording for Musicians", 1981, Amsco Publications, New York, NY.

-
- [MOOG] Bob Moog, "MIDI: Musical Instrument Digital Interface", J. Audio Eng. Soc., Vol. 34, No. 5, May 1986, pp 394-404.
- [MOORE] F. Richard Moore, "The Dysfunctions of MIDI", Computer Music Journal, Vol. 12, No. 1, Spring 1988, pp 19-28.
- [NAZAR] Bruce Nazarian, "Recording Production Techniques for Musicians", 1988, Amsco Publications, New York, NY.
- [SCAL] C. A. Scaletti and R. E. Johnson, "An Interactive Environment for Object-oriented Music Composition and Sound Synthesis", OOPSLA '88 Proceedings, September 25-30, 1988, pp 222-233.

END

03·03·92

FIN