A NETWORKING APPROACH TO SHARING MUSIC STUDIO RESOURCES

THESIS

Submitted in fulfilment of the
requirements for the Degree of
DOCTOR OF PHILOSOPHY
of Rhodes University

by

RICHARD JOHN FOSS

January 1996

Abstract

This thesis investigates the extent to which networking technology can be used to provide remote workstation access to a pool of shared music studio resources.

A pilot system is described in which MIDI messages, studio control data, and audio signals flow between the workstations and a studio server. A booking and timing facility avoids contention and allows for accurate reports of studio usage. The operation of the system has been evaluated in terms of its ability to satisfy three fundamental goals, namely the remote, shared and centralized access to studio resources.

Three essential network configurations have been identified, incorporating a mix of star and bus topologies, and their relative potential for satisfying the fundamental goals has been highlighted.

Keywords: Networking, music studio, MIDI.

Acknowledgements

I would like to thank my supervisor, Peter Clayton for his willingness to involve himself in this project, his sound guidance regarding the direction of the thesis, and his astute comments on the content. His encouragement and positive attitude have been a great source of inspiration.

Bruce Pennycook freely shared his time and wealth of music and audio engineering experience, providing an invaluable perspective on the thesis. A leader in the field of computers and music, he provided an essential confirmation of direction, and I am most grateful to him for doing that.

The implementation of the remote studio access was a joint effort, and I would like to thank all the students who worked on projects related to remote studio access under my supervision. Teaching is by no means a one way process, and I learnt an immense amount from them throughout the implementation. In particular, I would like to thank Arne Bier, Peter Koeslag, Thabo Mosala, Mahlomola Ntene, and Antony Wilks.

The Computer Science Department at Rhodes University has consistently supported this project, providing space for a laboratory, resources to equip it, and time to work on it. I am most grateful for this support.

The Joint Research Committee at Rhodes University has funded the project generously. They provided funds for the music resources within the remote, shared studio, without which this thesis could never have materialized. I am grateful for their trust and willingness to support a new field of research.

Although I am acknowledging my family at the bottom of the list, this should not be regarded as reflecting prioritization, but rather the baseline support which they have provided throughout my work on this thesis. I am continuously grateful for the life and love provided by Lisl, Kanina, Jeremy and Torey.

Contents

List of Figures

Chapter 1

Introduction

Recording and electro-acoustic music production studios have been established throughout the world to create the music which is so much a part of our everyday lives. Some music studios are used to record groups of musicians ranging from classical orchestras to popular bands, while others are used to record music for advertisements, films and videos. As in many other spheres of life, computers have found their way into music studios, to the point where they often play an indispensable role in the process of music recording. Indeed many music studios have become centres of advanced music technology. Much of this music technology significantly speeds up and enhances the recording process, but the technology is often expensive, must often be shared by numerous users, and requires a steep learning curve for effective usage. This thesis investigates the sharing of music studio resources. The contention is that computers and network technology can be used to effect this sharing in a controlled, cost-effective manner, which shields users from technical complexities, and allows for remote access from a single interface.

Before we look at the role that computers and networking can have in the sharing and remote access of music studio resources, we need to look at the nature of a typical music studio and define more clearly the resources which are to be shared.

1.1 Music Studios and their Resources

The overall goal of a music studio is to produce music in a recorded form so that it can be disseminated to a wide audience of music 'consumers'. There are two major types of music studios in the world of commercial music production, each type producing music for a different market. Music post-production studios produce music for film and video. Typically, the music will be recorded on an audio track alongside the film or video track [Huber 1987]. Music recording studios, the second type, are geared towards recording individual or groups of musicians such as popular bands, classical orchestras, choirs and jazz ensembles.

In both types of studios, there are sound generators which create the sound to be recorded. The range of possible sound generators is vast, and includes the human voice, acoustic instruments such as violins, pianos, and marimbas, electronic instruments such as electric guitars, as well as analogue and digital synthesizers and samplers. Analogue synthesizers utilize electronic circuitry such as voltage controlled oscillators, voltage controlled amplifiers, and voltage controlled filters to allow for the synthesis and shaping of interesting sound

timbres [De Furia 1986]. Digital synthesizers use computer-based technology and a variety of algorithms to create and modify complex waveforms in the digital domain. Samplers use analogue-to-digital conversion circuitry to convert an analogue representation of sound into its digital equivalent. They have the capability to store the digital representation of the sound and to retransmit it in analogue form with the help of digital-to-analogue converters. Many samplers now pick up their samples directly in digital form via a storage medium such as CD-ROM.

The sounds from these generators may be processed by sound processors, which can add a number of effects to the sound, such as reverberation, chorusing, flanging, and echo. They can also process the sound by compressing or limiting the dynamic range, filtering it or shifting the pitch. For more information on the range of effects and processing options, refer to [White 1989]. In many cases, these processors attempt to give the impression that the sound is emanating from a particular acoustic space, such as a large reverberant hall, indeed many multi-purpose sound processors provide settings which the user can select to emulate particular acoustic spaces [Yamaha 1988].

Once processed, the sounds are usually mixed together by an audio mixing console, which combines the outputs of microphones and instruments and routes these to various destinations. One mix of outputs may be sent to a multitrack tape recorder for recording, another to a monitor output for the sound engineer to listen to. Monitoring may be done using headphones, or via an amplifier and speakers. Each audio output is plugged into an audio channel on the mixer. There are usually buttons and sliders on the mixer which allow for control of equalization and volume levels of each channel. A mixer will usually allow the incoming audio signal on a channel to be routed through one or more signal processors and then to return to the mixer. A short introduction to mixing is provided by [Huber 1991].

A mixer can allow a particular mix of sounds to be routed to a multitrack tape recorder, which, as its name implies, is a device which allows for the recording of sounds on distinct tracks. Multiple sound generators can be recorded simultaneously onto separate tracks of a multitrack tape recorder, or they can be recorded in succession. The technology associated with multitrack recording has changed rapidly over the past few years. The traditional analogue multitrack tape recorder allows for the recording of multiple tracks onto magnetic tape [Woram 1989]. Tracks are recorded onto physically separated areas of the tape and tape widths range from a one or two inches all the way down to the width of domestic cassette tapes. More recently, digital multitrack recorders have appeared on the market which allow for the recording of audio in digital form [Hurtig 1992], where the recording medium can either be a hard disk or a magnetic tape. Hard disk based recording systems allow for the editing of the digital audio, whereas tape-based digital systems are used in a manner similar to the traditional analogue systems.

In a typical audio production, once multiple tracks have been recorded on a multitrack recorder, they are once again mixed together using the audio mixing console. The total volume of each incoming track can be apportioned between two output channels. This two channel, or stereo output, is then recorded by a mastering device. The mastering device could be a two-track analogue tape, or, as is more common these days, a digital audio tape recorder (DAT). In the case of a post-production studio, the eventual destination of the audio will be one or two tracks alongside the film or video signal. The entire recording process is well described in [Woram 1989].

Of course, there needs to be audio connections between the sound generators, sound processors, sound mixers, and sound recorders. Audio cables can be balanced by having two conductors twisted around one another, or consist of a single, unbalanced conductor [Wilkinson 1994], and will always be shielded to reduce noise transmission and unwanted radio frequency interference. Figure 1.1 gives a diagram of the resources of a typical music studio and the audio connections between these devices.

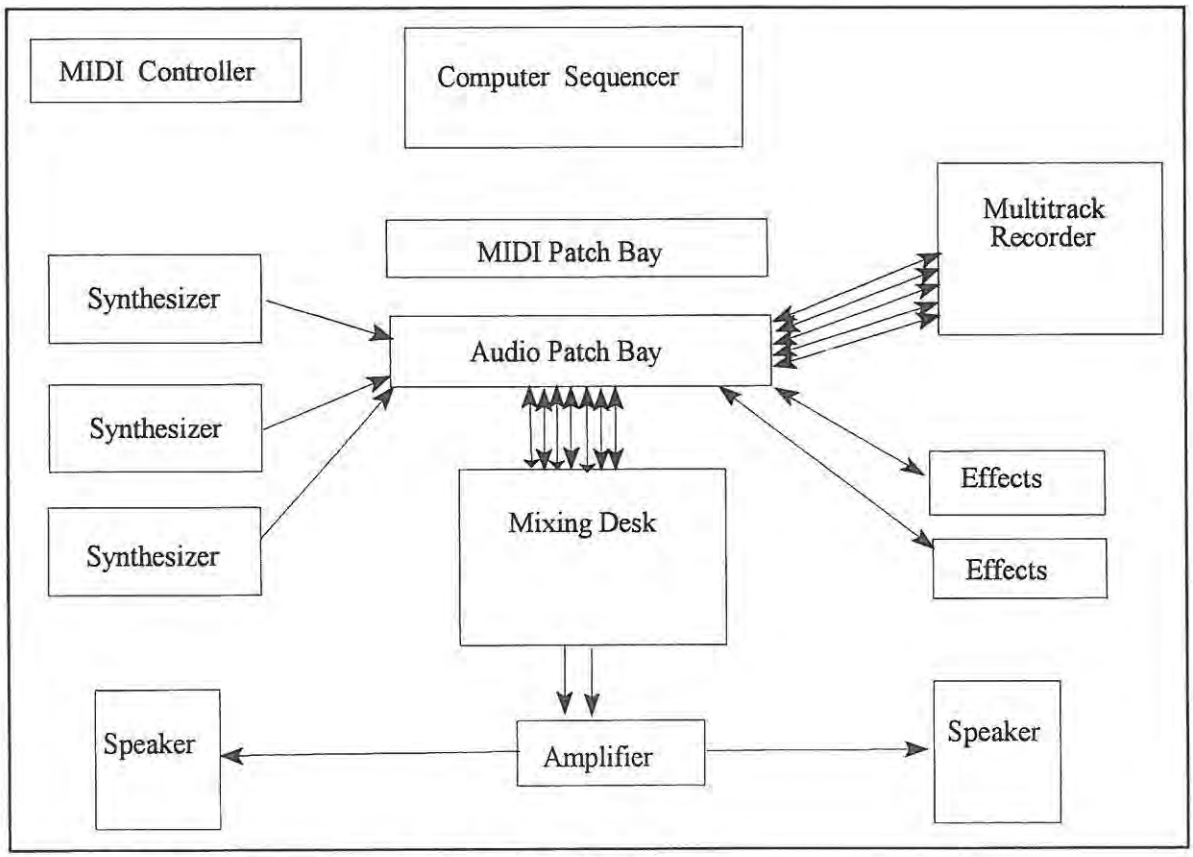

Figure 1.1 - Audio connections in a typical music studio

Figure 1.1 is a simplified representation of the audio resources and their connections in a typical studio. Even with this simplified diagram, it can be seen that there is a complex network of interconnection paths. The complexity grows with the number of audio generators, processors and recorders. Interconnections need to be flexible, particularly in a multi-user studio. This has led to the incorporation of audio patch bays into music studios, which allow the interconnections between audio resources to be modified via a patch panel. The inputs and outputs of most of the studio resources are connected to the patch panel. Input to output connections can then be made at the patch panel, rather than between the actual resources. The operation of manual audio patch bays is described in [Brighton 1992].

1.2 Control over Music Resources

The diagram in Figure 1.1 and the associated review of the range of music studio resources has given an idea of the complexity of a music studio. What should also be apparent is the range of devices which need to be controlled. In a typical large studio recording session, the various musicians will control the sound generators. It will be left to a sound engineer to control the signal processors, mixer, multitrack recorders, and patch bay connections, once microphones have been correctly placed. In a post-production studio, the sound engineer will often be responsible for generating sound effects and possibly adding additional tracks. A good studio design can ensure that these various resources are reasonably accessible from a central point. However, it is both unproductive and tedious for a sound engineer to move between the front panel of a multitrack tape recorder, a mixing desk, and an array of sound processors. In some situations, the nature of the recording might make this impossible.

A number of multitrack recorders, particularly the bigger units which occupy a lot of space, have small, remote control units associated with them. These units can be placed close to the studio engineer and duplicate the basic functions found on the tape transport system front panel , in particular, auto-location to a particular time point [Woram 1989]. The engineer can quickly turn from multitrack recorder to mixer control. Indeed, both could be controlled simultaneously. The hardware and software protocols for communication between remote units and recorders vary from manufacturer to manufacturer. At the hardware level, a number of manufacturers use the Electronic Industries Association RS-422 standard [EIA]. The Audio Engineering Society has also incorporated this standard into their communications interface standard (PA-422) for sound reinforcement systems [AES 1991]. At this point, the standard incorporates codes for control over equalizers, delay units and gain units. Note here that the remote control unit does not address the problem of shared usage. The actual device and remote unit are intended for use by a single studio engineer in a particular studio.

1.2.1 The MIDI standard for Music Resource Control

The most significant factor contributing to music studio control in the last decade has been the introduction of the MIDI (Musical Instrument Digital Interface) standard. The MIDI hardware and software protocol standards are described in the MIDI standards document published by the International MIDI Association [IMA 1989]. Curtis Roads describes MIDI as a "software language and hardware interconnection scheme for communication between computers and computer-controlled devices (such as synthesizers)" [Roads 1989]. MIDI does not have the syntactic and semantic richness of a typical software language such as C, but is rather a set of commands for controlling computer-based, music-related devices. Indeed, it is better referred to as a 'communication protocol'.

MIDI was originally devised to allow commercial synthesizers (sound generators) to be connected together so that they could share control information [Loy 1985]. This control information would typically be provided by a musician playing some sort of MIDI-compatible controller. MIDI information, originating at the controller, could be received by a number of synthesizers. The synthesizers could respond by activating sound generating circuitry to produce pitched notes, drum sounds, or sound effects. MIDI messages are divided into a number of groups. The most used group in the context of standard music production is the group known as 'Channel Voice Messages'. Each of these messages has associated with it a channel number which appears in the low nibble of the first byte of the transmitted message, thereby allowing for 16 channels. The high nibble contains an indication of the message type. The complete first byte is known as the 'status' byte. A brief overview of the MIDI protocol is given in Appendix A.

A receiving synthesizer can be assigned a particular channel. This synthesizer will only respond to MIDI channel voice messages which carry its particular channel number. Thus, in the scenario of multiple synthesizers connected to a single controller, the controller could select particular synthesizers to activate by sending channel voice messages with the appropriate channel numbers. The most commonly used channel voice messages are the 'Note On' and 'Note off' messages. A note on message is a 3-byte message comprising channel information, the pitch of the note and 'velocity' information. When a keyboard is being played, the velocity is an indicator of how fast the key was depressed. A musician could select a transmit channel on a keyboard controller, and by playing the keyboard, send note on and note off messages to the connected synthesizers. Only those synthesizers with their receive channels set to the same number as the controllers transmit channel would respond by playing the notes. Note that only performance information is transmitted by MIDI channel voice messages, not waveform information.

The MIDI connection from controller to synthesizer is a serial one. At the controller side, parallel data is converted into serial form by a standard UART (Universal Asynchronous Receiver/Transmitter). The UART

transmits serial data at a rate of 31.25 kbaud, defined by the MIDI standard. Transmission occurs via a current loop with an opto-isolator at the receiving end. A UART at the receiver picks up the serial data and converts it to parallel form for reception by the computer-controlled synthesizer. Note that transmission is uni-directional, with no acknowledgement of byte reception built into the protocol standard. Synthesizers can be 'daisy-chained', allowing MIDI messages from a controller to be received by multiple synthesizers. 'Thru-boxes' are devices which also allow for this multiple reception of MIDI messages. They achieve it by transmitting messages on a single incoming line over multiple outgoing lines. At a higher level, there is also no acknowledgement of message arrival, except in the case of the 'Sample Dump' standard, an addition to the original (1983) MIDI specification [IMA 1989]. The acknowledgement messages in the Sample Dump standard are sent back over a separate uni-directional MIDI cable.

Once established, the MIDI standard was adopted with surprising enthusiasm by the music industry, although there have been its critics. Many of the criticisms are well-founded, and include the slow speed of MIDI, the limited number of channels, and the limited parameter space (most parameters are allocated 7 bits, and have values ranging from 1-128). However these 'failures' of the standard stem from a requirement that the interface be inexpensive and easy to implement. Meeting this requirement has ensured the widespread use of MIDI. Virtually all music synthesizers are now manufactured with a MIDI interface. Applications of MIDI did not stop at simple controller-synthesizer interaction. It was obvious that MIDI messages generated by a controller could be read by a computer with a MIDI interface. MIDI messages can be stored and manipulated by computer programs commonly known as sequencers. Sequencers allow MIDI data, generated by a controller, to be recorded onto separate 'tracks', the name being derived from the tracks of a multitrack tape recorder. They typically provide the means to edit meaningful representations of the MIDI data. These representations are usually either in piano-roll style, where the notes are represented as horizontal bars, or the more familiar common music notation. Once recorded, the MIDI tracks can be 'played back' by the sequencer. In effect, the sequencer transmits the recorded MIDI performance data to one or more synthesizers and the synthesizers respond to the MIDI messages just as they would in a controller-synthesizer interaction. The sequencer will use either an internal or external timer to determine when the MIDI data was received and when it should be transmitted at playback time. A useful account of the design of a MIDI sequencer is provided by Garvin [Garvin 1987]. Rothstein provides some brief guidelines for MIDI program construction and a clear introduction to the MIDI file format [Rothstein 1992].

A question at this stage might well be whether the multitrack tape recorder has a place in this MIDI world, where recording and editing is so simple, noise-free, and takes up so little memory space. However, recordings of voice and analogue instruments do need to be made, and often need to be played together with sequencer recordings. For this to be possible, the multitrack tape recorder and sequencer need to be synchronized. A track of the multitrack recorder will usually contain synchronization information. The

SMPTE (Society for Motion Picture and Television Engineers) time code standard is the most commonly used format for the synchronization information. A modulated audio signal is used to lay down 80-bit frames. Each frame comprises absolute time information in the form of the number of hours, minutes, seconds, and frames [Woram 1989]. In the simplest case, the multitrack recorder will be the master device, and the sequencer, acting as slave, will listen to the SMPTE track, and transmit its recorded MIDI bytes at the correct times. A simple introduction to synchronization issues is given by Rona [Rona 1990]. The master device could also be a video recorder, where the SMPTE time code is commonly striped onto, and read off one of the audio tracks. This is particularly relevant for post-production studios, and is known as Longitudinal Time Code (LTC). It should be distinguished from the Vertical Interval Time Code format (VITC) which is stored within the video track at discrete points. Devices also exist to synchronize multitrack tape recorders and video recorders. One such device, the Tascam MIDiiZER, allows for the synchronization of sequencers, video recorders, and multitrack tapes [Many 1990].

The MIDI standard has been extended to allow for the transmission of absolute timing information over MIDI lines. This extension, known as MIDI Time Code, provides the same information as the SMPTE standard, though in a different format. Eight MIDI Time Code quarter frame messages are used to compile all the timing information for one frame [IMA 1989].

The role of the computer in a MIDI-based studio did not stop at the recording and editing of MIDI note data. Computers can also generate note data, and users can be in charge of this data generation process. An immediately obvious application of this concept lies in the computer-based generation of compositions [Chadabe 1986]. However, what is more relevant to this thesis is that manufacturers started building 'MIDI-awareness' not only into sound generators, but also other music studio resources. MIDI data which is either recorded or generated by a computer can be used to control all the various categories of music resource - sound generators, sound processors, sound mixers, patch bays, and even standard sound recorders. In the next few short sections, we will have a look at the application of MIDI to these five categories. These brief sections will not cover the full range of MIDI messages, nor all the applications of MIDI messages. For a more complete treatment, refer to the article by Yavelow [Yavelow 1989], and the more formal MIDI specification [IMA 1989].

1.2.1.1 Sound Generators

Sound generators can receive and respond to note on and note off messages. When an analogue synthesizer receives a note on message, it activates its sound generating circuitry to generate a note of the required pitch. A digital synthesizer will start up a sound generating algorithm - often hardwired into an LSI or VLSI chip. Synthesizers are capable of generating sounds of many timbres. The nature of the sound is determined by

parameters which control the sound generating circuitry, in the case of analogue synthesizers, and the path through the algorithm, in the case of digital synthesizers. Parameter settings are grouped into 'patches', a term derived from the patch panels of old analogue synthesizers. There are MIDI Program Change messages, part of the channel voice messages, which can cause synthesizers to switch to a new set of patch parameters, and subsequently play notes of a different timbre. A program change message can select one of 128 patches. It is a two byte message where the second byte determines the patch to be selected. At a higher level, a Bank Change message, one of the MIDI controller messages, can be used to switch between banks of instruments, each bank comprising 128 instrument patches.

The parameters for new sounds can be loaded into synthesizers by sending them MIDI 'system exclusive' messages. The make-up of a system exclusive message is largely defined by the manufacturer. Each message starts with the byte F0 (hex) and ends with the byte F7 (hex). The second byte is a manufacturer's ID number. The manufacturer can decide on the format of subsequent bytes. The subsequent bytes might contain information to load up a synthesizer with its full complement of patches (bulk dump), or provide enough information for just one patch. Synthesizers can also transmit these system exclusive messages containing patch information. Librarian and patch editing programs have been developed which allow for the storage, retrieval and modification of patches by computer [Yavelow 1989]. A sample dump standard has been developed to allow for the loading of sample data into a sampler.

Most synthesizers these days comprise a number of distinct parts. These parts can be viewed as independent child synthesizers within the parent synthesizer. Each part can be allocated its own receiving channel number and, in some synthesizers such as Roland's D110 [Roland 1988], can have their own independent outputs. MIDI messages can be sent to these parts just as they would to any synthesizer.

Controller messages are another important subset of the channel voice messages. Each controller message indicates by means of an identifying number what it is controlling, and a control value. A large variety of effects related to sound production can be controlled using controller numbers. Some controller numbers have been assigned for specific purposes such as volume and modulation control, while others can be assigned for purposes required by the manufacturer. For an overview of the MIDI protocol, refer to Appendix A.

1.2.1.2 Sound Processors

There are single- and multi-purpose sound processors. Single-purpose sound processors apply a single processing function to sound. This processing function might enhance or limit the sound in some way. For example, a reverberation unit will add reverberation to a sound, while a compressor will limit the dynamic threshold of the sound. Reverberation is one of the most complex sound processing functions, having to

emulate the reflection of sound in a physical space. It is performed by a digital processor which samples, filters and delays sound at a number of points in time. Many sound processors provide reverberation settings which emulate a particular physical space, such as a large hall [Yahama 1989]. Multi-purpose sound processors provide a range of processing functions in one unit. These might include compression, equalization (tone control), enhancement (adding harmonics), digital delay, reverberation, gating (used to remove unwanted noise), and many others. Once again, the sound processor will usually provide front panel switches for the application of a particular set of sound processing functions. A good high level overview of sound effects and processors can be found in [White 1989].

Most sound processors will allow their front panel settings to be selected using MIDI program change messages, where the program change number determines the sound space setting. They will also allow the parameters of the various sound processing functions to be changed using MIDI controller messages, however some, such as Roland's DEP-5 [Roland 1988] , require system exclusive messages to perform these changes. This capability of MIDI control means that sound processors can be modified via MIDI sequencers. A number of graphics-oriented MIDI sequencers allow controller values to be entered graphically. This means that during playback, sound from the various sound generators can be modified in real-time with visual feedback.

It must be noted here that the processor control messages and sequencer note messages will often leave via the same MIDI port, although multi-port MIDI interfaces do exist. Generating a large number of MIDI messages for the real-time control of sound processors will quickly use up the transmission capacity of a MIDI line, resulting in MIDI transmission delays. Moore has commented on the significance of even millisecond delays in the transmission of MIDI data [Moore 1988]. Each 3-byte MIDI message requires 1 millisecond for its complete transmission. Such a MIDI message will only change the value of one sound processing parameter. A further weakness of MIDI is its channel capacity. There are only 16 MIDI channels available. These channels must be shared amongst the full spectrum of sound resources in a studio, unless multiport MIDI interfaces are used, where each port has a 16 channel capability. If effects units are to be assigned channels for MIDI control, they will deplete the channels resource associated with a MIDI interface port.. These and related will be dealt with in more detail in chapter 2.

1.2.1.3 Sound Mixers

As in the case of Multi-track tape recorders, manufacturers of high-end mixing consoles have for some time provided automated control using their own proprietary protocols. However, with the advent of MIDI, manufacturers have begun to incorporate MIDI control over the various features of their mixing consoles. Probably the most well-known example was Yamaha Corporation's DMP series of mixing consoles [Aiken

1987], where MIDI control was provided over every parameter of the mixer, and, in some cases, there was even motorized control over the front panel sliders. Yamaha has recently enhanced the MIDI control capability of their mixers with the introduction of the ProMix O1 and OR2 MIDI controllable mixers. Some companies have provided retrofits to existing consoles, thereby upgrading old consoles to have automation via MIDI. A number of manufacturers including J.L. Cooper systems, Iota, and Akai have provided low-cost fader boxes with MIDI control over a series of voltage controlled amplifiers. A range of MIDI messages are employed to control the gain, equalization, pan, and fader controls on mixers. However, MIDI program change messages are usually used to switch between complete, pre-stored, mix setups, while controller messages are used for real time control over individual settings. Once again, the bandwidth limitations of MIDI and the possible contention with other MIDI devices in a studio must be borne in mind when assessing the desirability of complete MIDI control. A comprehensive layout of the controller messages required for the control of the 7-channel Mark of the Unicorn mixer is given by Meyer [Meyer 1991].

1.2.1.4 Sound Recorders

Three system exclusive manufacturer ID numbers have been set aside for specific purposes. The ID number 7F (hex) is used for real-time message extensions to the MIDI specification. One of these extensions is the MIDI Machine Control (MMC) standard [IMA 1992]. This standard defines a number of commands which can be sent to audio tape recorders, video tape recorders, and hard disk recorders. These commands can be simple transport commands such as 'play', 'fast-forward', and 'rewind', record control commands which can set particular tracks to be recorded on, locate commands which can locate a machine to a particular SMPTE time frame, and even chase commands which can allow slave machines to move synchronously with a master machine. MIDI Machine Control commands can be issued by a sequencer. From a user's point of view, the tracks of the controlled machine can be aligned with the normal sequencer MIDI tracks. When the user moves to a location in a MIDI track, MIDI Machine Control commands will ensure that the connected machines are also located to the correct point. 'Punching in' sections of music onto tape can all be done under sequencer control. The user need no longer switch from tape control panel to sequencer.

Some manufacturers implemented MIDI Machine Control features on their machines soon after the ratification of the standard [Westfall 1992]. A number of recording devices without an MMC implementation are in use. Many of these allow remote control via a parallel port, or serial port - often RS232 or RS422. A simple controller can be built as an MMC-compatible front-end to such a device, thereby making it MMC-controllable. This has been done for the Tascam 232 8-track multitrack tape deck, although it would work for any tape deck which adhered to Tascam's serial protocol [Ntene 1993].

1.2.1.5 Patch Bays

- **Audio Patch Bays**

Audio patch bays allow a studio engineer to modify the audio connections in a studio via a single centralized patch panel. This still requires the engineer to move away from the current locus of control to the patch panel. The engineer must also remember and restore connections made in a previous recording session. To avoid this, MIDI control has been extended to audio patch bays. MIDI controlled audio patch bays usually have sixteen audio inputs and sixteen audio outputs. Typically, a user will set up patch configurations, and can then use MIDI program change commands to switch between the configurations. Examples of audio patch bays are the 360 System's AM-16 series cross point switchers, and the Intone MIDI Maestro audio and MIDI patch bay [Oppenheimer 1991].

- **MIDI Patch Bays**

MIDI connections also need to be modified to accommodate various recording and playback requirements. For example, there might be a single MIDI input to a studio computer. At times, this MIDI input will be required to carry system exclusive patch messages from a synthesizer. At other times, it might be required to carry MIDI time code messages from a multitrack tape recorder. It is also often necessary to direct MIDI messages out to only select devices. In the worst case, MIDI cables have to be plugged and unplugged at the devices themselves. MIDI patch bays, MIDI thru boxes and MIDI merge units can make this process easier by centralizing connection points. However, just as for audio patch bays, MIDI patch bays exist which allow for MIDI control over the MIDI connections. These patch bays are typically 8 input, 8 output units which allow for MIDI configurations to be made from the front panel. Again, as for audio patch bays, MIDI program change messages can be used to switch between these configurations. An example of a MIDI patch bay is the Intone MIDI Maestro audio and MIDI patch bay [Oppenheimer 1991].. Many MIDI patch bays incorporate MIDI processing as well. The processing features usually include the merging of MIDI input streams, the filtering of selected MIDI messages and rechannelling MIDI messages. An example of such a patch bay is Music Quest's MIDIEngine, an 8 input, 8 output patch bay which can be controlled via MIDI, or via a parallel port [Music Quest 1995]. A good overview of MIDI patch bays and their processing features is given in [Yelton 1993].

A diagram of the flow of MIDI control information to the various music resources of a music studio is given in Figure 1.2 below. This diagram is idealized in the sense that not all studios utilize the control potential of MIDI. Indeed, in many cases, MIDI is only used for sound generator control. Studio mixers are often not

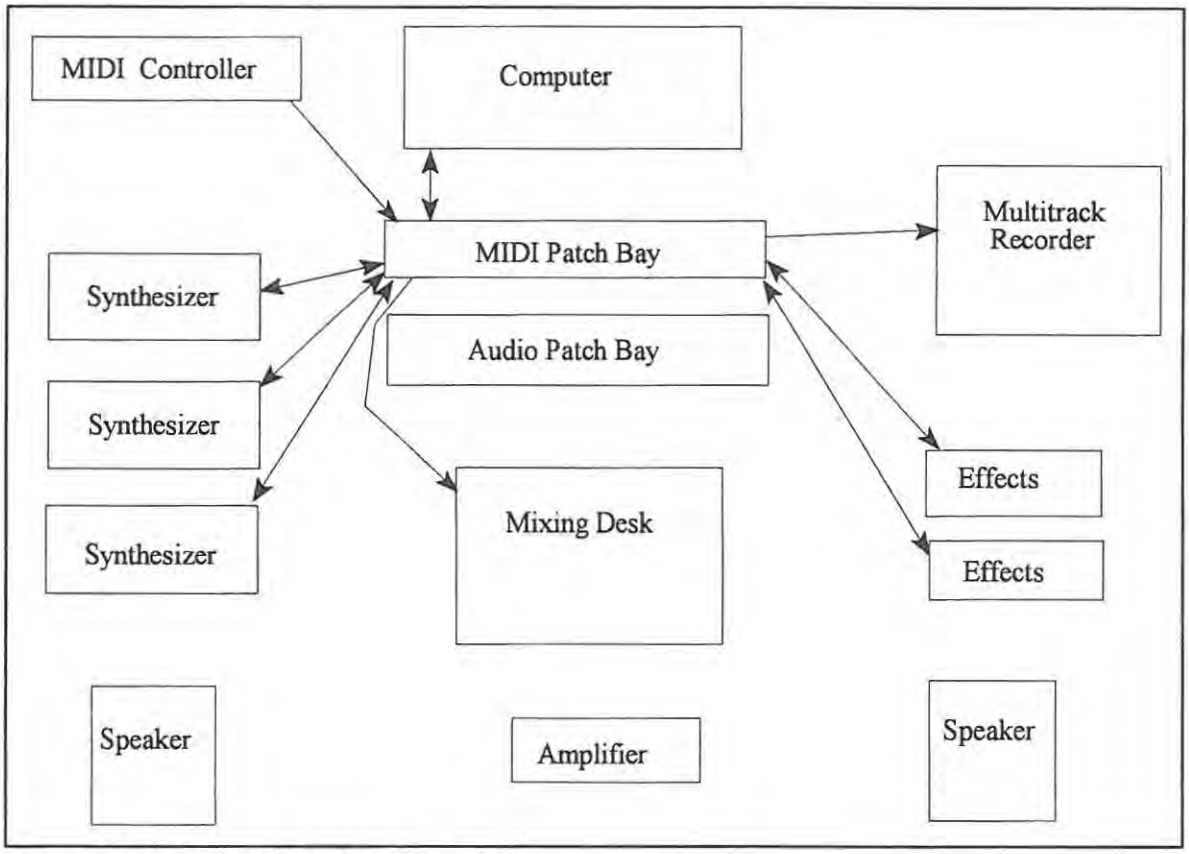

Figure 1.2 The Flow of MIDI Information

MIDI controllable, and automated mixing is done via MIDI volume controller messages, which control volume levels on the sound generators [Meyer 1989].

1.2.2 Centralization of Control

It is clear from Figure 1.2 that control over a music studio can now be centralized to one point. A studio engineer can control all the resources of a music studio from a single computer with an appropriate MIDI controller. This assumes, of course, that all the music resources are MIDI-controllable, and that suitable software exists for the control computer to perform the various control functions. The first sequencers to appear on the market were largely oriented towards note recording and note manipulation, the stress being on sound generator control. Current sequencers usually have graphic interfaces which allow fine graphic control over, for example, controller message values. These controller messages can be used for the control of sound processor and mixer parameters. Most sequencers now incorporate a software mixing console which will at least generates volume and pan control messages. Often, MIDI-controllable mixers are supplied with software which turns the computer screen into a virtual mixing console [Meyer 1991]. Many sequencers incorporate the MIDI Machine Control specification to provide control over sound recorders.

Music resources which are not directly MIDI-controllable may have their own control protocols and still be candidates for computer control. However, software for such devices is not readily available. The Tascam 238 multitrack tape deck is an example of one such computer-controllable music resource. A computer interface to the Tascam 238 is described by Wilks [Wilks 1994].

Most music programs are now being written to operate within multi-program, windowed computer environments. A studio engineer can move from window to window of different programs to control the various facets of a studio. Usually, these programs can all be synchronized via a common external or internal SMPTE timer. A good example of such a suite of programs is 'MaxPak' which runs under the IBM PC Windows environment [Kendall 1994]. This comprises a sequencer, a MIDI mixer automation module, a patch librarian, and a tape controller. The modules are tied together by the 'MIDI director' which allows multiple programs to run simultaneously, in synchronization. In general, major studios use a single calibrated 'house sync' to generate stable time code. This synchronization signal will then be fed to slave devices.

1.3 Digital Audio Processing in a MIDI environment

Although MIDI now provides comprehensive control over music resources, this is not to say that all studios do or even should utilize MIDI exclusively. There are strong arguments for not using MIDI [Loy 1985]. The major problems with MIDI lie in the slowness and time variance associated with its response to instrument control, and the paucity of real-time control over timbre modification [Roads 1989]. Harris does not even consider MIDI in his ideal computer music system, but prefers the 'rich sound realm' made available by software synthesis techniques on workstations [Harris 1987]. The advent of real-time software synthesis techniques, in large part made possible by more powerful workstations, has made this compositional environment even more attractive [Vercoe 1990]. On the other hand, the software-specified instruments of compositional languages such as csound can now be controlled via MIDI, because the synthesis can occur in real-time [Vercoe 1990]. This approach provides the best of both worlds, the ability to have complete flexibility in instrument creation, and MIDI controllable triggering of sonic events. Ballista *et al* have recognized the desirability of this approach in their proposed sound processing environment [Ballista 1992]. A number of commercial digital audio systems have appeared on the market which allow for the simultaneous recording, editing and playback of MIDI and audio tracks. Miller and Lehrman have pointed out that the key to integrating MIDI and digital audio has been the development of sound recording and sound generating hardware placed inside the computer that is handling the MIDI data [Miller 1991]. Systems which provide this ability to manage MIDI and digital audio flexibly are ideal for post-production environments and for experimental musicians who find the restricted parameters of MIDI confining. The controlling computer now contains the necessary hardware and software for the recording, editing and playback of digital audio, and also provides MIDI control over music studio resources.

A number of digital audio systems have hardware external to the computer, and are connected to the computer via interface cards connected onto the computer bus. Communication between the computer and external hardware can be via proprietary software and hardware protocols, or more standardized protocols such as a SCSI (Small Computer System Interface). MIDI cannot be used for the fast transfer of digital audio information required for digital sound editing. An overview of a number of digital audio systems is given by Lehrman [Lehrman 1992]. Figure 1.3, given below, shows the integration of MIDI and digital audio.

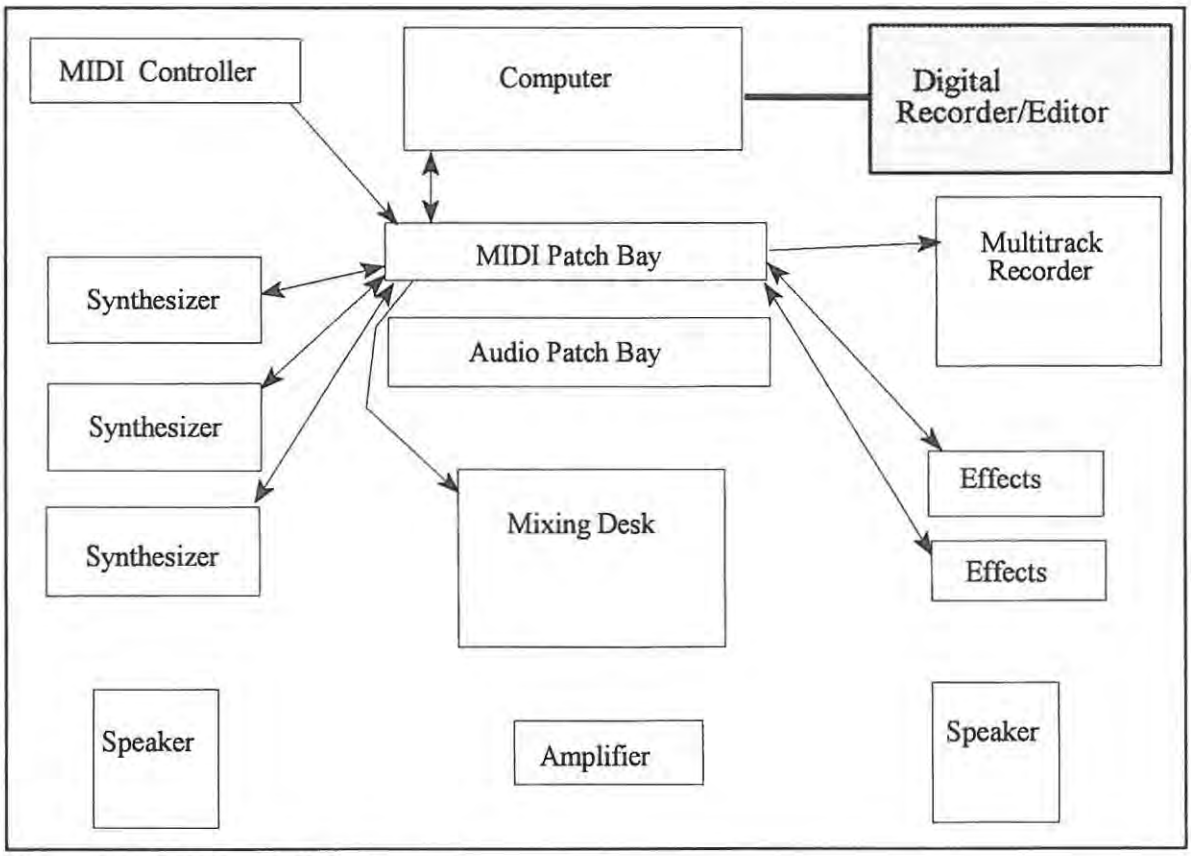

Figure 1.3 - MIDI and digital control

A description of a popular IBM PC-based digital audio system which works in conjunction with MIDI is given by Oppenheimer [Oppenheimer, 1993]. Usually such systems also allow MIDI control, to a level specified by the MIDI Machine Control specification. If used in this mode, then the 'digital recorder/editor' could be grouped into the 'Multitrack Recorder' set.

Yavelow has made the point that a computer-controlled music studio is, essentially, a local area network [LAN] using a variety of protocols such as MIDI, SCSI and SMPTE, to communicate between its constituent parts [Yavalow 1989]. However, a fact that is central to this thesis is that the LAN is oriented to use by a single user at any point in time. There is only one controlling computer. Control messages are issued from

the computer to all the constituent parts.

1.4 Sharing the Resources of a Music Studio

MIDI-based studios have been set up in a number of different environments. The music resources in these environments are shared to a greater or lesser degree.

The low price of MIDI controllable synthesizers and personal computers has meant that professional sound recording and playback capabilities are available at a reasonable price. It is now quite common for musicians to create their own project studios, some of them quite sophisticated [Molenda 1993]. A single user will usually create, record and produce the final product in such a studio. Some project studios have grown to the point where their facilities are rented out. Often, this is done to offset the cost of equipment.

It is also quite common for music departments in schools, technikons, colleges and universities to have their own MIDI-based music studios [Tywoniak 1992] [Wilkinson 1993]. In those institutions with more established electronic music programs, there will often be studios devoted to the teaching of MIDI, while other studios have workstations for the generation of digital audio using music languages such as cmusic and csound [Clarke 1992]. The studios in educational institutions are usually highly utilized. Students have regular music projects to complete, and, because there are limited studios, the demand for studio time is great. Typically, equipment will be moved between studios, as it is required for specific tasks.

MIDI is used extensively in professional post-production and recording studios for the control of music resources. Usually these studios will have equipment which is more sophisticated than that found in home and educational studios. The nature of the equipment in recording studios will tend to differ from that in post-production studios. Post-production studios are almost always equipped with some type of digital workstation equipment which works in conjunction with MIDI equipment. The digital workstations range from high-end systems such as the Synclavier (no longer in production, but still extensively used) and the Lexicon Opus, down to more moderately priced systems such as Digidesign's 'Protools' system.

Recording studios are more likely to have high-quality multitrack analogue and digital tape recorders as their medium for recording sound. Often the mixing consoles and tape decks have their own proprietary automation systems, although this is changing with the widespread use of MIDI, and the applicability of the AES-24 sound system control standard to mixer automation [Karlin 1995].

The larger music production houses will have a number of studios within a complex. The studios are rented out to recording artists, advertising companies and film and video producers. Typically, each studio will have

a certain complement of music equipment, and equipment will be moved from one studio to another according to sound requirements. If equipment cannot be easily moved, then either ad hoc or manufacturer-supplied methods are used to access the equipment remotely. A good example of a manufacture-specific solution is the AMS Logic 3 digital console, where hard drive digital storage is supplied by a remotely accessible unit. The advantages of sharing studio equipment and guidelines for doing it are given by Porter [Porter 1990].

If a number of different users are going to be using a studio facility, it is essential that they can save and restore information about their recording sessions. This information should include their sound generator sounds (synthesizer patches and sampler samples), sound processor patches, and their audio and MIDI connections. This is possible in a MIDI-based studio, where there is complete hardware and software control over all different categories of music resource. However, the studio configuration must remain stable. This implies that all the music resources needed to satisfy the requirements of a range of studio users must be resident in a single studio. Music resources cannot be disconnected from the studio and used in other studios.

A problem with this arrangement is that it leads to inefficient utilization of resources. A particular user may only use a small proportion of the music resources in a studio during a session. In large multi-studio complexes it often happens that a user in one studio requires one of the resources in a different studio. The temptation is great to disconnect the required resource from its studio configuration and use it. In practice, this scenario occurs frequently in studios and leads to a lot of time wastage. Often, if the resource is too large to move, some sort of ad hoc scheme will be developed to use the resource remotely. Usually this still involves manual patching and unpatching. Manual booking schemes are often in place in such complexes to control the movement of resources and to provide some form of planning for studio sessions.

When a large number of users share a limited number of music resources residing in one or more studio locations, a scheme has to be devised to share these resources efficiently. The next chapter (chapter 2) provides the desirable features of such a scheme, while chapter 3 explores the attempts which have already been made to share studio resources. Throughout the thesis, the focus will be on providing shared, remote access to the studio resources introduced in this chapter, and providing this access from a single interface.

Chapter 2

Desirable Features of a Shared Music Studio

Chapter 1 has introduced the various resources of a music studio and grouped them into functional categories. We have seen how the MIDI protocol has developed from its unassuming beginnings into a protocol which is used to control resources in all of these groups. Each step in the process of studio automation has moved control away from the outlying studio resources to a central computer interface. This means that a user can work more efficiently in the studio and also can be presented with a clear, simplified interface to the studio. The user does not have to become familiar with the front-panels of a range of music resources, but rather can have the essence of their functionality presented via a graphical computer interface. In this chapter, we look at the current state of studio automation from and stretch the concept of complete automation to a shared environment. When studio resources are shared in large educational and commercial music studios, studio engineers and users often manually patch and unpatch resources as they reconfigure studios to suit their needs for a particular session. Manual booking schemes are often set up to avoid conflicts of resource usage. From our experience with single user studios, this situation would seem ready for total automation.

2.1 Complete single user MIDI studio control

The MIDI software and hardware protocol can now be used to control a range of the resources in a music studio. Manufacturers have accepted the standard to such an extent that there are sound generators, sound processors, sound mixers, sound recorders and patch bays which can be controlled via MIDI. A music studio can be configured in such a way that all its resources are controlled via a single computer interface. A simplified diagram of this centralized control is given in figure 2.1 below, where all the MIDI controllable music studio resources are clustered into a networked 'cloud'.

MIDI cables carry MIDI information to and from the resources. Two audio cables deliver stereo sound to the user at the computer for monitoring purposes. The analogue audio cables could be replaced by fibre optic or copper twisted pair cables, carrying digitized sound from a digital mixer according to the AES (Audio Engineering Society) format [AES 1985], or SPDIF (Sony-Phillips Digital Interface). In this case a single cable would carry the stereo signal from the network. If the studio is used for the live recording of musicians, they can be regarded as sound generators inside the network cloud. Although MIDI can be used to control those digital recorders that accept MIDI Machine Control Messages, it cannot be used to transfer digital audio

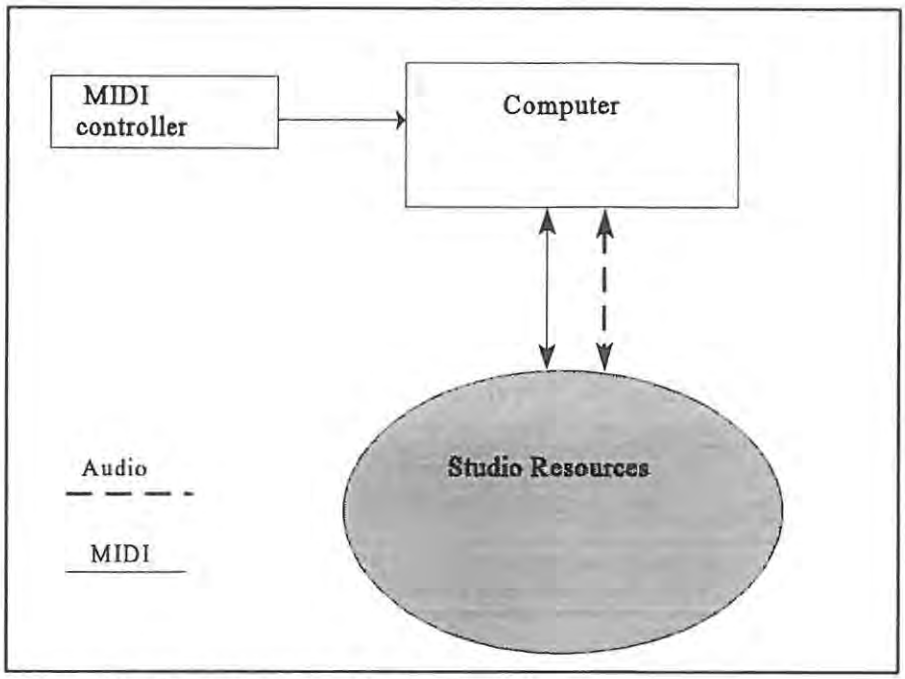

Figure 2.1 - Single Interface Studio Control

to and from a hard disk based digital audio system for the purposes of visual digital audio editing. For this reason a hard disk recording unit would be shown outside the MIDI network 'cloud'.

2.2 The Problem of Sharing Music Studio Resources

The studio configuration shown in figure 2.1 has been realized in a number of studios around the world, while many other studios approximate the ideal of complete computer control over music studio resources. The configuration has advantages in a shared studio environment. If there is complete MIDI control over all studio resources, this will usually mean that MIDI and audio patcher connections, sound generator sounds, and sound processor settings, can all be saved and later restored by users. In this way continuity is easily maintained from one session to another. However, it does not solve the problem of under-utilization of music studio resources.

Commercial and educational music studio complexes typically have a number of studios and small rooms where MIDI-controllable and hard disk based audio editing equipment is used. A good example of such an educational complex is the Middle Tennessee State University [Zweibel 1991]. Inevitably, there will be a music resource in one studio which is needed in a different studio for a limited time period. If this music resource is not currently being used, the resource will be unplugged, unpatched from MIDI and audio patch

bays, and repatched in the studio where it is needed. Equally inevitably, the resource will not be returned to its original studio! Sound recorders such as large 24-track tape machines and hard disk recorders cannot be easily moved from one studio to another, even though they need to be accessed from various studios.

A large Johannesburg-based audio-visual company, Videolab, instituted a manual booking scheme to overcome this problem. All MIDI controllable equipment resided by default in one studio, and was booked out from this studio for use in other venues. Although this did not overcome the problem of patching and unpatching, it at least provided a measure of control over the movement of equipment.

The digital audio company New England Digital, devised a scheme to allow users to access their Synclavier machine from a number of outlying personal computers. The Synclavier is a computer-based multi-purpose music resource, and although it is not produced any longer due to financial reasons, it is still widely used and incorporates many innovative concepts in its design. It provides sound generation, sound processing and hard disk recording and editing capabilities [Mellor 1989]. The various capabilities of the Synclavier can be viewed as 'soft' resources, they are all created in the digital domain and are under computer control. The computer can be time-shared to provide these soft resources to more than one user. Synclavier devised a protocol called SYNCnet to provide this shared access to multiple music resources on the single machine [New England Digital 1989].

The Synclavier system represents an ideal system in the sense that it provides access to a pool of resources by many users. It avoids the situation where one user, using only a fraction of the resources, monopolizes them all. On the downside, the available resources in the pool are only those on the Synclavier machine itself. The user is limited to those facilities which Synclavier can provide. Inevitably, there will be sound generators or sound processors produced by other companies, which users will want to utilize for their recordings. This ability to have an open architecture which accommodates a range of products from many manufacturers is an important one, and is a guiding principle in the work of this thesis. New England Digital did incorporate MIDI interfaces into their machine, but these MIDI devices were 'add-ons' and didn't fit into the complete resource-sharing philosophy.

A generalized diagram of the Synclavier environment is given in figure 2.2 below. The SYNCnet documentation does not specify the number of audio outputs available to each user and the means of audio transmission.

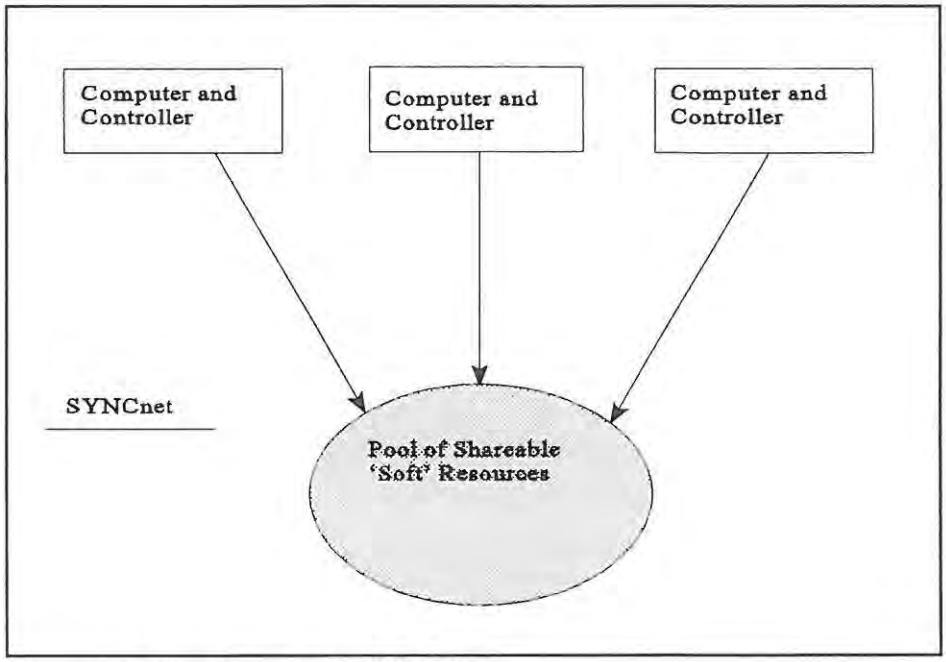

Figure 2.2 - Sharing a Pool of 'Soft' Resources

It would be ideal if all the MIDI-controllable resources in a shared studio could be clustered into a networked 'cloud', from which users could extract the resources they needed for sessions, via a workstation. Ideally then, we would like to expand figure 2.1 to allow for access via multiple workstation sites. Figure 2.3 shows this idealized situation.

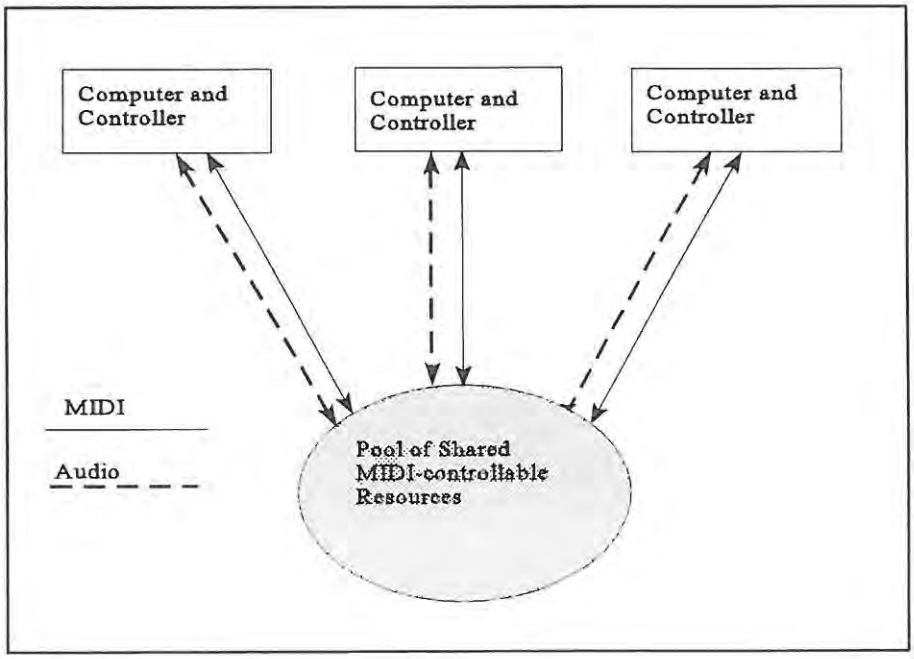

Figure 2.3 - Sharing a Pool of MIDI-controllable resources

The number of audio inputs into the network will vary, dependant on the application environment of the shared workstation. If the workstation is in a recording studio environment, then audio inputs must be available for each musician, and these will extend to the shared patcher/mixer facilities within the network cloud of resources. These multiple audio input cables may, in the future be replaced by a single cable carrying multiple digital audio channels. Such an arrangement is specified by the MADI protocol [AES 1991]. The impact of digital audio technology on the sharing of audio resources will be described more fully in chapter 6. If the workstation is to be used in a post-production, or single user environment, then one audio input for the laying down of voice or acoustic instrument tracks will probably be sufficient.

Unlike the case of the Synclavier, each MIDI-controllable resource can only be effectively controlled by a single user at a particular point in time. Each MIDI-controllable device typically has one MIDI input to allow for MIDI control over its processes. A number of synthesizers have multiple parts which can be assigned to different channels. If a MIDI patcher unit such as Lone Wolf's MIDITap [Lone Wolf 1989] is available, multiple users could access a single synthesizer by sending messages on channels particular to them. MIDI messages sent by the different users could be merged into one input stream. However, often the audio signals generated from the triggered notes are transmitted through the same stereo output, resulting in a mix of all the user's notes. Some synthesizers, such as Roland's D110 [Roland 1989], do allow parts to be assigned to their own audio outputs. They also allow certain minimum voice processing allocations to be made to the parts. In this case, however, a number of small single user synthesizers are being configured, all accessible via a single MIDI cable.

Since a MIDI-controllable resource must be assigned, in total, to a particular user during any given time period, there are only two ways in which a network of MIDI resources could be accessed by the workstation users in figure 2.3.

The first way is to switch control over the whole studio from user to user. When the studio is free, a user could see this from a workstation and request usage. Immediately, control over the studio would be transferred to the user. There might be a priority scheme devised whereby, during particular time periods, a particular user has highest priority, and will be able to switch out other users. This approach would have advantages in that it would allow access to the studio from other locations. A user could be set up and ready to work when studio time became available. Some work could also be done during another user's time period, although the possibility of being preempted at any time would hardly be conducive to musical creativity! A major disadvantage is that the user will have control over all the resources during any particular time period, when only a subset of them may be needed.

The second way is to allow users to access one or more of the resources in the network cloud of MIDI

resources for limited periods of time. However, during these time periods, the resources are under the sole, uninterruptable control of the user who has booked them. Having booked the resources, a user would have to configure a subset of resources by making MIDI and audio patch connections from a workstation.

From a user's point of view, both of these techniques to share music resources have the common advantages of simplicity and security. They allow users to access complex, interconnected music resources via a simplified interface. They also allow this to be done remotely. The studio resources can be in an area separated from the user's workspace. There needs to be no hands-on access to the equipment. In both cases all the studio equipment can be confined to a single small area.

In the initial stages of this project, the first model was adopted and a specification drawn up for user interaction with the studio resources. The system was analysed using Ward and Mellor's analysis tools [Ward 1986]. This model of the system was taken to various Johannesburg-based studios for comment. While the idea of remote access was appreciated, the model of total control over resources was unworkable in their environment. Resources had to be shared between satellite studios during any particular time period. This led to the second approach being investigated.

2.3 A Description of the Interface to Shared Music Studio Resources

Music resources can be categorized according to their ability to be remotely controlled. There are those which must be controlled manually from a front panel and which do not concern us. In the group of remotely controllable devices, there are those which can be controlled via the MIDI protocol, those which have their own manufacturer specific control protocol, and the digital audio hard disk recorders which also have their own protocols for control and for digital audio editing.

In this section we look at the desirable features of a system which allows sharing of MIDI resources, as well as sound recorders with a standard hardware control protocol such as RS232 or RS422. For the moment, hard disk recorders with digital editing devices are assumed to be local to the remote workstation, and under its sole control. Methods of sharing hard disk recorders with editing capabilities will be investigated in chapters 4 and 5. The desirable features of the user interface correspond closely to the requirements specification of the network-based shared music studio system whose configuration and construction are described in chapters 4 and 5, respectively. The following subsections describe the desirable features in broad terms. The complete, and more detailed specification for this system is given in Appendix B. The desirable features listed should comply with the overall goals, namely to provide remote shared access to all the resources of a music studio via a single interface.

2.3.1 Logging on and Booking

Each workstation user, when logging on to a workstation, should provide identity via a usercode and a password. Apart from limiting access to legal users, this will allow records to be kept of workstation and resource usage.

All workstation users must have access to a booking sheet from which they can book their required resources for the duration of a particular session. The system should ensure that only these booked resources will be accessible to the user at the booked times. A user should be able to book in advance and move from booking slot to booking slot, selecting or deselecting any of the available booking slots.

Selection and deselection should all happen by simply selecting the slot. The state of the slot determines the action of the system. The booking sheet of a particular user will be updated to reflect the booking changes made by other users.

If a user does not log on in the hour in which he or she has booked a resource, the resource should again become free for other users. Those freed resources must be displayed to other users involved in booking.

When a user's booked time for a resource expires, this resource should become available to other users and be reflected as such on any booking sheets. Any audio patches involving the resource should be unpatched and this unpatching displayed via the audio patcher/mixer interface described below . Any MIDI patches should also be unpatched and this unpatching displayed via the MIDI patch interface.

The time that a user actually uses a resource, as well as workstation usage time should be recorded. This will allow for control over resource usage. Users could pay for resources usage or be allocated fixed amounts of resource usage.

2.3.2 The Audio Patcher/Mixer Facility

The audio patch facility should allow a user to route audio signals from one resource to another, mix the outputs of resources onto an audio input, and to automate the mixing process. All this functionality should be accessible via a patch 'grid'. This facility incorporates both the ability to patch audio resources and perform remote mixing. Included in the mixing facility is the ability to adjust gain and equalization controls. In the ideal, MIDI-controlled, single user studio, patching and mixing would usually be viewed as two separate functions, often having separate programs to perform them. Novice users usually find that the operation of a studio mixer is difficult to understand. This is particularly so when audio needs to be routed

to and from other devices such as effects units. The grid layout of the audio patcher/mixer screen is intended to simplify the process of routing and mixing audio signals. Audio paths and mix levels should be clearly laid out on the grid.

Only resources which have been booked by the user for the current time period should appear on the patch grid. A user can patch the output of one resource to the input of another, by selecting the appropriate intersection block of the patch grid. The output of a resource may not be patched into the input of the same resource. The system should ensure this. The amplitude of a particular patch point should be user-controllable. In this way, the levels of a mixed signal sent to the input of an external device are user-controllable. A user should be able to record changing patch point input levels, and synchronize these changes to an external time code. Once recorded, it should be possible to 'play' back the patches in synchronization with the time code. It should be possible to record new patch level changes as old ones are being played back.

The user should have control over gain and equalization on each of the resource outputs. As for the amplitude values, a user should be able to select and alter the gain and equalization levels.

Every new booking slot period, the user might gain some resources and lose others. The audio patch screen should be altered to reflect this.

2.3.3 The MIDI Patch Facility

Figure 2.3 shows MIDI data entering and leaving the workstation. The user must be able to control how MIDI reaches resources and how it is directed back to the workstation. This control is provided by the MIDI patch facility. This patch facility also takes the form of a grid showing MIDI inputs and outputs. Once again, only resources booked by the user will appear on this patch grid.

The MIDI patch facility only needs to provide patches from the workstation to studio resources and from resources to the workstation. Patches internal to the MIDI resource pool do not need to be made. Multiple patches will usually have to be made to remote resources such as synthesizers, effects units and recorders. The full spectrum of MIDI messages will be sent via these patches. A single MIDI patch will be required to send synchronization messages from a remote recorder to the workstation. A MIDI patch may also be needed for the transfer of patch information from a synthesizer, or sample dump data from a sampler. However, these system exclusive messages will not be required at the same time as synchronization messages, and so only one patch will ever be required from resource pool to workstation.

It must be possible to change the channels of the devices in the resource pool. Each device may have multiple

parts associated with it, and the receive channels of these parts must be assignable. The MIDI protocol only allows for sixteen channels, and this number is insufficient for all the devices in a large resource pool, particularly since many of these devices have sixteen parts, each with their own channels. The user should have the capability of controlling channel assignments to the range of booked resources, otherwise many of these resources will have the same receive channel allocations. This issue will be discussed in more detail when the implementation of the system is discussed in chapter 5.

It should be possible to save both the MIDI and audio patch settings and to reload the setups, thereby restoring the original configuration. If instruments in a saved setup are not currently booked, then they, and their corresponding patches, will not appear on the patch screen. The system should utilize an identical booked resource if the resource saved in the setup file is not booked.

Every new booking slot period, the user might gain some resources and lose others. The MIDI patch should be altered to reflect this.

2.3.4 Sound Generator and Sound Processor Control

MIDI-controllable sound generators and sound processors in a single user studio are typically controlled via a sequencer program. There are a large number of sophisticated sequencers on the market. For an overview and review of a number of these sequencers, see [Helstrip 1993]. For two reasons, it would be best to allow users to use widely available commercial sequencers at the workstations. Firstly, there has been a huge investment made in the better commercial sequencers. It would require an equal investment to produce a sequencer for a shared environment. Secondly, many users have invested a lot of time in learning the features of the sequencer of their choice. The ideal situation would be one where each user could use his or her own particular sequencer in the shared environment.

2.3.5 The Audio Recorder Control Facility

An audio recorder which responds to MIDI Machine Control can be controlled via a sequencer with MIDI Machine Control (MMC) capabilities. A number of sequencers have MMC capabilities [Helstrip 1993]. A recorder without MMC capabilities, but with remote control capabilities, could be made MMC compatible in the manner described by Ntene [Ntene 1993]. Ntene's approach to a non-MMC recorder will be described in more detail in Chapter 5. The workstation could also provide its own audio recorder interface to both MMC and non-MMC devices. This section describes such an interface.

The recorder must previously have been booked. The screen will provide certain basic functionality to the

user. The user should be able to rewind, fast forward, stop, play, pause, record, and be able to select the track on which to record. A counter on the front panel will be continuously updated with the current tape position. Only recorders which can provide this basic functionality remotely will be supported. Usually this level of functionality is provided, and it can be quickly gleaned from the protocol specification of the recorder.

The user should be able to generate a SMPTE synchronization signal onto a particular track of the recorder and have this synchronization signal returned to the workstation. Firstly, however, a SMPTE start frame and frame type must be specified. If SMPTE reading is on, then MIDI Time code bytes should be sent to the workstation computer for a local sequencer to synchronize to. This SMPTE synchronization capability is incorporated into recorders which adhere to the MIDI Machine Control specification. However, for non-MMC recorders, this synchronization requirement implies the presence of a remotely controllable SMPTE striping device, and a SMPTE-to-MTC converter.

2.4 Towards a Network-Based Shared Music Studio

This chapter has introduced the concept of shared workstation access to a pool of remotely controllable music resources. Ideally, each workstation should have a MIDI and audio interface to a pool of shared resources. We have seen that the MIDI standard has been expanded to the point where it can control all the major groups of music studio resources. However, the protocol was not designed with a view to sharing studio resources. If each user at a workstation had uncontrolled usage of all the MIDI resources in the pool, chaos would result. Any user, at any time, would be able to patch MIDI control over, and audio output from, a music resource. This would possibly interrupt another user's work on the resource.

The remainder of this thesis investigates whether networking technology can be used to provide shared access to music studio resources. In other words, we are looking to find an approach which, together with a set of hardware and software tools, will manage the sharing of music studio resources. Preferably, the access to the studio resources should be via a single, personal computer interface, which can be located at a site separate from the studio itself. To begin with, Chapter 3 will describe and discuss research related to this field.

Chapter 3

An Overview of Related Research and Implementations

In a report on activities at the 1992 International Computer Music Conference, Carla Scaletti noted that there was greater interest than in previous years in the use of networks for musical activities [Keislar 1992]. Papers and discussions included 'dreaming' about the topic, and descriptions of actual examples. Networks were being used for live musical interaction, sharing resources, connecting equipment in a large studio, and forming a collaborative 'virtual' studio with local and remote equipment. This list covers a substantial number of the goals which have motivated researchers in universities and music equipment manufacturers to use networks for musical purposes. In this chapter, music networks will be categorized according to their fundamental purposes and a section devoted to each of these purposes. It should be noted that some networks have been created to achieve many goals. Also, there is often overlap in the implementation techniques used to fulfill varying goals.

3.1 Access to Remote Computing Resources for Sound File Creation

For many years, computer musicians have accessed powerful mainframe computers from remote terminals and used the processing facilities of these machines to generate sound files. Music languages such as MUSIC V, MUSIC 360, MUSIC 11 [Walruff 1983], and more recently csound [Vercoe 1990], and cmusic [Moore 1990] are used to generate digital audio sound files from instrument and score files. This is not done in real-time, and the processing facilities of the mainframes and mini-computers could be used concurrently by a number of users. This way of working was not unique to musicians working on musical applications, and a music application would simply be one of a number of jobs fired from remote terminals and processed by the CPU. Terminal connection to the mainframe would typically have been via a serial link in the past. It is more usual these days to have a TCP/IP connection to the computing resource, which will often be a powerful workstation.

3.2 Using an Expensive Music Resource from Remote Locations

It often happens that a university or commercial studio invests in an expensive music resource and needs to allow access to this resource from multiple sites to fully utilize it. This is the case at CNMAT (Centre for New Music and Audio Technologies) at the University of California, Berkeley. Here Freed has implemented

HTM, a real-time software synthesis system, on an SGI Indigo computer running under Unix [Freed 1992]. This machine is linked via Ethernet to Macintosh and NeXt computers in surrounding offices. Applications on the remote machines send control information to the HTM server using the UDP protocol, a connectionless protocol from the TCP/IP suite. A manual audio patching system is used to patch the audio output from the system to a remote user. The HTM server will respond to MIDI messages played from the remote sites and transmitted down the Ethernet. This allows for the use of MIDI-controllable synthesizers and effects units in conjunction with the Indigo-based HTM server. However, each remote site will need its own complement of MIDI equipment. There is no attempt here to share all studio resources.

The Synclavier is an expensive synthesizer, sampler, and direct to disk recording and editing device which is found in a number of professional sound recording studios. Like the Berkeley HTM server, its facilities can be accessed via remote Macintosh workstations and its sounds can be triggered via MIDI. As described in Chapter 2, a protocol known as SYNCnet is used to provide communication between the Macintosh workstations and Synclavier. The physical link uses synchronous RS-422 running at one mbit/sec. The SYNCnet protocol allows the Synclavier to be incorporated easily and efficiently into a number of remote satellite studios. However, as in the case of the Berkeley Indigo, its sound generation and processing facilities cannot be used concurrently by a number of users.

3.3 Remote Access to Digital Audio Files and Audio Processing

A number of university-based computer music studios have extended local campus networks to suit their own particular needs. Their requirements include the sharing of sound files between workstations dedicated to music processing and the utilization of the processing resources of other workstations. A good example of this phenomenon is the University of Santa Barbara's Center for Computer Music Research and Composition [Kuchera-Morin 1992]. The Center comprises a number of studios each with their own sound processing workstations, some with MIDI-compatible equipment. The workstations include NeXt's, Sun's and Macintosh's. A real-time studio incorporates Waveframe Corporation's AudioFrame digital audio workstation with a 386 front-end controller. Sound files can be generated and processed on a remote workstation and can be incorporated as samples in the AudioFrame's memory space. From here they can be played back in real-time. This is one of many applications of such a shared resource environment. The Center has enhanced resource sharing by providing a common sound file format for a range of sound processing and compositional programs such as csound, cmix and cmusic.

This capability of sharing sound files and processing resources has also been added to a few of the more sophisticated digital audio workstations. SoundNet from the company Solid State Logic allows digital audio files to be shared between workstations [Yonge 1993]. There is also a central store from which users can

access audio segments. This avoids the need for duplicated sound libraries. In the SoundNet system there is a hybrid networking structure which supports a central resource of hard disks, optical disks and backup tape drives, and makes them available to users on request. For the audio disk assignments it uses high-speed, point-to-point SCSI interconnects in a star configuration. A separate Ethernet network connects a number of processors on a bus to coordinate and control activities. Users can communicate with the processors remotely via RS422 links. Sixteen SCSI storage devices can be resident on the system and users can select the devices most appropriate to their current activities. In essence, this system allows the sharing of digital audio and hard drive resources. There are some parallels between this system and the remote studio access configuration discussed in Chapter 4, although the specific goals of the two systems differ. An outline diagram of the Solid State Logic system is given in Figure 3.1 below.

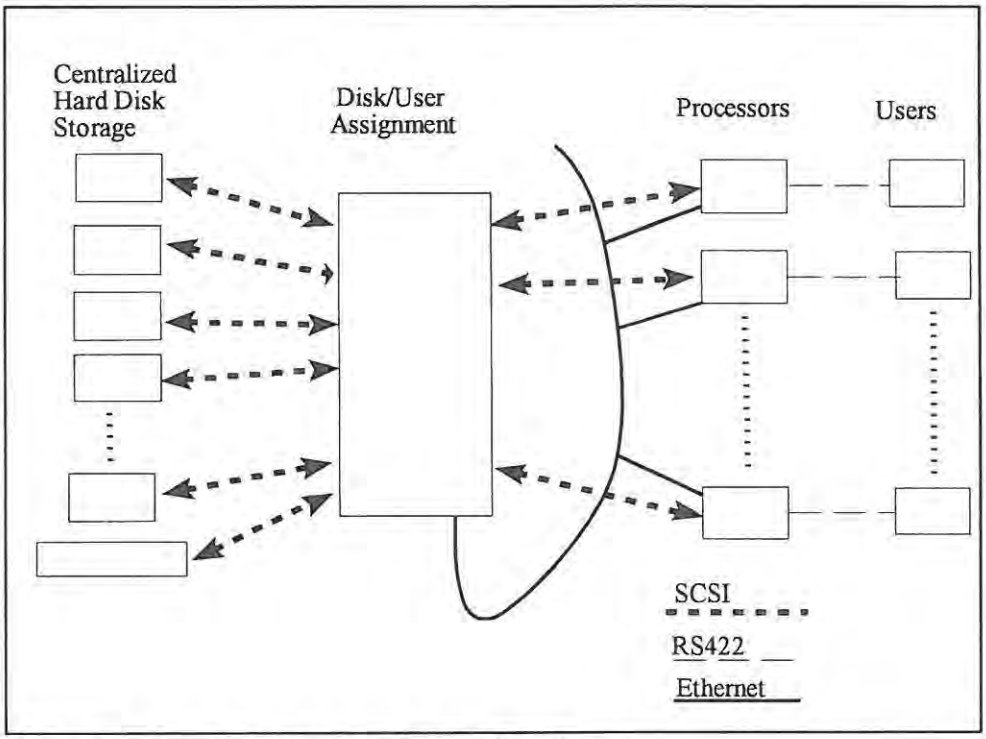

Figure 3.1 - The Solid State Logic SoundNet System

The SonicNet system from the company Sonic Solutions also allows users on different workstations to share sound files [Sonic Solutions 1993]. A user on one workstation can play a file which resides on another user's workstation hard disk. There is no need to first copy the file over to the local hard drive. This is made possible by the incorporation of an FDDI (Fiber Distributed Data Interface) network into the system. FDDI, a token ring network, has a data transfer rate of 100 MegaBits per second and allows for the real-time transfer of up to 80 channels of digital audio [Wilkinson 1993]. SonicNet also allows digital processing resources to be shared by multiple workstations. Special processing programs such as NONOISE for sound restoration and

Time Twist for time adjustment can be applied to any file stored on the network.

The goals of the three systems in this section are similar in many respects to the goals of typical commercial networked information systems, although they are biased towards audio production. The goals are to share expensive disk storage, audio information and audio processing resources. The workstations are geared towards digital audio processing, and the aims of these networks do not include sharing all the resources of a typical studio.

3.4 Music Clients and Servers

A number of researchers in the sound and multi-media fields have proposed the creation of a device server. Typically, the clients transmit time-stamped requests to the server. The server performs any necessary computation, time control and device control to satisfy these requests. A generalized diagram of this client-server model is given in Figure 3.2 below. There are various motivations for providing this type of client-server system.

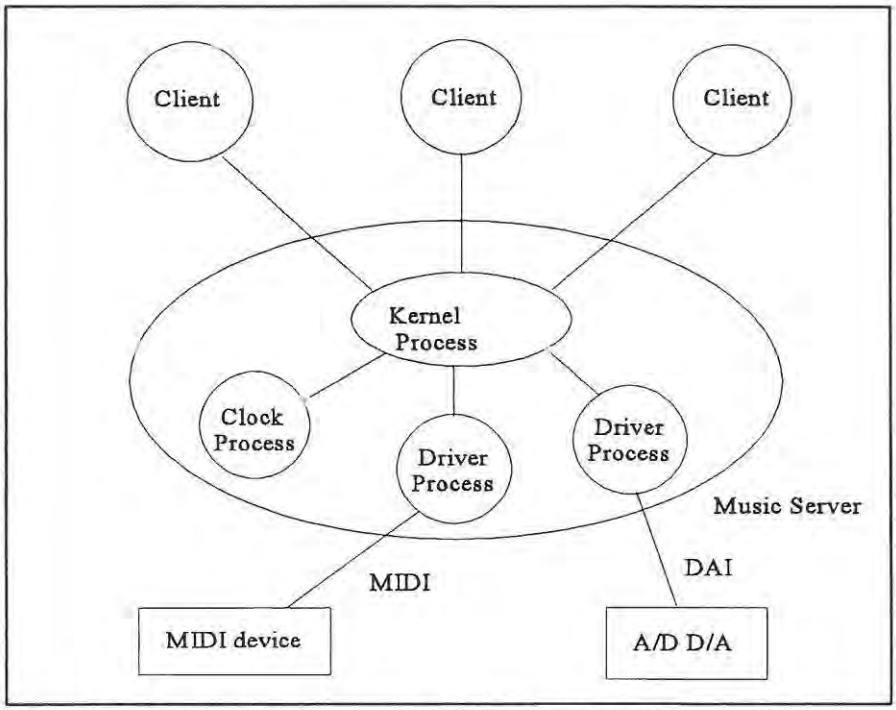

Figure 3.2 - Client-Server Interaction in a Distributed Music System

Aoyagi and Hirata have described the construction of a distributed music system known as the 'Music Server System' [Aoyagi 1992]. Their contention is that many computer music applications require a large amount of calculation for the generation of audio events and data. Their music server comprises a mix of powerful

workstations and personal computers with audio device drivers linked over a local area network. The device drivers could be MIDI or digital audio drivers. The server comprises kernel processes and clock processes running on workstations and driver processes running on personal computers. When playing musical events, the clock and kernel processes on workstations perform coarse-grain real-time scheduling and then pass the events on to the personal computers with device drivers for more accurate real-time control.

This interaction between a coarse time-grain server and more time-accurate drivers is found in the Tactus server [Dannenberg 1993]. However, the Tactus server is more ambitious in its goals, in that it synchronizes and controls a variety of media including audio, video, graphics and MIDI. Clients send time stamped data to the server ahead of real time. The Tactus server buffers this data until the appropriate output time and then forwards this data to the requested device. In the case of media with stringent timing requirements such as MIDI, the Tactus server dispatches data slightly ahead of real-time to the device, which, in its turn, performs some buffering. Tactus uses global clocks for flow control between client and server. Clients know when data will be output, and can thus send data just before it is needed. The Tactus server and its clients form a distributed system and the assumption is that there will be transmission delays between clients and servers. Real-time access of a server-controlled music device from a client-based controller would not be possible with this system. Music applications could be run with a limited start-up latency. These music applications could share the resources under server control.

A rather ambitious network standard called ArcoNet has been proposed by Allik, Dunne and Mulder for carrying real-time multi-media information [Allik 1990]. ArcoNet comprises an unspecified physical medium to carry the multimedia information, and a number of basic node types to provide a conversion between the ArcoNet protocol and protocols such as MIDI and RS232. ArcoNet is intended to meet the real-time demands of multi-media performances. The assumption here is that the interaction between nodes and a fast carrier will meet these stringent demands. Although there is no mention in the standard of a higher level client-server configuration, the standard identifies requirements for a physical substrate which would be well-suited to such a system.

Nieberle et al. have also looked to a distributed system (the CAMP system) to solve the problems of real-time performance [Nieberle 1988]. In particular, their goal was to support multi-player performance. Their proposed distributed system comprises a number of Atari ST computers connected by a token ring or token bus network. MIDI devices and specialised signal processors can be connected to the Atari computers for real-time access by performing musicians. The musicians could be playing controllers connected to other nodes on the network. In this way, the instruments do not form one big MIDI network affected by the danger of MIDI overload. Like the Music Server of Aoyagi and Hirata, the CAMP system is also intended to overcome the limitations of processing capabilities found on the personal computers which typically control

MIDI-based studios. The distributed system of Atari ST's can be viewed as a central server system which is fed data by the performing musicians, processes the data, and controls diverse sound generating devices.

The Music Server system and CAMP system are both required to generate musical events and to respond to real-time performance events. Both use the event buffering and deadline scheduling scheme of Anderson and Kuivila to meet their timing constraints [Anderson 1986].

3.5 MIDI Clients and Servers

The systems described in section 3.4 were either general purpose music servers or multi-media servers. In 1987, Bill Buxton proposed the creation of a MIDI server designed specifically to overcome the problems associated with MIDI-based networks [Buxton 1987]. He identified two major problems - bandwidth and configuration. Looking at bandwidth, problems generally occur when a computer or sequencer is controlling a large number of MIDI devices. If the music is complex, there may be intensive bursts where the amount of data to be communicated to the slave devices saturates the channel capacity of MIDI. There are ways around this problem, the sequencer's hardware can be enhanced with a number of MIDI ports and MIDI messages can be split among these ports. This reduces the load on any one MIDI line. This approach also increases the number of addressable MIDI channels. Each port adds a further 16 addressable channels.

The configuration problem relates to the fact that MIDI ports are unidirectional. A MIDI master controller can be connected to one or more slave devices using daisy-chaining or a thru box which splits up the MIDI signal. However, the MIDI specification does not naturally allow a number of masters to control a single slave device. This may be required when one or more performers or a performer and sequencer need to control a multi-voice synthesizer. MIDI merging boxes have been constructed to overcome this problem, but Buxton's contention is that these are 'kludge' or 'rubber-band' solutions and don't fix the basic problem. His proposed configuration resembles that of the CAMP system. Central to his proposal is the concept of a MIDI server. The MIDI server is a node on a LAN which serves one or more MIDI devices. It comprises a microprocessor, LAN controller, and MIDI controller. Any device connected to a node can communicate with any other device connected to a node. Control software and network connection data can be down loaded from a host computer on the LAN to the nodes. This software and the connection tables enable the microprocessor to manage connections between devices attached to nodes. The LAN is fast, thereby overcoming MIDI transmission speed problems. The bidirectional nature of the LAN together with the processing logic of the nodes allow more than one master to access a single slave device. A diagram of this configuration is given in Figure 3.3 below.

Buxton suggests that this configuration permits resources to be shared among a number of workstations and

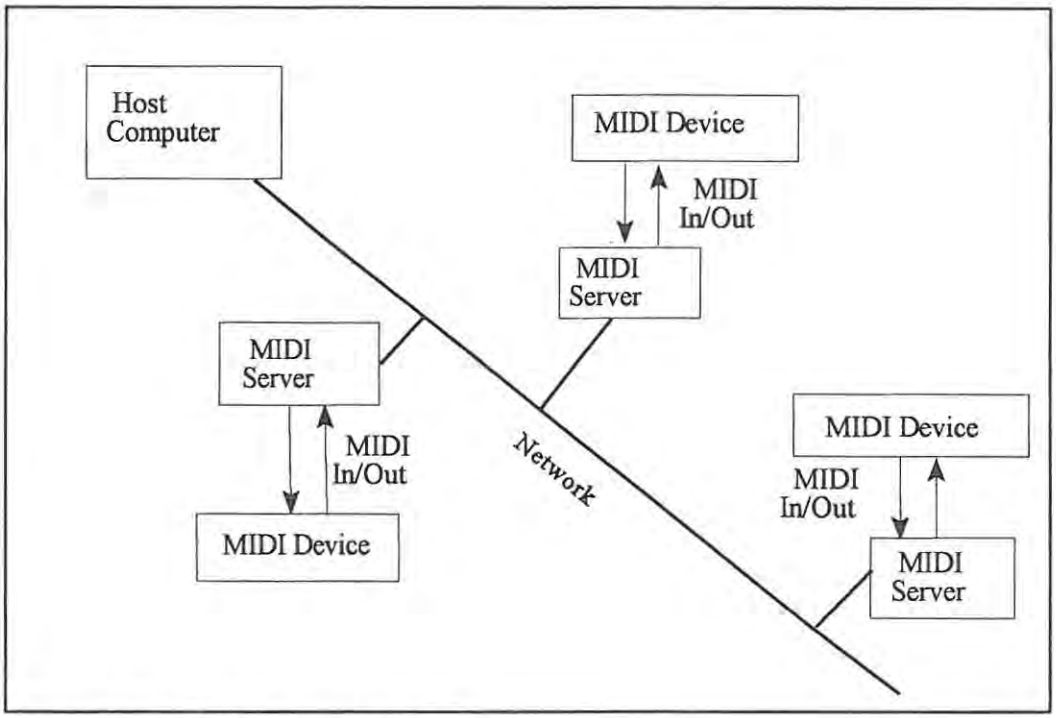

Figure 3.3 - MIDI Servers on a LAN

he cites the usefulness of this for educational situations. While the proposal addresses the problem of the routing of MIDI control information, it does not attempt to solve the problem of audio routing. In order to share studio resources effectively, the routing of audio to workstations must be addressed.

This need to patch audio is alluded to by Chris Meyer in an article which speculates about the possible future of MIDI LAN's [Meyer 1987]. His article refers to Buxton's proposal and applies it in the context of two studios sharing an array of machines. In his application, he suggests the need to book studio devices and patch audio (using a MIDI-controlled audio patch bay) to the studios using these devices. The MIDI servers control the flow of MIDI to and from devices. Also in 1987 Jim Cooper predicted that LAN's would be a feature of recording studios within a year [Cooper 1987]. His motivation for a LAN arose from the desire to save the status of all MIDI equipment in a studio and to load it back again at the start of a session. His predictions were not far wrong, and in 1989 the Californian company Lone Wolf announced the launch of its MIDITap unit and proposed the MediaLink network protocol for efficient LAN data transmission.

3.5.1 MIDITap and the MediaLink Protocol

In his forward to the MIDITap user manual, Scott Wilkinson describes the conceptual knot of keyboards, sound modules, sequencers and synchronizers occurring in many studios, and compares it to the mythological

Gordian knot [Wilkinson 1990]. He suggests that "MIDITap and MediaLink represent the sword that will cut the Gordian knot of MIDI once and for all". Like Buxton, he sees MIDI mergers, patchbays, and processors as ad hoc solutions to the problem, not really dealing with it elegantly and comprehensively.

The concept of a MIDITap unit is much the same as Buxton's MIDI server concept. Each MIDITap unit has four MIDI input ports and four MIDI output ports. It has an RS232/422 port which allows for connection to a computer's serial port. It also has two fibre optic connectors, which allow it to be connected into a daisy chain of MIDITap units. Each device connected to a MIDI port can be given a symbolic name. This applies to input and output ports. The devices connected to output ports are known as destination devices, while those connected to input ports are source devices. Any source device connected to a MIDITap unit can be connected to a destination device on any MIDITap unit in the network. Devices can be grouped together and these groups named. A group can act as either a source or destination. The output from a single source device could be sent to more than one destination device. There can also be more than one source device controlling a single destination device. MIDI processing such as filtering and transposition can be associated with a source or destination device or group. A number of MIDITap connections can be saved and later recalled. Such a collection of connections is known as a 'LanScape'. Lone Wolf have created a control program which displays connections on a network and which allows connections to be made.

Transmission of MIDI data depends on the MediaLink protocol. MediaLink is a protocol for high speed fiber optic networks which has been designed for real-time applications such as MIDI control and digital audio transmission. Up to 253 MIDITap units can be linked with this protocol, and deterministic transmission of MIDI is provided. Devices connected via the network can be up to 2.5 kilometres apart, overcoming MIDI distance limitations by two orders of magnitude.

Lone Wolf's studio networking concepts do not end at the MIDITap unit. They have proposed a more extensive networking strategy which involves carrying SMPTE, audio and video [Wilkinson 1993]. These different types of data will all be encapsulated in MediaLink packets and will be deciphered by taps specific to particular data types. Lone Wolf envisage a network comprising MIDITaps, AudioTaps, SMPTETaps, and even VideoTaps. These different tap units will send and receive data specific to their unit types. Also envisaged is an upgrade in the speed of the MediaLink physical carrier which is currently 2 mb/s. Upgrades to 5, 20 and then 100 Mb/s are proposed. In their proposed scheme, several MediaLink sub-networks could be linked together on a backbone, each sub-network connected to the backbone via a bridge.

3.6 Peer-to-Peer Networking

Although Buxton described a network of MIDI servers [Buxton 1987], which implies that there are MIDI

clients, the role of these clients and servers are interchangeable. It would be preferable to call them MIDI 'nodes'. Each node can accept MIDI messages and transmit them to the network. Each node can also accept network messages and transmit them to attached MIDI devices. This type of peer-to-peer networking has found acceptance in the field of sound system control, where the sound systems range from those used to provide sound reinforcement in stadiums and concert halls, through to systems in convention centres and airports.

The MIDI has not been widely used for professional sound system control. The limited transmission length and unidirectional nature of MIDI make it unacceptable for many sound system control applications. For some time the Audio Engineering Society (AES) has been investigating the field of computer control of sound systems. In 1988 Martin and Young proposed that the audio industry create a new standard for controlling and monitoring equipment in sound systems [Martin 1988]. The new standard, PA-22 has the following characteristics [Moses 1994]:

- A physical interface consisting of two DB-8 connectors (in and out).
- An RS422 electrical interface which is daisy chained.
- A master slave polling architecture.
- 19200 bits per second transmission speed
- A maximum transmission distance of 1.2 km.

PA-422, as a standard, did not have the success that MIDI had in the field of music system control. At 19.2 kbps it was slow, and the master-slave polling system was inefficient. There could only be one master in the system at a time. In 1992 the AES established the SC-10 Subcommittee on Sound System Control. The SC-10 subcommittee is charged with the mission of creating local area networking standards for sound systems. Two working groups within this committee are involved with the creation of two aspects of the standard.

The mission of the SC-10-2 working group is to create an application protocol for the remote control and monitoring of audio devices via a digital network. 'Application Protocol' means a set of data elements and exchange rules that implement the actual control and monitoring functions that end users see. [Berryman 1994]. The protocol is to be known as AES-24.

The task of the SC-10-1 working group is to formalize one or more transport systems for the AES-24 standard [Karagosian 1994]. Since a number of technologies are already used for the transmission of sound control information, the group diplomatically decided not to choose any particular technology. Rather they chose to institute a method for developing 'Recommended Practices' for carrying AES-24 on commonly available transports. Task groups have been assigned to form Recommended Practices for Crown Bus,

MediaLink, Ethernet, RS-485, Crest Audio Network, and more recently ZIPI.

To facilitate future expansion of the protocol, AES-24 was organized conceptually using the object-oriented notion of a class structure. The classes in the class structure have names such as Sensor, Actuator, Device, and Container. Certain classes inherit methods and attributes from classes higher up in the class structure. For example, Pan Control is a specialization of the Actuator class. The class structure was first proposed in an AES-24 draft information document named AES-24ID-xxxx. An introduction to the architecture and philosophy behind AES-24 is given by [Tyler 1994].

There has been general agreement on the outline architecture for AES-24, but the actual software protocols have been slow in materializing. In 1995, there were two detailed protocol proposals. One came from the company QSC, and is named QSC-24 [Berryman 1995]. The other came from Harman Industries and is named HCA, the Harman Communications Architecture [Karlin 1995]. QSC-24 was generated by a company involved with sound reinforcement applications, while the HCA architecture is built to accommodate networked control over a complex mixing desk. The controller can be a computer interface, as is the case for the workstation patcher/mixer interface specified in Chapter 2, or a more specific device such as a fader button. These two proposals pointed to an interesting convergence of the worlds of sound systems and music systems. Both protocols adhere to the spirit of AES-24, but differ in the message structure and design details. There is agreement that all objects in a sound or music system have to have application level addresses, and that there must be some form of address resolution to the level of transport addresses. There was some discrepancy as to how the sound or music system should ascertain what objects were attached to it. As will be seen in Chapter 5, this process is not automated in the remote studio access system. The manager of the system enters details of each object in the system at configuration time.

Following these two protocol proposals, it was decided that all interested parties should put forward protocol proposals, preferably in terms of actual C code. A set of meetings have been scheduled with the aim of completing the protocol specification by May 1996.

Progress at the transport level is dependant on the AES-24 protocol set. Recommended practices for implementation on the various transport technologies rely on a detailed high level protocol specification. It is interesting to note that both QSC and Harman are using the Ethernet protocol for the physical and data link layers. QSC uses the User Datagram Protocol above the common Internet Protocol to complete its transport level. These protocols will be discussed in more detail in Chapter 5. Harman use their own network routing and address resolution protocols.

One of the transport level protocols, ZIPI, deserves further mention as it has also been embraced by the music

system and sound system communities.

3.6.1 The ZIPI Hardware and Software Protocol

The invention of ZIPI was largely motivated by the shortcomings of MIDI [McMillen 1994]. The first goal in the design of ZIPI was to allow a new generation of highly expressive, high bandwidth digital musical instruments to communicate among themselves. ZIPI has been modelled on the OSI protocol stack [Tanenbaum 1989]. More will be said about OSI in Chapter 5. For now, all that needs to be known is that it comprises seven layers, with each layer implementing some facility required for reliable, and conceptually simple communication between two computers. The physical and data link layers are the bottom-most layers of the OSI stack and have been fully defined for ZIPI. Logically, ZIPI devices are connected in a ring, in which each device passes data to the next one around the ring. Physically, the devices are connected in a star-shaped configuration in which each device is connected to an active hub at the centre of the star. The hub maintains the logic of the ring by sending the data coming out of each device to the next device in the star. Devices are connected to the hub by a 7-wire cable with two directions of ZIPI data flowing through it. Each direction has a clock, data, and current line. The seventh wire is for shielding the entire cable.

A ZIPI-compatible device would need to have a serial chip, preferably the 8530 which automatically handles most of the data link protocol requirements. A token circulates around the ring and devices are only allowed to transmit when they receive the token. This makes the ZIPI network deterministic, unlike Ethernet. Each ZIPI network must have a designated monitor, which could reside in the hub. The monitor provides a clock for all ring communications.

ZIPI includes a network-layer naming protocol to allow devices on the network to find each other by name, or by characteristics.

ZIPI will contain multiple application layer protocols. They include a Music Parameter Description Language, which contains a far more extensive musical parameter set than MIDI. The MIDI protocol can also be sent through the ZIPI network, converted back to the MIDI physical layer, and used to control a MIDI devices. Other application layer protocols could be used for such tasks as mixer automation and machine control synchronization.

As will be seen in Chapter 4, Ethernet was used as the basis for workstation to server communication in the remote studio access system built at Rhodes University [Foss 1994] . When the ZIPI protocol becomes an implemented reality, it may well be considered as a replacement for Ethernet.

3.7 The Relevance of Current Music Network Strategies

Many aspects of the various networking approaches are relevant to sharing the resources of a music studio. If a powerful digital audio processor is one of the studio resources, or if digital audio cards are installed in network workstations, then access to a distributed library of sound files would be very useful. The method used by Solid State Logic to access this audio data shows a clear separation of control flow, which coordinates and connects devices, and the actual audio flow which employs a star configuration. This approach allows for audio to flow down paths with no possibility of channel contention and thus potential delays. This suggests a model for the flow of audio in a more generalized shared studio environment.

Sonic Solution's SonicNet provides workstation users with access to sound files and sound processors anywhere on their FDDI-based network.Their network, like that of Solid State Logic, allows sharing of specific resources which are of particular importance in post-production work However, synthesizers and effects units are assumed to be local to the workstations, and have to be patched to and mixed with the Sonic Solution's workstation inputs and outputs.

The use of the UDP protocol by Freed for the real-time control of Berkeley's remote SGI Indigo is interesting, and further reference to the UDP protocol will be made when protocol choices are discussed in Chapter 5. There is, however, no software-based management to control access to the resource.

The use of a network to enhance event and sound processing capabilities, as described by Aoyagi and Hirata [Aoyagi 1992], is not really relevant to this thesis. The assumption is that the studio resources have sufficient synthesis, processing, and recording potential. The question is how to share this potential and provide easy, well-managed access to it.

The proposal of Buxton and Lone Wolf's implementation provide a means whereby MIDI control information can be efficiently transported to and from controlling workstations. Lone Wolf have gone a step further, and proposed a more complete studio data highway on which all types of studio data can be transferred. However, what is missing in this scheme is overall management of the shared resources. There has to be a level of management which provides controlled access to the various resources of the shared studio.

From the above observations, it is clear that the various approaches to sharing studio resources are either partial, in the sense that only one type of resource is shared, or they do not provide managed access to the resources. The goal of our investigation is see whether all the resources of a music studio can be shared by a group of users working at various sites, not necessarily within the studio. This implies that all the resources should be controllable from a single workstation (personal computer) interface.

Chapter 4 investigates a configuration which will facilitate the management process of a broad range of music studio resources, including all the categories described in Chapter 1.

Chapter 4

A Hardware Configuration to Enable the Sharing of Music Studio Resources

Chapter 2 has described some desirable features of a system which would provide shared access to music studio resources. In essence, this system would allow a user at a remote workstation to book resources, patch the audio inputs and outputs of these resources to each other and to the workstation, patch resource MIDI inputs and outputs to and from the workstation, and provide control over various recorders. From the same workstation, the user should also be able to make use of the entire range of software currently used to control the resources of single user studios. This would include sequencers, patch librarians, patch editors, scoring programs, and any other MIDI-based software. This system provides the type of centralized control found in sophisticated MIDI studios.

An important feature of this proposed system is the ability to perform mixing at the patch points. In a typical MIDI-based studio, the audio patching and mixing would be performed as two separate functions in two distinct stages. Another significant feature is the provision of a booking facility. It is essential that a user have complete, uninterrupted control over a subset of the studio resources for a particular time period. It would be useful, particularly in a commercial environment, if a user's access to resources and workstations could be timed and recorded. These represent high level management features which have particular significance for an environment where studio resources are shared.

Hardware and software development for music studios has typically been oriented towards single user studios. Even some network products such as Lone Wolf's MIDITap units are geared towards easing the configuration problems in single user studios [Wilkinson 1993]. Chapter 3 has described attempts to use networks to share resources in shared studio environments. True sharing has been realized when the resources are either program resources or sound file resources in networked digital audio systems. Expensive, single user resources, such as Berkeley's SGI Indigo [Freed 1992], and New England Digital's Synclavier [New England Digital 1989], are made available to a number of users by providing remote access from a number of points. There have been no recorded attempts to provide software management of such resources, or to integrate them into a shared pool of resources and provide controlled access to this pool.

The desirable features described in chapter 2 have to be implemented in software on workstations which act as the access points to the shared studio resources. However, this software must interact with a hardware

configuration which is capable of supporting the desirable features. The goal of this chapter is to describe a suitable hardware configuration and to explain its suitability.

4.1 The Client-Server model applied to sharing music studio resources

LAN-based networks such as Novell provide workstation access to shared resources such as printers and disk drives. Software running on workstations connected in to these LANs is 'LAN aware'. A user running an editor program will be able to print a document on a remote printer. The file is down loaded by the editor program to the remote printer server. The server queues the file until the printer is available, and then down loads the file to the printer, adhering to the printer's hardware and software protocol. Some aspects of this client-server model are applicable to sharing music studio resources and some are not. LAN's are also used extensively in process control environments. Programmable logic controllers are often connected via serial links to processors (sometimes IBM PC's) which in turn are linked via a LAN to operator workstations. The control interface remains the same as when a direct serial link to the PLC's was used.

As in the case of the wordprocessor and remote printer, it is desirable to run commercially available music software at the workstations to control remote music studio resources. There is a large amount of high quality commercial software available and it would require a massive effort to custom-build this and to maintain it in such a way that it remains competitive with the commercial market. Furthermore, users have their own software preferences. It makes far more sense to provide a configuration on which this software can run. However, it is not possible to modify this software to become 'LAN-aware'. The software will only interact with the hardware configurations found in the single user situation.

It is also desirable to have a server which controls access to music resources, however requests for these music resources cannot be queued waiting for resource availability. A user must have sole access to a resource for a booked time period, and control must be in real-time. If a server is to have control over each music resource and is able to allocate dedicated access to each resource, then there must be a MIDI port dedicated to each resource. Multi-port MIDI interfaces do exist with as many as eight uniquely addressable ports [Yelton 1993]. However, if multiple workstations were sending MIDI control information along a common network, multiple servers would be needed to maintain real-time communication and prevent information overload.

Since our intention is to allow a user at a workstation to run commercial music software designed for a single user studio, it is appropriate to look at the hardware interface between the single user with a computer, and the studio resources. This can serve as a starting point for a more extensive hardware configuration for

sharing resources.

When a single user accesses a MIDI-based music studio from a controlling computer, he or she usually listens to the music being played via a pair of audio cables that transmit a stereo signal from a mixing desk. The signal could be amplified and sent to two speakers or could be transmitted to stereo headphones. Sometimes, the user will sing or play an acoustic instrument into a microphone. The microphone will be plugged into a mixer and the analogue signal could be recorded onto one of the tracks of a multitrack mixer. MIDI control signals are sent from the computer along one or more MIDI lines to the MIDI-controllable music resources. A single MIDI line is usually used to transmit synchronizing MIDI messages or system exclusive messages back to the computer. A simplified diagram of the hardware interface between user with computer and the studio is given in Figure 4.1 below.

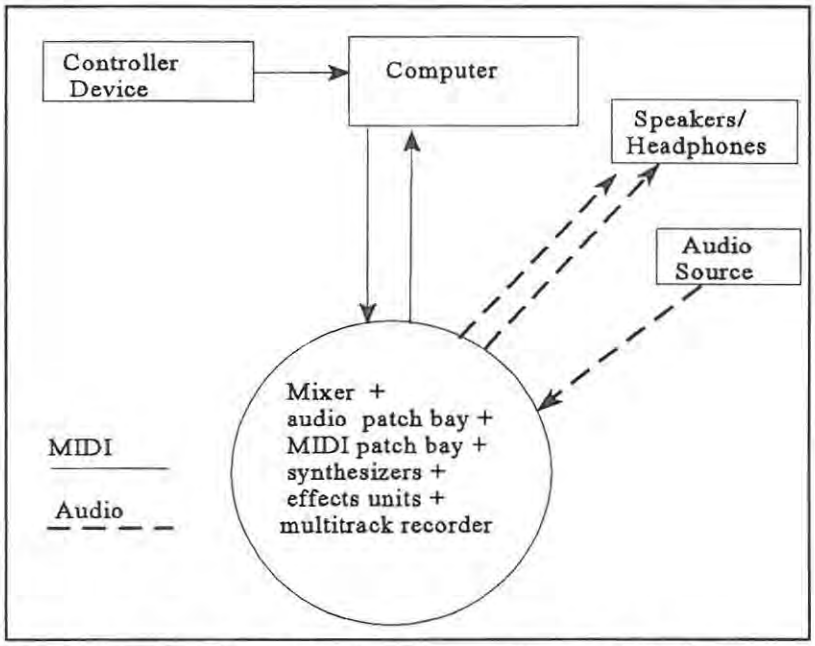

Figure 4.1 - Hardware Interface between User and Studio

Two audio lines carry a mixed and processed stereo audio signal to the user. One audio line carries mono audio from the user. One or more MIDI lines carry control information from the user's computer. One MIDI line carries synchronization or system exclusive information to the user's computer. A multitrack recording device has also been shown in the resource circle, assuming that it can be controlled via MIDI Machine Control messages from the studio computer. In this chapter, it will be assumed that the shared music studio system will cater to multiple users, all with these defined requirements for audio transmission and feedback, and control signal transmission and receipt. A hardware configuration must now be found which allows this audio and control signal transmission to and from each of a number of workstations, and also ensures exclusive use of a subset of shared resources by each workstation user.

4.2 Transmitting MIDI Control Signals in the Client-Server Model

A simple solution to the control problem is to run MIDI cables directly to the shared resources via a shared MIDI patch bay. A number of commercially available MIDI patch bays exist [Yelton 1993]. Typically, these patch bays will have their MIDI routings set up and changed via MIDI messages. In a single user studio, these controlling messages will be transmitted from a sequencer. Like other MIDI controllable resources, MIDI patch bays are designed to have a single master controller. A merge unit could be used to allow multi-user access to the patch bay, but this would not be desirable. Users would have access to the patch bay and would be able to perform routings to and from unbooked resources. It makes far more sense to send all patch requests to a server. The server can be the single master controller for the patch bay and will ensure that workstation users only perform patches on their booked resources. With this arrangement, there must be communication between the workstations and server to transmit the patch requests. This communication can take place over a networking medium such as Ethernet. A hardware configuration which will allow the controlled routing of MIDI messages between workstations and studio resources is given in Figure 4.2.

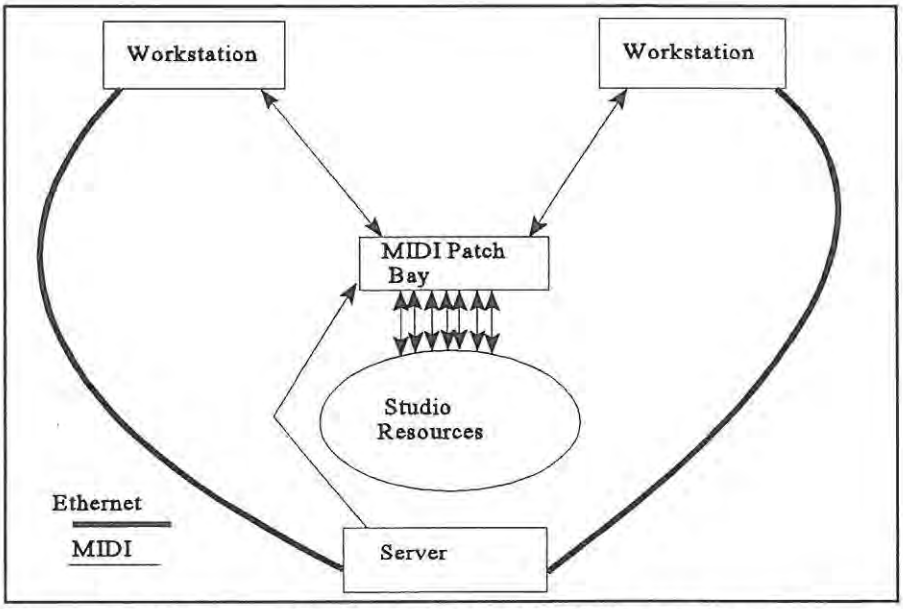

Figure 4.2 A hardware Configuration for MIDI Message Routing

The server will be in close proximity to the patch bay and resources and will adhere to its control protocol, typically the MIDI protocol. The server plays a role analogous to the print server in an office LAN. The advantage of the above configuration is that MIDI is routed directly to the resources with no intervening processing. The response will be identical to the single user situation, and all single user software will run with no modification. A disadvantage is that MIDI cabling needs to be run between the workstations and MIDI patch bay. The MIDI specification states that MIDI cables may have a maximum length of fifteen

metres [IMA 1989]. This may not be sufficient for certain studio layouts. This limitation can be overcome by using a different transmission specification, such as RS422 or RS423, for the transmission of MIDI data [Scott 1986]. In this case the cable length could be 1200 metres. Lone Wolf supplies a fibre optic transmission system for the direct transmission and reception of MIDI data. There is also the option of wireless transmission of MIDI, where cabling ceases to be a problem. Gambatte's MIDIStar Pro system provides a range of 120 metres [Westfall August 1989], although it is expensive, and probably not cost-effective for sharing resources amongst a number of workstations.

4.3 Transmitting Audio Signals in the Client-Server Model

A similar strategy can be adopted for audio. Audio can be routed from the shared resources to the workstations and from the workstation to one or more resources. The routing can be performed by a remotely controllable audio patch bay. A number of MIDI-controllable audio patch bays are available commercially, examples being Intone's MIDI Maestro audio and MIDI patch bay [Oppenheimer 1991] and 360 Systems' AM-16 Series Audio Cross point switchers [360 Systems]. Both are 16x16 audio patchers, although a number of AM-16 modules can be connected to expand the number of inputs and outputs. The audio patch bay will be controlled by the server.

An audio patch bay which simply provides routing facilities is not capable of providing the patcher/mixer grid capability described in chapter 2, where a mix level could be specified at each patch point. A MIDI-controllable fader box, such as the Iota Systems MIDI Fader [Westfall 1988] could be used to provide attenuation of each patched signal. A number of audio lines could then be run to each workstation, where mixing down to stereo could take place. This approach has a number of disadvantages. At least eight audio lines would need to be run to each workstation, and this means that the audio patcher would have to have a large number of outputs. A large amount of cabling would have to be run to each workstation. This would be difficult to implement and expensive for distant workstations. Each workstation would require a mixer, and the user would have to understand the operation of the mixer, something which the patch grid was designed to avoid. A MIDI controllable mixer such as Yamaha's DMP7 Pro [Westfall 1988], or Mark of the Unicorn's MIDI mixer 7s [Sofer 1991] could be used at the server side to mix down the audio patch bay outputs for each workstation, and allow two lines to carry the mixed audio signal to a workstation. In this case, each workstation would have to be allocated its own mixer, an expensive option, and the number of MIDI resources a workstation could access simultaneously would be limited by the number of input channels on the mixer. This option would also require the audio patcher to have a large number of outputs.

A far better option is to provide a closer match between the patcher grid requirements on the one hand, and the implementation hardware on the other. Richmond Sound Design manufacture 8x8 and 8x16 patchers

which allow for attenuation at the patch points [Richmond 1989]. The units are built up from digitally controlled attenuator (DCA) modules, each having 8 DCA's. A DCA module is assigned to each output line, allowing a controllable mix of eight input signals to be placed onto each output line. Patchers of various sizes could be built using these modules. A patcher such as this with a fader at each cross point mirrors the high level patch grid in chapter 2. However, the DCA modules are expensive, and this makes the construction of a large patch matrix costly. TRAILS, a patcher built for sound localization, also uses attenuation modules, and reduces cost by mixing 24 inputs onto an 8 channel mix bus [Bernardini 1989]. The mixed signals can then be dispersed to 24 outputs. However, this system does not provide the full functionality required to meet the specifications in chapter 2.

The ideal, then, is to have an audio patcher/mixer which incorporates remotely controllable attenuators at the patch points, and which allows input signals to be mixed at varying levels onto the output lines. This would enable a particular workstation to use any set of MIDI resources, and for a mixed stereo signal to be sent along only two audio lines to the remote workstation. Just as for the MIDI patch bay, it is not feasible to allow each workstation to have control over a communal audio patch bay. This control should rather be allocated to a single server. This may be the same server which performs MIDI patch bay control. Patch and mix control messages can be sent over a network to the remote server from the workstations. The server can carry out these requests by sending the required control messages to the audio patcher mixer, and it can check their legality. A hardware configuration which will allow this controlled patching and mixing from outlying workstations is shown in figure 4.3 below. Audio cable lengths are not a problem in this configuration, as a maximum length of 15 metres has already been set by the MIDI length limitation. However, just as for MIDI, wireless transmission is a possible, though expensive option.

4.4 Custom-Built Hardware Subsystems

Figures 4.2 and 4.3 show network-based hardware configurations which are capable of providing the desired functionality described in chapter 2. At the heart of these configurations are remotely controllable patch bays. The commercially available MIDI patch bays and audio patcher/mixers did not fulfill the requirements of this project for a number of reasons. The next two sections describe a MIDI patcher and audio patcher/mixer which were specially constructed for the project. The motivations for their construction are included.

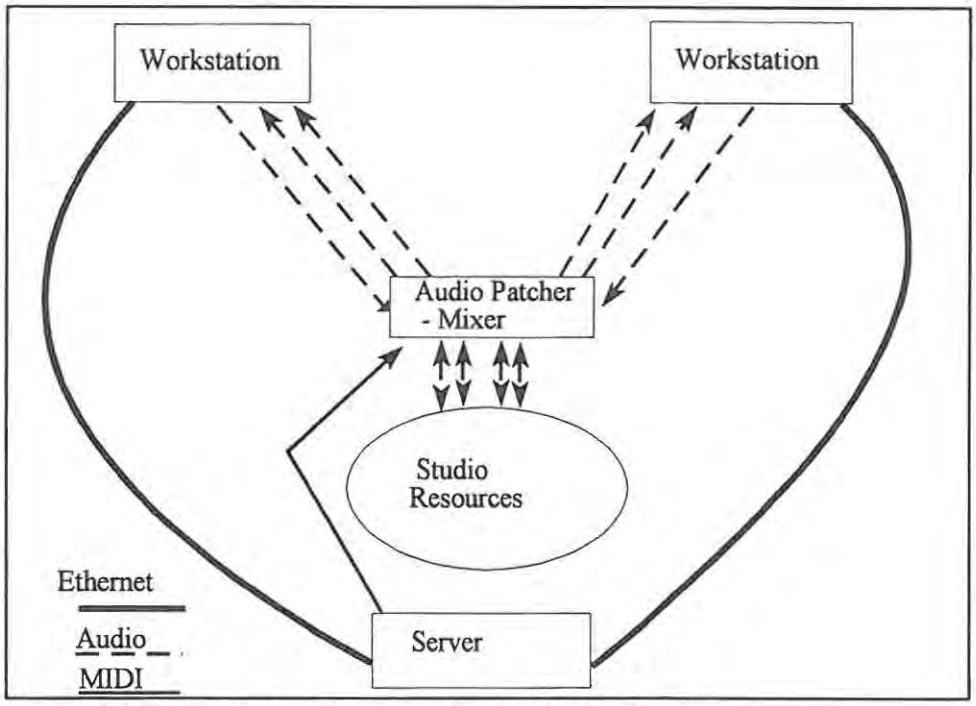

Figure 4.3 - A Hardware Configuration for Audio Routing

4.4.1 A MIDI Patcher for distributing MIDI control signals in a shared studio.

Two MIDI connections need to be made between each outlying workstation and the remote studio. One MIDI connection is used to transmit MIDI messages from a commercial program such as a sequencer to resources in the remote studio. The other transmits synchronization signals and possibly patch or sample dump information back to the workstation. The MIDI patcher which was built for this project was designed with these specific requirements in mind. There is a component which performs the workstation to device routing and another component which performs the device to workstation routing.

MIDI patchers can either perform dumb or intelligent MIDI signal routing. The dumb MIDI patchers simply transmit all the bits of a MIDI message from source to destination without modification. Intelligent MIDI patchers are capable of performing some form of processing on the messages. This processing may include filtering particular message types, or altering the values associated with messages. This type of message filtering or mapping can usually be performed by a sequencer, and so the MIDI patcher was designed with no processing capabilities. This made the design and construction simpler.

An important decision had to be taken at design time regarding the number of MIDI inputs and outputs. Each MIDI workstation would probably require more than one resource at any point in time. It would also be a rare case for all workstations to be occupied. If this case did exist, then each user would probably require some

sort of sound generation hardware, and also some sort of signal processing hardware. The decision was taken to provide twice as many sound resource inputs and outputs, as workstation inputs and outputs. An essential requirement was that the MIDI patcher be expandable to take into account a growing network. This requirement was a major motivating factor for the construction of a MIDI patch bay. Commercial MIDI patch bays do not provide expandibility. Most provide an 8 input, 8 output patch capability, although some provide a 16 input and output capability. The J.L. Cooper synapse tops the log with a 16 input 20 output capability. Often remote control applies only to switching between patch settings as opposed to individual patch control [Yelton 1993].

The number of workstations and MIDI devices connected to a single patch bay had to be related to the technology used for the implementation of the routers. Multiplexors capable of routing 8 and 16 channels abound, and consequently, the MIDI patcher was designed to support 8 workstations and 16 MIDI-controllable devices.

The MIDI patch unit circuitry comprises three sections:

> The MIDI input interface circuitry to convert MIDI logic levels to the TTL levels of the routing circuitry. It also provides MIDI extend inputs for patcher extension. A further 16 devices could be connected to the 8 workstations.
>
> The resource to workstation and workstation to resource routing circuitry. The routers are placed in parallel, allowing a single source (input) to be routed to several sinks (output). This is important, for example, if a sequencer is sending out MIDI messages to multiple devices.
>
> The output interface circuitry which converts the logic levels of the routing circuitry back to MIDI. It also provides MIDI thrus which are exact replicas of the MIDI inputs for expansion purposes. This allows the 16 MIDI inputs to be routed to more than 8 workstations.

The layout of the routers within the MIDI patch bay is shown in Figure 4.4 below.

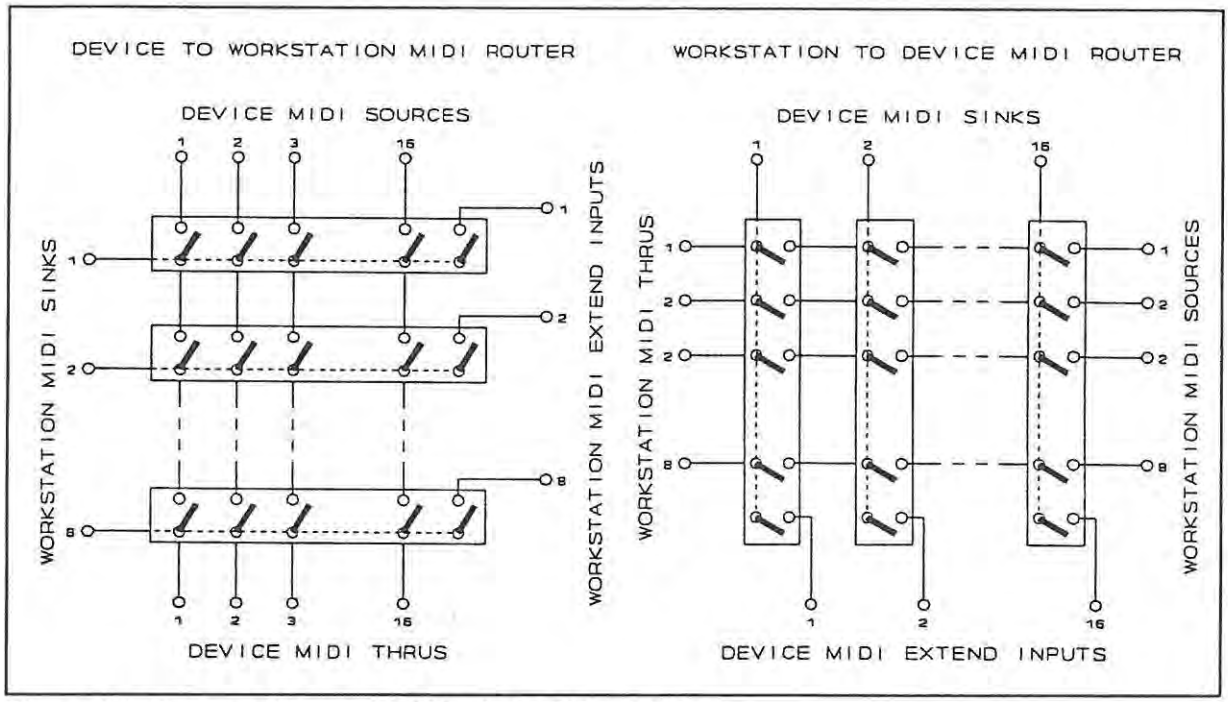

Figure 4.4 - Layout of Routers within the MIDI Patch Bay

Internal control over the process of MIDI patching is by way of an internal microprocessor and associated circuitry. The same microprocessor was used for the MIDI and audio patch bays, and it will be described in more detail in section 4.4.3. More detailed information regarding the design and construction of the MIDI patcher can be found in [Wilks 1994].

4.4.2 An Audio Patcher for Patching and Mixing Audio Signals

The main aim behind the design of the audio patcher/mixer unit was to provide each user at a workstation with the functionality of an audio patch bay, as well as a professional audio mixer. Two components were originally designed, an audio processor component, which was to provide gain and equalization control, and a patcher/mixer component, which was to provide audio patching and mixing capabilities. The audio processor circuitry was designed and prototyped, but was not constructed into a complete unit. The audio patcher/mixer unit was completely constructed and successfully used to provide surround-sound for a 1993 South African National Arts festival production. The relationship between the audio processor and audio patcher/mixer units is shown in Figure 4.5 below.

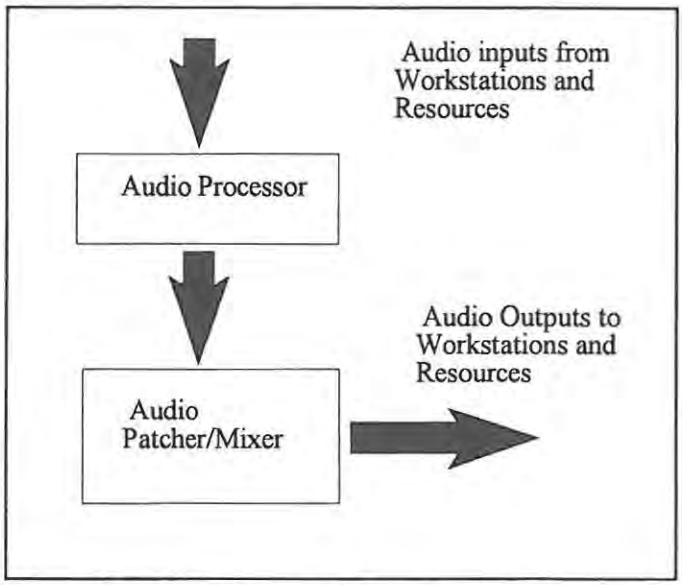

Figure 4.5 - The Relationship between Audio processor And Audio Patcher/Mixer units

In the current remote studio access system, source levels are manually adjusted on the input side of the patcher/mixer to provide sufficient dynamic range. Equalization and effects are provided via sound processors in the main studio. The audio patcher/mixer can be used to route sound generator signals to the sound processors.

The possibility of converting all input signals into the digital domain, and performing the patching and mixing digitally, was considered. However, multiple analogue-to-digital and digital-to-analogue converters would be required at the input and output sides of the patch bay. Multiple processors would be required for the routing and mixing of the digital streams, and time-division multiplexing would have to be performed on a high speed bus. Such an implementation would have been expensive in terms of time and hardware, and would not have added to the investigation of remote shared studio access. It was thus decided to leave the audio signals in the analogue domain, and control their processing by some form of digital-to-analogue interface. Digitally Controlled Attenuators (DCA) were constructed to provide this control.

Each DCA consists of a series of potential divider ladders, each performing a different, but related level of attenuation. The attenuator with the finest level control is first in the audio signal patch, with the coarsest level control coming last. This is to maintain the maximum signal-to-noise level ratio within the DCA. Figure 4.6 below shows the construction of the DCA.

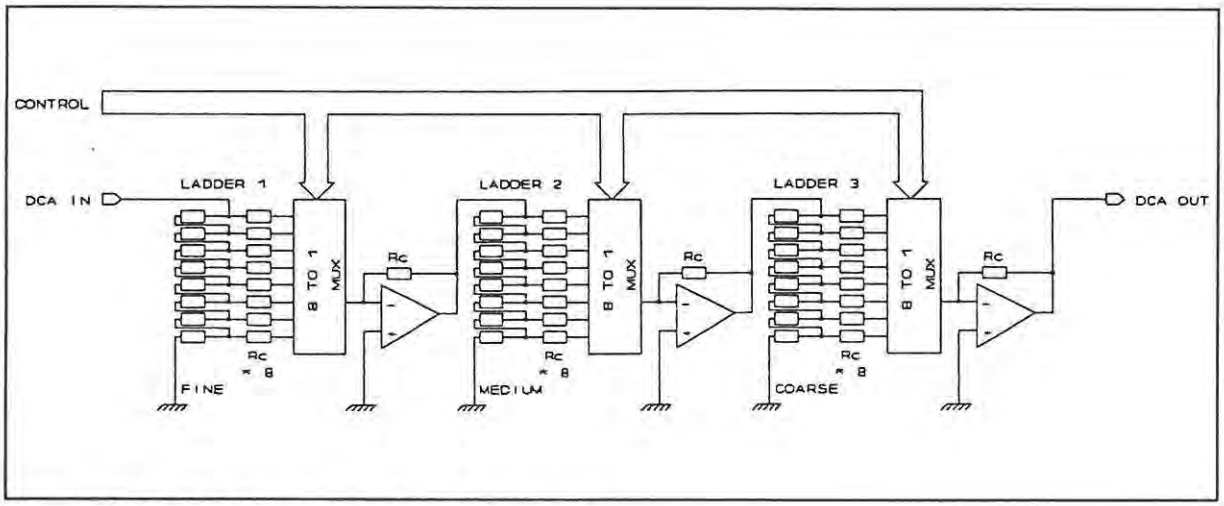

Figure 4.6 - The Digitally Controlled Attenuator

In the design of the processor component, DCA's were used to control gain and equalization levels. Equalization is provided by a set of three filters, providing bass, midrange and treble control. In the audio patcher/mixer component, they are used to provide amplitude control at the patch points.

The internal structure of the audio patcher/mixer component can be conceptualized as a two-dimensional matrix with inputs on one axis and outputs on another. A patch node had to be created which would patch an input to an output at each patch point, and control the level of the output signal. The output of the node needed to be mixed with the outputs of other nodes on the same input. The signal level at the patch point had to be digitally controllable.

Such a node was created, but, to reduce the amount of electronics and thus the expense of the unit, the node was made to be 'floating'. The nodes do not have a fixed point within the matrix, but can be assigned to any patch point in the matrix. Conceptually, there is a pool of nodes. A node can be assigned to a patch point from this pool, and after use, returned to the pool. This concept is shown diagrammatically in Figure 4.7 below.

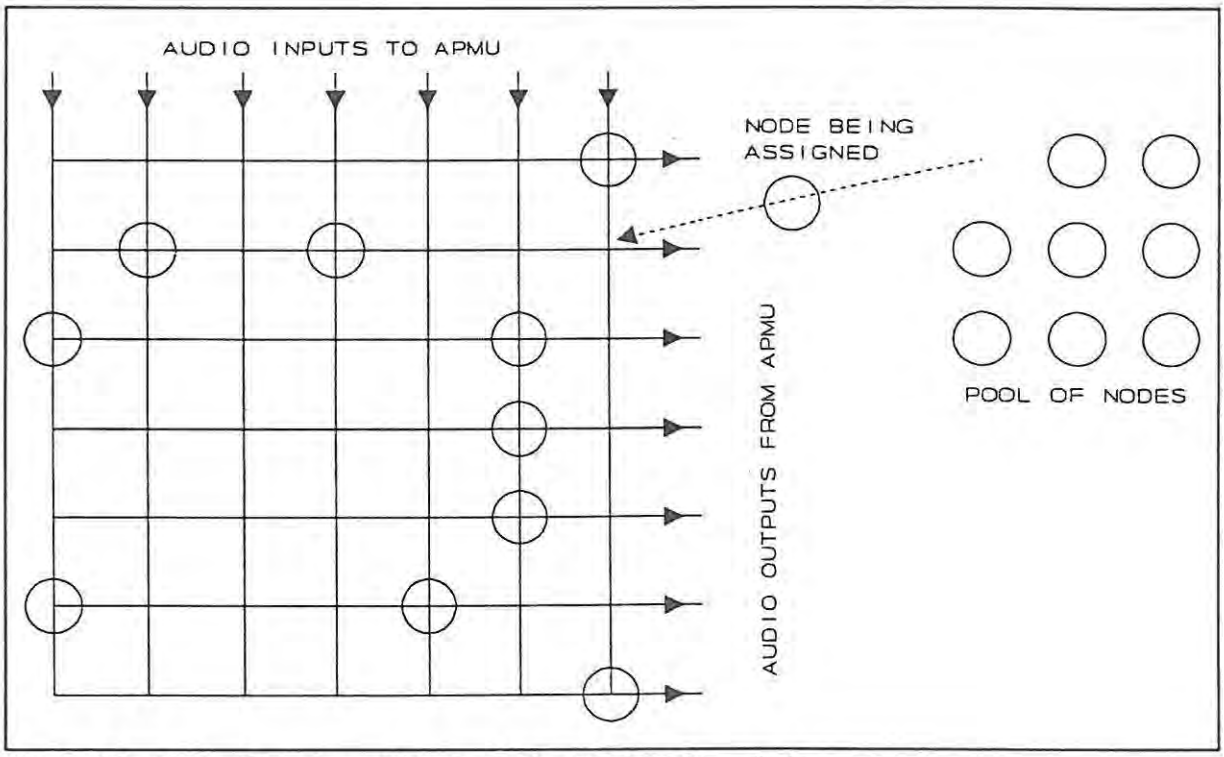

Figure 4.7 - Node Allocation within the Audio Patcher/Mixer Unit

Each audio patcher/mixer unit contains 16 audio inputs and 16 audio outputs. Patcher/mixer units can be grouped to provide more than this number of inputs and outputs. For example, a 32 x 32 patch matrix can be constructed from four such units. A larger patcher mixer unit would be clumsy in size and difficult to construct due to the excessive amount of electronics. Each unit can contain up to 48 patch nodes. This allows each audio signal entering the unit to be patched to three output points.

The structure of a patch node is shown in Figure 4.8 below. It comprises an analogue input multiplexor to select the source of the audio signal [Analog Devices 1988], a DCA to attenuate this signal, and an analogue output multiplexor to mix the signal created by the DCA onto the correct output line. The user has enough dynamic range to entirely mute an audio signal. The total dynamic range is 100 decibels. The mix level is quantized into a series of attenuation steps, each step being of the order of 0.2 decibels. Since the MIDI message value used to control the node level ranges from 1-128, some of the steps have to be larger than 0.2 decibels. The larger steps were chosen in the region of low volumes as it is more difficult to perceive changes in audio level at low volumes. There is an exponential increase in step size starting at -11 dB and ending at -101 dB.

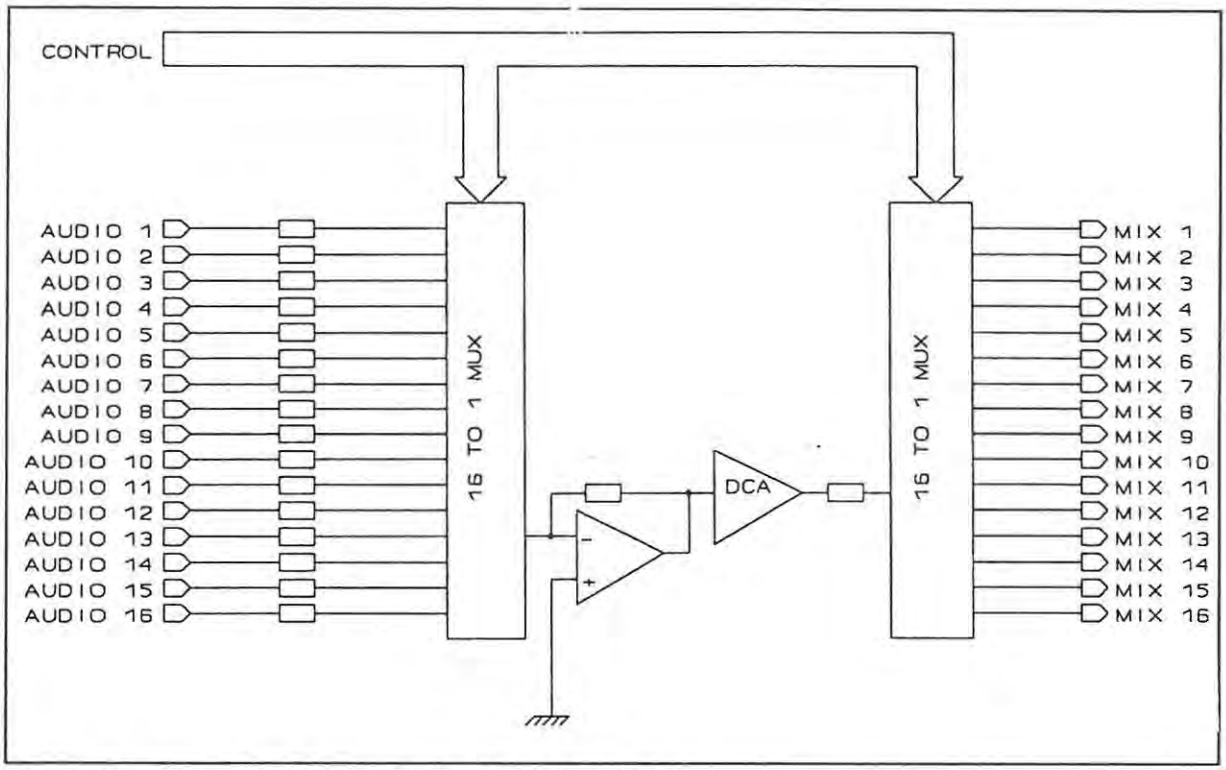

Figure 4.8 - Structure of an Audio Patcher/Mixer Node

The audio patcher/mixer circuitry comprises three sections, line buffer circuitry, node circuitry, and line output buffer circuitry. The line input circuitry ensures expandability by supplying a buffered thru for each audio input. The output buffer circuitry takes the mix bus, containing the outputs from the nodes and the mix extend audio inputs, as its input. The mix extend audio inputs allow external access to the patcher/mixer's mix busses, thereby providing for expansion. A more detailed description of the audio patcher/mixer design and circuitry can be found in [Wilks 1994].

4.4.3 The Microprocessor Control Unit

The functionality of both the MIDI patcher and audio patcher/mixer is controllable via a microprocessor control unit which resides within each of the patchers. The microprocessor control units in each of the patchers read MIDI system exclusive messages, and send data to their addressable components, depending on the contents of these messages. Each system exclusive message contains patcher identification information, parameter identification information, and the new values of the parameters. The manufacture's identification number, which is part of any system exclusive header, is set to 7D hex, a number which has been reserved for research purposes. For each of the patchers, the body of the message contains a number of patch specifications. Each patch specification comprises an input and output connector, as well as an indication of the nature of the patch. A patch value will usually accompany an audio patch specification. The

format of the system exclusive messages is given in Appendix C.

System exclusive messages are made as small as possible to prevent bottle-necking on the MIDI connections from server to patchers. Data is only sent for parameters which need changing, and an 'end of exclusive' byte is only transmitted when a message is to be sent to a new unit. Two or more parameters are placed within a single byte, where possible. The possibility of bottle-necking increases with each patcher which is connected to the server. This could be alleviated if the server were to have a multi-port MIDI interface.

The 8031 micro controller from the MCS-51 family of 8-bit micro controllers was chosen to implement the control unit within the MIDI and audio patchers. It has the required processing speed and a serial port ideally suited for the receipt and transmission of MIDI data. It has two on-chip timers and allows for the addition of 64K of program memory as well as 64K of data memory. Each patcher required a large number of address decoded outputs for selecting the range of multiplexors, as well as a number of data busses for transmitting data to them. Data entry circuitry had to be provided for reading patcher identification switch settings. Each patcher has two MIDI thrus which allow multiple units to be chained together via MIDI connections.

Software for the patcher control units was written using a C cross-compiler. After successful compilation and linking, the program was transferred to an Erasable Programmable Read Only Memory (EPROM) using an EPROM programmer attached to an IBM PC. The EPROM was then plugged into the control unit and the patcher powered up.

The software was derived from analysis and design specifications which were drawn up using the Ward and Mellor real-time modelling tools [Ward 1986]. In Ward and Mellor terms, the context schema of a system describes the high level interaction of a system with its environment. It provides a useful summary of the system's operation, and to this end the context schemas of the MIDI and audio patchers are shown in Figures 4.9 and 4.10 below. The Ward and Mellor toolset will be described in more detail in Chapter 5.

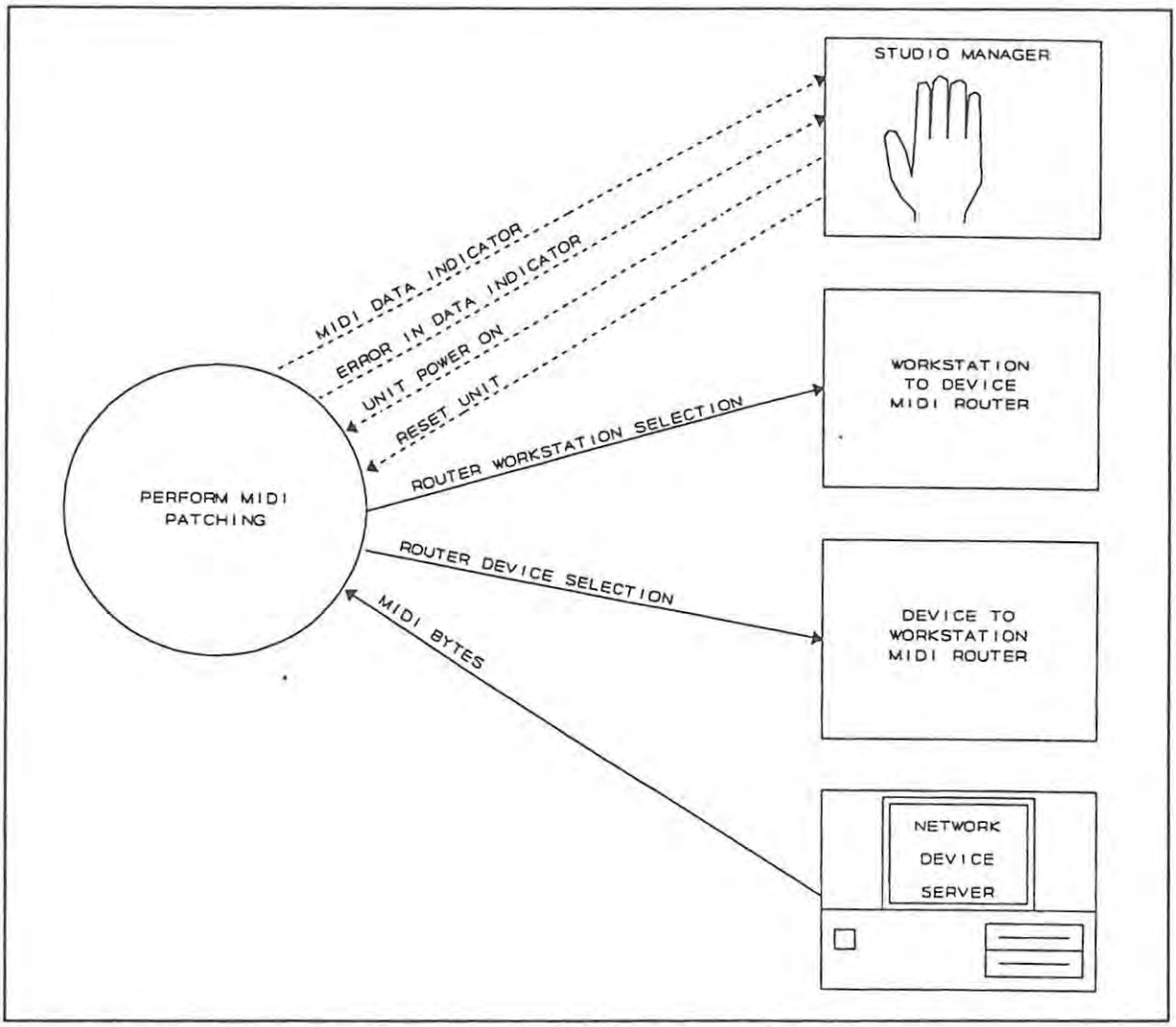

Figure 4.9 - Overview of Interactions between MIDI Patcher and Environment

4.5 Towards a Software Implementation

This chapter has described a hardware configuration which should allow remote workstation access to a pool of shared music studio resources. One can think of the server in this configuration as a busy studio manager who receives patching and mixing requests from a number of studio clients, and performs these tasks by sending messages to a MIDI patcher and an audio patcher/mixer.

One area where control is not completely performed by the 'virtual' studio manager, is that of multitrack recorder control. If the remote studio contains a multitrack tape recorder which is MIDI Machine Controllable, then the tape will have to be loaded by a user at the start of a session and unloaded at the end. A hard disk based multitrack recording unit will usually have a file system for the management of audio files. If such a hard disk recorder is to be accessed remotely, the user will have to open a file at the start of a

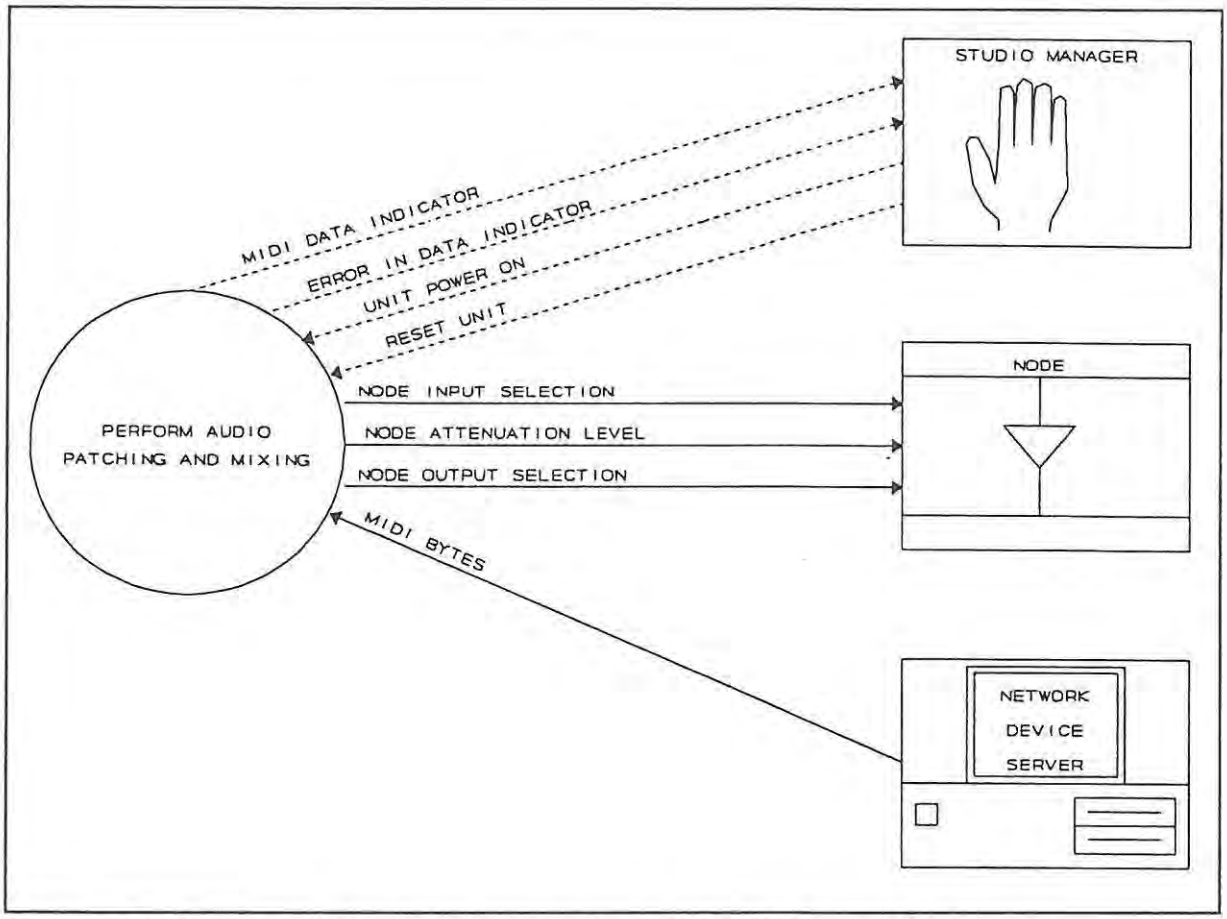

Figure 4.10 - Overview of Interaction between Audio Patcher/Mixer and Environment

session, then close it and possibly back it up at the end of the session. There are no file management commands built into the MIDI Machine Control specification which would allow this type of operation to be performed remotely. However, most current hard disk recording units are MIDI Machine controllable. This level of interaction with a remote device is contrary to the spirit of the remote studio access system, which aims to provide remote shared access to a studio from a single user interface. A number of hard disk recording units are controllable at the hardware level via a proprietary parallel, or SCSI interface. The control software for these units resides on a standard personal computer such as an IBM PC. With such a unit, the hardware interface card could reside in the remote studio server, and the control software on a workstation. This possibility will be explored further in Chapter 5.

The response times of the MIDI-controllable resources in this shared environment will match those in a single user studio. The nature of the hardware configuration, with the star-shaped arrangement of MIDI lines, ensures this. There is no contention from other workstations on the MIDI lines. Each workstation has a clear MIDI control path to its booked resources. This fact was the major motivating factor for the choice of a star configuration for the transmission of MIDI data.

The quality of audio being received from the studio resources is only dependant on the audio patcher/mixer. The audio specifications for the audio patcher/mixer can be found in [Wilks 1994]. The patcher/mixer was designed to have signal-to-noise ratios and a dynamic range comparable to current analogue mixers. As in the case of MIDI control, each workstation has its own analogue audio lines feeding and returning from the audio patcher/mixer.

The network carries patch level changes from all the workstations to the remote server. There are three areas where delay can be introduced into this patch change transmission:

- The Ethernet used for the transmission of patch changes from workstations to server can cause delays
- The server may be overloaded by the number of patch level requests.
- The MIDI link between the server and audio patcher/mixer unit may be incapable of carrying the number of system exclusive messages generated by the server.
- The audio patcher/mixer unit may be incapable of dealing with the MIDI messages being passed to it.

The processing speed of the audio patcher/mixer unit is dependant on the speed of the in-built serial port and CPU of the 8031 chip used as the microprocessor control unit. The audio patcher/mixer unit was physically tested by sending a continuous stream of MIDI messages to it [Wilks 1994]. A diode on the unit is used to present a clear error condition when buffer overflow occurs or when an incorrect system exclusive message is read. An overflow condition was never registered on the unit during these tests.

The other potential delays could only be evaluated when the audio patcher/mixer software had been constructed on the workstation and server side. Only then could the maximum patch level changes generated by a workstation be determined. Chapter 5 will now describe the implementation of the software which makes the sharing of remote studio resources a reality, and following this, potential transmission delays will be more fully evaluated.

Chapter 5

A Software Implementation Above the Hardware Configuration

Chapter 4 has described a particular hardware configuration which should enable the requirements discussed in Chapter 2 to be realized. Of course, software is required to marshall this hardware in such a way that it actually fulfills these requirements. This chapter describes the construction of the software from its analysis to its final implementation. Critical choices that were made in the course of the construction process are discussed and motivated. Particular emphasis is given to the process of analytical modelling. There were a number of iterations through the requirements specification and modelling steps. Modelling on paper is a far quicker process than coding a specification. The speed of the iterations, and clarity of the model were major factors contributing to the success of the project.

Before embarking on the process of modelling, a small test system was built to determine the feasibility of the project and to ensure the correct functioning of the hardware. This test system is described in the first section.

5.1 Feasibility Study.

The test system was written under the MS-DOS operating system, a non-multitasking operating system. This created a problem, in that the network software should run concurrently with commercial MIDI software such as a sequencer, event editor, universal librarian, or algorithmic composition software. In order to get around this limitation on concurrency, the network software was implemented as a 'Terminate and Stay Resident'(TSR) program. As their name implies, when TSR programs are run, they become resident within the run-time memory of the computer, where they stay after running to termination. They will run again at any time after becoming resident, if a certain condition arises in the computer. Usually, this condition takes the form of several keys pressed in unison. The TSR assumes control by switching control from the current running program to itself. After running to completion, it transfers control back to the interrupted program. The technicalities of TSR programming are well described in a book by Stevens [Stevens 1988]. The advantage of this technique in our case is that it allows a user to switch between the controlling network software and commercial sequencer software via a key-press combination.

The DOS-based network software comprises a booking system, an audio patching and mixing grid, a MIDI

patching facility, and a multitrack tape control screen. Only one of these subsystems can be active and displayed at any point in time. The facilities provided by the system satisfy many of the requirements described in Chapter 2. The booking system is not time-based, and booking is performed on a connect and release system. The server permits a user at a workstation to connect a resource to his workstation, but only if the resource is not currently connected to another workstation. When a workstation user has finished using the resource, he or she simply releases it. This makes the resource available to other workstation users. An example booking screen is given in Figure 5.1 below.

DEVICE	USER	BOOK	COMMENTS
DX 7	-		YAMAHA DX7 SYNTHESIZER
DX 21	WS 1	X	YAMAHA DX21 SYNTHESIZER
DX 11	WS 2		YAMAHA DX11 SYNTHESIZER
D 110	WS 1	X	ROLAND D110 SYNTHESIZER
CZ 1000	SERV		CASIO CZ1000 SYNTHESIZER
S 220	-		ROLAND S220 SAMPLER
IOTA	-		IOTA FADER
SPX 900	-		YAMAHA SPX900 EFFECTS PROCESSOR
DEP 5	-		ROLAND DEP5 EFFECTS PROCESSOR

Esc for Main Menu

Figure 5.1 - Booking Screen from Feasibility Study

Fig 5.2 below shows the Audio Patcher/Mixer screen. The gain and equalization settings are shown on the left and the patch points on the right. The screen scrolls, allowing the user access to all gain and equalization controls and to patch points. Although the gain and equalization controls are implemented, they do not function, as only the APM part of the APPM was physically constructed. Patch point edit functions permit the workstation users to connect audio sources to audio sinks, to then change the mix level at that patch point, and to disconnect a connected audio source and sink. In order to edit a patch point, a user must have booked both the audio device producing the audio and the audio device to which the patch point will route audio.

GAIN dB's	BASS BOOST /CUT dB's	MID BOOST /CUT dB's	TREB BOOST /CUT dB's			AUDIO RESOURCE OUT		PATCH POINTS AND MIX LEVELS dB's				
-6.0	+0.8	-1.2	-6.8			DX 7	A	-10.0		-20.0		
-6.0	+0.8	-1.2	-6.8			DX 7	B		-10.0		-20.0	
+4.8	0.0	-12.8	-12.8			DX 21	A	-10.0		-22.4	Muted	
+4.8	0.0	-12.8	-12.8			DX 21	B		-10.0	Muted	-22.4	
+2.0	0.0	0.0	0.0			DEP 5	L			-28.0		
+2.0	0.0	0.0	0.0			DEP 5	R				-28.0	
Enter to Edit Patch Point T to Select Gain/Equ. Process Esc for Main Menu						RESOURCE IN		DEP 5 L	DEP 5 R	WS 1 L	WS 1 R	WS 2 L

Figure 5.2 - Audio Patcher/Mixer Screen from Feasibility Study

When a workstation user books a MIDI device, the MIDI input of that device may be automatically coupled to the MIDI output of the workstation. This means that all MIDI devices booked by the workstation will automatically receive MIDI from that workstation. The workstation user may also need to receive MIDI from a booked device, such as synchronization and system exclusive information. The workstation user can patch any MIDI output from any booked MIDI device in the main studio to the workstation. The MIDI patcher user interface screen is shown in Figure 5.3 below.

DEVICE	MIDI OUT	MIDI IN	COMMENTS
DX 7		X	Transmit Channel - Receive Channels - 6, 7
DX 21	X	X	Transmit Channel - Receive Channels - 13
DX 11		X	Transmit Channel - Receive Channels - 10
D 110			Transmit Channel - Receive Channels - 2, 3, 4, 5(R)
CZ 1000		X	Transmit Channel - Receive Channels - 1
S 220			Transmit Channel - Receive Channels - 8, 9
IOTA			Transmit Channel - Receive Channels - 16
SPX 900			Transmit Channel - Receive Channels - 14
DEP 5	-		Device has no MIDI Out Receive Channels - 15

Figure 5.3 - MIDI Patcher Interface from the Feasibility Study

The test system also allows for workstation control over a Tascam 238 multitrack tape recorder. Once again, this control was provided to test the feasibility of such remote machine control, both in terms of technical feasibility and ease of use. The Tascam 238 has an RS232 interface and can be controlled via the serial port of a PC. Tascam provided a protocol for the control of all the functions of the 238. In the test system, the server is connected to the Tascam 238, and tape control messages are sent from the controlling workstation, via Ethernet, to the server. The server then transmits the relevant serial bytes to the 238. The user interface was designed to closely mirror the Tascam 238 control panel, and is shown in Figure 5.4 below.

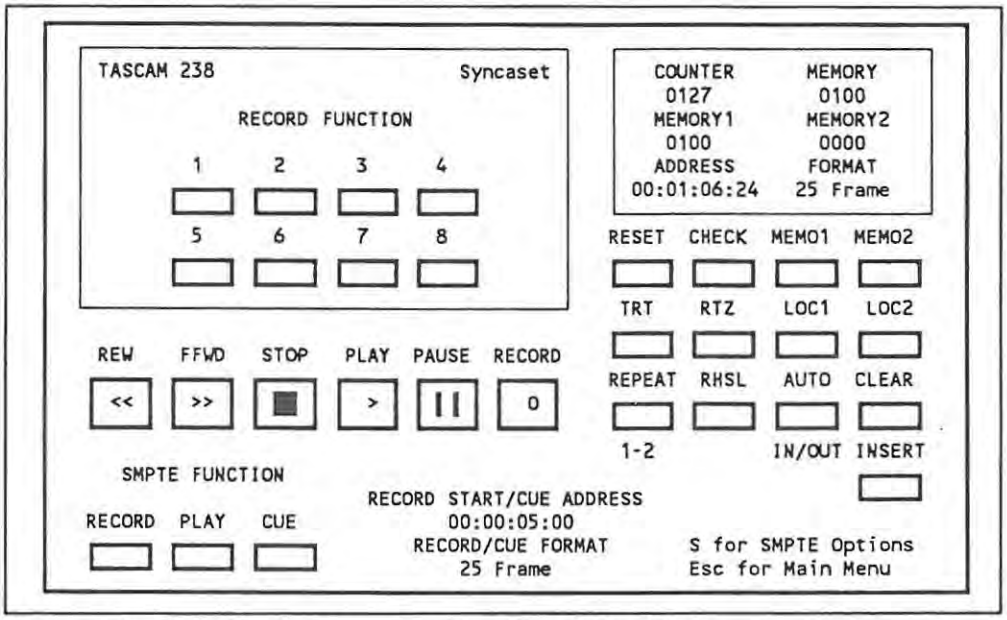

Figure 5.4 - Tape Control Interface from Feasibility Study

This type of machine control can be applied to video recorders and other tape machines which have remote machine control capabilities. There are two problems associated with supplying network access to remote audio and video recorders. The first problem is that of changing the tape. Each user must insert his or her own tape at the start of a session. This problem is not new in the field of shared remote access to devices. Users of shared printers must pick up their printouts, or organize for an operator to do the job. Typically, machine access will only need to take place at the start and end of a booked session.

The second problem is that of setting audio record levels on the recorders. The record level meters which indicate the record levels on the recorders are not visible to the remote workstation user. In the case of the Tascam 238, the record levels are not part of the protocol set. This problem can only be overcome by having a level meter at each workstation. The record level controls of the recorders should be set such that the level shown on the workstation meter gives a true indication of the audio level being recorded on the recorder.

This feasibility study was performed by MSc student Antony Wilks, and is more fully described in [Foss 1990]. It is interesting at this stage to consider Wilks' comments about the system [Wilks 1994]. On the positive side, the response at the workstation side was sufficiently fast to give users the impression that their commands were being executed immediately. As far as the audio patcher/mixer screen was concerned, the clear layout of the source and destination devices and their connections was far easier to comprehend than the block diagrams explaining the connectivity of conventional mixers. However, once comprehended, a conventional audio mixer with its arrays of buttons and sliders is far easier to operate than the mouse controlled patcher/mixer interface.

A major limitation of the MIDI patching interface is the lack of control over music resource channel numbers. These are manually set in the remote studio and documented in the system for the user to see. Even low-end synthesizers these days have multiple parts, each with their own channel assignments. With the sixteen channel limitation of MIDI, there are bound to be channel conflicts. The MIDI specification [IMA 1989] does not include a channel change message and the only option for a user is to send instrument-specific system exclusive messages to alter channels and mute parts to produce a desirable configuration. This modifies the configuration and places it in an unknown state for the next user. This issue of channel modification will be dealt with more fully in Chapter 6.

The TSR approach to multitasking was found to have too many limitations to be a viable option for further network development. These limitations stem from the fact that the TSR program cannot run in a time-shared manner alongside a sequencer and communicate with it. This fact prevents mix level automation. The lack of communication between the TSR program and the sequencer prevents MIDI data (in particular MIDI clock and timecode data), generated by the sequencer, from being used to time-tag mix requests and later play them back. More serious is the fact that the TSR program displaces the sequencer. MIDI bytes are still received by the interrupt handlers, but they quickly fill the input buffers, causing them to overflow.

A system could be configured where the sequencer runs on one machine, and the workstation network software on another. This configuration will be discussed in section 5.3. As a general point here, this approach solves the problem of concurrency, but increases the workstation cost and diffuses control, making the studio facilities more difficult to operate.

Another point that should be considered here, is that the trend these days is for sequencers to be provided with a graphical user interface. DOS-based sequencers, though still sold, are losing popularity.

Even before the initiation of the feasibility study, work had begun on the thorough modelling of the remote studio access system. Indeed, the rough design of the test system depended on these modelling efforts. In the

next section, the complete analysis and design of the remote studio access system will be described.

5.2 Analysis and Design of Remote Studio Access System

The remote studio access system can be classified as a real-time distributed system. Processing and information must be distributed across workstations and a server. The system is concerned with the communication of audio control information. If this information is not transmitted fast enough, the error will be perceived by a highly sensitive detector, the human ear. MIDI messages must both be transmitted within a particular critical bound - the criterion for a system being 'real-time' [Ward 1985]. Mix level requests and MIDI messages must both be transmitted between remote locations within a certain time interval.

With this in mind, it was appropriate that real-time analysis and design tools be used for the analysis and design process. At the time of choosing, there were two major contenders, the real-time tools of Hatley and Pirbhai [Hatley and Pirbhai, 1987], and those of Ward and Mellor [Ward 1985]. Ward and Mellor provided real-time extensions to the familiar Yourdon and Constantine approach to structured analysis and design [Yourdon 1979]. They also provided tools from the high level analysis all the way through to design. This toolset was thus chosen for the toolset project. This project had a number of complex entities and relationships which needed to be modelled and which could not be adequately described by the modelling tools of Ward and Mellor. The information model was constructed using the modelling tools of Sally Shlaer and Steven Mellor [Shlaer 1988].

Although Ward and Mellor state categorically that they have provided a toolset, not a methodology, they do provide a broad approach to system construction together with the toolset. At the kernel of their approach is the concept of modelling. Developers are advised not to rush into implementation, but to develop a model which can be reviewed and modified. Two models are built for a system, an essential model which is an implementation-free statement of what must be done, and an implementation model, which is derived from the essential model by adding information about the implementation technology.

The essential model consists of two sub-models, the environmental model which is a description of the environment in which the system operates, and the behavioural model which is a description of the required behaviour of the system. Both these models have two components. The environmental model comprises a context schema which describes the boundary between the system and its environment, and an event list, which describes all the events occurring in the environment to which the system must respond. The behavioural model comprises a transformation schema and a data schema. The transformation schema describes the transformations that the system makes in response to events, while the data schema describes the information which the system must have in order to respond appropriately to events.

5.2.1 The Modelling Process.

The modelling of the remote studio access system was not performed sequentially from essential model to implementation model. There were a number of iterations, and the iteration loop incorporated the specifications which were altered radically during the course of the modelling process. Modelling on paper provides a relatively fast way of building a complete system and assessing its practicality. The first model built was based around a very impractical specification, where the entire studio was used in a timesharing manner. Each user occupied all the resources of the studio in a particular booked time slot. Through building the model and reviewing the operation of the system, it quickly became clear that an altered specification where there was concurrent use of shared resources was more practical.

Buhr has pointed out the importance of 'playing' design diagrams while exploring solutions during the early stages of designing systems, particularly real-time distributed systems [Buhr 1993]. Buhr describes a new notation called 'time threads' which enable the understanding of behaviour patterns. While these time threads were not used in this system, the mix of control transforms [described below] and state transition diagrams had the same effect of highlighting behaviour patterns.

5.2.1.1 Essential Model

It is essential at the start of system construction to define the boundaries of the system, what is within the system and what is part of the environment with which the system communicates. This boundary is defined by the context schema. A context schema comprises a single circle representing a large transform. This transform denotes the functionality of the entire system. Data flows are drawn to the system transform from terminators. Terminators represent the things in the environment with which the system interacts, for example, users and audio patcher/mixers.

At this stage, there is no representation of the workstation/server configuration of the system. The system is viewed as one large black box which receives data from the environment, transforms it, and sends its own data back to the environment. Indeed, throughout the essential model, modelling is performed in terms of the subject matter of the system. There is no reference to the underlying technology, unless the technology itself relates closely to the subject matter. A good example here would be a MIDI connection to an audio patcher/mixer. The context schema for the remote studio access system is shown in Figure 5.5 below.

The number of data flows within the context schema is typically large, and there has been some grouping of flows in Figure 5.5. It is important at this stage to start documenting these flows within a data dictionary.

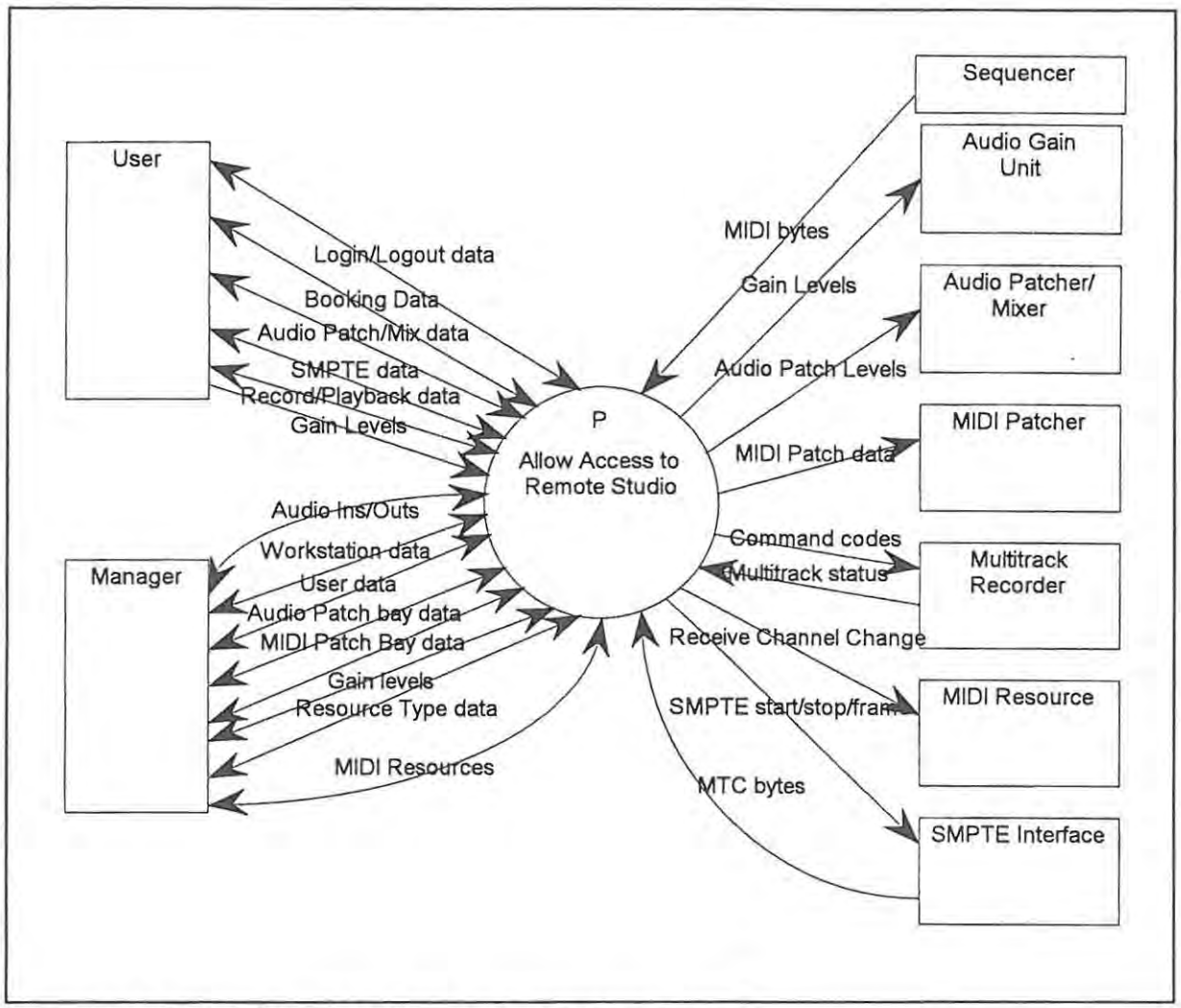

Figure 5.5 - Context Schema for Remote Studio Access System

Programmers have to refer to a data dictionary to find the intended meaning of the analyst's data flow names. Ward and Mellor provide guidelines for a syntax to be used within a data dictionary, and the data dictionary for the remote studio access system is given in Appendix E.

An events list is packaged together with the context schema to form the environmental model. An event as defined by Ward and Mellor has three characteristics. It occurs in the system's environment, it elicits a preplanned response, and it occurs at a specific point in time. Some events have data associated with them, such as when a user requests a new mix level at an audio patch point. Some are simply triggers, such as when a user starts playing back a recorded mix. The preplanned responses to the events are usually listed alongside the events list. The events and responses for the remote studio access system appear in Appendix D. As can be seen, these responses are in narrative form and are not a rigorous description of the behaviour of the system. This more rigorous description appears in the behavioural model.

In creating a behavioural model, the analyst uses a few simple but powerful modelling tools. Each response is represented as a transform circle and the transform can be either a data or control transform. Data transforms respond to events which have data associated with them, discrete data flows, and usually produce output data flows. Data transforms can read from and write to stores which are represented as two parallel lines with a descriptive name between the lines. Control transforms accept events which don't have data associated with them, and often produce event flows which start up or terminate data transforms. Data transforms and discrete data event flows are drawn with solid lines, while control transforms with their input and output event flows are shown with dotted lines. Control transforms with their associated event flows are essential for modelling the dynamics of real-time systems and are an important addition to the structured modelling tools of Yourdon and Constantine. The analyst uses these modelling tools to formally describe the responses to events occurring in the environment. Each of the event responses is modelled, and in so doing, a preliminary transformation schema is built up. Two of the transforms relating to audio patching and mixing are shown in Figure 5.6 below.

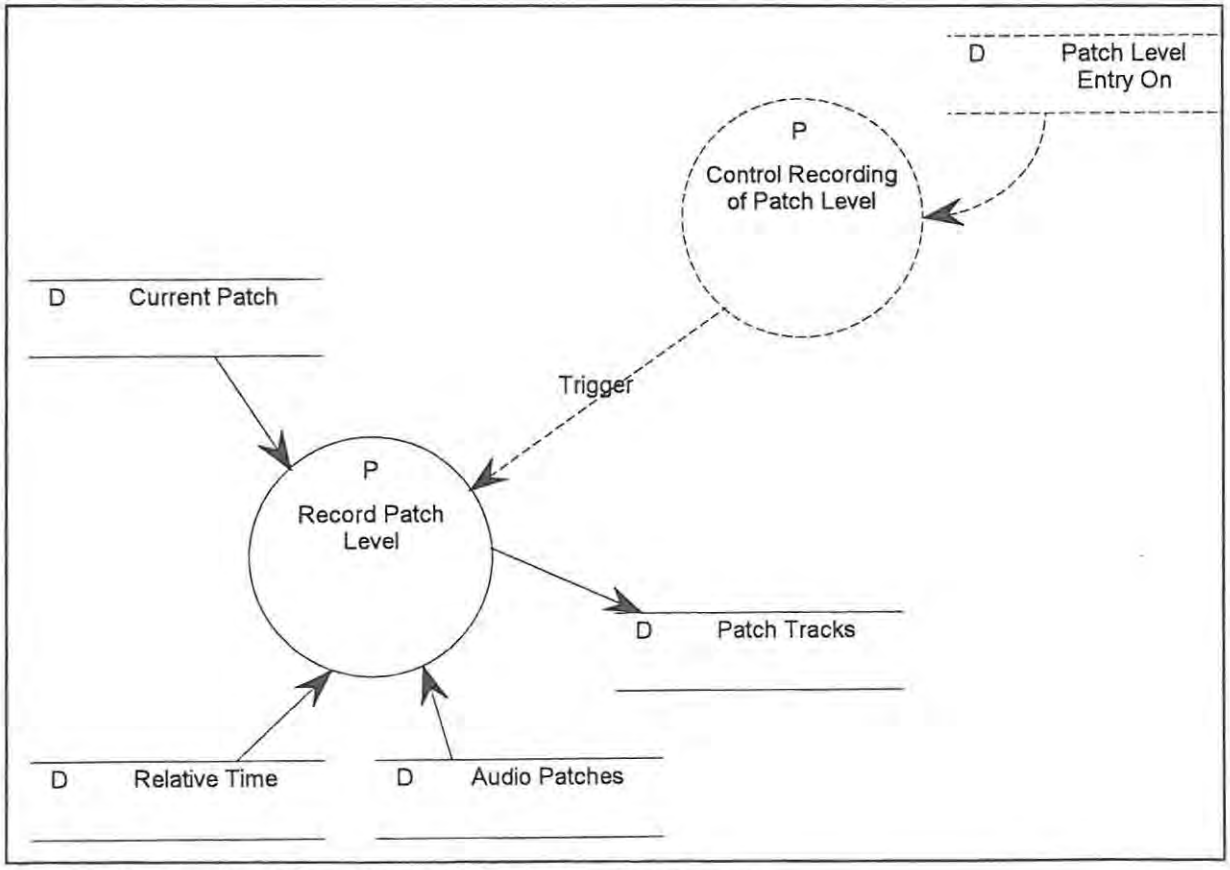

Figure 5.6 - Low level Transforms for Recording Patch levels

Figure 5.6 displays the interaction between a control transform and data transform to perform the process of recording a patch level, after the receipt of the next SMPTE frame in the form of MIDI Time Code messages.

The behavioural model comprises two components, the transformation schema and the data schema. Ward and Mellor suggest that there are two fundamentally different perspectives which one can take in describing a system. The active view pictures the system as a mechanism for transforming inputs into outputs. The passive view sees the system in terms of associations among real-world things which it must keep track of. The transformation schema reflects the active perspective, while the data schema reflects the passive one. Both views are important for gaining a complete understanding of any system, and both models are required for a sound implementation.

The data schema has been given more prominence in this project, and the information tools of Shlaer and Mellor [Shlaer 1988] have been used to more accurately model objects and their associations. The information model for the remote studio access system is given in Figure 5.7 below. Shlaer and Mellor's book is entitled 'Object Oriented Analysis - modelling the world in data', and it stresses the importance of finding the fundamental objects in the problem domain. The objects and associations in the diagram bear a close resemblance to the entities and relationships of the familiar entity relationship diagrams first proposed by Chen [Chen 1976].

At a first glance, the information model may be rather intimidating as there is a large amount of information contained within it. The rectangles are the objects, each object being divided into three layers. The topmost layer contains the name of the object, the second layer the attribute(s) which uniquely identify the object, and the third layer non-identifying attributes. The lines drawn between the objects indicate relationships between the objects, and the symbols at the end-points indicate the nature of the relationships. A diamond shape appearing midway on a relationship line indicates that there is an associative object associated with the relationship and that it contains information about the relationship.

A CASE tool will typically allow one to show or to hide various levels of detail within an information model such as the one in Figure 5.7.

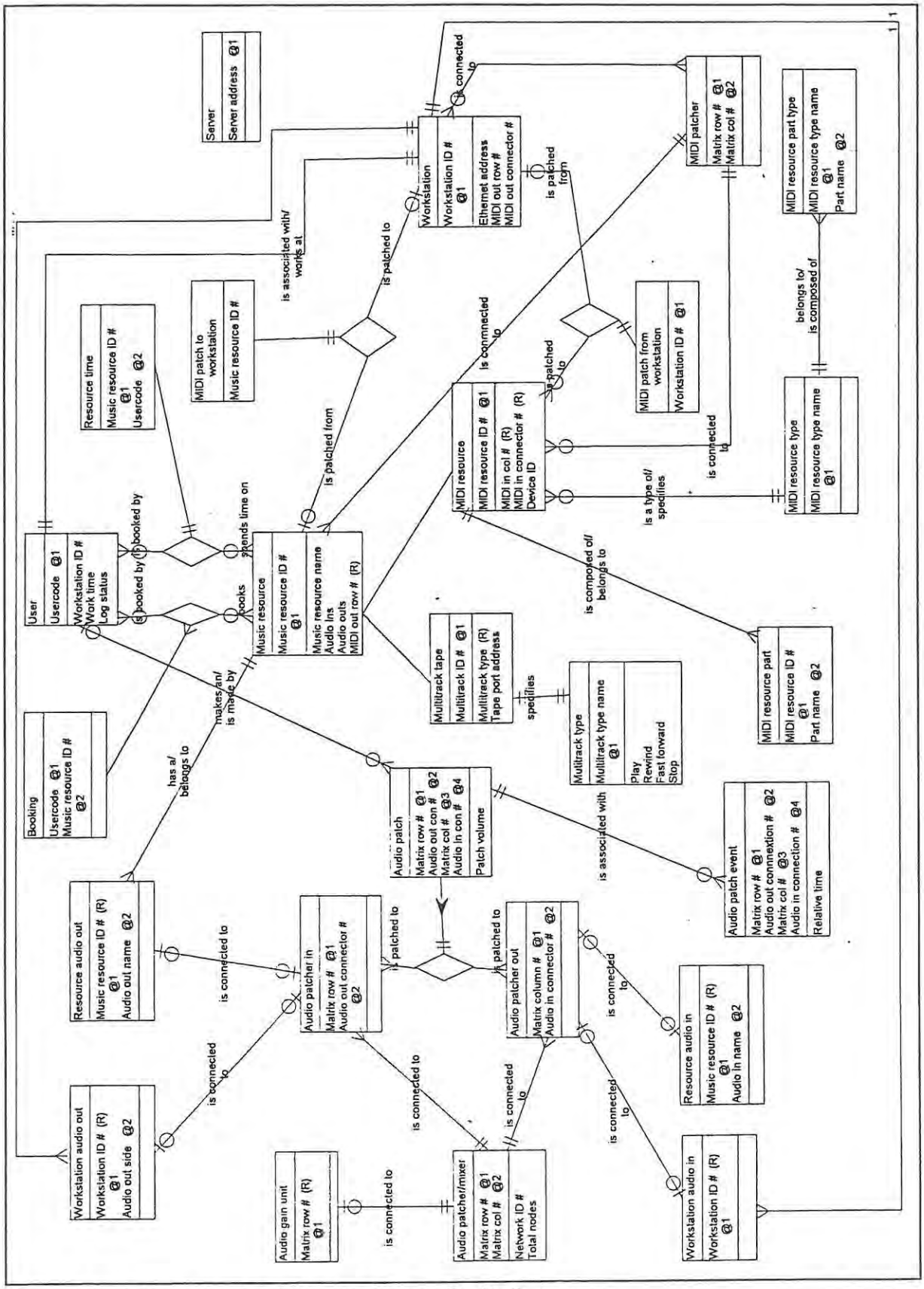

Figure 5.7 - Information Model for the Remote Studio System

Although the preliminary transformation is a very accurate and detailed description of the system's responses to external events, it is not an easy model to read. To facilitate presentation, Ward and Mellor suggest that the model be levelled. Levelling is a process whereby related transforms are clustered into groups, and then these groups further clustered. Each group is represented as a higher level transform with all the input and output flows of the component transforms. In this way, a hierarchy of transforms is developed, the process being known as upward levelling. Ward and Mellor provide a number of heuristics which can guide an analyst when doing the clustering. A very general criterion for a reasonable upper level grouping is the ability to assign a subject-matter-specific name to the grouping. There is a numbering scheme which allows for easy referencing of the various transforms within a hierarchy. Various levels of transforms for the audio patching component of the remote studio access system are given in Figure 5.8 below. These are shown in hierarchical form, where each box in the hierarchy represents an analysis screen and will comprise a number of transforms. At the top of the hierarchy is the context schema, and at the bottom, the transforms which were derived from the event-response table.

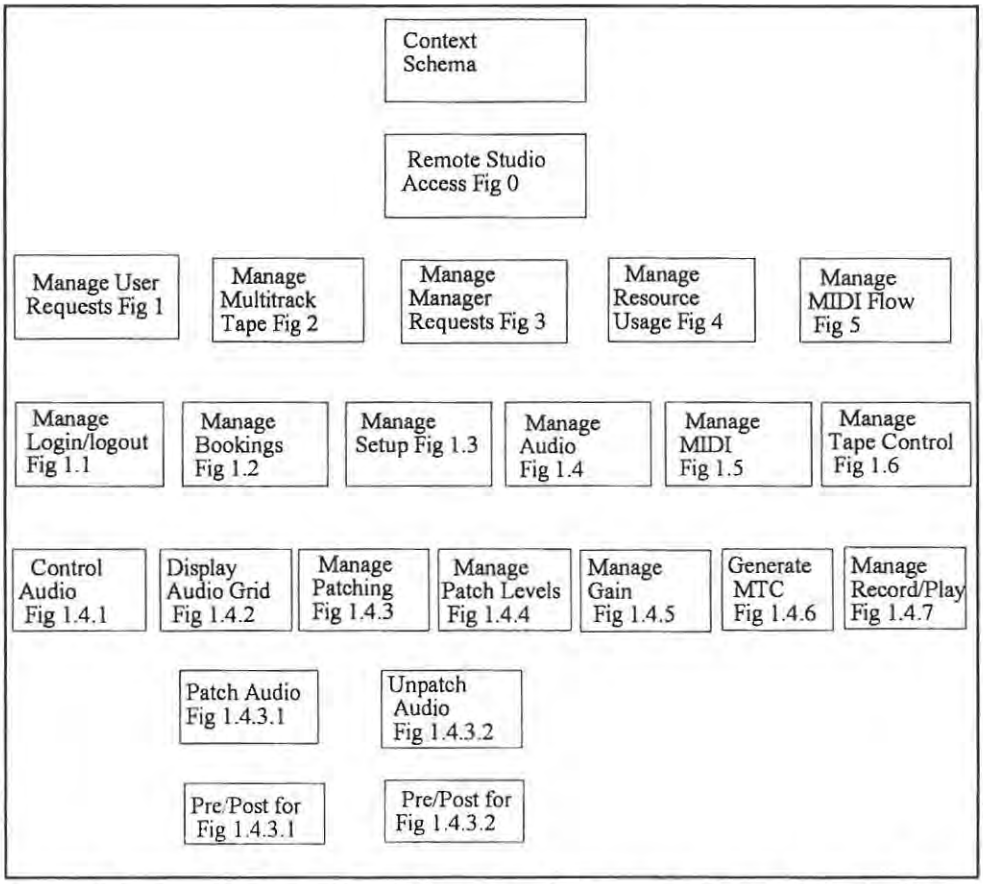

Figure 5.8 - Levelled Set of Transformation Schemas for Remote Studio Access

In addition to being unpresentable, the transforms derived from an event list are often incomplete. Very often a transformation in the preliminary transformation schema can be broken down into further, lower level transforms. Ward and Mellor provide guidance for doing this and stress the importance of not introducing implementation bias at this stage of the essential model. There were very few instances where a transformation from the remote studio access system required further break down. Of more significance were specification tools for the more accurate specification of control and data transforms. Pseudocode or pre/postconditons can be used to more accurately specify the data transforms. Ward and Mellor suggest pre/postconditions as they do not prescribe a particular algorithm for the transform, instead deferring this decision to implementation time when hardware constraints are better known. A mix of pre/postconditions and pseudocode was used to accurately specify transforms in the remote studio access system.

Control transforms are specified using state transition diagrams. These were found to be particularly useful when modelling the time-dependant interaction between events captured by the control transforms of the system. States are used to represent intervals over which some externally observable behaviour persists. Transitions represent points in time at which behaviour changes. Transitions between states are caused by events occurring. It is often useful to model state transition diagrams first, and then include their corresponding control transforms into the transformation schema. An example of a state transition diagram which accurately specifies the recording of patch levels is given in Figure 5.9 below.

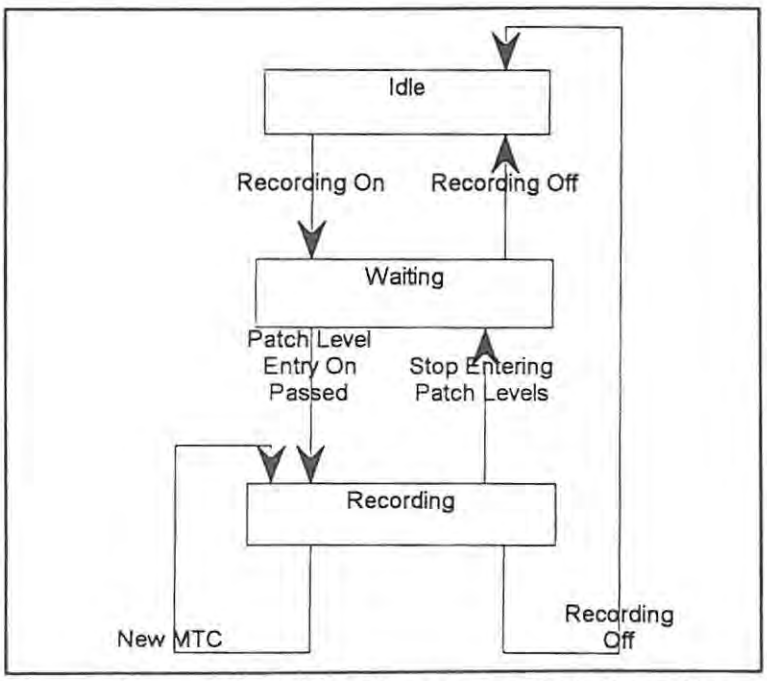

Figure 5.9 - State Transition Diagram for Patch Recording

5.2.2 Implementation Model

The essential model described above contains no implementation details. Eventually, of course, the model must be implemented on real hardware. The transforms of the essential model must be modified in such a way that they can be assigned to hardware components. Ward and Mellor provide guidelines for this process and describe it as 'implementation modelling'. The most important guideline is that the implementation model should distort the essential model as little as possible.

The first task of the modeller is to assign the essential model to one or more hardware processors, if the system hardware has more than one processor. In the case of the remote studio access system, there are two processor types - the workstation and server processor types. Allocation is performed in a top-down fashion. If a top-level transform can be assigned in totality to a processor, then it is tagged as belonging on that processor. Otherwise, the analyst descends to the next level in the hierarchy, and determines whether transforms at this level can be completely allocated to processors. This procedure is followed right down to the lowest level transforms. At this stage, the work of the transform must often be split across more than one processor.

This was the case for many of the low level transforms in the remote studio access system. A good example of this is the low-level audio patcher/mixer transform which, in its essential form, accepts audio patch/mix requests from a user and outputs patch/mix requests to a hardware audio patcher/mixer unit. The audio patch/mix requests are accepted from a user on a workstation. The workstation processes these requests and transmits them over Ethernet to the server. The server in its turn processes the requests and transmits them to the audio patcher/mixer using the hardware unit's message protocol. The processing of these requests is split between workstation and server, and the data needed to process the requests also needs to be split between the two processors. The data can be accessible to both processes via some sort of shared memory or distributed file system.

Distributed file systems (DFS's) are well-established, with the better known ones being Sun Microsystem's Network File System (NFS), AT&T's Remote File Sharing (RFS) and Transarc's Andrew File System (AFS) [Sanderson 1991]. The high level goal of these systems is to allow collaboration and data sharing. Comeau provides a good practical introduction to RFS and NFS [Comeau 1990], while Satyanarayanan provides a thorough overview of the development of the Andrew File System [Satyanarayanan 1990]. A distributed file system designed specifically for the IBM PC is described by Cheng and Sheu [Cheng 1991]. Fundamental to the file sharing mechanism is the Remote Procedure Call (RPC) which manages the request/response exchange between the client and server on a network. These RPC's depend on a more fundamental communication protocol such as TCP/IP. Whereas these distributed file systems are usually add-ons to

existing operating systems, some operating systems such as the Amoeba operating system are built on an assumption of shared data and processing resources [Tanenbaum 1990]. In all these distributed systems, the client-server model is most pervasive. A distributed system is organized as a number of distributed server processes which offer services to client processes across the network. This form of interaction can be viewed as a data-passing model, and it extends the underlying message passing communication mechanism of the operating system. This model is in contrast to the shared memory model which provides processes in a system with a shared address space. Approaches to providing such a shared address space are provided by Stumm and Zhou [Stumm 1990].

Ward and Mellor provide guidelines for modelling communication between the two components of a split transform. If a shared memory or distributed file system is available, the modelling is simple and both components access the same store. If not, then shared data must either be replicated or belong to one of the components. Guidelines are provided for modelling data store control mechanisms.

In the remote studio access system, low-level transforms associated with login, booking, audio patching/mixing, MIDI patching and machine control have to be split into workstation/server components. These mirror the client-server interactions of typical message-passing distributed systems, although the servers have quite varied roles including data and device control. The variety of client-server interaction and the choice of operating system, discussed in section 5.3, prevented the use of a generalized data sharing mechanism. All client-server interactions, including data sharing were modelled and later implemented. An example of an audio patcher transform split between workstation and server is given in Figure 5.10 below. Bal and Tanenbaum have pointed out that many implementations of shared data models are based on replication of data, and this has typically been the mechanism used in the remote studio access system modelling [Bal 1990].

Once the hierarchical transformation schema has been assigned top-down to available processors, a similar procedure must be carried out for tasks. A multitasking operating system running on top of a processor can perform a number of tasks. The operating system schedules processing of these tasks and switches between them, making their execution appear simultaneous. When transforms are derived in the essential model, no thought is given to the presence or lack of concurrency. However, when a group of transforms are assigned to a task, there can be no concurrency in transform execution. Ward and Mellor provide mechanisms for reorganizing control transforms grouped onto a task, such that they perform sequentially. Allocation to tasks was simple in the remote studio access system, and followed the categorization already described - booking, audio patching/mixing, MIDI patching, and machine control. All these activities run concurrently with

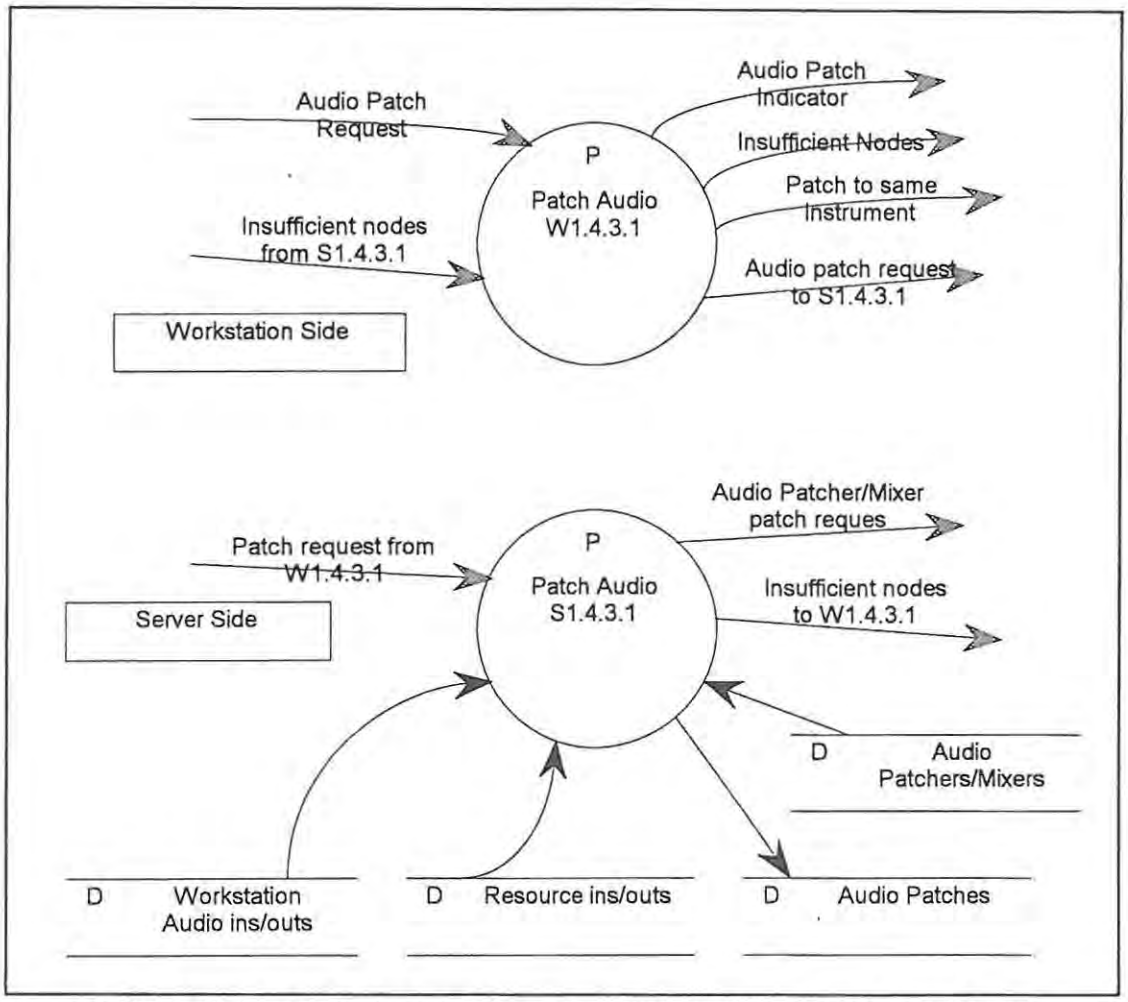

Figure 5.10 - Splitting the Audio Patch Transform Between Workstation and Server

commercial MIDI programs, a task assignment which is clear-cut and complete.

The data and control transforms developed at this stage of the modelling process have a networked structure. A network of transforms are joined by data and event flows. The more common structured implementation languages such as Pascal and C have a hierarchical structure based around the subroutine call mechanism. Ward and Mellor submit that a model which allows visualization and refinement of a subroutine hierarchy can be extremely important at this stage. They provide guidelines for converting the transform network into a structure chart, a hierarchy of modules connected via calls, and incorporating shared data areas. Yadav takes this even further, and develops a software design technique for organizing programs from a structure chart specification [Yadav 1990]. As will be described in the next section, implementation of the remote studio access system was performed under an operating system based on an event-response model of computing. The network of transforms, itself based on events and responses, matches this model closely, and the conversion to a structure chart was not performed. This approach has also been adopted by Robson and

Henderson [Robson 1993], where data flow diagrams are directly implemented by the concurrent language VAXELN Pascal.

All the diagrams in this chapter were produced using the CASE tool, System Architect [Popkin Software 1990]. In the analysis and design of the system models were drawn freehand and the data dictionary typed. Entry of diagrams into the CASE tool was found to be too slow for the entry and change of modelling diagrams. The CASE tool was used as a diagramming and notating tool. This does not exploit one of the major advantages of a CASE tool, the formal control of change. For example, modification of a transform should be reflected in a corresponding data dictionary modification. This role of CASE tools is more fully discussed by Lehman [Lehman 1991].

5.2.3 Alternative Analysis and Design Approaches

The Ward and Mellor toolset is one of the few that allows an analyst to model the dynamics of real-time systems accurately. However, it is not the only approach, and it is appropriate to look at alternative approaches with a view to enhancing the model and thus the clarity and maintainability of the eventual system.

Since the late 1980's the object oriented approach has had a strong influence on the way systems are analysed and designed. It has taken longer to establish itself in the area of real-time system analysis and design, simply because the dynamics of real-time systems require specialized modelling tools. The essence of the object oriented approach is the encapsulation of the information and behavioural aspects of a system into a number of communicating objects. These objects can inherit both information and behavioural characteristics from higher level objects. Shlaer and Mellor describe these higher level or parent objects as supertypes, and the lower level, inheriting objects as subtypes [Shlaer 1988].

Shlaer and Mellor have extended the object-oriented information modelling approach found in their first book into state and process modelling. A state model provides a life cycle model for each object in the information model, while the process model depicts the actions associated with each state in the state model. Fayad, Hawn, Roberts and Klatt have described the successful application of the Shlaer and Mellor approach in the area of mission planning [Fayad 1993]. Two other popular variations on the object oriented approach are described in Yourdon and Coad's three volumes of texts [Coad 1991] and the second edition of Grady Booch's popular book [Booch 1995]. A key motivating feature of these object oriented approaches is the direct mapping that is possible between the analysis, design, and implementation phases. Object oriented languages such as C++ have a structure which is based around the fundamental object oriented concepts of encapsulation, inheritance, polymorphism and message passing. There is no conceptual gap between the

object oriented analysis and design model and the program constructs. Nerson stresses the advantage of this in his description of an application of object oriented analysis and design. The same ideas and concepts were manipulated from the requirements phase down to the implementation phase [Nerson 1992].

Even though such arguments for the object oriented approach appear formidable, it is not clear that the object oriented approach is always preferable. Yamashiro has provided a comparison of object oriented analysis and real-time structured analysis approaches and found the analysis results of real time structured analysis to be easier to understand than those of an object oriented approach [Yamashiro 1993]. Alabiso confirms this, stressing the usefulness of data flow diagrams for expressing the flow of data to and from loci of functionality, and proposes a method to convert an analysis flow model into an object oriented model [Alabiso 1988]. Ward, aware of the popularity of object oriented languages, has shown a way of integrating object orientation with structured analysis and design [Ward 1989]. Objects can be represented in structured analysis and design by grouping the lower level transformations and stored data into higher level transformations. Inheritance can be incorporated into structured analysis/design by embedding operations and data for high level objects in the transformations representing lower level objects. Kerth has proposed a structured approach to object oriented design, where the problem and the solution to that problem are analysed with a number of complimentary modelling techniques [Kerth 1991]. Kelly and Sherif confirm this approach, suggesting that the various methods be considered as a collection of organizational tools which can be blended and customized to an individual project.

These observations are well borne out by the analysis and design approach used for the remote studio access system. Shlaer and Mellor's object oriented approach was used for the information modelling aspects of the system, while Ward and Mellor tools were used for the behavioural aspects. The model lays the groundwork for any further object oriented design and implementation efforts, providing a clear specification and dynamic description of the system.

5.3 The Choice of an Operating System

An operating system has a large influence on the implementation of a system, and the choice of an operating system is an important one. An operating system determines the nature of the mapping from design model to code. The remote studio access system is a real-time distributed system and has been modelled using real-time analysis tools. The essential model comprises a number of concurrent transforms which would be most naturally mapped onto multiple tasks created under a multitasking operating system. It must be said at the outset that the only operating systems considered were those which could run on an IBM PC platform. This constraint was simply due to the availability of the hardware. This ready availability extends to the music industry more generally, and has resulted in the IBM PC being a commonly used platform for music

applications [IMA 1992].

The feasibility study described at the start of this chapter, was implemented under DOS, and gained a measure of multitasking from the Terminate and Stay Resident mechanism. There are other, truly multitasking, alternatives to using the TSR mechanism. Kryszak has described the use of Multiuser DOS (DRMDOS), from Digital Research for a factory control system application [Kryszak 1992]. At Rhodes University, extensive work has been done on the PC-Xinu multitasking operating system originally developed by Comer and Fossum [Comer 1988]. The operating system has been converted into a resident form [Foss 1992], and has had a Windows-like event-driven programming interface added to it [Rehmet 1993]. Both these enhancements were added with a view to providing an implementation environment for the remote studio access system.

The full source code of PC-Xinu is available, and all aspects of the operating system can be modified. The advantage of this is that the operating system can be more finely tuned, particularly for the real-time aspects of the application. Orlarey has described an efficient scheduling algorithm for real-time musical systems, which could be incorporated into the PC-Xinu operating system [Orlarey 1990]. Stankovic has confirmed the usefulness of this approach, describing the inability of current real-time operating systems to handle hard deadlines [Stankovic 1991]. Berryman and Sommerville describe the scheduler as the central component influencing the timing behaviour of any real-time system [Berryman 1992].

However, the disadvantage of the PC-Xinu approach is that any other studio control software, such as a sequencer, must either be developed under PC-Xinu, or must run on a separate station. There are some advantages to having commercial studio software running on a separate machine from the remote studio access software. There is no contention for machine processing resources, and any commercial hardware and software can be utilized for studio control. This separation of studio access and studio control software and hardware is described more fully in [Foss 1992]. A hardware configuration which allows this separation is shown in Figure 5.11 below.

The remote studio access workstation sits between the commercial station and the remote studio. MIDI is passed between the two stations, and this allows the access workstation to filter MIDI bytes. More significantly, it can pick up any MIDI Time Code bytes and thereby synchronize to the same clock as the commercial station.

If, as in this configuration, there is no need for co-residence of commercial software and remote studio access

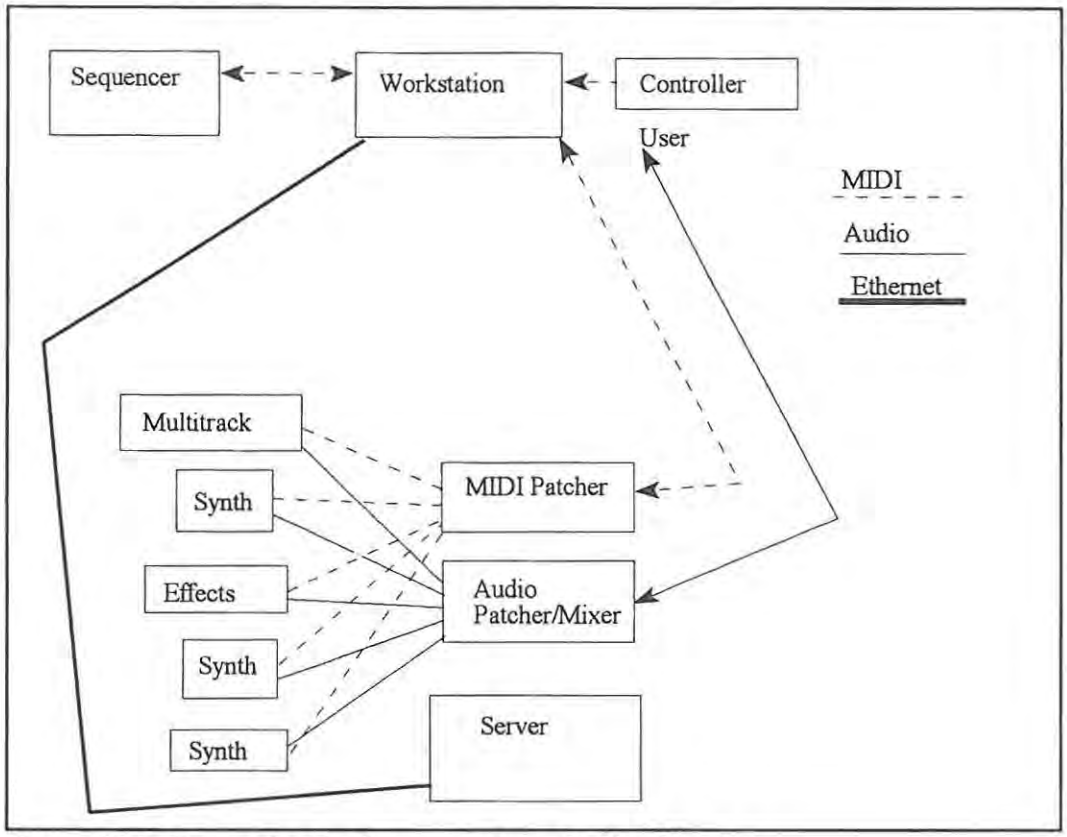

Figure 5.11 - Separating the Sequencer from the Workstation

software, a greater range of operating system choices become available. Wells has provided a comparative study of four commercial operating systems for IBM PC-compatible microcomputers, namely UNIX, OS/2, QNX and FlexOS [Wells 1993]. The study evaluates the predictability and performance of the operating systems in several areas critical to real-time systems. OS/2, QNX and FlexOS all provided the kind of performance required of real-time operating systems, and could be used as for the implementation of the remote studio access system. The lack of real-time characteristics in UNIX is confirmed by Fichera [Fichera 1991].

The argument for separation of remote studio access and studio control processes onto discrete machines is certainly attractive. However, there are also strong arguments against the approach. The remote studio access system was created to share resources, partly for logistical reasons but also for financial savings. It makes better financial sense to have a single all-purpose workstation. Control over a single workstation, where different functions reside in distinct windows on a screen, is faster and easier than switching between two distinct control stations.

Once the decision to have a single workstation was taken, the choice of operating system was constrained by the availability of commercial studio control software. By far the most popular operating system platform

for commercial studio control software is Microsoft Windows. Sequencers such as Cakewalk from Twelve Tone Systems, Orchestrator Plus from Voyetra, Powertracks Pro from PG Music, Mastertracks Pro from Passport Designs, and MaxPak from Big Noise Software, represent the most popular IBM PC sequencers, and run exclusively under the Windows operating system. The same can be said of editors and librarians such as the Music Quest editor from Sound Quest and UniEdit from Mark of the Unicorn software.

The user interfaces for these software tools appear in the form of graphical windows, generated with the help of the Windows operating system. Most sequencers provide a number of possible window interfaces, each window providing the user with different types and levels of control over the components of a MIDI studio. There will typically be a track window, allowing for track by track recording, and including channel and patch selection. A measure control window will allow manipulation of recorded music at the level of musical bars. Notation and piano roll windows allow for note manipulation within individual bars. Most sequencers also provide a mixer window which emulates a real mixer with graphic fader buttons, but which generates MIDI controller messages. These windows can all be open at the same time, allowing a user to switch quickly from one level or type of control to another. Users have become familiar with this mode of operation.

The remote studio access software has been written under the Windows 3.1 operating system, both at the server and remote workstation side. At the time of implementation its upgrade, Windows 95, had not been released. Like the MIDI control software, the workstation functionality is encapsulated in a number of graphical windows generated by the Windows operating system. A user can move between the windows which provide remote studio access and those which provide standard MIDI control. In a typical single user MIDI studio, a user will move between a mouse-controlled user interface on a computer, a mixing desk, an audio patch bay, and a MIDI patch bay. The remote studio workstation has moved the patching and mixing facilities into windows which the user can move between. With the advent of MIDI-controllable MIDI patch bays, audio patch bays and mixers, this multi-faceted mode of operation is possible and becoming more prevalent in single user studios. A further feature of the remote studio interface is the presence of a booking sheet window, necessary for the shared resource environment.

While the multi-window mode of operation appears useful and familiar, a question which must be asked is whether the Windows 3.1 operating system will allow for real-time control over multiple processes from multiple windows. More than one task can run concurrently under the Windows 3.1 operating system, but multitasking is non-preemptive. The operating system does not assign slices of time to each task and switch tasks when an allocated time-slice is complete. Each program is written in the form of a loop, where the loop consists of reading a message from a message queue, and then processing that message. The messages usually originate from user actions, such as dragging a scroll bar, or selecting an option via a graphic symbol. The message will be dealt with by a procedure associated with the window which the user interacted with. These

window procedures define the tasks running under the operating system. A window procedure can utilize the processor exclusively, as there is no timer-based preemption mechanism. Task switching can only occur at the point when a new message is retrieved from the message queue, and it is the responsibility of each window procedure to cooperate with other tasks and make these message retrievals within reasonable intervals. Interrupts can result in task switching if the interrupt is tied to a user defined call-back procedure (effectively an interrupt handler), but usually an interrupt is handled by the operating system and simply results in an addition of data into the message queue. Further details regarding the architecture of the Windows operating system and the structure of programs running under it, can be found in [Petzold 1992].

In the remote studio access system, there are two levels of multitasking. There is the obvious concurrent operation of commercial studio software and remote studio access software. Within both these application programs, there are multiple windows with their associated tasks. The most crucial real-time constraints lie within the commercial sequencer, where the intervals between successive notes must be maintained in order to retain the congruence of the musical piece. Ideally, this process of note output should be assigned a high priority, and the operating system would then ensure the timely delivery of notes. This mix of prioritization and preemptive multitasking is possible under the Windows 95 operating system, and could be obtained by running a sequencer program at a higher priority than the remote studio access program. If Windows '95 is to be used in the future, its scheduling and context switching times would have to be measured to see whether they fall well below the upper limit of MIDI message delivery time. Under Windows 3.1, with its cooperative multitasking, it is important that the remote studio access software provide adequately short intervals between potential task switch points. The mix level changes, which originate from the audio patcher mixer component of the studio access system, should also be processed in a timely fashion, but do not have the same critical time constraints as the sequencer events. Timing issues surrounding the remote studio access system will be discussed in more detail in section 5.8.

A major advantage of selecting the Windows operating system is the close correspondence between Ward and Mellor's event-driven approach to analysis and the event-driven orientation of the operating system. An events and responses list provides the source for a Ward and Mellor transformation schema. All processing in Windows is based on extracting messages from a message queue. These messages arise from events occurring in the environment of the system. This correspondence allowed for the transforms of the transformation schema to be implemented directly as procedures triggered by operating system messages.

5.4 Network Communication in the Remote Studio Access System

The hardware configuration of the remote studio access system comprises a number of workstations connected to a remote server via Ethernet. Login, booking, MIDI patching, audio patching/mixing, and

recorder control messages must all flow from workstations to server. Having decided on a hardware communications platform to transmit these messages, a software protocol had to be chosen for their transfer. Typically, communications protocol software is written in several layers, each layer dealing with various aspects of communication. There are two dominant approaches to protocol layering. The first is that of the ISO reference model of open system communication. The ISO system comprises seven protocol layers, the physical, data link, network, transport, session, presentation and application layers. Successive layers provide different forms of transmission reliability. For example, the data link layer includes timeout and retransmission algorithms at the frame level between two communicating machines. The network layer provides error detection and recovery for packets transferred onto the network. The session layer provides end to end reliability. This level of error detection and recovery across all the layers is not present in TCP/IP, the other dominant approach.

TCP/IP software is based on four conceptual layers which build on a fifth layer of hardware. The four layers are the network interface, Internet, transport and application layers. The transport layer ensures end-to-end communication of packets, which it does by acknowledging packets sent from a source to a destination port, after first establishing the connection. The underlying Internet layer is only responsible for routing packets, there are no reliability checks. At the application layer, an application transmits data to a destination port. A TCP/IP port comprises an Internet address and a port number. The transport layer adds header data to this application data, creates a packet and passes the packet to the Internet layer. The Internet layer adds its own header data, including the source and destination Internet addresses and finally passes this on to the network interface layer. The network interface layer comprises a physical hardware driver. In the case of an Ethernet driver, the Internet packet will be bundled into the data part of an Ethernet packet. An Ethernet packet comprises source and destination addresses for Ethernet communications hardware on physical machines. The Internet layer is responsible for providing a destination physical address, which it does by a process of address resolution. The ISO layering scheme is well described in [Tanenbaum 1989], while Comer provides comprehensive details of the TCP/IP layers [Comer 1988]. Diagrams of the ISO and TCP/IP layers are given in Figure 5.12(a) below.

The ISO scheme is often used as a standard by which other protocol layering schemes are compared. Leslie Tyler has based his description of the AES-24 protocol for sound system control on the ISO reference model [Tyler 1994]. However, not many protocol stacks implement the full functionality of the various levels of the ISO model. In contrast, there are numerous implementations of the TCP/IP protocol on a host of different machines. This is not remarkable, considering that the TCP/IP protocol is the basis for all Internet communication [Comer 1991]. TCP/IP implementations are available for both Windows and DOS operating systems on the IBM PC.

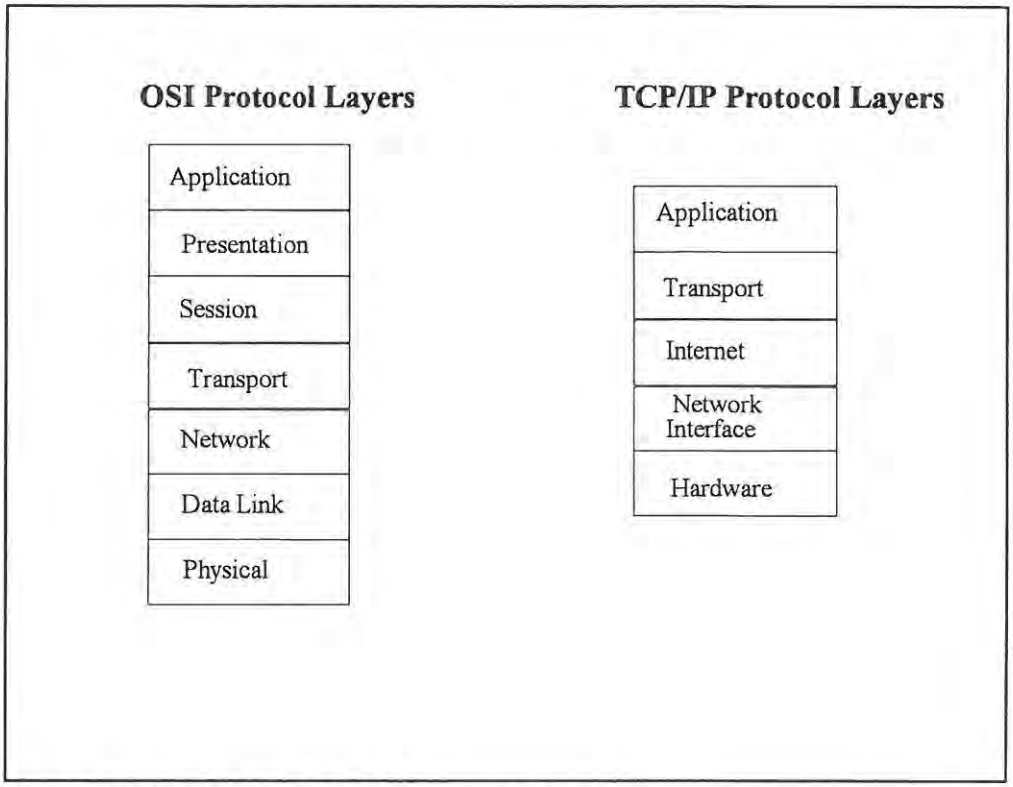

Figure 5.12(a) - OSI and TCP/IP Protocol Layers

In the remote studio access system, data delivery speed is a fundamental issue. Mix levels have to be transmitted from source to destination in real-time, that is, there is a bound on the time for the delivery of this level information. This real-time requirement is even more pronounced when, as described in Chapter 6, MIDI data is passed across the network. The TCP/IP protocol suite has taken account of this real-time requirement by providing an alternative to the full connectivity in the transport layer. This alternative is the User Datagram Protocol, or UDP protocol. UDP also allows an application program on one machine to communicate with an application on another machine, and also uses a port mechanism to transfer data, using the underlying Internet protocol for routing. However, the protocol is connectionless, providing no means of acknowledgement, and therefore allows for faster transmission times. This protocol has been extensively used in the context of real-time systems, and more particularly, real-time sound systems. Freed has described the use of UDP for the efficient transfer of performance data from outlying workstations to a real-time synthesis engine [Freed 1992]. Crane and his team at the sound system company, QSC, have used UDP for the transfer of amplifier control information in sound reinforcement applications [Crane 1995].

The motivation for adopting the UDP protocol in the above two cases, depended partially on the better data transfer speeds, but also on the fact that routing is incorporated in the protocol. Because UDP lies in a layer above the Internet protocol, messages can be routed via gateways to hosts on other networks. There is an overhead associated with this facility, both in terms of packet size and speed of packet delivery. Twenty four

bytes of Internet protocol header data appear directly after the frame header, and a further eight bytes of UDP header data appear after the Internet header data. Processing time is required to assemble the packet at the source, as it passes down the protocol stack, and to disassemble it at the destination, as it passes up the stack. The Internet address needs to be resolved to the actual physical address of a host, if the destination lies on the same physical network, or of a gateway, if the host is on a remote network.

In the case of the current version of the remote studio access system, workstations and server reside on the same physical network, this network being isolated from other data traffic. The internetworking routing capability, was not required, and in the interests of better real-time performance, it was decided to strip out the UDP and Internet layers and simply pack the studio access messages in the data section of Ethernet frames. Initially, drivers were written specifically for 3-Com Ethernet cards [Haliburton 1988], however it was later decided to incorporate hardware independence, and transmit Ethernet packets according to the FTP packet driver specification. This specification was originally written in 1986 to allow the company FTP to have its TCP/IP software co-exist with a proprietary LAN system. Drivers which comply with this specification have now been written for almost all IBM PC Ethernet interface cards [Romkey 1991]. From the application's point of view, the functionality of this approach is much the same as if the Internet protocol were used. Instead of Internet addresses, Ethernet addresses are used to transmit messages to host machines. The protocol is connectionless, and any reliability must be built in at the application layer.

The packet driver interface revolves around a software interrupt, which network applications call to perform a variety of functions: getting information about the network interface, transmitting packets, and registering to receive packets. A problem with the packet driver interface is that it was written for the DOS operating system, and DOS uses the functionality of the 8086 processor. The Windows operating system rests on top of DOS, but uses the extended memory capabilities of the 80386 and later Intel processes. An interface, known as DPMI (DOS Protected Mode Interface) has been written to allow Windows programs to cross the boundary into DOS and utilize facilities in DOS, such as software interrupt calls [Glass 1990]. This interface was used to write a packet driver for the Windows operating system, and was subsequently used for all remote studio access communication.

The TCP and UDP protocols which reside above the Internet protocol layer, both incorporate the concept of a port. A host machine can have a number of ports, each application program, or even processes within a program, having their own communication ports. To provide this functionality within the remote studio access system, a small message dispatcher has been built for both the workstation and server. A simple protocol, implemented as two bytes within the Ethernet data frame, has been used to direct packet data to the relevant process within either the workstation or server. The nature of the protocol is given in Figures 5.12(b) below.

Description of Message	Two Byte Code
Workstation Startup	1 0
Usercode entry	2 0
Logout	2 1
Validate Login	2 2
Already Logged in	2 3
Login Valid	2 4
Login invalid	2 5
Booking Sheet request	3 0
Booking Sheet arrival	3 1
Resources List request	3 2
Booking request	3 3
Unbooking request	3 4
Audio In list request	4 1
Audio Out list request	4 2
Audio Patcher/Mixer configuration request	4 3
Make patch	4 11
Free patch	4 12
Change patch	4 13

Figure 5.12(b) - Protocols for Workstation-Server Interaction

5.5 MIDI Transmission and Receipt in the Remote Studio Access System

MIDI data has to be transmitted by both the server and remote workstations. The server needs to transmit MIDI system exclusive messages to the Audio Patcher/Mixer and the MIDI patcher. The workstation's sequencer transmits the full range of MIDI messages to the MIDI patch bay, for onward transmission to studio devices. Microsoft's Windows multimedia extensions provide a standard interface to MIDI hardware [Microsoft 1991]. This interface comprises a set of well-documented function calls to open a device, transmit MIDI messages and close the device again. MIDI hardware manufacturers provide software device drivers which adhere to this standard.

The multimedia extensions standard also specifies a set of MIDI messages which can be placed in an application's message queue, when MIDI data is received by a hardware interface. An alternative is for the application to provide a callback function which is called immediately upon receipt of a MIDI message. This approximates an interrupt handler and allows for faster MIDI processing.

Commercial MIDI software and remote studio access software have to co-reside at the workstation side. More importantly, both must synchronize to the same clock. This will allow MIDI and digital audio sequences to be synchronized with audio mix level changes. The most popular MIDI synchronizing signal is MIDI Time

Code, which is analogous to SMPTE time code. It is not common for commercial sequencers to generate MIDI Time Code, and thus the synchronization signal must come from another source on the workstation, or from an external source such as a multitrack recorder. Ideally, one or more programs on the workstation should be able to read the same MIDI signal, or one program should be able to route MIDI messages to another program.

This facility has been provided for some time by Apple's MIDI Manager [IMA 1989], where applications address logical ports. Thus, one application can transmit MIDI messages to a logical port, while a second application can read these messages from the same logical port. MIDIShare for the Apple Macintosh also allows for this dynamic connection between MIDI applications through internal links [Orlarey 1990].

An Apple MIDI Manager type of facility does not form part of the Multimedia extensions of Windows. Some MIDI drivers allow more than one MIDI application to attach to, and read MIDI data from them. An example is PG Music's Multi-MPU driver [PG-Music 1994]. This driver allows both commercial MIDI sequencer software, and the workstation audio patcher/mixer software, to read synchronization messages coming from an external recorder into the workstation. Also, 'virtual' MIDI drivers exist which effectively provide logical ports. MIDI software can read from and write to such a driver, allowing for inter-application MIDI message transfer. Turtle Beach provide such a driver with their Quad Studio software [Turtle Beach 1994]. This allows the digital multitrack Quad Studio to act as a synchronization master for both a commercial MIDI sequencer and the audio patcher/mixer software.

5.6 The Implementation of the Remote Studio Access System

Once all the system support structures were in place, the information and implementation model could be realized in C code. After the Windows operating system had been chosen, C was an obvious implementation language, since the majority of Windows based programming texts are based on the C language. Borland C's system development tools were used for the implementation, since at the time of choosing they were the only ones which allowed for development in a Windows environment [Borland C 1991]. The upper level transforms, arising from the process of analysis and design, will be used to provide a structure for the discussion of the implementation. This hierarchical structure also provides a framework for the understanding of the system. Directly below the context schema in the analysis hierarchy, the remote studio access system is partitioned into six major transforms. These transforms are shown in Figure 5.13 below.

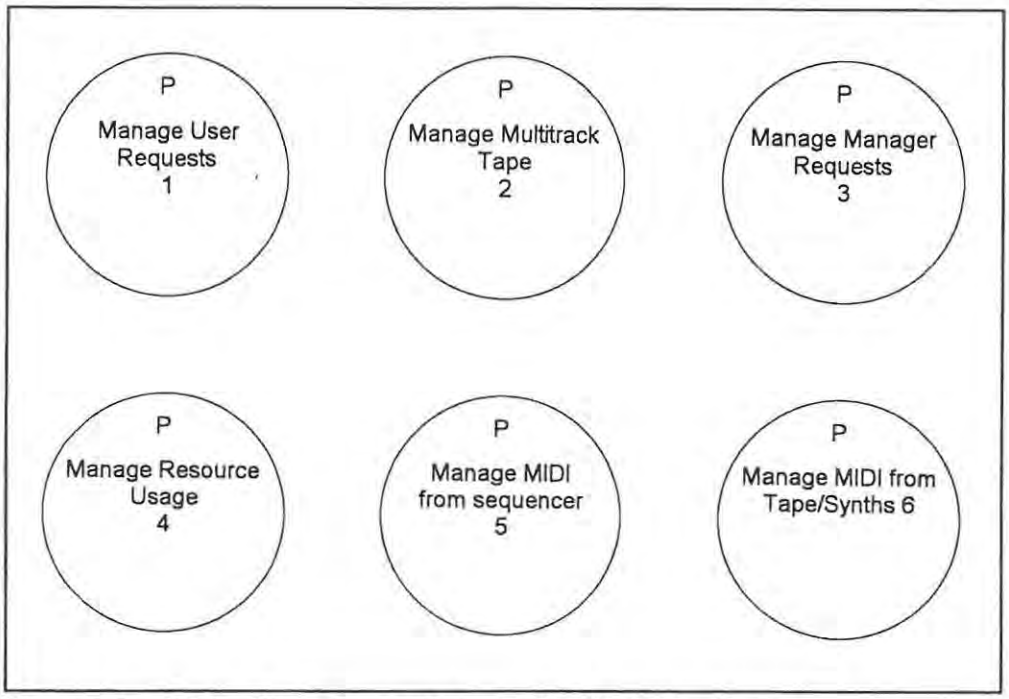

Figure 5.13 - Major Transforms of Remote Studio Access System

The stores and data flows which should appear on the diagram have not been included for the sake of clarity. The two major transforms are those labelled 'Manage User Requests' and 'Manage Manager Requests'. The 'Manage User Requests' transform is partitioned across workstations and server, while the 'Manage Manager Requests' is allocated to the server alone. It was decided to have all configuration data entry, a task of the system manager, performed via the server. The manager must have contact with the various remote studio resources and the patchers to perform configuration, and this approach therefore seemed the simplest and most logical.

5.6.1 User Request Management

The 'Manage User Requests' transform will be dealt with first, as it provides the best insight into the remote studio operation. Investigation of the other transforms will then tie up any loose ends, for a complete understanding of the system. Moving one step down in the hierarchy, the 'Manage User Requests' transform is further subdivided into six transforms. These transforms are shown in Figure 5.14 below, once again without the complication of stores and data flows.

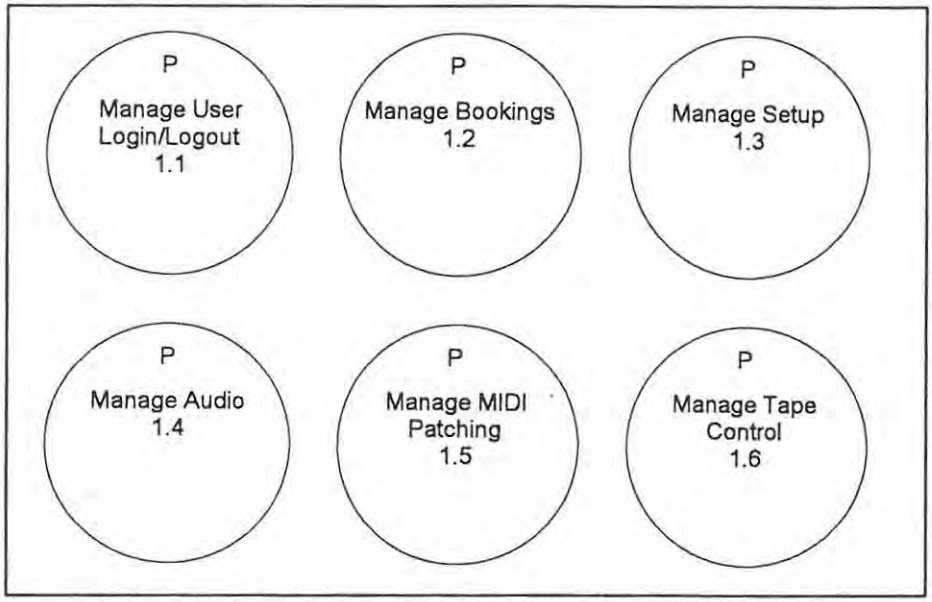

Figure 5.14 - The Transforms which deal with User Requests

The human interface to these broad functional components is provided via a simple row of buttons which can reside above a commercial sequencer, as shown in Figure 5.15 below.

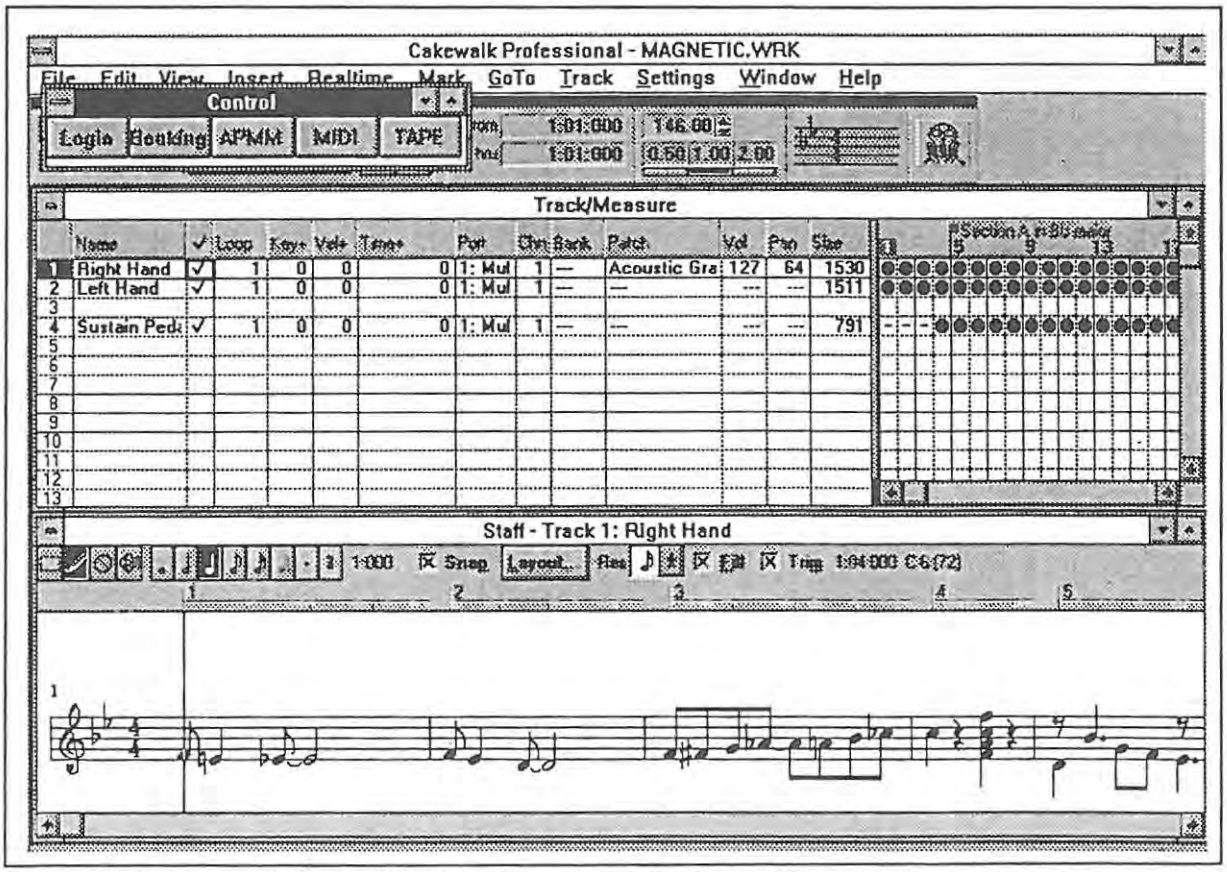

Figure 5.15 - Button Selection Panel for Remote Studio Access Components

By clicking on one of the buttons, a window relating to the transform and allowing relevant data entry will pop up. By clicking the same button again, the window will disappear. The control interface is designed to intrude as little as possible into the sequencer interface.

5.6.1.1 Logging and logging out

The first button in the row is the login/logout button. The transform associated with this button is given in Figure 5.16 below. Before the row of buttons appears on the screen, a workstation establishes contact with the server and requests its workstation identifier. Each workstation has an identifier, which allows for less cumbersome identification than its Ethernet address. When the login button is pressed, a user is requested to enter a usercode via a dialog box. The usercode uniquely identifies the user for studio usage logging, and for the retrieval of studio setups. The usercode is passed to the server for verification. The studio manager is responsible for the assignment of workstation identifiers and usercodes.

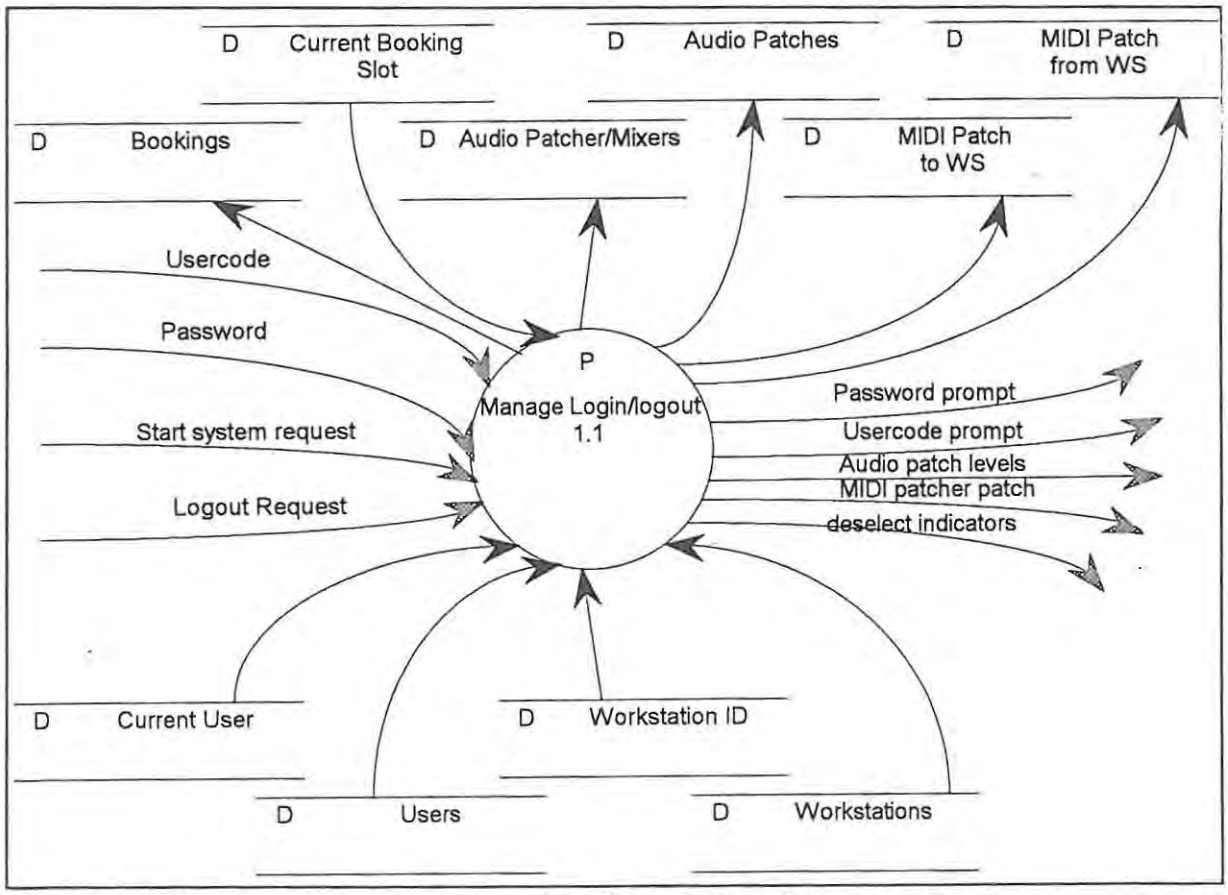

Figure 5.16 - The Login/Logout transform describing the User Entry procedure

5.6.1.2 Booking

Once logged in, a user can book studio resources for a session. The high level transform which encapsulates the booking functions and data is shown in Figure 5.17 below. The booking sheet is opened by pressing the booking button in the control panel. An example of a booking sheet is shown in Figure 5.18.

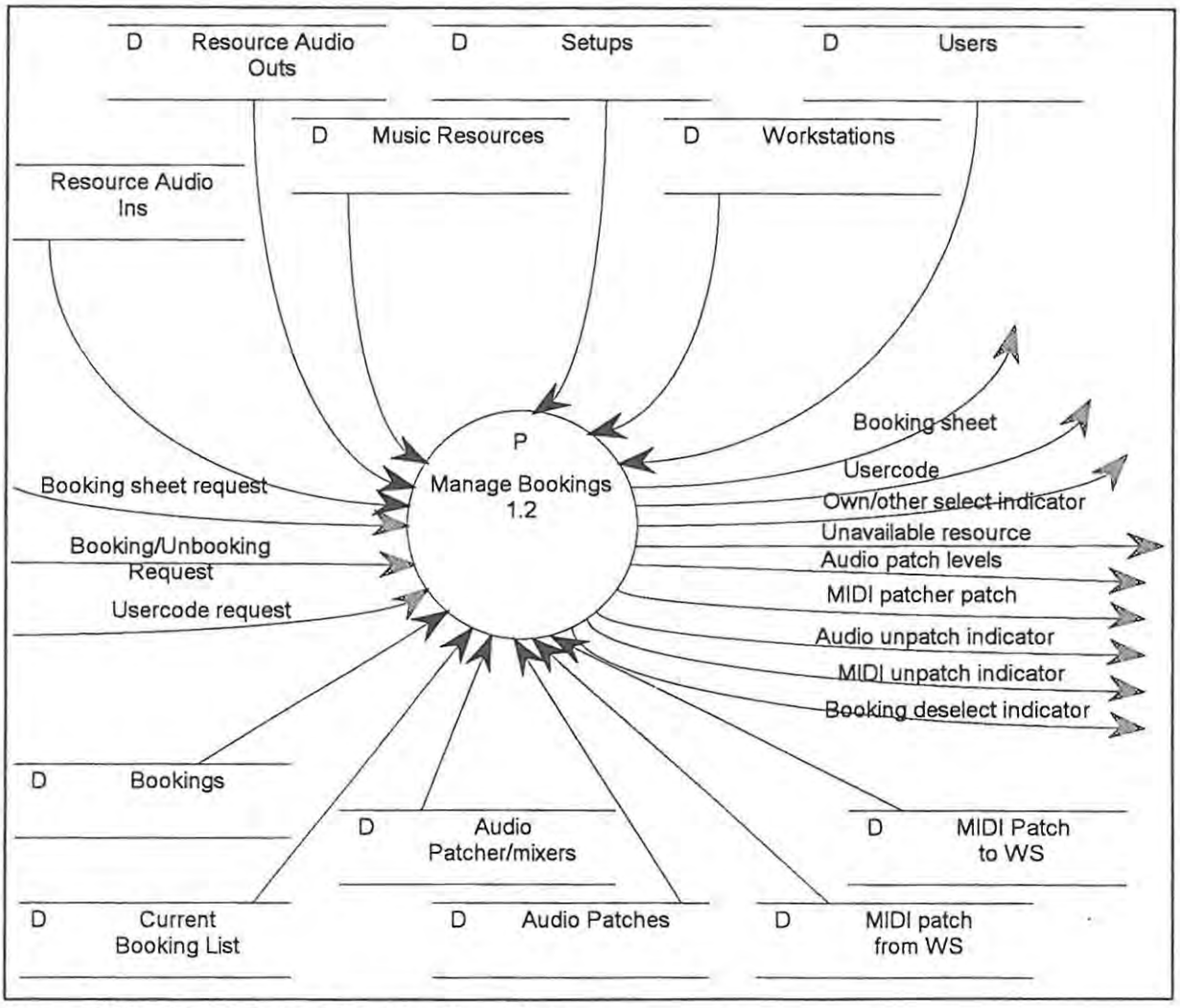

Figure 5.17 - The high-level Booking Transform

All the studio resources appear on the booking sheet and can be booked at particular times by pressing the appropriate grid point. Hour-long slots can be booked, and a user can book up to seven days in advance. An 'x' indicates whether the current user owns a slot, while a 'u' indicates ownership by another user. The booking sheet comprises a list of resources and associated users for each hour of each day. The booking sheet is duplicated on the server and workstations. When a user requests a resource at a particular time, the server fields the request, and then acknowledges it by broadcasting the new booking to all workstations. All workstations then update their booking sheets, thereby retaining exact duplication across the entire system.

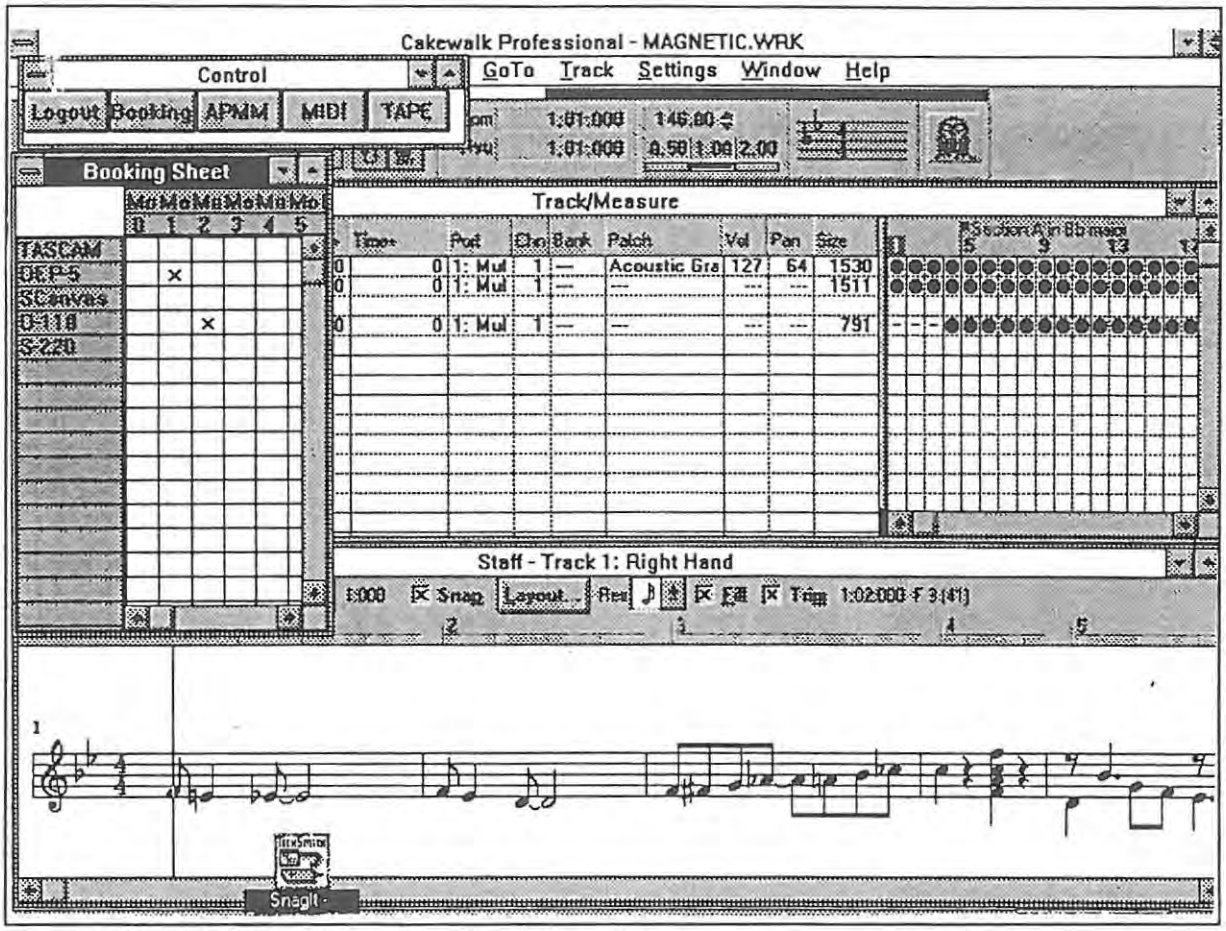

Figure 5.18 - A Booking Sheet above the Cakewalk Sequencer

The booking system is essential to prevent contention for resources amongst users. It also introduces complexity into the system, in that booking slots must be monitored, and users excluded from or allowed usage at appropriate times. Resources whose booked time is up must be unpatched from users workstations, both in the MIDI and audio domains. If MIDI unpatching occurs while the resource is being sent MIDI messages, a complete MIDI message may not be sent, possibly resulting in the resource moving into an error state. This possibility provides a motivation for an alternate form of MIDI patching, which will be explored in Chapter 6.

5.6.1.3 Audio Patching and Mixing

Once a set of resources have been booked, the user can specify how the audio inputs and outputs of these resources are to be patched and what the mix levels are to be at the patch points. This is done via the audio patcher/mixer interface, which is selected from the control panel buttons. Pressing the control button will cause the interface to alternately pop up and be cleared, allowing for a quick patch or mix level change to be made, while working on a sequencer. The high level transform which describes the user interaction with

the audio interface, and the subsequent control of the hardware audio patcher/mixer, is given in Figure 5.19,

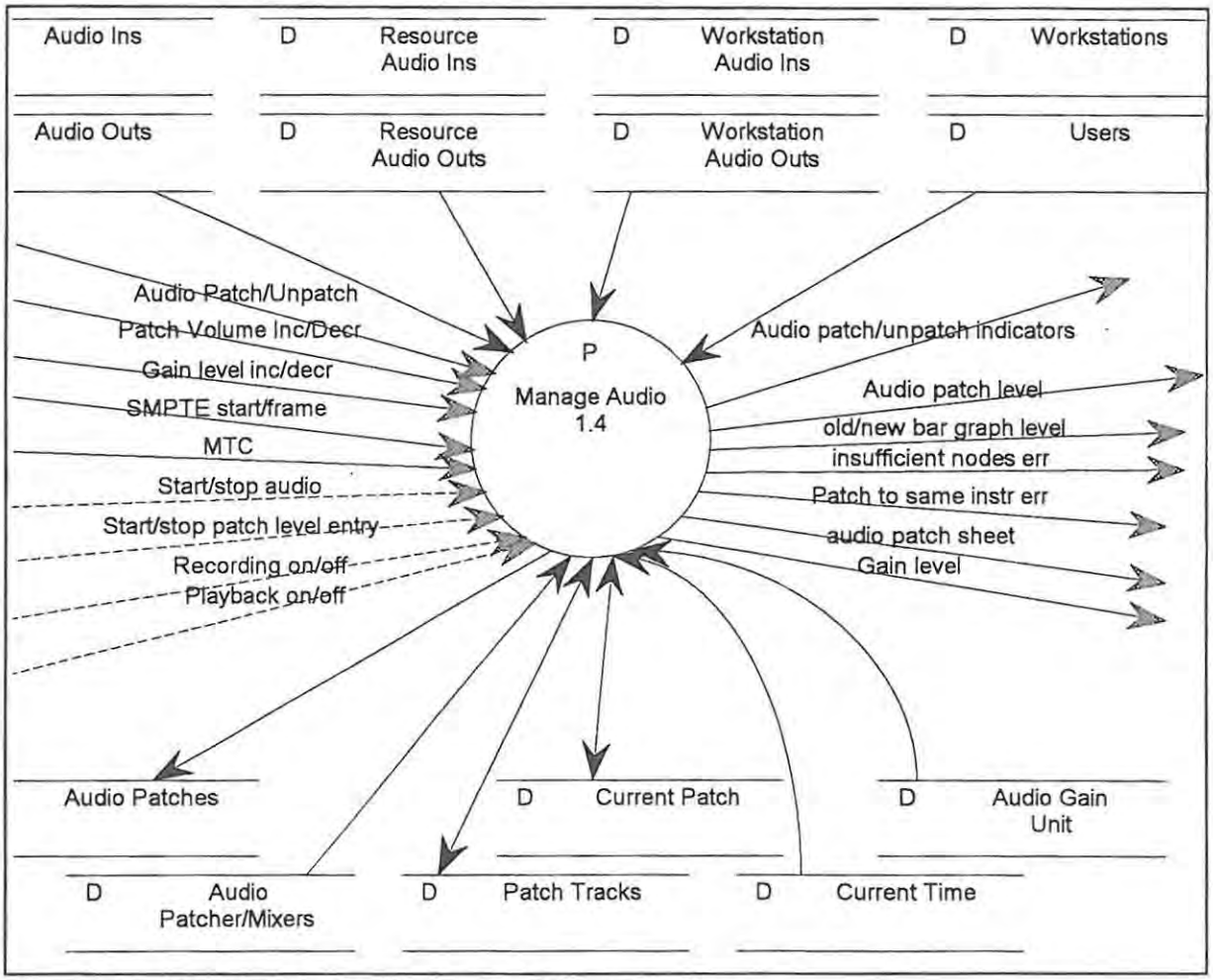

Figure 5.19 - The high-level Audio Patcher/Mixer Transform

and the user interface to the audio patcher/mixer module is given in Figure 5.20. When the user interface is first selected, the workstation requests a list of audio resource inputs and outputs from the server. The server provides these lists together with information about the physical connections to the audio patcher/mixer.

A patch can be created by 'clicking' (pressing and releasing) the left mouse button on a patch point. The patch volume of a selected patch can be increased or decreased, by moving the mouse up or down with the left button depressed. An indication of the current patch volume is displayed at the patch point on the patch grid, as shown in Figure 5.20. This volume is a percentage of the full amplitude possible at the patch point, not a loudness value as in the case of the feasibility study. The patch volume can also be changed by moving the scroll bar button of the 'New' scroll bar. A double click on the left mouse button will unpatch a created patch. A right mouse button click on a patch will set the patch to a default value. When a patch is created, deleted, or its volume changed, information is sent to the server. This information includes the physical

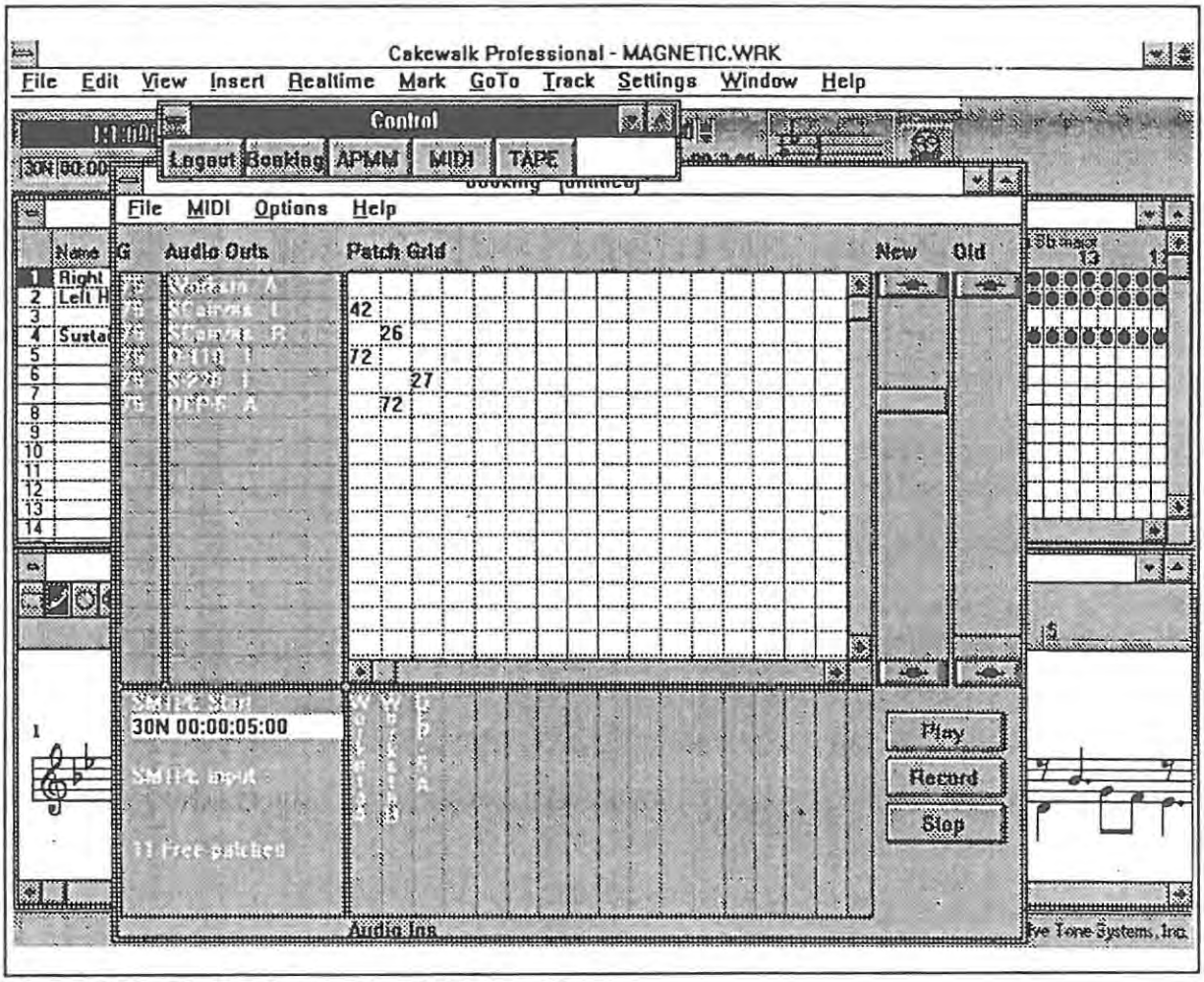

Figure 5.20 - The Audio Patching/Mixing Grid

connections of the inputs and outputs, as well as the patch status and current volume. The server fields this information and constructs the relevant MIDI system exclusive message for transmission to the hardware audio patcher/mixer.

A SMPTE start value can be entered in the SMPTE Start edit box. SMPTE timing frames reach the audio patcher/mixer in the form of MIDI Time Code messages via the selected input MIDI device. The source of these messages can be either a remote multitrack recorder or a digital recorder program running concurrently on the workstation. For testing purposes, the Quad Studio digital recorder from Turtle Beach Systems was used. MIDI Time Code messages are captured and assembled into complete frames. An internal message alerts the program each time a complete frame is assembled.

To record patch changes, the Record button is pressed. Patch changes are only recorded if the current SMPTE frame value is greater than the start frame specified by the user. Each patch change is recorded together with information about its grid position. Once recorded, these patch changes can be 'played back' by pressing the

'Play' button. Patches will only be played back if the current SMPTE value is greater than the SMPTE start value. The 'Old' scroll bar is provided to indicate the current recorded levels of the selected patch. This provides a direct comparison with the new levels input as indicated by the 'New' scroll bar.

The control aspects of the audio patching/mixing component are complex. The transform diagrams which accurately model this complexity are shown in Appendix F. The Ward and Mellor enhancements to the traditional structured analysis /design were introduced to represent this type of real-time interaction, and were found to be quite adequate.

5.6.1.4 MIDI Patching

According to the original configuration, MIDI messages flow from each workstation, down MIDI cables, to a remote MIDI patch bay. These MIDI messages must be routed to the appropriate MIDI resources. As shown in the original specification in Appendix B, there was to be a MIDI patch screen, analogous to the audio patch screen which allowed for MIDI routing, and for resource channel changes. A complete analysis and design was performed for this specification, but was not implemented. The configuration was tested by using Lone Wolf's MIDITap unit to perform manual patching from workstation to devices, and from device to workstation. The analysis and design highlighted the need for a better approach to MIDI routing, and this will be discussed in Chapter 6. In this section, the original analysis and design will be shown, and the problems described.

As always, the MIDI patch screen interface can be accessed by pressing the MIDI control panel button. The high level transform which displays the user interaction with the MIDI interface is shown in Figure 5.21 below. When a user requests the MIDI patching facility, then information about each booked resource and its associated parts must be transmitted from the server to the workstation. Each part of each resource is set to a particular channel number, and this channel number must be associated with the part. A user is able to change the channel numbers of the various parts. The MIDI specification does not incorporate a 'channel change' message, and all channel changes must be performed via manufacturer-specific system exclusive message. The server keeps information about the system exclusive message for each part of each resource type. Thus, although there may be multiple DX7's in the resource pool, system exclusive information will only be retained for the single DX7 resource type. The relationships between MIDI resources, resource types, resource part types, and resource parts are shown in the system information model in Figure 5.5. The server accesses the system exclusive message format and transmits it to the workstation. The workstation then transmits the message over its MIDI link to the relevant resource.

When a user requests to make or delete a MIDI patch from the MIDI interface, patch change information is

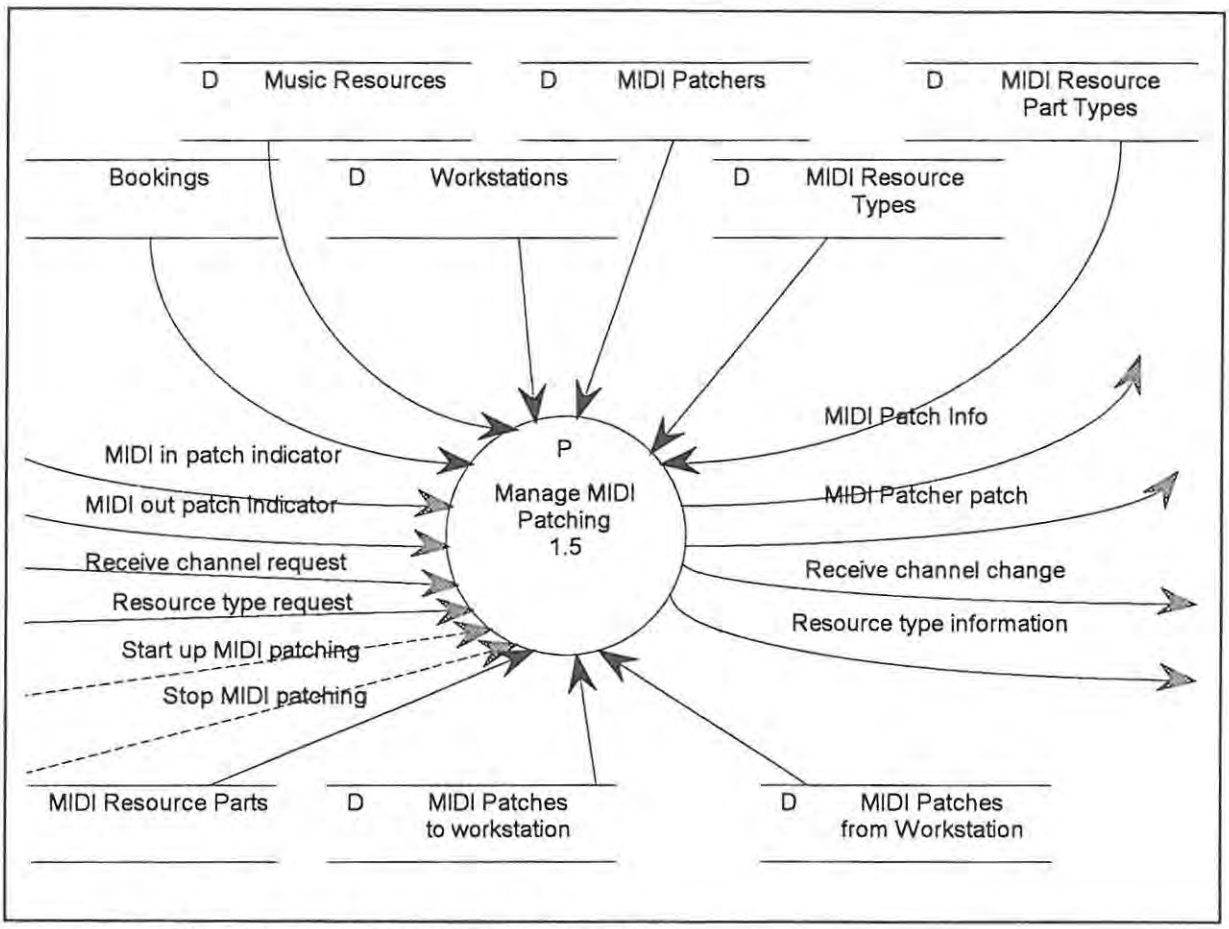

Figure 5.21 - The high-level MIDI Patching Transform

sent to the server, which then transmits the patch change to the hardware MIDI patcher unit using a system exclusive message. This is similar to the approach used for audio patching.

The manager of the remote studio access system must enter the system exclusive message format for any music resources added to the system. This requires a thorough knowledge of MIDI, and often, the system exclusive message formats for channel changes are not easily accessible from the instrument manuals. Sometimes, instruments have to have their states changed to accept system exclusive messages while some instruments do not provide a channel change capability at all. The difficulties with the approach stem from the fact that a non-standard message type is being used. The approach described in Chapter 6 avoids the use of system exclusive messages, and also provides an enhanced configuration.

5.6.1.5 Tape Control

The Tascam 238 multitrack cassette recorder was the only multitrack recorder available for testing purposes in the remote studio access implementation. In the feasibility study, an exact replica of the Tascam 238 front

panel was provided as a user interface. The Tascam can be controlled via a serial link using the RS232 hardware protocol and a custom software protocol. In the feasibility study, mouse moves and button presses on the workstation interface were translated into Tascam software protocol messages and sent to the server via Ethernet. The server forwarded these messages to the Tascam via the serial link. This same approach can be used for all recorders which provide a hardware and software protocol for machine communication. The Tascam 238 software protocol [TEAC 1989] is analogous to many of the other machine protocols and it is instructive to have a brief look at it.

There are four sets of messages, namely, status request, status response, control, and error messages. The control messages are the simplest to understand, since they provide the familiar functionality of a tape recorder. There are messages to control play back, recording, fast forward, rewind, return to zero, locate to a particular position, and many others which correspond to front panel controls.

The status request messages are sent to the Tascam to obtain information about such parameters as the current counter value, the counter values within each of the two memory locations, the record status of each of the tracks, and the cassette protection status. Status response messages are returned to the machine controller, and contain the information requested by the status request messages.

Error messages are returned by the Tascam 238 if there is an illegal command, if there is no cassette in the deck, or if there is record protection.

Remote machine control was specified for the remote studio access system, and the specifications analysed and designed. Figure 5.22 shows the high level transform which encapsulates the management of a remote recording and playback device.

In 1992, the MIDI specification was upgraded with a MIDI Machine Control (MMC) specification [IMA 1992]. MMC has been adopted by the MIDI Manufacturer's Association (MMA) and the Japanese MIDI Standards Committee (JMSC) as a protocol for controlling audio and video recorders, and other timecode-based devices via MIDI. MIDI Machine Control uses two Universal Real Time System Exclusive ID numbers, one for commands (transmissions from controller to controlled device), and one for responses (transmissions from controlled device to controller). All Universal System Exclusive messages have the manufacturer identification number within the typical system exclusive format set to 7F hex. The MMC system exclusive messages thus have the following format:

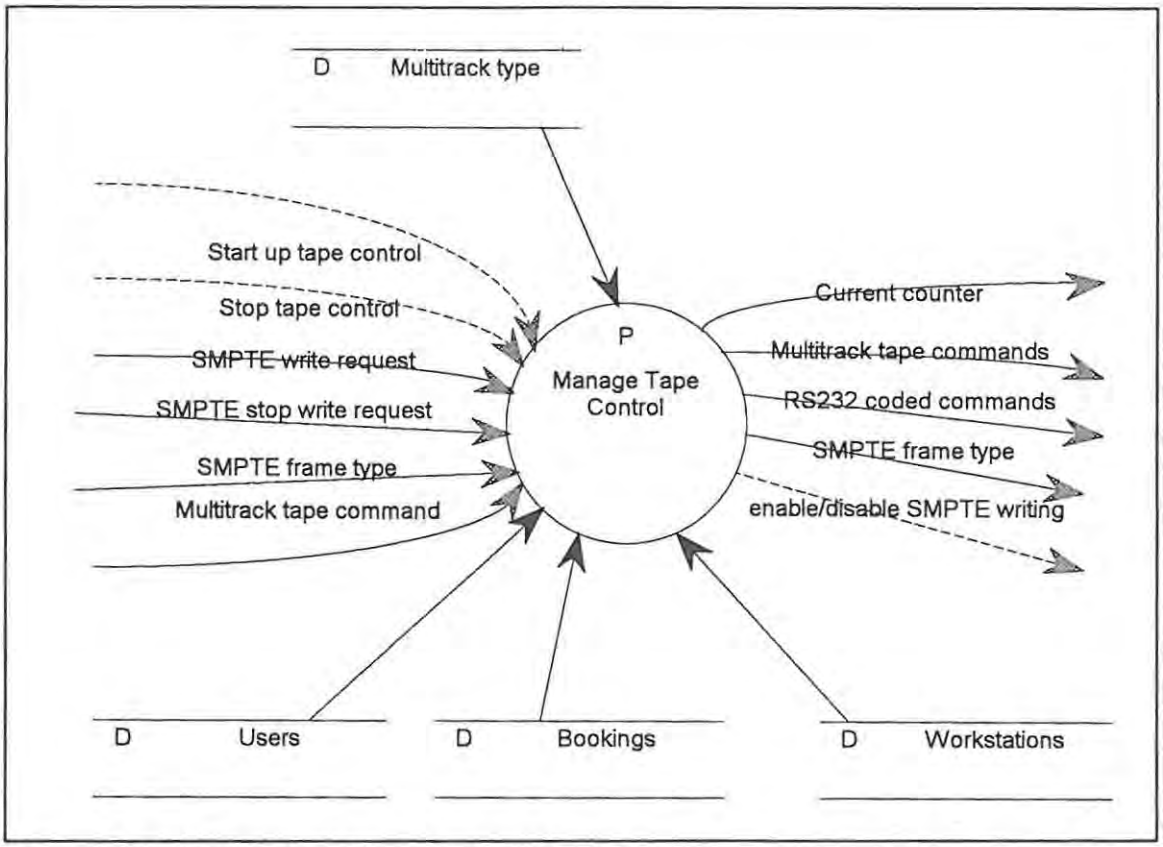

Figure 5.22 - The high-level Tape Control Transform

F0 7F <device_ID> <mcc> <commands... > F7 (controller to controlled device)

F0 7F <device_ID> <mcr> <responses... > F7 (controlled device to controller)

<device_ID> is either a controller or controlled device address.

Many sequencers have incorporated the MIDI Machine Control Protocol. A good example is the Cakewalk sequencer into which it was first introduced in the 3.0 release [Twelve Tone 1994].

When the 'Play' command is specified in Cakewalk, an MMC 'Locate' message is first transmitted, and can be received by one or more MMC compatible devices. These MMC devices will move to the given SMPTE time code position on their recordable medium, whether it be audio tape, video tape, or hard drive. An MMC 'Play' command will then be sent, and the device will start playing from the correct timecode position. The device will, in its turn, produce MIDI Time Code messages which are returned to the sequencer. The

sequencer can synchronize itself to these messages. The great advantage of this approach is that the sequencer is now the master device from which all the studio resources can be controlled.

In the current remote studio access configuration, a MIDI connection runs to and from each workstation, and so the workstation sequencer can be used to control remote recorders. It was decided to adopt this approach to machine control rather than the network based one, as the number of user interface control screens are reduced. In our particular situation, this was problematic, since the Tascam 238 only had a custom control protocol. However, many of the Tascam protocol messages could be directly translated into MIDI Machine Control messages. An IBM PC-based system was constructed which read MIDI Machine Control commands, translated them into Tascam 238 software protocol messages, and then transmitted them to the Tascam 238. The IBM PC system, together with the Tascam 238, now form a complete MMC compatible device. This same approach could be used with any other non-MMC compatible device.

Direct translations from the MIDI Machine Protocol to the Tascam 238 software protocol could be performed for the set of control commands, such as 'Play', 'Stop', 'Fast Forward', etc. The essential 'Locate' MMC command was more of a challenge. There is no 'Locate' command on the Tascam which allows location to a particular SMPTE point. Furthermore, SMPTE cannot be read while the Tascam is in 'Fast Forward' or 'Rewind' mode. However the tape counter can be read in these two modes. A table was built up which provides a mapping between SMPTE time code and counter values. From this table a target counter value could be calculated on receipt of an MMC Locate command. Tascam 'Fast Forward' or 'Rewind' messages were transmitted to move the tape to just before the required SMPTE time, and then a 'Play' message sent to move the tape to the exact SMPTE time. A more detailed account of this system can be found in [Ntene 1993].

Although remote multitrack recorder control is possible from a sequencer on any workstation, there are problems with this approach, in that there has to be user interaction with the remote studio. For multitrack tape recorder control, there is just a mechanical procedure involved, a user must mount and unmount a tape, or request an operator to do this. The situation is not as simple for multitrack hard disk recorders, where a user must open a file before recording, and is excluded from all the editing features which make hard disk recording systems so attractive. There are ways around this, however. A number of hard disk recording units utilize the IBM PC as a hardware and software controller, and for providing a user interface. The hardware control is either via a proprietary interface card, as in the case of the Soundscape and Yamaha CBX-D5 hard disk recorders, or a more standard SCSI interface card, as in the case of the Vestax unit. Editing and recording software has been written for the former two units, and is being written for the latter. In principle, it is possible to receive the hard disk control commands from the software and transmit them down Ethernet to the server. The server can then forward the commands to the recorder unit. Chunks of digital audio and

acknowledgements can be returned to the workstation software in a similar fashion. Both Yamaha and Soundscape corporations have indicated that they will make their PC-to-recorder protocols available.

Even with this possibility, there is still a second problem of disk space availability in a shared environment, where all users have access to a single, limited file space. Most hard disk units provide the capability of backup to, for example, DAT tape. There should, at least, be some sort of remotely controllable file backup system. Solid State Logic do this within their SoundNet system by providing tape and magneto-optical backup facilities all accessible via SCSI [Yonge 1993]. A separate server with a SCSI interface could do this. Exclusive access to a particular user file space would also be desirable, but not essential if the backup facility exists.

5.6.1.6 Manage Setup

The specifications for the remote studio system required that the MIDI and audio patch settings be stored in a setup file and that these setups could be restored in future sessions. The high level transform which describes this process of storing and retrieving state information, is given in Figure 5.23 below.

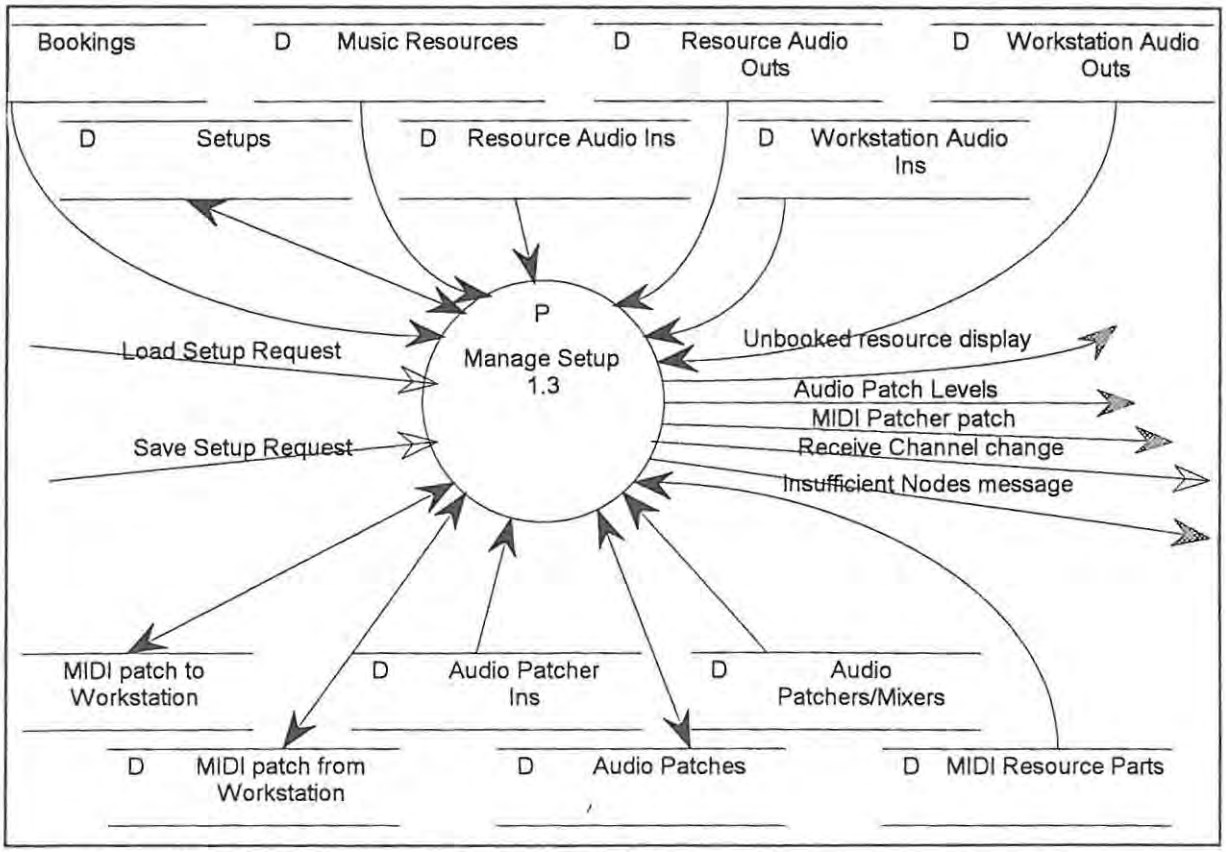

Figure 5.23 - The high-level Transform for Set-Up Management

Loading a setup requires information retrieval from a number of stores. Current bookings are checked to see whether the previous resources are now present. Identical resources have to be used in cases where the actual setup resources are not booked. Music resource and workstation information is used to perform the saved MIDI and audio patches. Patching has to be done on the server side. All the necessary channel changes must be performed by sending the relevant system exclusive message formats to the workstations, which then transmits them to the actual devices.

5.6.2 Managing 'Manager' Requests

The main task of the Manager of the remote studio access facility is to configure the system. Configuration implies specifying all the music resources and workstations in the system, and listing the users who can use the facility. The Audio and MIDI patch connections of these resources to the audio and MIDI patch bays need to be known. Without this information, the server cannot fulfill the workstation's patch requests. It should be noted here that most music resources will have multiple audio output connections to the audio patch bay, and these connections must be identified. This also holds true for the audio input connections to certain music resources such as effects units and multitrack recorders. The 'Manage Manager Requests' transform, shown in Figure 5.13, was upward-levelled from a number of sub-transforms. These transforms are shown in Figure 5.24 below. As always, with the more complex schemas, the data flows and stores have not been shown on the diagram.

The manager's interface to these configuration transforms is shown in Figure 5.25 . The 'List Configuration Commands' transform has been implemented as a series of buttons at the top of the configuration screen. These are toggle buttons, allowing a particular part of the configuration to be shown or hidden. The workstation configuration screen has been built according to the user interface specifications. The remaining configuration screens are in the form of simple editors which allow the various configuration parameters to be typed in and edited. This was done simply to speed up the implementation, and all the screens can later be fleshed out to match the specification interface. Each of the above configuration transforms breaks down into lower level transforms, resulting in quite a lengthy set of analysis diagrams. Note that the MIDI menu in the configuration interface allows for MIDI driver selection, for transmitting MIDI messages to the patch bays.

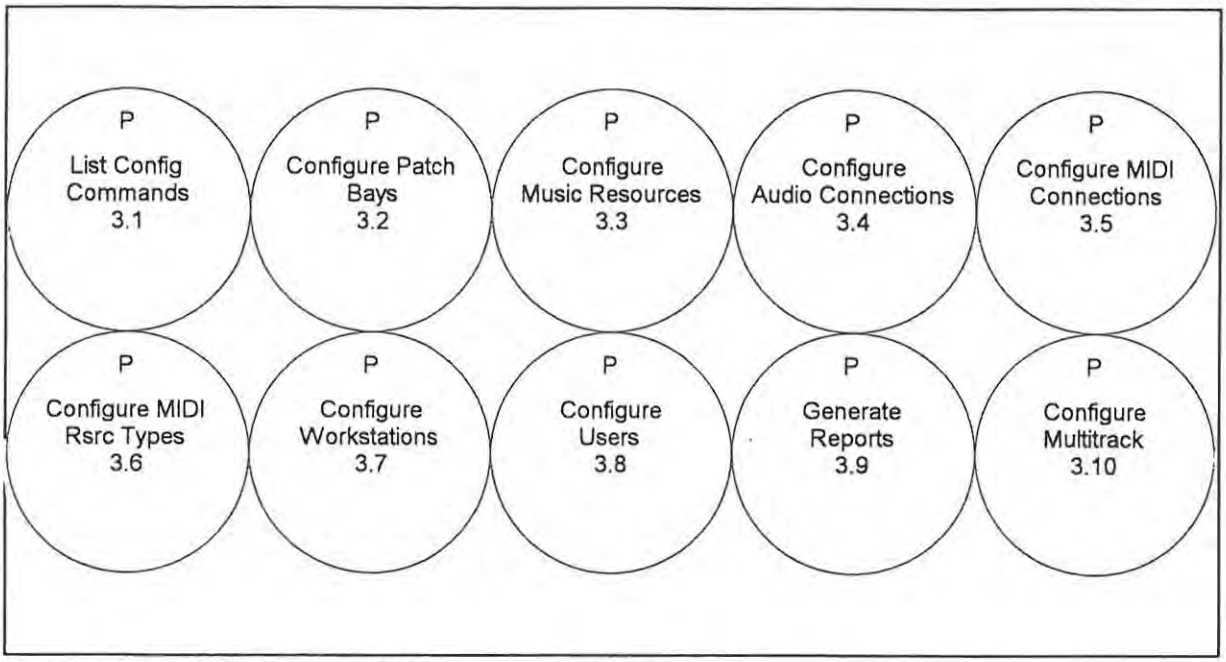

Figure 5.24 - All the 'Manage Manager Request' Transforms

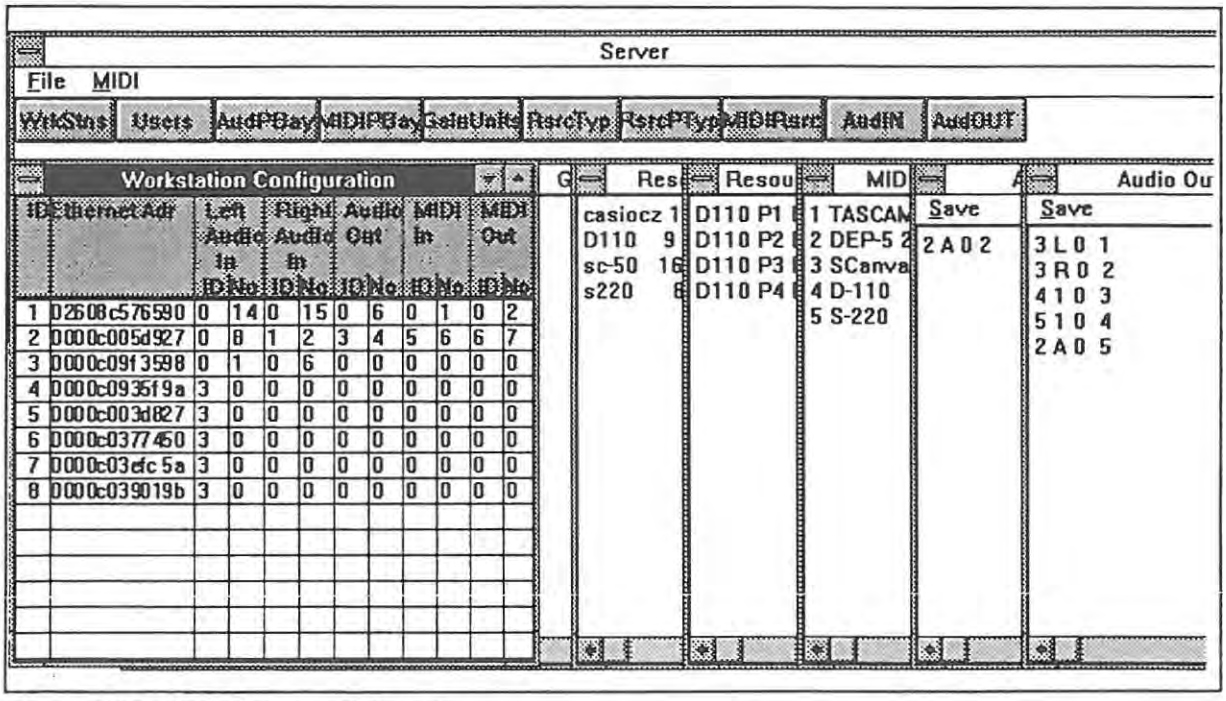

Figure 5.25 - The Manager's Interface

5.6.3 Manage Resource Usage

Transform 3.9 in Figure 5.24 describes the interactions and processing required for the display of workstation and music resource usage by users of the remote studio. Information about this resource usage is accumulated throughout the life of the system. The information model of the system shown in Figure 5.7 has provided placeholders for resource usage, in the form of a 'Resource Time' object and the 'work time' attribute of the 'User' object.

However the management of resource usage extends beyond keeping an account of resource usage, it is also concerned with the management of bookings, ensuring that users can access their booked resources at the appointed times. Two control transforms are used to model the lower levels of resource management. Simplified diagrams showing the interactions between control and data transforms, are given in Figures 5.26 and 5.27 below. Only event flows have been shown on these diagrams. There are two 'event generator' transforms, namely 'generate resource timer' and 'generate booking timer'. There is a direct correspondence between these timer event generators and the Windows operating system timers which generate timer messages into the Windows message queue.

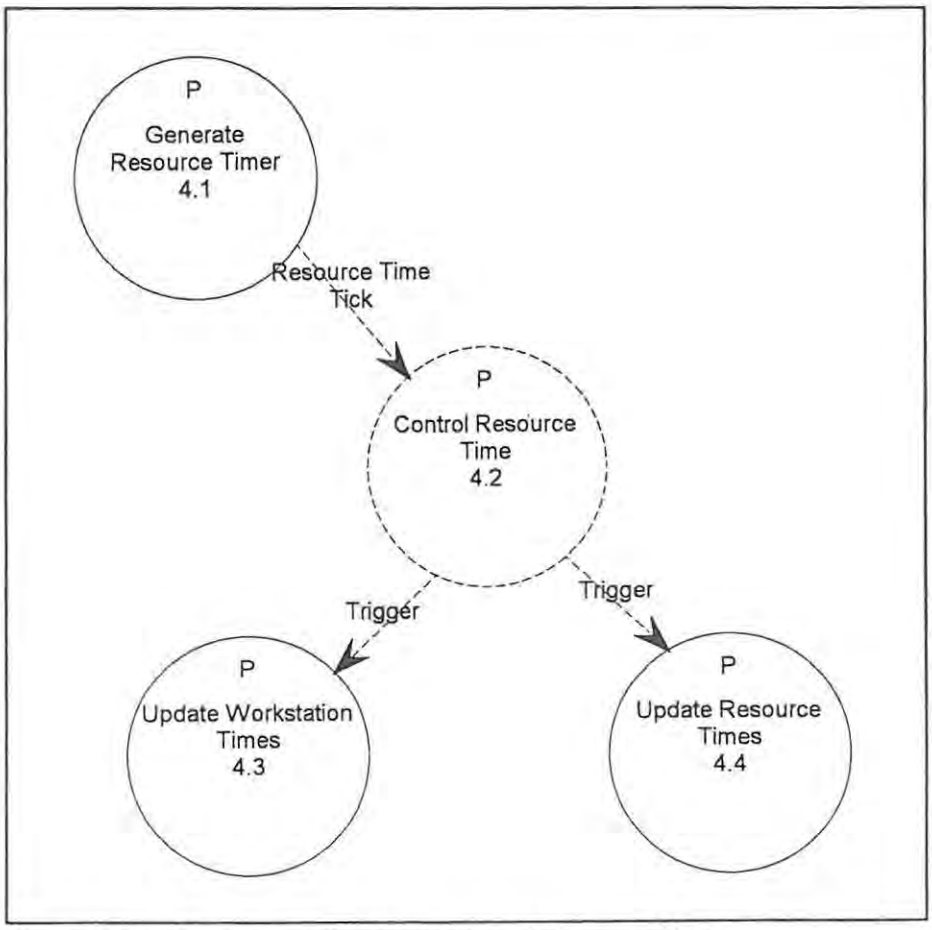

Figure 5.26 - Transforms for the Timing of Resource Usage

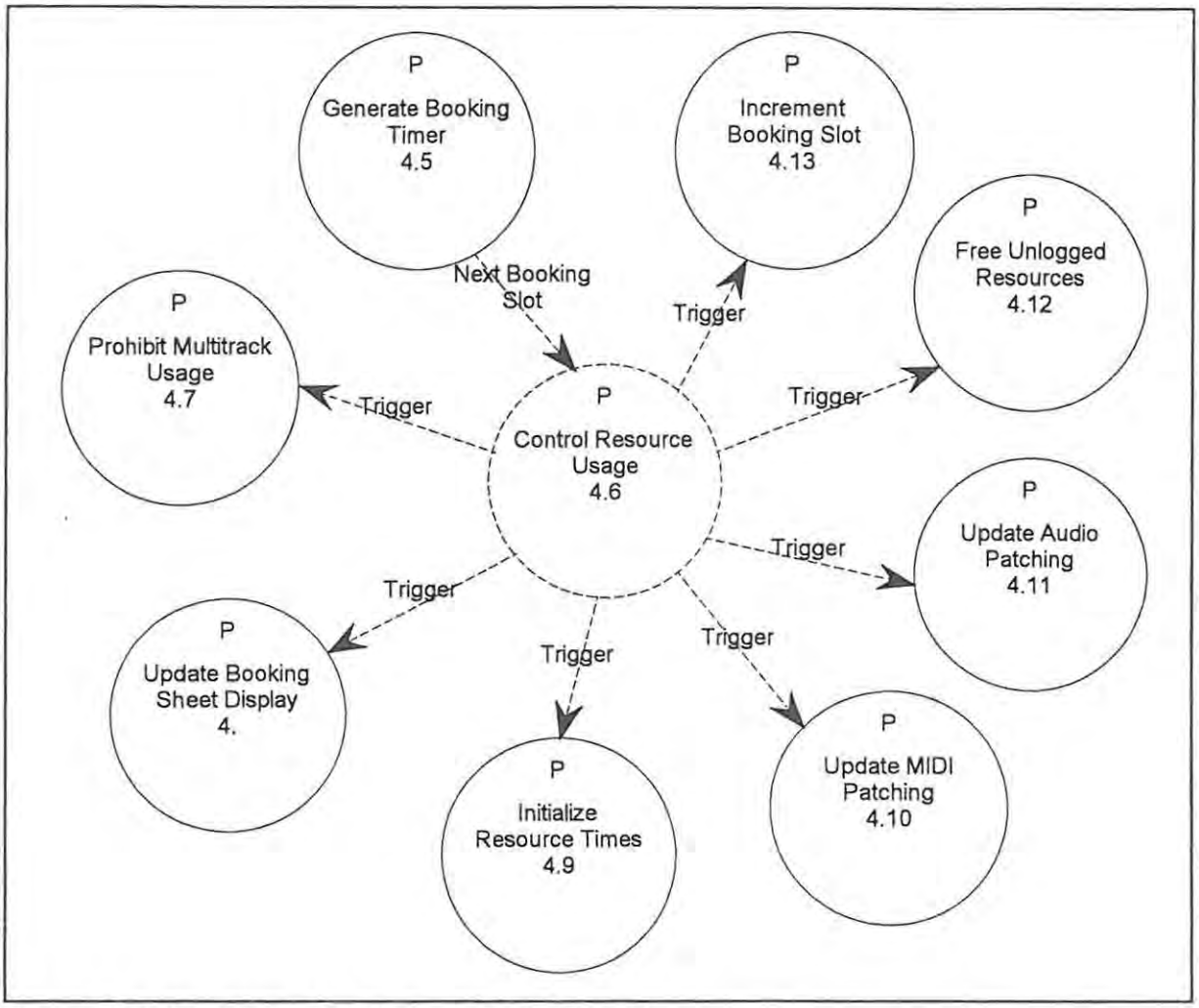

Figure 5.27 - Transforms for Resource Usage Control

5.7 An Evaluation of the Remote Studio Access System

Two aspects of the remote studio access system need to be evaluated, usability and speed. Of course, there is some overlap here, since a user interface which does not respond sufficiently fast to a mix level change will also be unusable.

5.7.1 The User Interface

The workstation user interface to the remote studio provides a user with a number of facilities, all accessible via different windows. These windows can be made to appear or disappear via button presses on a control panel. The two main activities of a user will be based around a commercial sequencer and the audio patcher/mixer facility. It takes a bit of time to juggle the sizes and positions of the windows associated with

these programs, such that usage is optimal. Once positioned, however, control is quick, and a user can move rapidly between activities. This centralization of control was one of the main motivations for the construction of the remote studio access facility.

The user can watch the value relating to the current patch volume change as the mouse control is moved. This value is given as a percentage of the full amplitude value. If the mouse is moved at a speed where the volume is incremented or decremented in steps of one unit, the mixes will be smooth. This is caused by the low decibel increments on the audio patcher/mixer [Wilks 1994]. However, if the mouse is moved up or down very rapidly, packets will be transmitted with patch level increments greater than one unit, and volume changes are not as smooth. When the mouse is moved rapidly, 'Mouse Move' messages are not transmitted to the program's main message handling loop for every pixel incremented [Petzold 1992]. When a message is received and processed, the new pixel value on the y axis is used to determine the new patch level to be transmitted to the server. This approach allows for quick volume changes

In order to determine the number of patch level requests sent from workstation to server, the server program was modified to report the number of patch level requests received. When the mouse was moved at maximum speed from the lowest to highest amplitude, 15 patch level change messages were generated. There are 128 volume levels at the patch point of the audio patcher/mixer, and at the mid-range volume, each level change represents a loudness change of 0,2 decibels. Thus, at the mid-range, there will be approximately 1,7 decibel changes in loudness, with these getting larger at the lower volumes.

The mixer response was tested with the Cakewalk sequencer running the Scott Joplin piece 'Magnetic', where timing inconsistencies are quite obvious. The Joplin piece ran smoothly during fast mix changes, which was to be expected since MIDI output is usually triggered via a timer interrupt and direct call-back to a MIDI output routine, as opposed to messaging. What might rather have suffered in this concurrent operation of sequencer and patcher/mixer hardware, was the speed of patch level changes. Patch level changes were counted at the server side, and this count displayed, while changes were transmitted as fast as possible from the workstation side, both with and without sequencer playback. There was found to be no difference in the number of messages transferred. The slight variation in message count showed no consistent decrease with the sequencer running.

These tests were performed with an unloaded network. It is now time to evaluate more closely the potential overloading of the remote studio access system.

5.7.2 Transmission Delays

As discussed in Chapter 4, there are four potential sources of patch level transmission delay in the remote studio configuration: Ethernet transmission delays, server overload, MIDI overload, and audio patcher/mixer overload. The first three are of concern here, since the audio patcher/mixer has been dealt with in Chapter 4. Before dealing with the problem of overload, some thought needs to be given to the overall size of the remote studio access system. A factor which immediately constrains the size of the network is the size of the audio patcher/mixer. Each audio patcher/mixer has 16 inputs and 16 outputs. Although there is currently only one 16x16 audio patcher/mixer in the system, more units could be built to extend the inputs and outputs by multiples of 16.

With a 16x16 audio patcher/mixer the following configuration is possible:

Four workstations, each having 2 audio inputs and 2 audio outputs,
One 4-track multitrack recorder, having 4 inputs and 4 outputs,
Three effects units, each having 1 input and 1 output,
One sampler, having 1 input and 1 output,
Four synthesizers, each having 1 output.

This represents a small studio resource, and it would certainly be desirable to expand this by providing a 32x32 audio patcher/mixer capability. The current audio patcher/mixer has 16 patch nodes which allows a single patch to made on every audio input. An audio input will often be routed to an effects unit and then mixed with the original signal. Also, the audio input may be monitored, and at the same time sent to a track of a multitrack recorder. A 32 node unit would be more realistic, although it should be remembered that the nodes are floating and can be allocated on demand.

5.7.2.1 Ethernet Transmission Delays

Assuming that an audio patcher/mixer configuration comprising four 16x16 audio patcher/mixer units, each with 32 nodes, is used, then it is possible to ascertain the maximum number of messages which could flow from workstations to server, and then on to audio patcher/mixer units. To do this, one patch was selected, and its value incremented and decremented as fast as possible for a period of time. The patch level packet count was recorded and the number of packets per second calculated. After many iterations, the number was established as 33 packets/sec. The total nodes in our assumed configuration is 4*32, i.e. 128. To take a worst case scenario, if each of the 128 patch nodes were to have its level changed 33 times a second, the data flow across Ethernet would be:

33(packet rate) x 72 (minimum packet size) x 128 (number of nodes) x 8 (bits per byte) =
2.43 Megabits/sec

This represents close to a 25% loading of the 10 Megabit/sec Ethernet. According to Hutchinson and Merabti, there will be a maximum transmission delay time of 7,5 to 28 milliseconds for a network load variation of 10 to 90% [Hutchinson 1987]. Even 28 milliseconds is an acceptable delay for the onset of a mix level change, but more importantly this rate of mix level change would be most extraordinary. To get a more realistic idea of possible loading, a few audio mixes were performed on sequenced data over set time periods and their results averaged. The average packet rate was found to be 4 packets per second. With this figure the network load becomes:

4 x 72 x 128 x 8 = 0,29 Megabits/sec

Even this figure, which represents a 3% loading of the Ethernet, is high, as it will be most unusual for every node to be having its patch levels changed continuously. The efficiency of the network could be enhanced if patch level data was grouped into the data section of each packet. Ethernet is most efficient when large packets are transmitted infrequently [Metcalfe 1983].

Another factor which should be borne in mind when assessing the effect of large patch level packet rates, is that synthesizer level changing can also be performed via standard MIDI volume control changes. Most sequencers now provide MIDI 'mixing desks', which provide graphic faders for the easy adjustment of volumes.

5.7.2.2 Server Overloading

The Borland C Profiler, which is incorporated into the Borland C development package [Borland 1991], was used to determine the amount of time used by the server to completely process a patch level change request from a workstation. The profiler allows an analysis to be made of the time spent within each routine within the program. 2000 packets were sent and the recorded time divided by 2000. From this it was found that each patch level change required 3.2 milliseconds of processing time, where the server was an IBM 486 with a 33MegHz clock rate. The server is thus capable of processing 1000/3.2 = 312 patch level changes per second. The maximum number of packets which can be sent from all the workstations, where a 32x32 audio patcher/mixer is being utilized, is:

128 (number of patch nodes) x 33 (maximum packet per second) = 4224 packets per second.

This would clearly overload the server. Using the more realistic mix level rate of 4 level changes per second, the packet rate becomes 128 x 4 = 512 packets per second, still higher than the service rate of the server. As was pointed out in the previous section, this packet rate assumes that all patch nodes are being continuously modified, an unlikely scenario. However this packet rate does point to the need for a fast server and a substantial buffer to deal with time periods when patch change activity is high.

5.7.2.3 MIDI Overload

Each time a patch level change request is received by the server, it builds a system exclusive message to perform the required change, and transmits it to the audio patcher/mixer(s). The system exclusive message comprises 7 bytes, unless there is grouping of patch changes within a single message. It is unrealistic to look at the very worst case maximum here, since the server itself is incapable of dealing with it. With the mix level rate of 4 per second, the server will process and transmit 512 packets per second to the audio patcher/mixer(s) over a single MIDI line. The MIDI line can be daisy-chained from audio patcher/mixer to audio patcher/mixer. This will correspond to a bit rate of :

512 (packet rate) * 7 (message size) * 8 (bits per byte) = 28,67 bits per second.

This is below the MIDI bit rate of 31,25 bits per second. This bit rate could be reduced by using a multi-port MIDI interface on the server side and connecting a separate MIDI cable to each audio patcher/mixer.

From the above analyses, it appears that only the server could be overloaded in a large remote studio access system. This could be rectified by enhancing the power of the server, and fine-tuning the code which performs patch level control. Ethernet and MIDI transmission characteristics do not present an obstacle to the required operation of the system. It is now time to evaluate the system from a broader perspective and see whether there are ways in which it could be improved. Potential improvements should be judged according to whether they fulfill the three-way goal of shared access to remote studio resources from a centralized interface.

The hardware configuration described in Chapter 4, and the software implementation described in this chapter, provide multiple users with shared, centralized control over a large number of remote resources. The MIDI configuration limits the extent of remote communication to 15 metres, unless specialized, expensive, alternative carriers are used. Channel changing will not be possible on those MIDI resources which don't provide the necessary system exclusive messages.

Audio carriers also limit the length of remote communication, but here there is no distinct cut-off point as in the case of MIDI. The resistance of audio lines carrying analogue audio increases with the length of the line,

and energy is dissipated as the audio signal travels down the line. Whether this dissipation is critical, depends on a number of factors such as the composition of the cable, its diameter, the power required at the destination, and the power of the signal at the source [Wilkinson 1994]. What is of more concern is the possibility of electro-magnetic interference in long cable runs, a factor which will certainly diminish audio quality.

Chapter 6 describes approaches to address the problems associated with the hardware configuration described in Chapter 4, while still retaining the positive attributes of the user interface described in this chapter.

Chapter 6

Alternative Hardware Configurations

Chapter 4 proposed a hardware configuration which was ideal in many respects. MIDI and audio for each workstation are channelled to and from studio resources, without any contention from other workstations. The network, with its remote access protocol ensures that only booked devices are used by the respective workstations, and that MIDI and audio are correctly patched. The only real-time messages travelling down the Ethernet are mix level messages, and the bandwidth of Ethernet easily caters for these messages.

However, there are problems with this configuration. MIDI cables have a maximum length of 15 metres, and, while there are solutions available to solve this problem, they are expensive and not easily accessible. This problem limits remote access to the studio resources to 15 metres, although it doesn't limit resource sharing. Channel changing is difficult with this MIDI configuration. The studio manager has to determine the channel change system exclusive message type for each studio resource. Some resources do not have channel change system exclusive messages and the process of channel changing is convoluted. When the user requests a channel change, the server has to transmit a channel change system exclusive message to the workstation, which in turn sends the message down MIDI lines to the appropriate studio resource. Some resources need to be placed in a state where they will respond to the system exclusive message, which might require further messages. The server must keep track of the current receive channel of a resource or resource part, in order to display this to the workstations. It is possible for commercial workstation software to transmit system exclusive messages, and to modify the channel, without the server being aware of these changes.

There are also problems with the audio side of the current configuration. There is a limit to the transmission length of a cable carrying an analogue audio signal, although there is not the same stringency. The length of cabling will depend on a number of factors, such as the make-up of the cable, the width of the cable, the power at the source, and the power required at the destination [Wilkinson 1993]. The strength of the audio signal gets dissipated as it travels along the cable, and this is more marked when a thin, high resistance cable is used. Furthermore, the analogue audio signal in the cable is prone to electromagnetic interference, and the chance of this interference will increase as the length increases. Solutions to the problems of length and interference, such as having balanced lines and higher quality materials, are expensive.

Both MIDI and audio transmission require large amounts of cable in the current configuration. For each

workstation, there have to be two MIDI lines and three audio lines. There is already an Ethernet cable connecting all the workstations and the server, and the idea of this cable carrying MIDI and digital audio is tempting. Of course, there is the problem of contention on the single line. Ethernet is a non-deterministic data transmission medium, the data link protocol being based on an algorithm which detects the presence of a signal on the common bus line, and which backs off for a random time interval if there is one [Tanenbaum 1988]. However, there is no need to retain Ethernet as the network transmission medium.

This chapter will look at alternatives to the current configuration. Firstly, the easier problem of an alternative for MIDI transmission will be looked at. This will be followed by an investigation into alternative audio transmission methods. The first priority here will be to move the audio into the digital domain.

6.1 An Alternative to MIDI Routing in the Remote Studio Access System

Two solutions to the problem of routing MIDI down Ethernet in the remote studio environment have been implemented. Both are based on the MIDITap model of the Lone Wolf Corporation [Lone Wolf 1990]. Both solutions consist of a number of IBM PC machines linked together via Ethernet, each IBM PC being termed a 'MIDINet unit'. A number of MIDI interface cards reside in each PC and provide the physical links to and from MIDI-compatible devices. A diagram of this configuration is given in Figure 6.1 below. The entire configuration is termed a 'MIDINet network'.

The number of MIDI devices attached to each IBM PC is dependant on the number and capability of the MIDI interface cards within each PC. Some PC-based MIDI interface cards have more than one MIDI input and ouput port, indeed some have 8 input and output ports [Music Quest 1995]. A MIDI device attached to a MIDINet unit can transmit MIDI messages to any other MIDI device in the MIDINet network.

It is important to define the meaning of the term 'MIDI device' in the context of a MIDINet unit. Each physical MIDI device, such as a Proteus ultra-sound expander unit, may have a number of MIDI parts, each part being associated with a different receive channel. In the context of the MIDINet network, each of these parts is viewed as a separate device to which MIDI messages can be sent. Some MIDI devices, such as sequencers can transmit MIDI messages on multiple MIDI transmit channels. Here again, in the context of a MIDINet network, each of these transmit channels would be viewed as a separate device. From now on, the term 'MIDINet device' will be used to distinguish the MIDINet view of a device from the more common physical MIDI device. There are MIDINet receiving devices, such as particular parts within a synthesizer, and there are MIDINet transmitting devices, such as the channels of a sequencer.

More accurately, now, any MIDINet transmitting device can transmit MIDI messages to any MIDINet

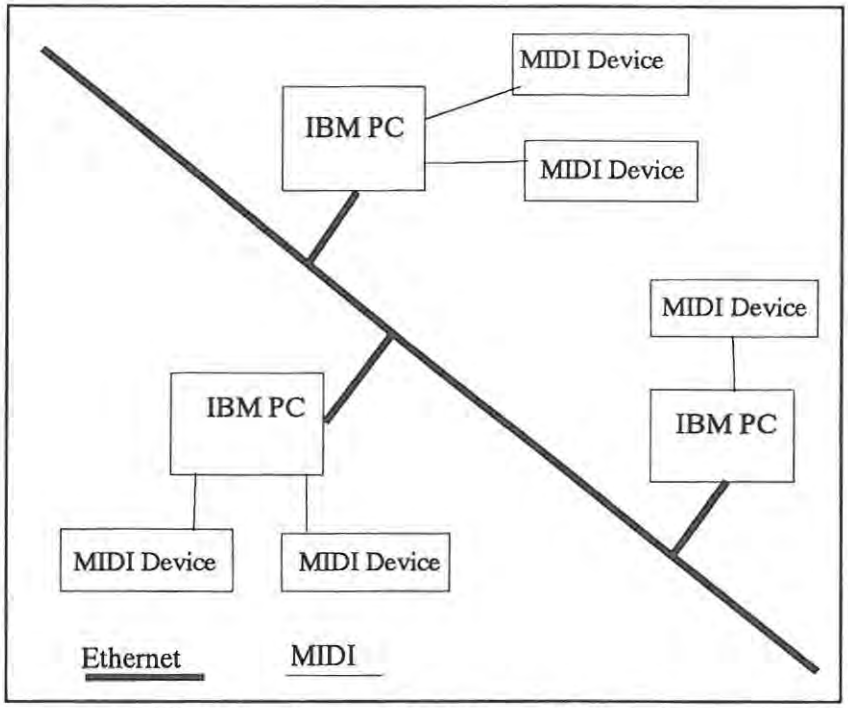

Figure 6.1 - MIDI Routing via IBM PC's and Ethernet

receiving device on the MIDINet network. A particular MIDINet device can transmit the same MIDI messages to more than one MIDINet receiver, providing the same capability as a MIDI thru box, and more than one MIDINet transmitter can transmit MIDI messages to the same MIDINet receiver, thereby allowing for MIDI merging. When a MIDI message is sent to a receiving MIDINet unit, the channel nibble of the MIDI message is altered to correspond with the receive channel of the receiving device. As far as the remote studio access system is concerned, this is one of the most important features of the MIDINet system. If the commercial sequencer running on a remote workstation utilizes a MIDINet network for MIDI message transmission, then the current awkward channel change protocol can be dispensed with. The various parts of MIDI devices do not have to have their channels changed. MIDI messages are directed to particular receivers, and at the point of reception, the nature of the MIDI message is changed to be compatible with the receiver.

A user interface allows a user to make connections by choosing source and destination MIDINet devices. These devices have symbolic names associated with them, but these symbolic names are not unique. The nature of its connections and its channel numbers make a particular MIDINet device unique, and these attributes can be seen by a user. Each MIDINet unit has a unique identification number associated with it, which is determined and assigned by the system manager. Each MIDINet unit also has a number of MIDI ports, corresponding to the physical serial MIDI ports on the interface cards. A MIDI device may have its MIDI output connector connected to one of the MIDI input ports of a particular MIDINet unit. The MIDI device may also have its MIDI input connector connected to one of the MIDI output ports of the same

MIDINet unit. A MIDINet receiving device is uniquely identified by the identification number of the MIDINet unit which its MIDI device is connected to, the port number to which it is connected, and the receive channel number of its part within the MIDI device. A transmitting MIDI device also has MIDINet unit and port number identification, but instead of a receive channel, has a transmit channel to completely identify it. A MIDINet network manager can specify these attributes of a MIDINet device attached to a MIDINet network.

A diagram showing the incorporation of a MIDINet network into the current remote studio access configuration, is given in Figure 6.2 below.

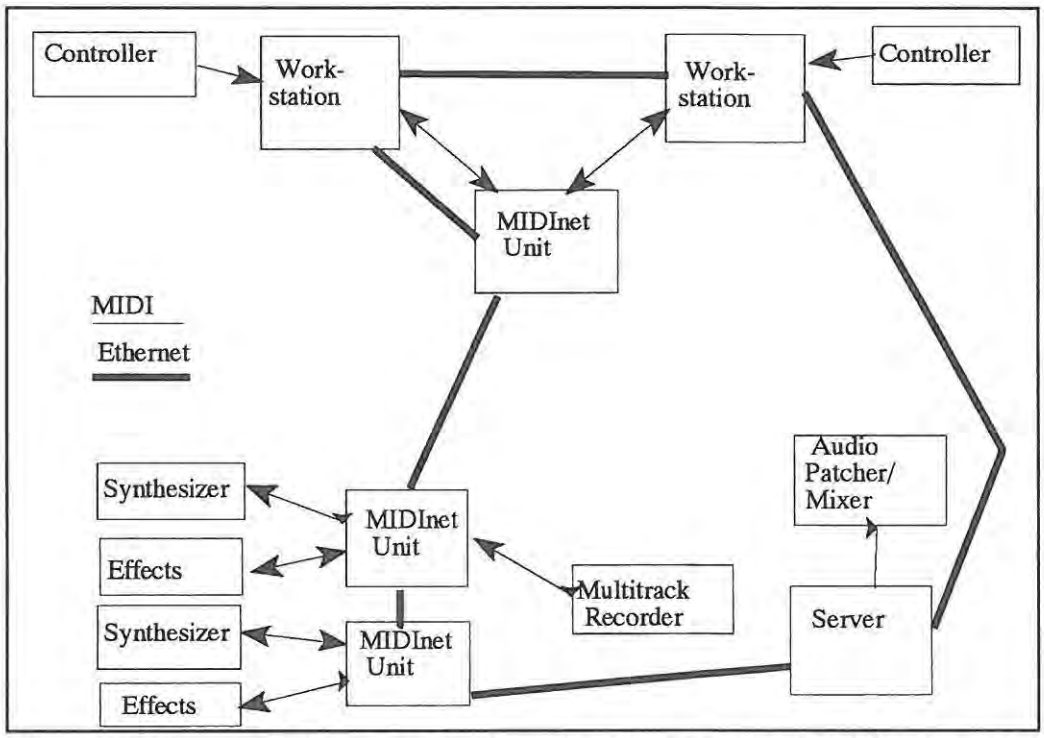

Figure 6.2 - Remote Studio Access Configuration Incorporating MIDINet Network

Each remote workstation has two MIDI connections to a MIDINet unit, an input and an output connection. The MIDI controller at a workstation could either be connected to a MIDINet unit or could be connected directly to the workstation, as shown in the diagram. In this case, a MIDI card with two MIDI input connectors should be installed at the workstation. Such cards are common commercially. More than one workstation could be connected to a single MIDINet unit. At the remote studio side, all the shareable MIDI resources such as synthesizers, effects units and MIDI-controllable recorders, are also connected to one or more MIDINet units. The workstations, server and MIDINet units are all connected onto the same Ethernet bus.

With this arrangement, there will be two protocols running simultaneously on the Ethernet. To prevent excessive processing at the higher protocol levels, an Ethernet packet type is allocated to the remote studio access packets, and a different type to the MIDINet packets. In this way the two packet types are differentiated at the data link level and no separation has to be done at higher protocol levels. The two allocated types are different from the types allocated to TCP/IP and Novell networks, allowing the studio control packets to travel without conflict on typical commercial and university networks.

The two implementations of the MIDINet network are based on essential and implementation models, built using the 'Shlaer and Mellor', and 'Ward and Mellor' toolsets. These tools have been fully described in Chapter 5. A diagram of the information model is given in Figure 6.3 below.

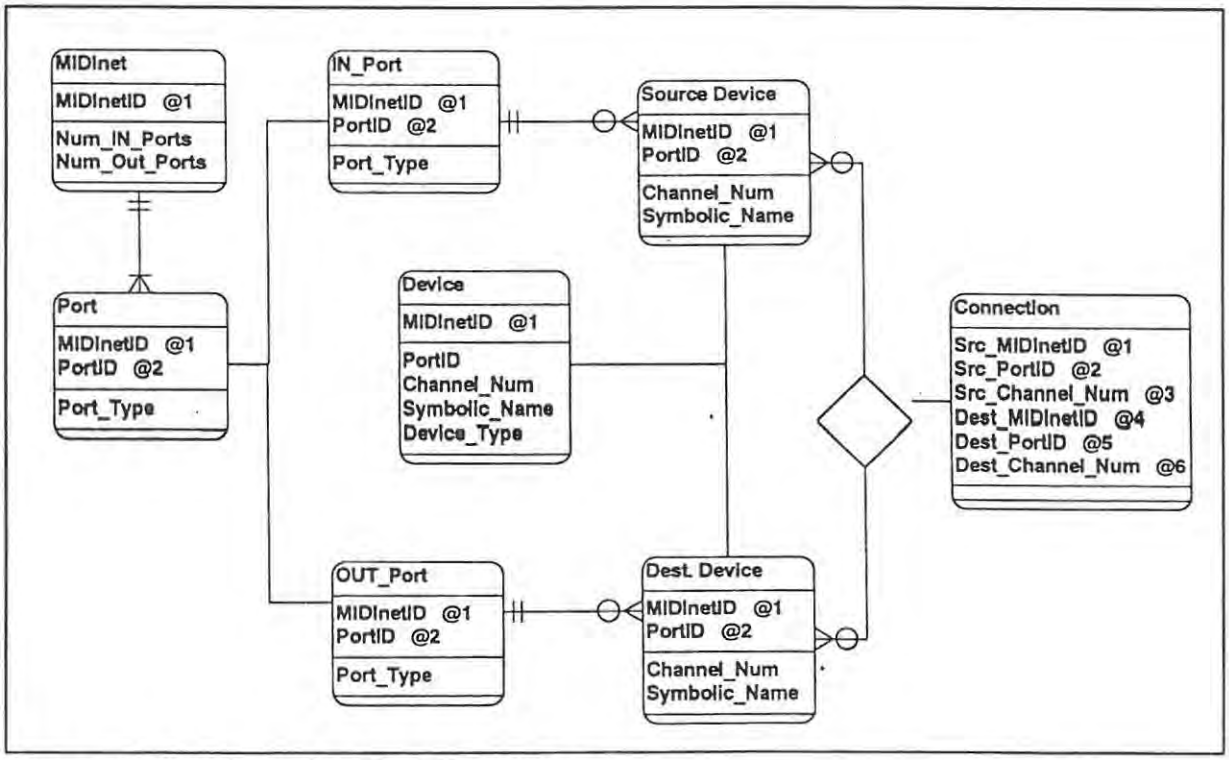

Figure 6.3 - Information Model for a MIDINet unit

This model shows in entity-relationship terms the relations between MIDINet units, ports, transmitting and receiving devices, and their connections. Figure 6.4(a) is a context schema for a MIDINet unit, and Figure 6.4(b) the first level of the upward-levelled transformation schema, simply showing the transforms.

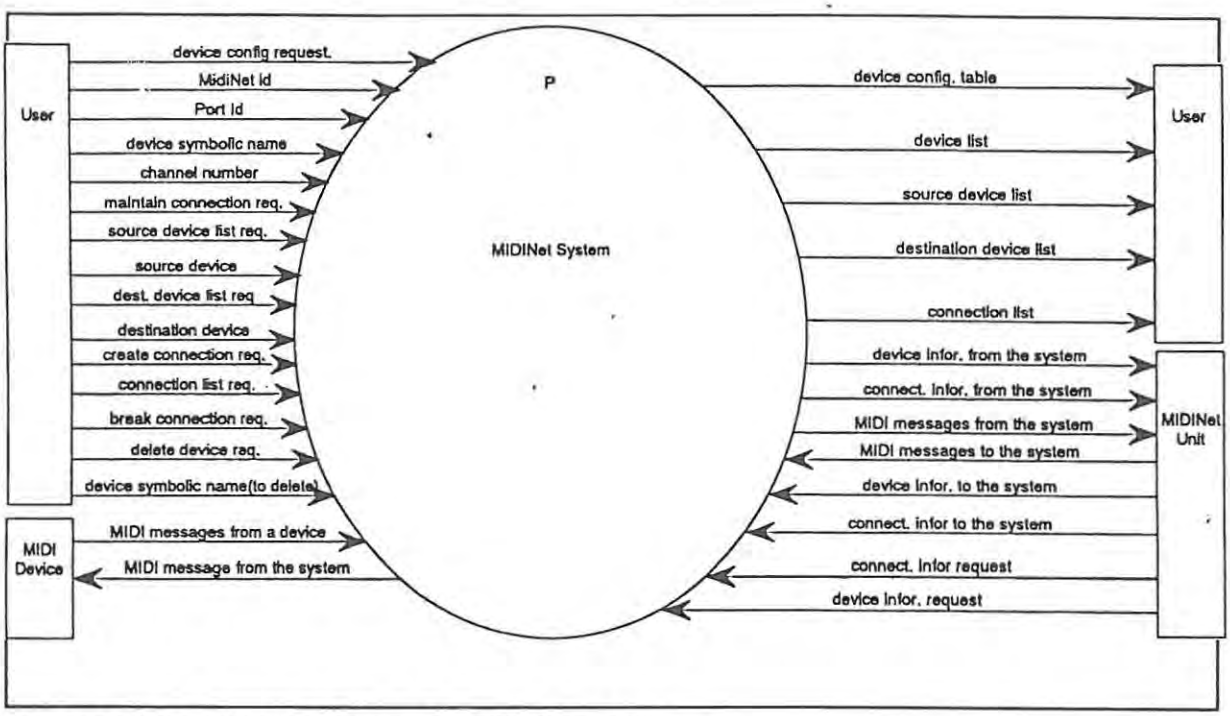

Figure 6.4(a) - A Context Schema for a MIDINet Unit

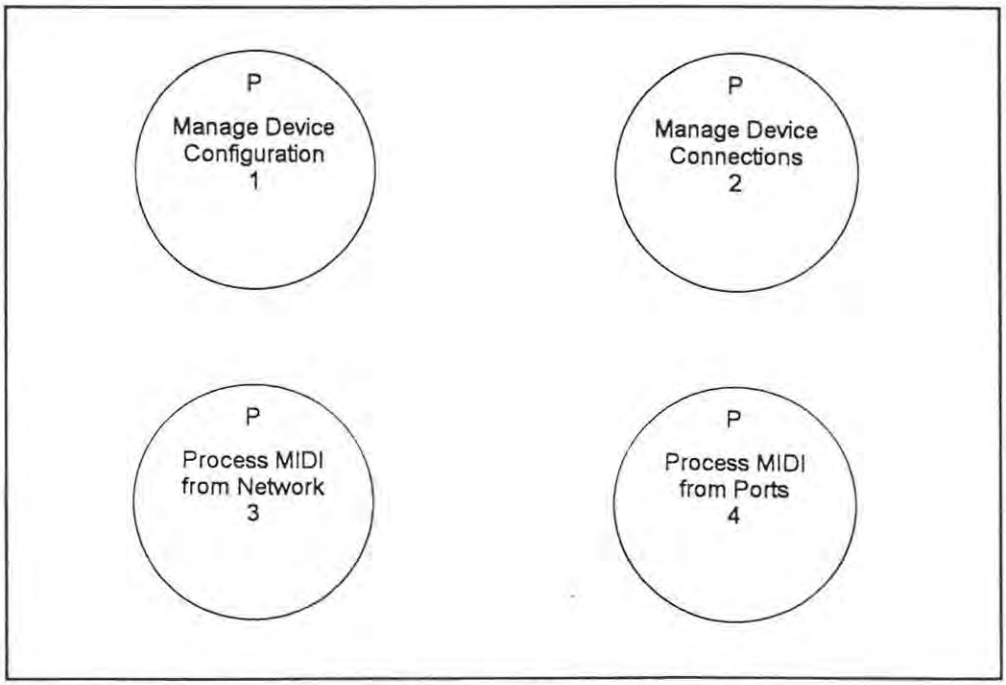

Figure 6.4(b) - High Level Transforms for the MIDINet System

The routing of MIDI messages to and from MIDINet devices is not processor-intensive, and the intention was to utilize inexpensive IBM PC's as MIDINet units. In many cases, a user interface would not be necessary, the MIDINet unit simply being a PC motherboard with necessary peripherals. For this reason, the Windows operating system was not considered as an implementation environment.

6.1.1 A First Implementation under PC-Xinu

The first implementation was developed using a modified version of the PC-Xinu operating system, and is described in [Foss 1995]. Xinu is a small, multitasking operating system originally developed by Douglas Comer for use on networked, diskless workstations [Comer 1984]. It was later ported by Fossum to the IBM PC, with an MS-DOS-compatible file system being added in the process [Comer 1988]. With the original IBM PC version of the operating system, any Xinu program was developed under DOS, then linked to the operating system to create a complete running program. When run, this program would install and initialize the operating system, then launch the user program with its Xinu operating system calls. In order to reduce linking time and reduce the size of executable files, a memory resident version of the operating system was developed [Foss 1992]. In this version, Xinu operating system calls are made to a resident operating system kernel via software interrupts. This was further extended to provide an event-driven programming environment similar to Microsoft Windows, with easily constructible text windows [Rehmet 1993]. This was the version used for the first implementation of the MIDINet system and it was ideal in many respects.

The Ward and Mellor essential model comprises multiple processes, and a multitasking operating system, with task synchronization mechanisms, allows for a more direct mapping to the implementation. PC-Xinu allows for task prioritization, and thus a higher priority could be assigned to tasks such as MIDI message routing, and a lower priority to user interface interaction. The Ward and Mellor approach to process modelling is based on an event-response mechanism, and once again the adapted PC-Xinu operating system allows for a direct modelling of this approach.

MIDI interface cards were custom-built for the first implementation. Each MIDI interface card is a modified serial interface card containing two 8250 UART chips. The transmit and receive ports of these two chips lead to a MIDI box containing the 5-pin DIN connectors and current loop electronics, prescribed by the MIDI standard. Each box has two MIDI input and two MIDI output connectors. The clock of the serial card was changed to allow division down to the MIDI clock speed. Each MIDINet unit contains two of these cards, with each card using a different hardware interrupt.

A device driver was written for PC-Xinu which allowed for high level read and write access to the MIDI interface cards. Device driver construction under PC-Xinu involves writing upper and lower half processes

which communicate via a shared buffer. Operating system primitives, buffers, semaphores, interrupt numbers and port addresses are all specified via an I/O configuration table, known as a 'device switch table' [Comer 1988].

A device driver also had to be constructed for the Ethernet packet driver interface. This was a somewhat different type of device driver, in that the lower half processes interacted at the DOS level with the packet driver interface. Packet writes are performed via software interrupts, and notification of packet arrivals are via packet driver call-backs. There is no direct interaction with hardware ports or the PC's interrupt mechanism. Semaphores were used to ensure that no more than one process accessed the non-re-entrant packet driver code.

6.1.2 A Second Implementation under DOS

A second implementation of the MIDINet system was inspired largely by a desire to provide a more standardized MIDI interface to the MIDINet units. There are a number of manufacturers of IBM PC compatible MIDI interface cards. Many of these cards are compatible with the original PC MIDI interface card produced by Roland corporation, the Roland MPU-401 [Roland 1985]. However, some do not adhere to this de facto standard. In order to provide an industry standard at the software level, the company Voyetra developed the 'Voyetra Application Programmers Interface'(VAPI) [Voyetra 1990]. This was derived from, and follows closely, the CAPI interface of the Yamaha corporation. [Yamaha 1989]. DOS VAPI drivers have been written for most PC MIDI interface cards, and any DOS-based program which uses VAPI software interrupts to access MIDI hardware, can utilize this range of hardware interfaces. VAPI drivers are not written to cater for multiple interface cards, but multiple VAPI drivers can be loaded if the software interrupt vector for each driver is saved and later restored [Koeslag 1995].A PC-Xinu driver has now been written for PC-Xinu, however, as for the Ethernet packet driver, there is a layer of time-consuming PC-Xinu driver software at a level above the DOS VAPI driver.

In the second implementation it was decided to implement under DOS and gain speed at the expense of implementation elegance. The program is based around a polling loop, where three components are polled, the user interface for any user interface events, the MIDI ports for any MIDI messages, and the Ethernet network for any MIDINet messages. The program was based on the analysis model of the first implementation, and required some distortion for implementation under DOS. The MIDINet protocol was originally derived as part of the first implementation and then extended during the second.

6.1.3 The MIDINet Protocol

In a MIDINet network, a MIDINet device can transmit a particular MIDI message to more than one MIDINet receiver. To implement this, the same message could be sent to each receiving device in turn, or the message could be broadcast, and the relevant receivers could then extract it from the network. In order to reduce network traffic and also to reduce transmission time, the latter approach was chosen. All MIDI messages are broadcast across the network using the broadcast network address of Ethernet. All MIDINet units view all the MIDI message packets, and determine from connection information whether the message is destined for one of their attached receivers. With this scheme, all MIDINet units must hold the same connection and configuration information. Whenever a system manager configures a MIDINet unit, this configuration information is broadcast across the network. Likewise whenever a user makes a MIDINet connection, this connection information is broadcast across the network. Of course there needs to be an exactly specified protocol for the transmission of these three message types. There also needs to be a protocol established for when a new MIDINet unit joins the network. This new unit will need to obtain all the current connection and configuration information.

The general form of each Ethernet packet transmitted across a MIDINet network is as follows:

MIDINet_Packet = Destination_Ethernet_Address + Source_Ethernet_Adress + Ethernet_Type +
MIDINet_Message_Type + MIDINet_Data

The MIDINet protocol is encapsulated within the data component of the Ethernet packet. There are four categories of MIDINet messages, one for MIDI messages, one for configuration messages, one for connection messages, and one for initialization messages. There are message types within each of these categories, each message type having a unique digit associated with it.

6.1.3.1 The MIDI Message Category

There is only one message type within this category, having the digit 1 associated with it. The data component of the message has the following form:

MIDINet_data = Source_MIDINet_Unit_ID + Source_Port + MIDI_Length + {MIDI_Message}

There can be multiple MIDI channel messages incorporated in the data component of this type of message, and all these channel messages must originate from the same MIDINet transmitter. This transmission of multiple messages optimizes the use of network bandwidth, since the minimum size of the data component

of an Ethernet packet is 46 bytes, while a channel message uses a maximum of 3 bytes.

The MIDINet transmitter will broadcast such a message across the MIDINet network. All the MIDINet units on the network read the message, and determine the transmitting device from the MIDINet unit identifier, the port number, and channel number of the first MIDI message. Each MIDINet unit then accesses its connection and configuration tables, to determine whether any of its MIDINet devices should receive the MIDI messages of the transmitting device. If they should, then the MIDI messages are extracted from the Ethernet packet, the necessary channel adjustments are made, and the messages are transmitted to the appropriate MIDI device. With this arrangement, messages transmitted from a MIDINet device with running status do not have to have the status byte added in again. The presence of the length byte means that MIDI message parsing need only take place at the transmitter side.

System Common and System Real-Time messages can also be transmitted using this message type. When a MIDINet unit detects one of these message types, it will transmit the message to all MIDI devices having MIDINet receivers to which MIDINet transmitters within the transmitting MIDI device are attached. Put more simply, the MIDINet unit considers connections at a MIDI device to MIDI device level when dealing with System Common and System Real-Time messages. A System Exclusive message may sometimes extend beyond the data space provided within a single packet, in which case it can be split and transmitted by more than one MIDINet message. Once again, the length byte facilitates the receipt of these messages.

6.1.3.2 The MIDINet Configuration Message Category

There are two message types in this category, a message type to add new devices, and a type to remove devices from the network.

- **MIDINet Message Type 2 - Add a Device**

  MIDINet_Data = MIDINet_ID + Port + Channel + Symbolic_Name + In/Out

  This MIDINet message could either be directed to a particular MIDINet unit or could be broadcast If a new MIDINet device is being added to a MIDINet unit, then this fact needs to be broadcast to all MIDINet units in the network. However, if a particular MIDINet unit has just joined the network and needs to have configuration information, this information can be sent directly from an established unit to the new one, preventing unnecessary processing by all other units on the network. This is done by setting the Ethernet destination field to the Ethernet address of the new unit.

The message contains unique identification information together with the symbolic name of the MIDINet device. The 'In/Out' field specifies whether the device is a receiver or transmitter.

- **MIDINet Message Type 3 - Remove a Device**

MIDINet_Data = MIDINet_ID + Port Channel + In/Out

The 'Remove a Device' message is only ever broadcast, to keep the configuration tables on all MIDINet units identical. The symbolic name is not needed for unique identification, and so is not included.

6.1.3.3 The MIDINet Connection Message Category

As for the configuration category, there are two message types, one for establishing a connection, and the other for breaking a connection.

- **MIDINet Message Type 4 - Establish a Connection**

MIDINet_Data = Source_MIDINet_ID + Source_Port + Source_Channel +
Destination_MIDINet_ID + Destination_Port + Destination_Channel

As for the 'Add a Device' message, the 'Establish a Connection' message is either broadcast or directly transmitted. A single message is broadcast when a user makes a connection, and multiple messages are directed to a new MIDINet unit in order to create its connection table. The message contains unique identification information for a MIDINet receiver (Source) and transmitter (Destination).

- **MIDINet Message Type 5 - Break a Connection**

MIDINet_Data = Source_MIDINet_ID + Source_Port + Source_Channel +
Destination_MIDINet_ID + Destination_Port + Destination_Channel

As for the 'Remove a Device' message, the 'Break a Connection' message is only ever broadcast, and this happens when a user breaks a connection via the user interface. Unique identifiers are provided for the transmitter and receiver.

6.1.3.4 The MIDINet Initialization Message Category

There are three messages in this category. The category was established to allow a new MIDINet unit, just joining the network, to obtain all the current configuration and connection information. The MIDINet unit asks who is currently attached to the network, upon which any attached units respond. The new MIDINet unit then requests configuration and connection information from one of these units.

- **MIDINet Message Type 6 - Who is out there?**

  MIDINet_Data = MIDINet_ID

  This message is only ever broadcast, and is sent on startup by a MIDINet unit which has just joined the network. It is a request by this unit for all other units on the network to identify themselves.

- **MIDINet Message Type 7 - I am here**

  MIDINet_Data = MIDINet_ID

  This message is sent by an active MIDINet unit in response to a 'Who is out there?' message. It is not broadcast, but is directed at the Ethernet address contained in the 'Who is out there?' message packet. The receiver of these messages can use them to build up a list of all MIDINet on the network.

- **MIDINet Message Type 8 - Send Request**

  MIDINet_Data = MIDINet_ID

  This message is usually sent as the last part of the initialization routine. It is directed to one of the MIDINet units which responded to the 'Who is out there?'message, and is a request to this unit to supply the current configuration and connection tables. The correct response to this message is a series of 'Add Device' and 'Establish Connection' messages.

6.1.4 An Evaluation of the MIDINet Approach

The MIDINet approach needs to be evaluated in terms of its suitability for the remote studio access system. To be suitable, it must:

- conform to the original goals of remote studio access,
- provide control over MIDI devices within acceptable time limits,
- be extendable to other network types.

6.1.4.1 The MIDINet Approach in the Context of Remote Studio Access

A MIDINet network is a general purpose MIDI-based network which can be adapted to a variety of uses, such as MIDI routing within single user studios, exhibition centres, and live sound installations. As shown in Figure 6.2, it an also be incorporated into the remote access system, but its generality detracts from its potential. The goals of the remote studio access system are to provide shared access to a remote studio via a single interface, where the sharing is managed to prevent contention for resources. In Figure 6.2, the MIDINet unit at the workstation side does indeed provide a remote MIDI patching facility, and allows for the transmission of MIDI messages to the remote resources. However, each user has MIDI patch control over all resources, not only booked resources, and the patching is done at an interface separate from the workstation.

Work has been initiated on a Windows interface to the MIDI connection and configuration tables [Freeman 1995].This allows a user to patch MIDI devices within a window on the workstation, and is based on the original MIDINet essential model. There are two approaches to dealing with the MIDI messages which are transmitted from the workstation sequencer. The MIDI messages can be encapsulated into a MIDINet packet and transmitted via an Ethernet driver which appears to a sequencer user as a MIDI driver. Such a MIDI-to-Ethernet driver has been written at Rhodes University and is described in [Howell 1994]. Otherwise a virtual MIDI driver which routes MIDI messages between programs can be used to route MIDI messages from the sequencer to the Windows-based MIDINet program. This was the approach used by Freeman, and the virtual MIDI driver used was the Turtle Beach Systems driver, packaged with Turtle Beach Quad Studio package [Turtle Beach 1994]. With this approach, the Windows-based MIDINet patcher is conceptually identical to the original MIDINet unit.

The Windows-based MIDINet patcher provides remote patching from the same, single workstation interface.

The next step is to make only booked resources accessible for MIDI patching. This will entail re-modelling the MIDI component of the remote studio access system to incorporate the MIDINet protocol. As in the case of the current MIDI patching specifications, only booked resources will be accessible for MIDI patching.

Although the current MIDINet network has limitations caused by its generality, it is usable within the context of remote studio access. However, a question which must be asked is whether the MIDINet network is capable of receiving, processing and forwarding the streams of MIDI messages arriving at its MIDINet unit ports. In the current MIDINet configuration, there are four MIDI ports within each MIDINet unit. Bearing in mind the MIDI transmission rate of 31.25 bits/sec, approximately 1000 MIDI messages could be transmitted down a MIDI line, although this is highly unlikely, as there will usually be delays between message transmissions caused by software at the source of a transmission. Furthermore, typical compositions do not usually generate this type of message stream. Indeed when a range of MIDI files were loaded and played from sequencers into the four ports of a MIDINet unit, a maximum of 125 MIDI messages/sec was recorded [Mosala 1994]. The time to process a MIDINet message, that is to extract the source and determine the destination from connection and configuration tables, was found to be 0.7 milliseconds, using a 486 IBM PC with a 33MHz clock rate [Mosala 1994]. This means that a MIDINet unit can service 1428 messages/sec, and is presented with a maximum of 125 messages/sec. Queuing theory can be used to determine an optimal buffer size for the receipt of messages from clients, but is really only useful when the mean transmission rate of the client is marginally less than the processing rate of the server. If the client's mean transmission rate exceeds that of the server, an unstable system will result [Jain 1991]. Assuming a Poisson distribution for the arrival rate of MIDI messages at a MIDINet unit, the probability of buffer overflow with a 10 message buffer, was found to be $3.17 * 10^{-11}$ [Mosala 1994].

6.1.4.2 Transmission Speed within the MIDINet Network

With the original remote studio configuration, the only area where real-time transmission was a concern was in the area of remote mixing. The bandwidth of 10 Mbps Ethernet is more than adequate to fulfill these real-time constraints. The transmission of MIDI messages over Ethernet from multiple workstations needs closer analysis. The multitrack sequencers on the various workstations have the potential to transmit far more data than the various patcher/mixers which have control over a limited number of patch nodes. The nature of the messages demand greater timing accuracy. A MIDI Note On message which arrives slightly late is more easily discernable than a lagging mix level change. Before starting out on an analysis of MIDINet transmission times, it is important to determine the meaning of 'late' in the context of MIDI events.

According to Moore, people cannot reliably distinguish which of two musical events happened first, when the time difference between them is less than about 30 milliseconds [Moore 1988]. Loy confirms this by

pointing out that sound travels at a speed of 1/3 metre per millisecond. A delay of 10 milliseconds represents a distance between musical sound sources of 3 metres. Players in a symphony orchestra are often spread out over 30 metres [Loy 1985]. However, both Loy and Moore stress that time delays even smaller than a millisecond, can be recognized in the pitch/timbre domain. Loy uses this as an argument against employing multiple MIDI synthesizers to create a single fused timbre. For our purposes, where MIDI is an assumed carrier of musical events, it is important that these events do not take longer than 30 milliseconds to be transmitted. However, if there is a transmission delay between the notes of a chord, then these notes will not appear simultaneous, they will appear 'smeared'. Ideally, there should be a transmission delay of no more than 30 milliseconds between the first and last notes of a chord. Although not common, ten-note chords can be played, and this means that a maximum transmission delay of 3 milliseconds between notes is desirable, although stringent. As an absolute rule, the time delay between two events should not exceed 30 milliseconds.

In order to get an idea of the workstation capacity of the remote studio access system, given the above time constraints, an operational model was built. This model comprised a number of IBM PC's linked together by Ethernet. All except two of these PC's generated a random number (1-10) Ethernet packets at random time intervals (1-500ms). The two exceptions each had a MIDI interface card. One of them was used to send a MIDI message over Ethernet to the second, and this second one retransmitted the message to the first via a MIDI link. The delay between transmission of the Ethernet message and receipt of the MIDI message was then measured for each of 10000 MIDI message transmissions. The packet numbers and time intervals of the load-generating PC's were chosen to provide a feasible amount of traffic from each workstation, and with the realization that there will be huge variation amongst the various stations. A diagram of the operational model is given in Figure 6.5 and a graph of maximum and mean delay times against number of workstations in Figure 6.6.

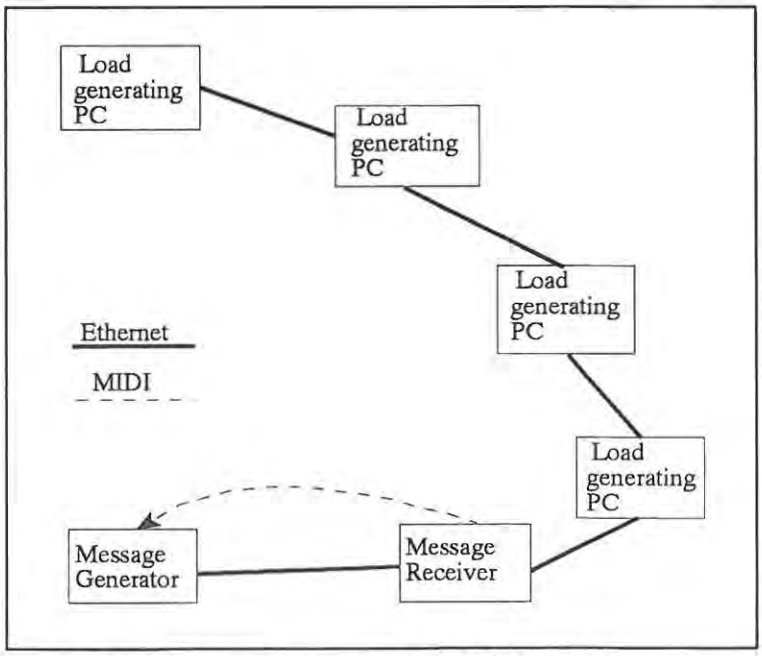

Figure 6.5 - Operational Model for MIDINet System

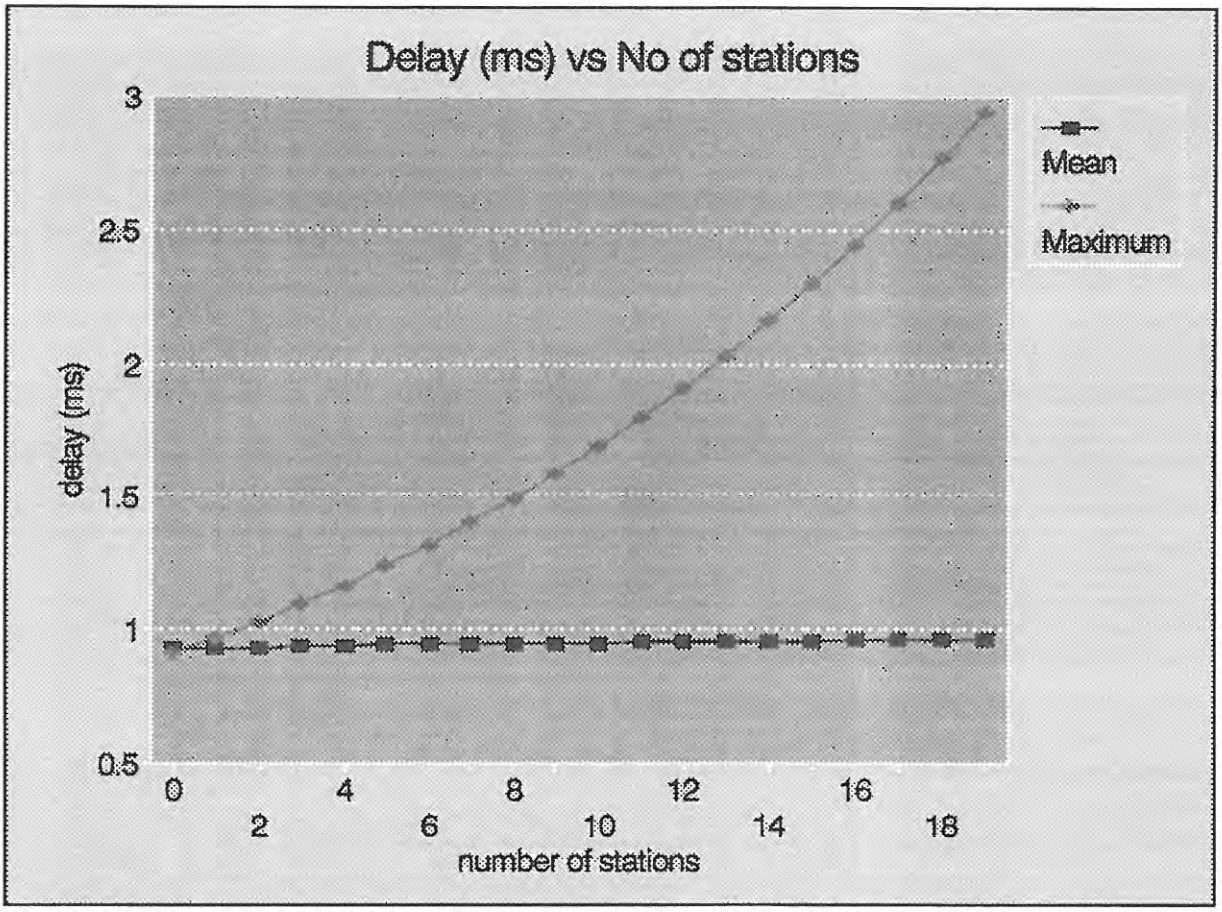

Figure 6.6 - Response Time of Operational Model with a Randomly Varying Load

The mean transmission time of 1 millisecond shown in the diagram is caused largely by the MIDI transmission delay on the return loop. Even with 19 load generating stations, the maximum packet delay was only 2 milliseconds, taking into account the delay caused by MIDI transmission. This low transmission delay is caused by the low load imposed on the network in this experiment. Hutchinson and Merabti have shown that if the network load is lower than a given threshold, the transmission delay time will always fit within a given period. Using simulation techniques, they got a maximum delay time variation of 7,5 to 28 milliseconds for a network load of 10 to 90% of the 10Megabit/second maximum [Hutchinson 1987]. Fober used an operational model comprising 7 stations, each transmitting and receiving one or more reference data flows. The reference data flow comprised 1000 note events per second, each note event corresponding a MIDI Note On and Note Off message. Fober's system does allow for grouping of MIDI messages, so it is unclear how many Ethernet packets this would resolve to. On loading the network with 25 reference data flows, the maximum transmission delay time was 22 milliseconds with a median of 3,5 milliseconds [Fober 1994].

6.1.4.3 Extending to Other Network Types

According to Tanenbaum, networks can be divided into two categories, those using point-to-point connections and those using broadcast channels [Tanenbaum 1988]. In this chapter, we have seen how MIDI messages can be transferred over a broadcast medium, rather than over point-to-point connections. As soon as a broadcast medium is used, there needs to be some method of resolving contention for the use of the common carrier. Two main protocols have evolved to resolve this contention, namely token passing, and carrier sense protocols. Stations which use a carrier sense protocol listen to see whether anyone else is transmitting on the common channel, and only if the channel is clear do they transmit. When a token passing protocol is used, a token is passed from station to station, and only when a station receives the token does it transmit data. Ethernet uses a carrier sense protocol, which is enhanced to detect collisions and act appropriately, if two stations happen to sense a clear channel and transmit simultaneously. Many variations on these two fundamental protocols exist, but two IEEE standards have emerged: the 802.3 carrier sense multiple access with collision detection (CSMA/CD) and the 802.4 Token Bus protocol. Ethernet, which has been used for remote studio access coordination, and now for MIDI transmission, is an implementation of the 802.3 standard.

Stations which use the Token Bus protocol are arranged physically in a bus configuration, but logically they are arranged in a ring, with each station knowing the address of its predecessor and successor. A token is passed from station to station on the logical ring. A station will only transmit data if it has the token, and when it passes on the token it specifically addresses its successor on the ring, irrespective of where it is physically located. The protocol has provisions for adding and deleting stations from the ring, as well as for determining whether stations are still active. These activities do make the protocol complex. The advantage of the Token Bus protocol is the determinism it introduces into transmission over a broadcast medium. A simplified version of the 802.4 Token Bus protocol has been implemented above Ethernet at Rhodes University [Foss 1995] [Mosala 1994]. All stations are assigned unique addresses, and these addresses are encapsulated, together with Token Bus protocol data and actual message data, into the data component of broadcast Ethernet packets. Effectively, the MAC sub-layer of Ethernet is viewed as the physical layer of the Token Bus protocol. There is no contention on the Ethernet network, since only the token holder transmits messages. This was borne out by analyses of MIDI message transmission [Mosala 1994]. However the Token Bus protocol, which was relegated to a high priority PC-Xinu task, did consume host processor resources and detracted from the overall efficiency of the MIDINet system.

An interesting implementation of the Token Bus protocol is the MediaLink protocol developed by the company Lone Wolf [Lacas 1993] and referred to in Chapter 3. The protocol has been specifically designed with multimedia applications in mind, and Token Bus was employed to provide determinism. There are a

number of packet types defined by the protocol, including MIDI, SMPTE, Video, Audio, RS232, and PA422 packets. The protocol is strictly defined from the data link up to the application layer, and a MediaLink chip comprising a CPU and I/O has been developed which implements the protocol. The data rate supported by the chip is 125 Kbps. The MediaLink protocol was investigated as a possible MIDI and studio access message carrier, but there is currently no IBM PC interface card or software library for MediaLink application development on the IBM PC family of computers.

Token Bus is a somewhat unnatural way of providing token-based transmission rights to a station on a network, in that a ring is created above a physical bus. A more natural way is to create a physical ring of stations. Each station is connected point-to-point to another station in such a way that all the stations form a physical ring. A special bit pattern, the token, circulates around the ring when all stations are idle. When a station wants to transmit data, it removes the token from the ring before transmitting, then regenerates the token. Any token ring network needs to have a monitor to oversee the ring, making sure that the token is not lost, taking action when the ring breaks, and ensuring that more than one token does not appear on the ring. The monitor function will usually be assigned to a station on the ring. There have been a number of implementations of the Token Ring concept, and an IEEE standard, 802.5, has been formulated. The ZIPI protocol, also described in Chapter 3, uses the concept of a Token Ring. In this case, the ring is encapsulated in a hub, and stations connect to the hub via a 7-wire cable with both directions of data flowing through it.

There are many arguments for and against the various network types described in this section. Ethernet is by far the most popular, and hence the least expensive option. Mbps Ethernet interface cards are about a quarter of the price of their 16 Mbps token ring counterparts, and the overall performance of the two are much the same. When the network is not heavily loaded, and when large packets are transmitted, Ethernet provides an efficient means of data communication. However, it has drawbacks. The smallest valid Ethernet frame is 64 bytes long, and for the type of real-time transmission performed in the remote studio system, there will be a lot of wasted bit space. Ethernet transmission is non-deterministic and at high loads the collision rate becomes a problem causing large delays in packet transmission. There is also no capability for message prioritization.

Unlike Ethernet, the efficiency of Token Bus and Token Ring is excellent under high load conditions, although there is some delay at low loads because a sender must wait for the token. Prioritization is possible with both protocols, and both short and large packets are allowed. Both protocols are deterministic when the networks are fully functional, but token loss can jeopardize this. The monitor function in token ring networks introduces a critical component, something which is not an issue for Ethernet where any station can be added and removed on the fly, without affecting the rest of the network.

There are high speed versions of star, bus and token ring networks, and these could all be used as carriers for control information. However price/performance ratios must be considered, and the control aspects of the remote studio system do not require the level of speed offered by these networks. A discussion of high speed networking will be deferred to the next section, when the problem of digital audio transmission is investigated.

It is important that the MIDINet protocol can be implemented above the various network types described in this section. At present, the protocol depends on the Ethernet data link layer to transmit packets directly to particular MIDINet units at connection time, and also to broadcast packets to all stations on the network. Both the IEEE 802.4 Token Bus and the 802.5 Token Ring destination fields have the same addressing capabilities as the Ethernet destination field, allowing both network types to transmit to individual and groups of stations. The ZIPI data link layer also provides the capability of directly addressing any station on the ring or broadcasting to all stations [Zeta 1994].

6.2 Alternatives to Audio Routing in the Remote Studio Access System

In the current remote studio system, audio lines run between an analogue audio patcher/mixer and the user workstations in a star-shaped configuration. For some time now, there has been a trend in recording studios towards converting sounds into the digital domain, and staying digital throughout the mixing, processing and recording phases. This trend is not all-consuming and there are many who still believe in the special quality of analogue audio [Hurtig 1992]. However, digital audio does not suffer from the accumulated noise levels which affects analogue audio as it is re-mixed and re-recorded.

There are two approaches which can be taken for the transmission of digital audio. The one is to use point-to-point connections, and effectively have the same configuration as for analogue audio. The second is to use some form of network to transmit the digital audio.

6.2.1 Point-To-Point Connections with the AES/EBU Interface

The Audio Engineering Society responded to the trend towards digital audio, and in 1985 produced a standards document with recommendations for the transmission of two channels of digital audio [AES 1985]. This standards document has formed the basis for all international standards documents concerning a two-channel interface. The document has been revised and updated, the most recent version having been published in 1992 [Finger 1992]. This standard is commonly referred to as the AES/EBU interface standard. There also exist a number of two-channel audio transmission formats specific to certain manufacturers such as Sony, Yamaha and Mitsubishi which are incompatible with the international standards and with each other.

For more information on these formats, refer to [Rumsey 1993]. AES have also produced a standard for multi-channel audio digital audio transmission, known as MADI - the Multichannel Audio Digital Interface - which carries 56 channels of audio data [AES 1991].

The AES/EBU interface is serial and self-clocking. The two channels of digital audio are carried in a multiplexed fashion over the same communications channel. The data is combined with a clock signal in such a way that the clock may be extracted at the receiver and used to synchronize reception. The transmission stream is divided into a number of frames, each frame comprising two subframes, handling channels 1 and 2 of the audio signal. Each subframe consists of a sync preamble, 20 audio sample bits, a validity bit, a user bit, a channel status bit, and a parity bit.

In the current remote studio system, two channels of analogue audio flow from the audio patcher/mixer to each workstation, and one channel flows from each workstation to the audio patcher/mixer. These lines could be replaced by lines carrying digital audio according to the AES3 format. Two physical lines would be required, one to carry the stereo signal to the workstation, and another to carry a mono or stereo signal back to the remote studio. The recommended physical carrier for AES3 format data is a shielded twisted pair cable, and the cable length can extend to up to 100 metres, although this can be increased with the use of repeaters and equalization at the receiver end [Finger 1992]. The transmission of digital audio data down the star-shaped audio transmission paths represents a huge improvement over the analogue system, which was subject to electrical interference and consequent signal corruption. What has to be determined is how to provide similar capabilities to the current remote studio system at both the workstation and remote studio sides, but now in the digital domain.

A number of interface chips are available from various manufacturers which allow for the receipt and transmission of digital audio in the AES3 format. An example is the Crystal Semiconductor CS8411 chip which receives AES3 format digital audio data, demultiplexes the data, and sends the actual audio sample data through a serial port [Crystal Semiconductor 1994]. This data can be read by a micro controller and transmitted to a digital-to-analogue converter for monitoring. There are IBM PC interface cards on the market which have AES/EBU input and output connections and which also have digital-to-analogue converters on board [ADB 1995]. A small 'desk unit' built for the United States senate chamber desks is described by Zweibel, and allows for the conversion between AES and analogue audio signals [Zweibel 1994].

On the remote studio side, some device is required to perform the patching and mixing function in the digital domain. However, before looking at the possibilities for patching and mixing in the digital domain, its important to be aware of the need for synchronization of digital audio inputs. If digital audio signals are to be combined with one another, their generating devices need to be synchronized to a common reference. In

this way their sampling frequencies are identical and do not drift in relation to one another. Although the sampling frequencies of the devices may be nominally the same, eg. 44.1 kHz, it is possible for differences in frequency of up to 10 parts per million to exist. When a signal drifts in relation to another combined signal, the audible result will be a glitch or click at a frequency equal to the difference between the two sampling frequencies, and at a level of around 50db below the signal. The Audio Engineering Society has published recommendations for the synchronization of digital audio signals, described in the document AES11-1991 [AES 1991]. The AES11 synchronization reference signal is identical in format to the standard two channel AES3 signal, and may contain sample data or not - the standard AES3 signal incorporates a clock signal using a bi-phase mark coding scheme. The AES11 standard states that machines should preferably be able to lock to the AES11 reference signal, and that they should have a separate input for such a synchronizing signal.

According to Rumsey [Rumsey 1993], there are essentially two ways in which standard two-channel interface systems may be routed and switched. One way is to use a conventional logic-based cross point switcher, and the other is to use a TDM (Time Division Multiplexed) router. A cross-point switcher simply switches inputs to outputs, and does not perform any processing at the patch points. In a TDM router the digital inputs are decoded and time-division multiplexed onto a fast parallel bus. Output routing is performed by extracting data in the appropriate channel time slot on the bus and assigning it to a particular output. An important feature of the TDM system is that different input formats can be handled by using appropriate interface cards. Output cards pick out the required samples from timeslots and reformat them into an AES bit stream. Thus analogue inputs are also possible, an essential feature in a remote studio where many of the synthesizers, samplers, effects units and recorders only have analogue inputs and outputs. In the Pro-Bel TDM router, it is possible to have 512 sources using a dual fast bus at 24 Megawords/sec [Ward 1993]. All inputs into a TDM router must be synchronous. Of course, for the analogue inputs, this is not a problem, since the analogue-to-digital converter clocks can be synchronized to the main TDM clock. Once the samples have been extracted from the bus, they could also be processed, if suitable digital signal processing hardware is available. A possible remote studio hardware configuration which utilizes digital transmission and audio patching/mixing is shown in Figure 6.7 below.

Only one synthesizer is shown generating digital audio, and this is a reflection of the slow rate at which the interface is appearing on synthesizers and effects units. After the introduction of the AES3 standard in 1985, there was a flurry of interest from manufacturers, and at the 1987 AES convention some effects units appeared with digital inputs and outputs [IMA 1987]. At the 1991 convention, Emulator, Yamaha and Kurzweil displayed synthesizers with digital interfaces [Schwartz 1991]. The trend towards digital has not been marked, although the advent of low-priced high quality digital mixers may well change this. The assumption in Figure 6.7 is that all the digital audio generators can accept AES11 synchronization signals, or that they can extract synchronization information from the clock within their AES/EBU inputs. This may not be the case, and buffering may be needed at their inputs as well, in which case the audio patcher/mixer must be capable of performing this buffering.

There are two IBM PC based products which allow for audio routing and processing of the audio at the patch points. Both are based around interface cards which contain Motorola's 56002 DSP chips. Peavey's MediaMatrix system is manufactured in a number of configurations, the most powerful being the MediaMatrix 900 which contains up to 192 inputs and outputs. Inputs and outputs can be either analogue or AES/EBU [Peavey 1995]. The system was used to provide routing of audio in the United States senate chamber and the construction of this application is described by Zweibel [Zweibel 1994].

Another, more recent product is the V8 card and break-out box from Digital Audio Labs [Digital Audio Labs 1995]. The V8 interface card, which contains the DSP hardware, is connected to the break-out box, which in turn has 8 analogue audio inputs and outputs, and 2 AES/EBU inputs and outputs. According to the Digital Audio Lab staff, the system can be expanded to provide up to a 32x32 input/output capability. This would imply an 8x8 AES/EBU input/output capability.

There are unresolved issues surrounding the use of these two systems for audio patching and mixing within the remote studio. A sound designer using the MediaMatrix system will select the audio processing devices required from a menu system, locate them on the screen, and then perform the necessary connections between inputs, outputs, and processing devices. This chosen configuration is then compiled, the compiler assigning the selected devices and wiring configuration to the DSP chips for the creation of a 'Virtual sound system' [Zweibel 1994]. In the remote studio access system patching and unpatching must be performed rapidly, and under remote control. The Peavey staff were unable to say whether this type of real-time operation would be possible. A software development kit is to be made available for the Digital Audio Labs V8 card and break-out box, but the functions provided by the kit are not yet accessible, and it is not possible to determine whether the hardware and software will have the functionality required for remote studio access. There is no indication in the specifications of either system whether there is the capability of using an external reference signal for synchronization purposes.

There are two major problems associated with the real-time routing of digital audio between sources and destinations. The one is the synchronization of the source sample transmitters, and the destination sample receivers. The other is the problem of bandwidth.

The designers of the ZIPI and Medialink protocols both had audio transmission in mind when they devised their hardware and software protocols, and both have suggested possibilities for synchronization. The data link layer of the ZIPI ring synchronizes the real-time clocks of all devices on the ring to within 50 microseconds, via a software protocol [McMillen 1994]. Interrupt latencies on most processors are in the 50 microsecond range, and this is the best resolution possible for a network-wide software-based clock. At a 44.1 kHz sampling rate, an audio sample needs to be clocked in every 22.676 microseconds, making such a software clock unfeasible for synchronization purposes. However, the authors of ZIPI have stated that it is possible to synchronize ZIPI devices to actual ZIPI clock line transitions, allowing for 4 microsecond resolution at the minimum clock speed. The Medialink designers have included the definition of a 'Sync Packet' at the data link layer of the protocol. The sync packet is used for precision system wide timing, and according to the authors has a 50 nanosecond resolution on a 20 megabit per second network [Lacas 1993]. It is not clear how this clock is generated.

A different approach is explored by Jacobs and Anderson [Jacobs 1994]. They suggest adjusting the sample clock rate of either the source or destination during transmission, such that buffer underflow or overflow does not occur. Of course, this assumes that some buffering, and thus a slight, inaudible delay can occur between the time of sample transmission and receipt. There are two alternatives, the source-driven alternative is to have the destination monitor the fullness of its buffers and vary its playback rate to match the source. There are two destination-driven alternatives: the destination can issue requests for data and the source can adjust its clock on the basis of the fullness of its buffers, or the destination can issue control flow packets to the source requesting that it adjust its clock. Destination driven protocols are problematic where a single source is broadcast to multiple destinations, since the source cannot match to more than one destination at a time. The simpler buffering scheme described in section 6.2.1 could also be employed, the buffer pointer simply being reset to the buffer midpoint. As Rumsey points out, a short crossfade could be introduced at the reset point to 'hide' the discontinuity, or the discontinuity could be arranged to occur within silent periods of the audio signal [Rumsey 1993].

Turning to the problem of bandwidth, the 10 Mbps Ethernet link connecting workstations and server would not be adequate for the transmission of multiple digital audio signals between remote resources and workstations. Furthermore, the transmission of audio would jeopardize the real-time transmission of MIDI and patch level change signals. Audio can be transmitted down Ethernet [Rybacki 1993], but where more than

one channel is used, the application is usually in the area of low-grade voice communication. Anderson suggests that Ethernet is realistically limited to a single 44.1 kHz stereo transfer [Anderson 1993]. Yonge is more dismissive, writing that "the idea of multiple users accessing multiple channels of digital audio over a conventional LAN simultaneously and reliably just isn't serious" [Yonge 1993]. In the same article where he discounts Ethernet (even 100 Mbps Ethernet), Anderson describes the use of FDDI for the SonicNet system of Sonic Solutions. The advantages of FDDI are that it is a high speed network, and its 100 Mbps data rate makes possible simultaneous transmission of more than 100 channels of 44.1 kHz (CD-quality) digital audio. FDDI's token ring configuration allows the network to provide close to hardware data rates under heavy load conditions, something which is not possible from 100Mbs Ethernet, with its collision detect random backoff medium access policy. SonicNet does not allow any device to transmit digital audio onto the network at any time. It applies a 'reservationist' policy to digital audio transmission. Before an application begins data transfer, it must reserve the needed bandwidth, and will not be able to perform the transfer if bandwidth is not available. However for applying the reservationist policies and for general control of the SonicNet network, standard protocol suites such as Appleshare and TCP/IP are used. This traffic is 'bursty' and does not impact the performance of the audio transfers.

A network protocol which incorporates the 'reservationist' policy and which is referred to by Anderson, is ATM (Asynchronous Transfer Mode). ATM was originally designed to carry both voice and data across the international telephone network, but is now being extensively utilized by the private networking industry [Alles 1993]. ATM is a connection-oriented technology, and before data transfer can occur between two points on the network, a connection needs to be established between those end points using a signalling protocol. Small fixed size (53 bytes) cells are used to carry all types of data. Each cell contains two fields - a Virtual Path Identifier and a Virtual Circuit Identifier, which jointly serve to identify connections. A set of VPI/VCI mappings is programmed across an ATM network, from one terminal to another through any number of intermediate switches. The switches identify cells belonging to the connection and switch them correctly. Before a source device sends data to a destination, it must request bandwidth and the connection must be set up. A problem with ATM at the moment is that the price of the switches is very high, each switch being the price of a fairly comprehensive single user studio. At least two of these switches would be needed, one at the workstation side, and one at the remote studio side.

It would appear then that FDDI offers the best solution for a carrier of control and audio data, unless the physical speeds of the ZIPI or medialink protocols are enhanced. As in the case of the AES/EBU digital audio transmission, the nature of the transmission endpoints has to be formulated. At the remote studio side, there must be some type of 'AudioNet' unit which can read either analogue or digital inputs, and transmit these over the FDDI network. There would have to be more than one of these units to connect to the range of remote studio devices. An AudioNet unit must also be able to pick up digital audio from the network and

transmit it to digital or analogue outputs. On the workstation side, each workstation will need to pick up multiple channels of digital audio, mix them and transmit the mixed stereo signal to a digital-to-analogue interface card for monitoring. From a conceptual point of view, the capabilities of the AudioNet unit are very similar to the MIDINet unit, where MIDI merging now translates to mixing in the audio domain.

A question which must be asked is whether this all-inclusive network concept could be implemented immediately with off-the-shelf components, or in the future with customized hardware. Up till now, the IBM PC has been used for the implementation of all the remote studio access concepts. There are FDDI cards available for the IBM PC, compatible with both the ISA and EISA busses. The price of these cards is approximately five times higher than 10 Mbps Ethernet cards and about twice the price of 100 Mbps Ethernet cards. There are also a number of manufacturers of 'sound cards' for the IBM PC [Rideout 1994]. These cards provide digital-to-analogue and analogue-to-digital conversion capabilities, and often incorporate MIDI input and output ports. They usually allow for DMA (Direct Memory Access) transfer of audio samples to and from the IBM PC RAM. Two interrupt controllers and two DMA controllers on the IBM PC motherboard provide the means for efficient communication between external I/O devices and the PC [IBM 1988]. On typical IBM PC systems, there are five hardware interrupt lines and five DMA channels available for use by interface cards plugged into the motherboard. This would allow for the use of a network card and four sound cards within a PC.

The use of four sound cards for the transfer of digital audio is feasible, and this configuration is suggested by the company SEK'D in their Samplitude hard disk editing/recording program, Samplitude [SEK'D 1995].However, SEK'D also suggest the use of a large memory configuration (16 Megabytes) with a Pentium processor for this configuration. With currently available hardware, it would be more feasible to incorporate the MIDINet and AudioNet concepts into one implementation. A single soundcard could provide MIDI input and output capabilities, as well as two audio inputs and outputs. Both Roland's RAP-10 [Roland 1993] and Turtle Beach's Monterey or Multisound [Turtle Beach 1992] sound cards provide this capability, and have selectable hardware interrupt and DMA channel numbers. A problem with most of these soundcards, which are not aimed at the professional market, is that they do not incorporate the AES/EBU interface for digital audio input and output. Furthermore, the sample clock rate on the cards is not controllable from an external synchronization source.

A second option is to treat the IBM PC as a controller and coordinator of activities, and to construct an interface card specially for the application. The card could comprise a fast processor, the necessary digital-to-analogue and analogue-to-digital circuitry, AES/EBU circuitry, FDDI circuitry, and possibly MIDI interface circuitry. The design of a low-cost FDDI interface card is described by Mahavadi [Mahavadi 1993]. Routing information could be passed from the IBM PC to the card. Digital audio would never have to pass through

the PC bus or its CPU. This is the approach used to implement SonicNet, where the host machines are Apple Macintosh's [Anderson 1993]. The two major advantages to this approach are the increase in speed and the control gained over sample clocks. The disadvantages are the time required for hardware development in a field where technology is continuously changing, and the dependance on specialist technology. A diagram of a possible remote studio access system using Audio/MIDINet units is given in Figure 6.8 below.

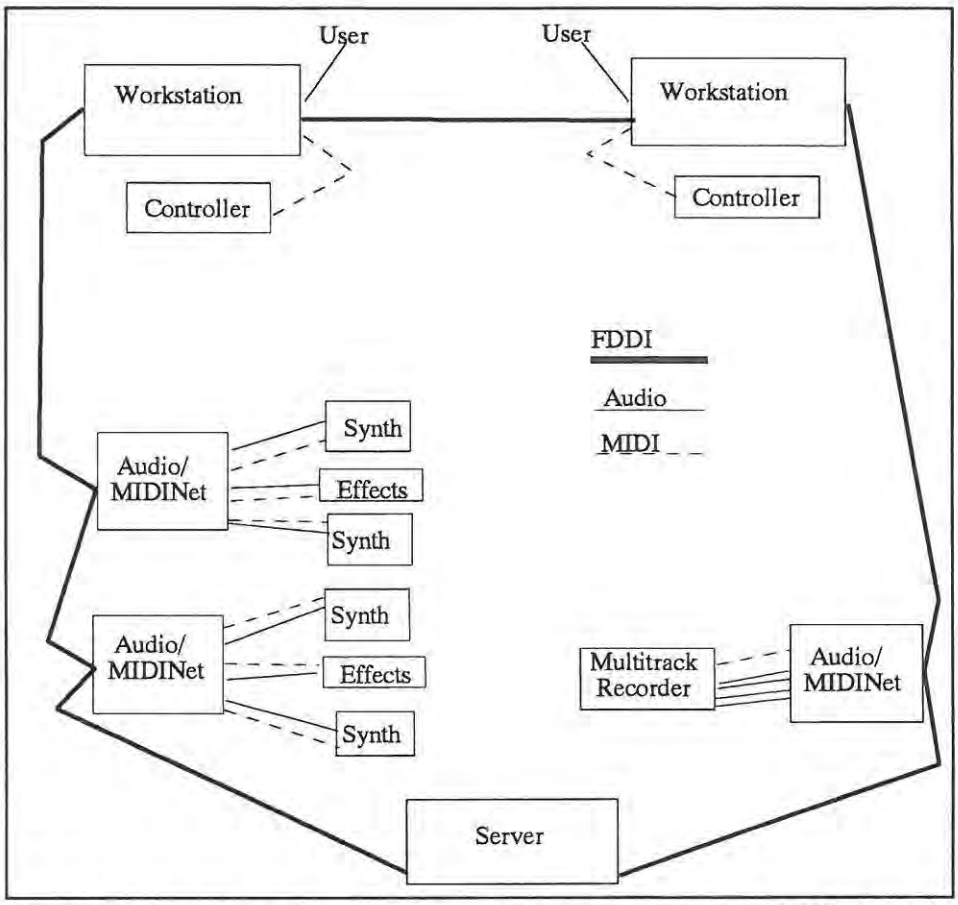

Figure 6.8 - Remote Studio Access System Incorporating Audio/MIDINet Units

In this network, the server only plays the role of a booking manager, ensuring that there is no contention in the access of remote resources by the outlying workstations. A master synchronization unit could be incorporated, as in Figure 6.7, thereby ensuring that the sample clocks of all digital audio generators run at the same speed. If the multitrack recorder is a hard disk based unit, it becomes far more feasible to have it directly connected to the network, instead of utilizing an Audio/MIDINet unit. This would mean building a recorder along the lines of the SonicNet hard disk recorders, or utilizing current hard disk recording/editing software and modifying its sample input and output drivers to read and write to the network.

The ideas in this section, while largely speculative, do indicate that there is a possibility of audio and control data transfer down a common network. While still in the domain of speculation, it is appropriate to consider the possibility of remote studio access in the wide area network sense, i.e. truly remote access. Up to now, our view of remote access has been in the context of local area network access.

6.3 Long Distance Access to Remote Studio Resources

The most obvious choice for transmitting audio and data over long distances is ISDN (Integrated Services Digital Network). ISDN was designed to be the digital upgrade to the existing public switched telephone network, where many of the exchanges are already digital. User information is conveyed on 64 kbits/sec channels, known as B channels. There are two access possibilities to the network:

- Basic Rate Access which offers two B channels at 64 kbits/sec and a D channel at 16 kbits/sec for signalling, and

- Primary Rate Access which offers 30 B channels at 64 kbits/sec and a D channel at 64 kbits/sec.

Nearly all telephone line connections can be used for Basic Access, whereas for Primary Rate specialized lines must be used. The telephone line on the user's side is terminated by a network termination unit (NT). The NT provides a so-called S-interface to which dedicated ISDN equipment can be connected. The S-interface in turn provides an S-bus to which up to 8 ISDN terminals can be connected. The terminals can be special ISDN terminals like an ISDN telephone, or terminal adaptors used for connecting existing non-ISDN equipment to the S-interface through common interfaces like RS232. ISDN cards exist which connect directly to the S-bus, and some also have Ethernet interface chips on-board which allow for direct connection between ISDN and Ethernet. [DIGI 1995].

Nielsen has implemented a small ISDN-based system which allows a user to transit MIDI data to a remote synthesizer, and receive the audio signal back from the synthesizer [Nielsen 1994]. Digitizing of 44.1 kHz audio leads to a bit rate of 705 kbits/sec, far higher than the capacity of a B-channel, and in Nielsen's application the returning audio was compressed using MPEG encoding techniques. A good overview of MPEG encoding and decoding techniques is given by Dietz et al [Dietz 1995].

An ISDN connection is bi-directional (full-duplex), and so two 64 bits/sec B-channels are available in each direction. Regardless of the remote studio configuration, two channels of audio and one MIDI channel need to flow from the remote studio to each workstation, while one channel of audio and one channel of MIDI are transmitted from each workstation to the remote studio. Assuming the remote studio configuration described

in section 6.1, where analogue audio was transmitted along point-to-point connections between workstations and patcher/mixer, there would need to be two computer stations allocated for the provision of single workstation remote access. Two ISDN lines would also need to be allocated. The station on the remote studio side would be fitted with analogue-to-digital and MPEG encoding hardware, MPEG decoding hardware, an Ethernet interface card, and an ISDN interface card with a 2-line capability. Four ISDN channels would be used to transmit MIDI, server control data, and two audio channels to the workstation. The workstation would need to have MPEG decoding and digital-to-analogue conversion hardware, an ISDN interface card with a 2-line capability, and MPEG encoding hardware. Three ISDN channels would be used to transmit MIDI, workstation control data and a single audio channel back to the remote studio. This configuration is shown in Figure 6.9 below.

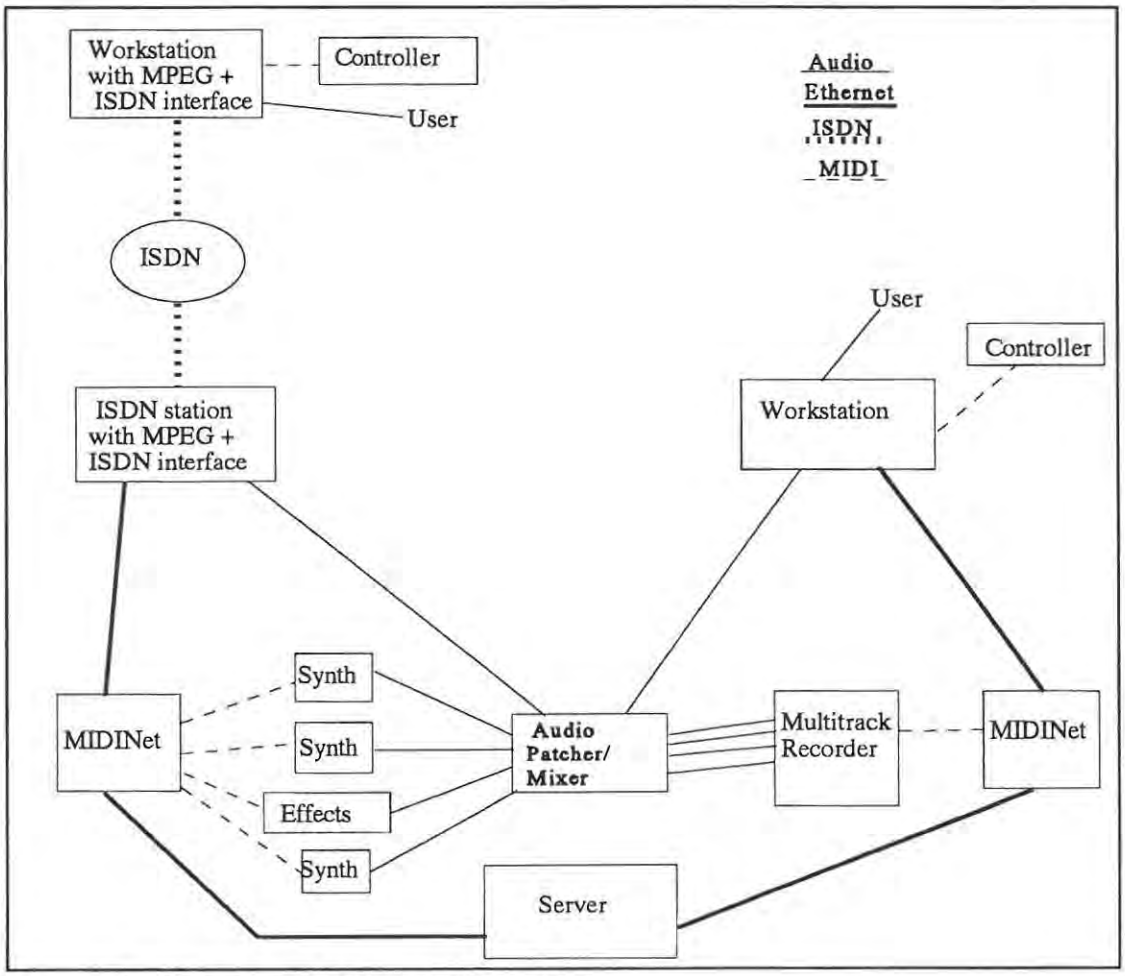

Figure 6.9 - ISDN Access to Remote Studio Resources

The above configuration, with its need for audio compression, is not ideal. Anderson has noted that compression is difficult to use in most areas of professional audio, because compression schemes which

achieve significant compression ratios tend to be 'lossy' [Anderson 1993]. Such compressed sound, when subsequently de-compressed, tends to exhibit artifacts of the original compression process. Nielsen has described methods for handling signals at higher bit rates over ISDN, such as 256 kbits/sec [Nielsen 1994]. By inverse multiplexing, this signal is split into four separate 64 kbit/sec B-channels at the transmitting side, and recombined into the original 256 kbits/sec at the receiver. A method known as BONDING (Bandwidth ON Demand INteroperability Group) provides for inverse multiplexing up to 63 B-channels. Nielsen also indicates that primary rate ISDN can be used for the transmission of AES/EBU digital audio signals. Videira and Casaca describe the design of an IBM PC interface card which will allow for the transmission of data between an Ethernet LAN and primary rate ISDN link. However, the applications which they describe are data oriented, where multiple streams of data could be transferred over the 30 ISDN B-channels, and they do not address the high speed transmission of a single stream of digital audio [Videira 1993].

ISDN as described up till now, is more accurately termed N-ISDN, Narrowband ISDN. In the mid-1980's a CCITT (Consultative Committee on International Telephone and Telegraph) began working on a successor to N-ISDN, known as B-ISDN, Broadband ISDN. In 1988, the CCITT decided to base the development of B-ISDN on ATM. ATM was described in section 6.2 and the current prohibitive expense of its switches pointed out. However, with its increasing deployment for the transmission of multimedia and data in public wide area networks, it is appropriate to look at its potential for providing remote studio access. This becomes economically feasible if ATM switching units are shared with other applications.

There are two major types of interfaces in ATM networks, the User-to-Network Interface (UNI) and the Network-to-Network Interface (NNI). An ATM network comprises a set of end-systems (terminals) and a set of intermediate nodes (switches) all linked by a set of point-to-point ATM links. The terminals are linked by UNI interfaces to the switches, and the switches by NNI links to each other.

There are three protocol layers which are fundamental to the transmission of ATM cells from a source to a destination. The ATM Adaption Layer (AAL), the highest of the three, takes whatever data is to be sent across the network, establishes the appropriate connections, then packages the traffic into 48 byte chunks which is passed down to the ATM, or second layer, for transmission. In the reverse direction, the AAL layer must receive these 48 byte chunks from the ATM layer and put them into a form expected by the higher level receiving software.

The operation of the ATM layer varies depending on whether it is an end-point or a switch. When the ATM layer is an end system, it exchanges a cell stream with the physical layer, generating cells from the information chunks passed down to it from the AAL layer. It is responsible for adding the appropriate Virtual Circuit values for the correct routing of the cells. In a switch, the ATM layer determines from the Virtual

Circuit value the port to which the cell should be relayed, the new Virtual Circuit value for the next stage of the link, and then transmits the cell. Multicasting is currently supported by the input controller on a switch replicating the cell, and submitting each copy to the switch fabric. This issue of multicasting is important for the remote studio application where the MIDINet protocol depends on a sub-carrier to broadcast its packets to all stations on the network.

The bottommost layer is the physical layer, and one of the attractions of ATM is that it can operate over a variety of physical layers. SONET (Synchronous Optical Network) based interfaces are used predominantly in public ATM networks [Rooholamini 1994]. However, the ATM Forum has defined a variety of physical layer standards, one of them being the 100 Mbps fiber interface used at the physical layer of FDDI. There are two choices of switches at the hardware level, the two options having a similarity to the two choices for the switching of AES/EBU signals. In backplane-based switches, ATM cells are transported across a bus linking ATM modules. This is similar to the TDM approach for digital audio routing described in section 6.2.1. Matrix switches, the second type of cell switches, are similar to cross point switches.

As pointed out in section 6.2.2, ATM switch units are expensive. ATM interface cards for IBM PC based end-point units are available, and they too are expensive, almost twice the price of FDDI interface cards. However, with the rapid growth of ATM units, these prices should decrease. It is interesting to note that at least one manufacturer of digital audio equipment is currently working on an ATM solution to audio file transfer. The company is SADiE, a UK-based manufacturer of hard disk recorders and editors for the IBM PC platform. The company foresees ATM as being the 'information superhighway of the future', and that customers will require the capability of local and wide area networking of audio [Lockwood 1995]. Figure 6.10 below gives a possible ATM-based hardware configuration for remote studio access.

There are many advantages to using ATM. The speed of the physical layer ranges from 45 to 155 Megabits/sec, although there are SONET implementations which allow up to 1.244 Gigabits/sec [Allis 1993]. The connection-oriented nature of ATM means that data rates are guaranteed. The fact that public networks are embracing ATM technology means that long distance access will be possible from a variety of locations, and that the price of ATM technology will drop.

Figure 6.10 extends the speculative configuration of section 6.2.2. Rather optimistically, it shows the audio and MIDI links of four resources connected to an Audio/MIDINet unit. Of course multiple Audio/MIDINet systems could be used with less load per unit. However this does raise the issue of station overload when high speed networks are being used. Steenkiste estimates the peak application-to-application network throughput that workstations can support to be in the region of a few tens of megabits per second, even though the network bandwidth is 100Mbps or more [Steenkiste 1994]. He has shown that data crosses the memory bus

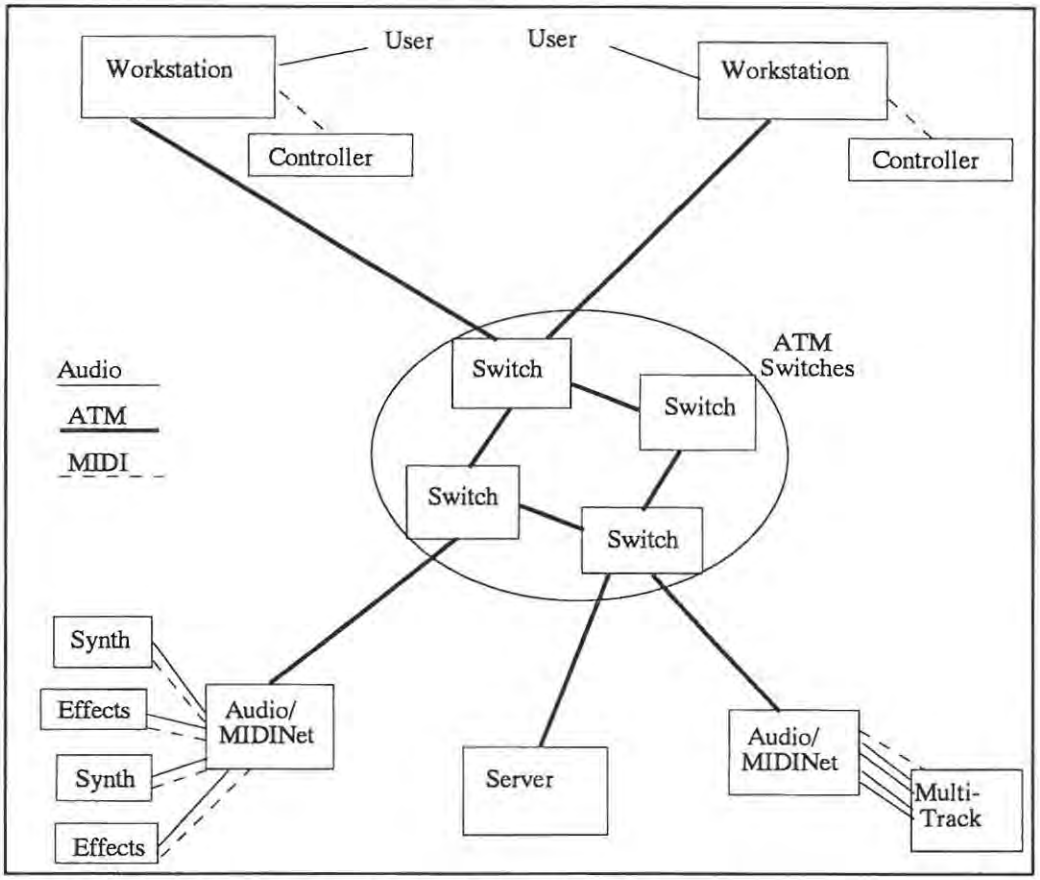

Figure 6.10 - ATM Access to Remote Studio Resources

six times in the traditional network interface, and that the host's memory bandwidth becomes a limiting factor for network bandwidths of 100 Mbps and upwards. Steenkiste has shown how throughput can be increased by optimizing the networking software and using more powerful network adaptors

6.4 A Perspective on Remote Studio Access Technologies

The ATM proposal shown in Figure 6.10 pushes the implementation of the remote studio access concept to its limits with currently available technology. Of course, this high speed state-of-the-art communications technology has its price, which must be balanced against the cost of the resources being utilized. It is time now to step back, get perspective on all the technologies that have been introduced, and see how the needs of the user can best be served. In particular, we need to glean the essential characteristics of the various technologies, and see how they support our three-pronged goal of shared access to a remote music studio from a centralized interface.

This will be the agenda for the final chapter, Chapter 7.

Chapter 7

Conclusion

This thesis was inspired by a desire to see to what extent networking technology could be used to enhance the use of music studios. Apart from this practical standpoint there was also the fascination of being able to manipulate all the resources of a large computer music studio from a remote site and from a single personal computer interface. Throughout the investigation, three themes, first introduced in Chapter 1, have continuously emerged:

- **Remote Access** - essentially meaning studio access from a location different from that of the studio resources.
- **Shared Access** - more than one person can access the studio at the same time.
- **Centralized Access** - all the music studio resources are accessible via a single interface, the user does not have to move from one resource to another to complete a task.

Various network-based solutions were modelled and implemented in an attempt to provide shared access by many users to a remote studio, each user controlling resources via a single interface. The investigation has followed an evolutionary path, each stage in the investigation pointing the way to the next evolutionary step. In this chapter we will first look back on these various stages and see to what extent they provided shared, remote, centralized access.

7.1 The Evolutionary Stages of the Remote Studio Access System

Three configuration types can be abstracted from the various configurations which have been proposed.

7.1.1 MIDI and Audio in a Star, separate from Studio Control data

This was the first configuration to be modelled and implemented. The philosophy behind this configuration was to use a bus network as a facilitator for a number of workstations. The workstations all have requests for a 'studio manager' which are transmitted down the network. The studio manager, residing on a server

machine in a remote studio, fields these requests and deals with them by performing MIDI patches and audio patches on booked resources. The power of this system lies in the fact that it is simply a facilitator, allowing current commercial MIDI software to control studio resources in the same manner as they would in a single user studio. All the system does is to provide clear control and audio paths to and from the workstations. There are two important items of hardware in this configuration, the audio patcher/mixer and the MIDI patcher. These must be capable of expansion and of receiving control commands from the server. Units were custom-built for this configuration, although other hardware can be used as long as it conforms to the specifications. The server can be modified in such a way that it issues control messages according to the protocol of this other hardware. The audio in this star configuration could be either digital or analogue, the principles behind the configuration remain the same. Digital sound generators need to have their clocks synchronized, and in this respect it is useful to have all the generators in a single remote studio location. This synchronization with audio from the workstations is more difficult, although a clock can be derived from an AES/EBU input signal.

A disadvantage of the configuration is that it limits remote access to the studio to 15 metres, the maximum length of a MIDI cable as prescribed by the MIDI standard, although alternative methods are used to convert the MIDI signal to a transmission standard with longer transmission lengths. A further disadvantage is the amount of cabling required. Two MIDI cables, three audio cables (for analogue audio) and an Ethernet cable all need to link a remote workstation to the studio.

From a user's perspective, the hardware configuration is not important, and the configuration certainly does satisfy the criteria of shared, centralized access. Studio resources can be booked and patched, their audio outputs mixed, processed and recorded, all from a single personal computer interface. Furthermore the full range of commercial MIDI software can be utilized from this same interface. One pitfall in the sharing of resources may be a limitation on channel changing. As discussed in Chapter 5, some MIDI controllable resources do not provide a system exclusive message for channel changing, while others may need to be in a particular state before receiving a channel change system exclusive message.

7.1.2 Audio in a Star, MIDI combined with Studio Control data

The next evolutionary step attempted to rectify the limitations on remote access and channel changing, and to move a step towards a more elegant configuration where all communication between workstation and server is carried across a single line. To this end the 'MIDINet' concept was evolved and then implemented. MIDI can be fed to and received from ports on MIDINet devices, which can be stationed in the remote studio, or in proximity to the workstations. Each MIDINet device provides a user interface which allows connections to be made between devices. MIDI messages are tagged with source device details and can be pulled off the

network by any connected devices.

The MIDINet data flows together with the studio control data and the question of whether real-time transmission is possible was posed. This question is a key one and points to a fundamental trade-off, gaining configuration simplicity at the expense of possible indeterminacy. The Ethernet solution used was found to have sufficient bandwidth for the relatively low requirements of MIDI and studio control data.

The configuration using MIDINet devices at the workstation side is not altogether satisfactory, in that it disperses MIDI patching to a separate interface. It also allows patching to be performed by a particular workstation user on all studio resources, not only the booked ones. To bring this back to our three guiding themes, it does not provide complete centralization of studio access, nor does it provide controlled contention-free sharing in the area of MIDI control. This situation can be rectified by re-modelling the MIDI patching aspect of the remote studio access system, and implementing MIDI patching software on the workstation side which conforms to the MIDINet protocol.

7.1.3 MIDI and Audio combined with Studio Control data

Although this is a logical evolutionary step, it represents a quantum leap in terms of the implementation technology. Audio has far higher transmission bandwidth requirements than studio and MIDI control data. Indeed, in most applications where control is required over sound systems, the sound control and audio transmission paths are separate. A number of transmission media were reviewed and FDDI found to be a good candidate. Even if a transmission medium is used which allows for the combined transmission of audio and control data, the receiving stations need to have the capture and processing capabilities to cope with these large amounts of real-time data.

The construction of an 'Audio/MIDINet' system was proposed, similar in concept to the MIDINet unit, but allowing for the transmission of audio streams from device to device. The MIDI patching and audio patching/mixing which was once centralized at MIDI and audio patching units within the remote studio, is now dispersed across a number of networked nodes. Audio/MIDINet nodes and workstations may need to pick up multiple channels of audio from the network, mix them and retransmit the mixed signal to the network or to an audio port. As soon as multiple streams of digital audio need to be mixed, the problem of synchronization of audio generators arises. When the streams are being read from networked file servers, this is not a problem, as the files can be transmitted faster than real-time over a network medium such as FDDI, and then mixed from buffers at the destination. However this is not possible when the audio streams are being generated in real-time, and methods to achieve synchronization are discussed in Chapter 6.

If the problems of bandwidth and synchronization are resolved, the configuration can be used to fulfill the goals of remote, shared, centralized access to music studio resources. The networking technology used will define the meaning of 'remote', up to 200 km for FDDI [Tanenbaum 1988]. The possibility of extending the access distance was investigated in Chapter 6. Both ISDN and ATM were proposed as possible transmission technologies. ATM with its high transmission rates and rapidly gaining popularity looks the most promising.

Interestingly, ATM has a star configuration, where virtual channels are reserved for communication between sources and destinations. Physical switch units are used to perform routing from sources to destinations. Separate channels could be opened for the transmission of MIDI, studio control and audio data. Just as in the original star configuration, MIDI and audio are guaranteed an uninterrupted pathway from sources to destinations. The difference is that the workstations and studio resources could be scattered around the world!

7.2 Resource Control in the Remote Studio Access System

In all these various configurations, there is always a server which maintains control over resource usage. The server will only release information to a workstation about the resources which it has booked, and only registered users can use workstations. While the ability to book resources may seem a trite management level add-on to more basic resource access, it has fundamental implications for the features and implementation of the system:

- **User Planning** - Users can plan their use of studio resources, booking relevant resources for the various stages of their projects.

- **Studio Resource Planning** - If user access to resources and workstations is logged and subsequently reported, statistics can be kept on resource usage, and the studio resources adapted to the demand.

- **Charging** - Users can be accurately charged for resource usage, making such a network a commercial proposition. Users need only be charged for the resources which they use, not the entire studio.

As far as the implementation is concerned, there must be a Resource Manager which accurately times resource usage and stores this information for later reporting. When each new booking slot starts, the Resource Manager must ensure that all workstations receive updated resource list and that all connections which were made with previously booked resources, are broken. Of course, the method for breaking connections will depend on the configuration in place. For this implementation to work, data needs to be maintained regarding all the resources and their connections, and control protocols have to be adhered to.

Most importantly, clear simple models of the interactions need to be built, and this principle applies to all components of the system

7.3 Modelling as an Integral Part of System Construction

The remote studio access system is a distributed system, requiring real-time response. It is a large system with a number of complex interactions. As seen in section 7.1, three fundamental hardware configurations have been proposed, and on top of each of these configurations, software needs to be constructed. Software modelling tools have been invaluable for clearly and rapidly describing the operation of the software. After modelling the system, its operation could be traced using the model, and its feasibility determined. The first specification and configuration for remote studio access was rejected after modelling it. If coding had been initiated directly a large time investment would have been wasted.

The modelling tools of both Shlaer and Mellor, and Ward and Mellor were used consistently from the modelling of the embedded patcher systems to the networked MIDINet system. As discussed in section 5.2, these tools provided the necessary mix of object description and control dynamics, while the event-response paradigm of the tools matched the event-driven nature of the implementation operating system, Microsoft Windows. Object-oriented modelling tools provide a more consistent path from modelling diagrams to implementation in a language such as C++. The ideal would be to retain the event-response paradigm, and to apply the object-oriented approach throughout the modelling stages. Shlaer and Mellor incorporate these two facets in their second book [Shlaer 1992].

7.4 Multitrack Recorder Control

Throughout this thesis there has been one resource type which has resisted complete 'hands-off' control from a remote site. In the case of multitrack tape recorders, a tape needs to be loaded at the start of a session, and unloaded when the session ends. In the case of a hard disk multitrack recorder, space on the hard disk needs to be allocated for a project, and this space backed up at the end of a session. While a session is in progress, multitrack control is possible via standard MIDI Machine Control commands, and there is also the possibility of remote control via proprietary software and hardware, as described in Chapter 5.

Interestingly, access to networked sound files is a capability which has been successfully provided by one or two commercial sound system manufacturers. This is mainly because high speed networks can transmit sound files at speeds which are faster than real-time, and synchronization between source and destination is not a problem. In these systems, there is usually networked access to large file storage centres, and backup can be done from a local workstation. Proprietary protocols are used for the rapid transmission of these sound

files. The third configuration discussed in section 7.1.3 would be enhanced by the provision of such a facility, and together with the 'Audio/MIDINet' units provide a more complete remote studio access facility. However, the facility would have to be a proprietary one and is contrary to the 'facilitator' role played by the current system. It might be that a protocol such as Sonic Solution's MediaNet becomes a standard and that hard disk recorders emerge which observe and respond to this protocol.

7.5 The Complete, Homogeneous Workstation, a preferable solution?

The remote studio access system provides access to a cluster of heterogeneous music studio resources, all manufactured by a range of different companies. It depends on the MIDI standard to provide a way of controlling all this diversity. The philosophy of the system is to be, as far as possible, inclusive, and allow access to a wide range of resources. At the other end of the spectrum is the 'complete workstation' approach which provides powerful signal processing capabilities from a single workstation. Sound generators and processors are implemented in software and run on fast general purpose processors or more specialized digital signal processing hardware. The system is flexible and open-ended in that new generators and processors can be incorporated in the form of software modules. This was the approach used in the Synclavier, which for many years was a commercial success. As described in Chapter 3, remote, shared access was provided to its 'Pool' of resources. However, competition is fierce within the music industry and this drives companies to research and implement new sound generation and processing techniques. Inevitably, musicians will want to add these to their sound palettes. Synclavier yielded to this trend and provided MIDI interfaces on their systems. In later years customers were attracted by their hard disk recording and editing capabilities rather than their sound generating potential. This attempt to be 'all things to all men' may well have been the downfall of the Synclavier, requiring huge effort, and thus expense, to maintain an all-purpose system in an industry of specialists.

A competitive music industry also leads to low priced music resources, and it may well be argued that multiple small studios are preferable to a networked pool of resources. However, there will always be leading edge, powerful resources which carry a high price tag, and which need to be accessed at times by different users. A mixed system of low-priced local resources and networked expensive ones is possible, but it runs contrary to the simple, centralized interface which characterizes the remote studio access system. The approach in the remote studio access system is to configure all resources into the system and let them be used in the same, congruent fashion. However, in a commercial setting, it would be possible for a user to develop a piece of music using simple, inexpensive sound resources, to transfer a MIDI file to a network workstation, and then use the more powerful remote studio resources to add timbrel quality.

7.6 Replication of Resources.

In Chapter 1, the resources of a typical music studio were categorized into four groups - sound generators, sound processors, sound mixers, and sound recorders. Chapter 2 described typical scenarios in educational and commercial music studios, where particular, often desirable resources could only be accessed from certain studios. The network configurations described in this thesis overcome the limitations of resources being bound to particular work areas, and the inconvenience of resource relocation.

However, another solution to the limitations of resource access is resource replication. If there are multiple studios, then each studio can be fitted out with a comprehensive set of sound resources, so that there is no contention for the use of resources. A question which is then posed is: at what point does it become more cost-effective to install a network rather than to replicate resources? However, this question does not take into account the dynamic nature of resource usage on music projects. It may well be simple and inexpensive to configure multiple studios each having the same minimal set of resources, but inevitably certain music projects will require more than the minimal set, and reconfiguration will occur. The network configurations described in this thesis allow this dynamic reconfiguration to take place quickly and easily.

The most cost-intensive part of the two implemented network configurations, is the audio patcher/mixer. The price of the custom-built patcher/mixer with a 32x32 input/output capacity approximates the price of a typical 24 channel studio mixer which would have to be located in one of a suite of studios in an educational or commercial complex. The patching and mixing capabilities of the audio patcher/mixer can be assigned simply and dynamically to various users at various times. The price of this custom-built unit is also on a par with the IBM PC-based digital audio patcher/mixers described in section 6.2.1. Additional costs include either a centralized MIDI patcher, or distributed MIDINet units. These costs compare favourably with the cost of providing automated MIDI patching capabilities in each of a number of studios.

7.7 Future Work

The concept of powerful yet simple remote studio access is an exciting topic and prone to fantasy! Some of this fantasizing has become reality in the current remote studio access system. Other ideas remain in a dubious grey area between fantasy and reality, and some of these have been described in Chapter 6. However in the world of computer control and networking, the time between fantasy and real implementation has proven to be remarkably short. The Internet is a prime example of this. The company Progressive Networks has produced a product called 'Real Audio' which provides 8 bit, 11 kHz mono sound in real time over the Internet. Internet links will inevitably get faster as we move towards realizing the fantasy of the 'Information Superhighway'.

For now, the next step in the 3-stage evolution of the remote studio access system will be investigated, namely the passage of audio, MIDI and studio control information down a single, high speed link. Hardware will be configured, software modelled, and the system assessed to see whether it can further the three goals of remote, shared, centralized access to music studio resources.

Bibliography

360 Systems "AM-16 Series Audio Cross point Switchers" 360 Systems product announcement.

ADB "Professional Digital I/O For Windows" Sales Document for the Multi!Wav Pro interface card, ADB, 1995.

AES "AES3-1985: Serial Transmission Format for Linearly Represented Digital Audio Data" Journal of the Audio Engineering Society, Vol 33.

AES "AES10-1991: Serial Multichannel Audio Digital Interface (MADI)" Journal of the Audio Engineering Society, Vol 39.

AES "AES11-1991 Synchronization of Digital Audio Equipment in Studio Operations" Audio Engineering Society Standards Document, 1991.

Aiken, J., Milano D. "MIDI-Controlled Digital Mixer" Keyboard, August 1987.

Alabisio B. "Transformation of Data Flow Analysis Models to Object Oriented Design" Proceedings of OOPSLA '88, September 1988.

Alles A. "ATM in Private Networking" Hughes LAN Systems, 1993.

Allik A., Dunne S., Mulder R. "ArcoNet: A Proposal for a Standard Network for Communication and Control in Real-Time Performance" Leonardo, Vol 23, No. 1, 1990.

Analog Devices "Data Conversion Products Data Book" Analog Devices, 1988.

Anderson D., Kuivila R. "Accurately Timed Generation of Discrete Musical Events" Computer Music Journal, Vol. 10, No. 3, 1986.

Anderson D., Doris R., Moorer J., Reichbach J., Roth J., Tellegen B. "High Speed Networking for Professional Audio" Proceedings of the Audio Engineering Society UK Conference, London, 1993.

Anderson D., Moorer J., Roth M. "Performance Issues in Digital Audio Networks" Presented at the 95th AES Convention, New York, 1993, AES preprint number 3734.

Andrews, B.R. "Taking up Residence with Coderunner" Dr. Dobb's Journal, June 1991.

Aoyagi T., Hirata K. "Music Server System - Distributed Music System on Local Area Network" Journal of Information Processing, Vol. 15, No. 1, 1992.

Audio Engineering Society "AES Recommended Practice for Digital Audio Engineering - Serial Transmission Format for Two-Channel Linearly Represented Digital Audio Data" Audio Engineering Society, Inc., 1991.

Audio Engineering Society "AES Recommended Practice for Digital Audio Engineering - Serial Multichannel Audio Digital Interface (MADI)", Audio Engineering Society, Inc., 1991.

Audio Engineering Society "AES Recommended Practice for Sound Reinforcement Systems - Communications Interface PA-422" AES Special Publications AES15-1991, 1991.

Ballista, A., Casali, E., Chareyron, J. Haus, G. "A MIDI/DSP Sound Processing Environment for a Computer Music Workstation" Computer Music Journal, Vol. 16, No. 3, Fall 1992.

Bernardini N. "TRAILS: An Interactive System for Sound Location" Proceedings of the 1989 International Computer Music Conference, Ohio, November 1989.

Berryman J. "QSC Application Protocol" Internal Document, QSC Corporation, 1995.

Berryman J. "SC-10-2 Scope and Status" Proceedings of the AES 13th International Conference, Dallas, 1994.

Berryman S., Sommerville I. "Modelling and Evaluating the Feasibility of Timing Constraints Under Different Real-Time Scheduling Algorithms" Real Time Systems, Vol. 4, No. 4, 1992.

Booch G. "Object Oriented Analysis and Design, 2nd Edition" Benjamin/Cummings, 1994.

Brighton, N. "The Patch Bay" Electronic Musician, May 1992.

Buhr R.J.A "Pictures that Play: Design Notations for Real-Time and Distributed Systems" Software-Practice

and Experience, Vol. 23(8), August 1993.

Buxton W. "Masters and Slaves Versus Democracy: MIDI and Local Area Networks" Proceedings of the Audio Engineering Society 5th International Conference, Los Angeles, 1987.

Chadabe, J. Zicarelli, D. "M-The Interactive Composing and Music Performing System" Albany: Intelligent Computer Music Systems.

Chen P.P. "The Entity-Relationship Model - Toward a Unified View of Data" ACM Transactions on Database Systems, Vol. 1, No. 1, March 1976.

Cheng H. Sheu J. "Design and Implementation of a Distributed File System" Software-Practice and Experience, Vol. 21, No. 7, July 1991.

Clarke, M., Bromwich, M, Smith, G. "Studio Report: Huddersfield Polytechnic Electronic and Computer Music Studios" ICMC Glasgow, 1990.

Coad P., Yourdon E. "Object Oriented Analysis" Yourdon Press, 1991.

Coad P., Yourdon E. "Object Oriented Design" Yourdon Press, 1992.

Coad P, Yourdon E. "Object Oriented Programming" Yourdon Press, 1993.

Comeau G. "Networking with Unix" Byte, 1989.

Comer D. "Operating System Design, The Xinu Approach" Prentice-Hall, 1984.

Comer D. "Internetworking with TCP/IP Volume 1: Principles, Protocols, and Architecture - 2nd Edition" Prentice-Hall, 1991.

Comer D., Fossum T. "Operating System Design - the Xinu Approach: PC Edition" Prentice Hall, 1988.

Cooper J.L. "LANs and MIDI" Keyboard, December 1987.

Crane B., et. al. "An Open Modular Architecture for Computer-Controlled Sound Systems" 99th Audio Engineering Society Convention Preprint, New York, October 1995.

Crystal Semiconductor "Crystal Semiconductor Audio Databook" Crystal Semiconductor, 1994

Cutmore N. "MADI-Based Fibre Optic Studio Routing" Proceedings of the Audio Engineering Society UK Conference, London, 1993.

Dannenberg R.B., Neuendorffer T., Newcomer J., M., Rubine D., Anderson D.B. "Tactus: toolkit-level support for synchronized interactive multimedia" Multimedia Systems Journal, Vol. 1993, No. 1, 1993.

De Furia, S. "The Secrets of Analog and Digital Synthesis" Third Earth Productions, 1986.

Digi "Product Profile - Digi PC IMAC and PC IMAC/4" Digi International Sales and Specifications Document, 1995.

Dietz M., Popp H., Brandenburg K., Friedrich R. "Audio Compression for Network Transmission" Presented at the AES 99th Convention, New York, 1995, AES preprint number 4129, 1995.

Digital Audio Labs "V8, The Next Phase in the Evolution of Digital Workstations" Digital Audio Labs Sales Document for the V8 Audio Processor Card, 1995.

EIA "EIA Standard RS-422A" Industrial Electronics Bulletin No 12, Electronics Industries Association, Engineering Dept., Washington DC.

Fayad M., Hawn L.J., Roberts M.A., Klatt J.R. "Using the Shlaer-Mellor Object-Oriented Analysis Method" IEEE Software, March 1993.

Fichera S. "Machine Tongues XIII: Real-Time Audio Conversion Under a Time-Sharing Operating System" Computer Music Journal, Vol. 15, No. 3, Fall 1991.

Finger R. "AES3-1992: The Revised Two-Channel Digital Audio Interface" Journal of the Audio Engineering Society, Vol. 40.

Fober D. "Real-Time Data Flow on Ethernet and the Software Architecture of MIDIShare" ICMC Proceedings, Arrhus, 1994.

Foss R., Wilks A. "A Network Approach to the Problem of Sharing Music Studio Resources" Proceedings

of the ICMC, Glasgow 1990.

Foss R., Wilks A. "The Remote MIDI Workstation" Proceedings of the ICMC, San Jose, 1992.

Foss R., Rehmet G., Watkins R.C. "A Multitasking Operating System above MS-DOS" Proceedings of the 7th South African Computer Research Symposium, South African Computer Journal, July 1992.

Foss R., Wilks A. "Using Network Technology to Share Music Studio Resources" Proceedings of the AES 13th International Conference, Dallas, 1994.

Foss R., Mosala T. "Routing MIDI Messages over Ethernet" Presented at the AES 99th Convention, New York 1995, AES preprint number 4057.

Freed A. "New Tools for Rapid Prototyping of Musical Sound Synthesis Algorithms and Control Strategies" Proceedings of the 1992 ICMC, San Jose, 1992.

Freed A. "New Tools for Rapid Prototyping of Musical Sound Synthesis Algorithms and Control Strategies" International Computer Music Conference Proceedings, San Jose, 1992.

Freeman R. "The RED-MIDI System" Honour's Thesis, Rhodes University, 1995.

Garvin, M. "Designing a Music Recorder" Dr. Dobb's Journal, May 1987.

Glass L. B. "DPMI: The DOS Protected Mode Interface" Byte IBM special edition, Fall 1990.

Haliburton G. "MIDI Communication over Ethernet" Honours project thesis, Rhodes University, 1988.

Harris, C. "A Composer's Computer Music System: Practical Considerations" Computer Music Journal, Vol. 11, No. 3, Fall 1987.

Hatley D., Pirbhai I. "Strategies for Real-Time System Specification" Dorset House, New York, 1987.

Helstrip, S. "Face the Music" Personal Computer World, November 1993.

Howell M. "Implementation of Windows MIDI Drivers for a Networked Music Studio Environment" Honour's Thesis, Rhodes University, 1994.

Huber, D. "Riding the Bus" Electronic Musician, March 1991.

Huber, D. "Audio Production Techniques for Video" Howard W. Sams & Company, 1989.

Hurtig, B. "The Future of Analog" Electronic Musician May 1992.

Hutchinson D., Merabti M. "Ethernet for Real-Time Applications" IEEE Proceedings, Part E. Computers and Digital Techniques, 1987, Vol. 134, No. 1.

IMA (International MIDI Association) "IMA Member Survey" IMA Bulletin, May 1989.

IMA "Apple Announces MIDI Manager" IMA Bulletin, February 1989.

IMA "MIDI Machine Control 1.0" the International MIDI Association, 1992.

IMA "MIDI 1.0 Detailed Specification, Document Version 4.1" the International MIDI Association, 1989.

IMA "MIDI Machine Control 1.0" International MIDI Association, 1992.

IMA "New Digital Audio Standard from AES" IMA Bulletin, December 1987.

Jain R. "The Art of Computer Performance Analysis" John Wiley & Sons, 1991.

Jacobs D., Anderson D. "Design Issues for Digital Audio Networks" Proceedings of the AES International Conference, Dallas, 1994.

Karagosian M. "SC-10-1 Scope and Status" Proceedings of the AES 13th International Conference, Dallas, 1994.

Karlin D. "The HCA Project and its Relationship to AES-24" Internal Document, Harman Corporation, 1995.

Keislar D. et al. "1992 International Computer Music Conference" Computer Music Journal, Vol. 17, No. 2, 1992.

Kendall, R. "Big Noise Software MIDI MaxPak" Electronic Musician, February 1994.

Kerth N.L. "A Structured Approach to Object Oriented Design" Proceedings of OOPSLA '91, October 1991.

Koeslag P. "Routing MIDI Messages Over Ethernet" Honours Thesis, Rhodes University, 1995.

Krysak R. "Multiuser DOS for Control Systems" Dr. Dobb's Journal, April 1992.

Kuchera-Morin, J., Scott D. "The Center for Computer Music Research and Composition, University of California Santa Barbara" International Computer Music Conference Proceedings, San Jose, 1992.

Lacas M., Warman D., Moses B. "The MediaLink Real-Time Multimedia Network" Presented at the 95th AES Convention, October 1993, AES preprint number 3736.

Lehman M.M. "Software Engineering, the Software Process and their Support" Software Engineering Journal, 1991.

Lehrman, P. "Tape Killers - The EM Guide to Hard Disk Recorders" Electronic Musician, September 1992.

Lockwood D. "Studio Audio & Video OCTAVIA Multitrack Editor" Audio Media, September, 1995.

Lone Wolf "MIDITap User Manual" Lone Wolf, 1990.

Loy, G. "Musicians Make a Standard: The MIDI Phenomenon" Computer Music Journal, Vol. 9, No. 4, Winter 1985.

Mahavadi R. "FDDI Dual Attachment Station on a PC-AT Interface Card" Microprocessors and Microsystems, Vol 17, No 2, 1993.

Many, C. "Tascam MIDiiZER MTS-1000 Multi Synchronizer/Controller" Electronic Musician, July 1990.

Martin H., Young E. "Computer Control of Complex Systems and Description of RS-422 Protocol" Proceedings of the 6th International AES Conference, 1988.

McMillan K., Wright M., Simon D., Wessel D. "ZIPI - An Inexpensive, Deterministic, Moderate-Speed Computer Network for Music and other Media" Proceedings of the AES 13th International Conference, Dallas, 1994.

Mellor, D. "Inside the Synclavier" Sound on Sound, June 1989.

Meyer, C. "Hardware Control of the Mark of the UNicorn MIDI Mixer 7s" The IMA Bulletin, July/August 1991.

Meyer, C. "Total Control (on the cheap)" IMA bulletin, May-August 1989.

Meyer C. "MIDI LAN's, A Possible Future?" the IMA Bulletin, August 1987.

Microsoft "Multimedia Extensions Programmers Workbook" Microsoft Press, 1991.

Miller, D., Lehrman, P. "When Worlds Collide" Electronic Musician, December 1991.

Molenda, M. "Dream Home Studios" Electronic Musician, February 1993.

Moore F.R. "Elements of Computer Music" Prentice-Hall, 1990.

Moore, F.R. "The Dysfunctions of MIDI" Computer Music Journal, Vol 12, No. 1, Spring 1988.

Mosala T. "Routing MIDI Messages in a Shared Music Studio Environment" MSc Thesis, Rhodes University, 1994.

Moses B. "Technical Advancements Leading to Growth of SC-10 and the SC-10 Committee Structure" Proceedings of the AES 13th International Conference, Dallas, 1994.

Music Technology Review "360 Systems Audio Matrix 16" Music Technology, April 1989.

Music Quest "MIDIEngine 8Port/SE User's Manual" Music Quest, 1995.

Nerson J. "Applying Object Oriented Analysis and Design" Communications of the ACM, Vol. 35, No. 9, September 1992.

New England Digital "An Introduction to the SYNCnet System: A communications Network and Protocol" New England Digital, 1989.

Nieberle R.C., Modler P. "The CAMP system: an approach for integration of realtime, distributed and

interactive features in a multiparadigm environment" International Computer Music Conference Proceedings, Cologne, 1988.

Nielsen O. "MIDI and Audio via ISDN" Proceedings of the 1994 International Computer Music Conference, Aarhus, 1994

Ntene, M. "MIDI Machine Control of a Tascam 238 Multitrack Tape Recorder" Honours project report, Dept. of Computer Science, Rhodes University, November 1993.

Oppenheimer L. "Intone MIDI Maestro Audio and MIDI Patch Bay" Electronic Musician, March 1991.

Oppenheimer, L. "Digidesign Session 8 Integrated Digital Studio" Electronic Musician, September 1993.

Orlarey Y., Lequay H. "MIDI Share" Proceedings of the 1989 ICMC,

Orlarey Y. "An Efficient Scheduling Algorithm for Real-Time Musical Systems" Proceedings of the 1990 ICMC, Glasgow 1990.

Peavey "MediaMatrix - Configurations for most needs and budgets" Sales document for Peavey's MediaMatrix, Peavey, 1995.

Petzold C. "Programming Windows" Microsoft Press, 1992.

PG-Music "Multi-MPU driver Programmer's Manual" PG-Music 1993.

Popkin Software "The System Architect User's Manual" Popkin Software 1990.

Porter, B. "The Case for Rental Rooms" Recording Engineer/Producer, April 1990.

Rehmet G. "An Event Driven Windowing System for PC-Xinu" Honours Project Report, Rhodes University, November 1993.

Richmond C. "COMMAND/CUE Systems: History/ Function/ Application/ Design" Proceedings of the 87th AES Convention, October 1989.

Rideout E. "Sound Cards" Keyboard Magazine, October 1994.

Roads, C. "The MIDI Interface - Overview" Selected reading from "The Music Machine" edited by Curtis Roads, Massachusetts Institute of Technology, 1989.

Roland Corporation "D110 User's Manual" Roland Corporation, 1989.

Roland Corporation "Digital Effects Processor DEP-5 Owner's manual" Roland Corporation, 1988.

Roland Corporation "The MPU-401 Technical Reference Manual" Roland Corporation, 1985.

Roland Corporation "Roland Audio Producer RAP-10 Owner's Manual" Roland Corporation, 1993.

Romkey J., Fisher S. "Packet Drivers" Byte, May 1991.

Rona, J. "Synchronization - from Reel to Reel" Hal Leonard Publishing Corporation, 1990.

Rooholamini R., Cherkassky V., Garver M. "Finding the Right ATM Switch for the Market" IEEE Computer, April 1994.

Rothstein J. "MIDI A Comprehensive Introduction" Oxford University Press, 1992.

Rumsey F., Watkinson J. "The Digital Interface Handbook" Focal Press, 1993.

Rumsey F. "Audio Programme Interchange: Networks, File Formats, and Real-Time Transfer" Presented at the AES 95th Convention, AES preprint number 3737, 1993.

Rybacki R., Robbins A. Robbins K. "Ethercom: A Study of Audio Processes and Synchronization" Proceeding of the 24th SIGCSE Technical Symposium on Computer Science Education, Indianapolis, 1993.

Sanderson D. "Distributed File Systems: Stepping Stone to Distributed Computing" LAN Technology, May 1991.

Satyanarayanan M. "Scalable, Secure, and Highly Available Distributed File Access" IEEE Computer, May 1990.

Schwartz R. "Digital Domain: Un-Conventional" Recording Engineer/Producer, November 1991.

Scott A. "Quark LRM2 MIDILink" Music Technology, November 1986.

Shlaer S., Mellor S. "Object Oriented Analysis - Modelling the World in Data" Yourdon Press Computing Series, Prentice-Hall, 1988.

Shlaer S., Mellor S. "Object Lifecycles - Modelling the World in States" Yourdon Press Computing Series, 1992.

Sofer D. "Mark of the Unicorn MIDI Mixer 7s" Electronic Musician, July 1991.

Solid State Logic "SoundNet, The World's First True Multi-User, Mass Storage Digital Audio Network" Solid State Logic Limited, 1990.

Sonic Solutions "SonicNet - the Open Studio" Sonic Solutions, 1993.

Stankovic J.A. "The Spring Kernel: A New Paradigm for Real-Time Systems" IEEE Software, May 1991.

Steenkiste P. "A Systematic Approach to Host Interface Design for High-Speed Networks" IEEE Computer, March 1994.

Stevens, A. "Writing TSR's" Computer Language, February 1988.

Tanenbaum A. S. "Computer Networks - 2nd Edition" Prentice-Hall, 1988.

Tanenbaum A.S., et al "Amoeba A distributed Operating System for the 1990's" IEEE Computer, May 1990.

TEAC "Tascam 238 Serial Port and Control Specifications" TEAC AV Division report, 1989.

Turtle Beach Systems "Multisound User's Guide" Turtle Beach Systems, 1992.

Turtle Beach Systems "Quad Studio User's Manual" Turtle Beach, 1994.

Twelve Tone Systems (Cakewalk Professional for Windows version 3.0" Twelve Tone Systems, 1994.

Tyler L. "The AES-24 Protocol: General Architecture and Philosophy" Proceedings of the AES 13th International Conference, Dallas, 1994.

Tywoniak, E. "MIDI Enters the Class" Electronic Musician, April 1992.

Vercoe, B. "Real-Time CSOUND: Software Synthesis with Sensing and Control" ICMC Glasgow, 1990.

Videira F., Casaca A. "An ISDN Primary rate Interface for Ethernet Access" Microprocessing and Microprocessors, Vol 38, 1993.

Voyetra Technologies "VAPI Programming Considerations" Voyetra Technologies, 1990.

Walraff D., Marshall P. "Computer Music Catalog - Digital Music Systems" Digital Music Systems, Inc., 1983.

Ward P., Mellor S. "Structured Development for Real-Time Systems" Prentice-Hall, 1985.

Ward P. "How to Integrate Object Orientation with Structured Analysis and Design" IEEE Software, March 1989.

Ward D. "Handling the AES Interface, Timing, Routing and Distribution" Proceedings of the AES UK Digital Audio Audio Interchange Conference, London, 1993.

Wells G. "A Comparison of Four Microcomputer Operating Systems" Real-Time Systems, Vol. 5, No. 4., 1993.

Westfall, L. "MIDI Machine Control Ratified" IMA Bulletin, Vol 9, No. 1, Summer, 1992.

Westfall L. "Wireless MIDI System from Gambatte" IMA Bulletin, August 1989.

Westfall L. "DMP7 Pro, Mac-Based Editing for Yamaha's Digital Mixer" the IMA Bulletin, January 1988.

Westfall L. "Iota Systems' MIDI Fader" the IMA Bulletin, June 1988.

White P. "Creative Recording Effects and Processors" Music Maker Books, 1989.

Wilkinson S. "Sound All Around" Electronic Musician, October 1993.

Wilkinson S., Rona J. "MIDITap User Manual" Lone Wolf, Inc., 1990.

Wilkinson, S. "Cool Schools" Electronic Musician, September 1993.

Wilkinson, S. "The Great Cable Debate" Electronic Musician, January 1994.

Wilks, A. "The Analysis of a Computer Music Network and the Implementation of Essential Subsystems" MSc thesis, Rhodes University, 1994.

Woram, J. "Sound Recording Handbook" Howard W. Sams & Company, 1989.

Yadav S. B. "Control and Definition Modularization: An Improved Software Design Technique for Organizing Programs" IEE Software Engineering, Vol 16, No 1, January 1990.

Yamaha Corporation "The C1 Programmer's Reference Manual" Yamaha Corporation of America, 1989.

Yamaha Corporation "Yamaha SPX900 Professional Multi-Effect Processor Operation Manual" Yamaha Corporation, 1988.

Yamashiro A. "Comparison of OOA and Real-Time SA - From the Experiment of Analyzing an Image Filing System" Proceedings of OOPSLA '93, September 1993.

Yavelow, C. "Music and Microprocessors: MIDI and the State of the Art" selected reading from "The Music Machine" edited by Curtis Roads, Massachusetts Institute of Technology, 1989.

Yelton G. "Making Connections - the EM Guide to MIDI Patch Bays and Processors" Electronic Musician, January 1993.

Yonge M. "Practical Networking Strategies for Audio and Control Data" Proceedings of the Audio Engineering Society UK Conference, London, 1993.

Yourdon E., Constantine L. "Structured Design" Prentice-Hall, 1979.

Zeta Music Systems "ZIPI Network Detailed Technical Specifications" Zeta Music Systems, 1994.

Zweibel, R. "Designing the MTSU Media Complex" Recording Engineer/Producer, November 1991.

Appendix A

MIDI Messages

Data	00 - 7F	MIDI Data
Channel Status Bytes	80 - 8F	Note Off
	90 - 9F	Note On
	A0 - AF	Poly Key Pressure
	B0 - BF	Control Change
	C0 - CF	Program Change
	D0 - DF	Channel Pressure
	E0 - EF	Pitch Wheel Change
System Common Status Bytes	F0	System Exclusive
	F1	MTC Quarter Frame
	F2	Song Position Pointer
	F3	Song Select
	F4	Undefined
	F5	Undefined
	F6	Tune Request
	F7	End Of Exclusive
System Real Time Status Bytes	F8	MIDI Clock
	F9	Undefined
	FA	Start
	FB	Continue
	FC	Stop
	FD	Undefined
	FE	Active Sensing
	FF	System Reset

Appendix B

REQUIREMENTS SPECIFICATION FOR REMOTE STUDIO ACCESS NETWORK

The goal of this project is to provide a number of workstation nodes. Each node enables access to a shared studio. The node will allow for the booking of studio resources, such as tape decks, effects units and synthesizers, audio patching and mixing of the inputs to and outputs from these resources, MIDI patching and channel selection of these resources. Mixing can be automated. Any MIDI-based sequencer can have its MIDI ins and outs connected to this node. Thus, users will be able to share the resources of a common studio, and the workstation node will provide configuration, mixing and channel selection capabilities which normally have to be done "hands on" in the studio. In this version, all synthesizer and effects unit bank loading will be done via the sequencer's universal librarian and patch editing software. The workstation node allows for two-way MIDI communication between any booked music resources.

User Entry

A user will be required to type a usercode and password to identify himself before using a workstation. A user will be able to change his own password. When a user logs out, all his resources will be freed and any patches, audio or MIDI will be unpatched. This freeing of resources must be reflected for other users using the booking sheet.

The Booking Sheet

This can be requested at any point. The booking sheet will be dependant on the resources available in a particular studio set-up. In all the following diagrams, the resources shown will be the minimal set required to fully test all facets of a complete networked audio production system. Given in figure 1 below is a typical booking screen. A user will be able to book up to 5 days in advance. A user will be able to move from booking slot to booking slot within a 5-day period. Every new booking slot time, the booking sheet on screen may need to be adjusted. This will happen if one of the booking slots is now no longer valid. A user will be able to access the usercode of the user currently using any one of the resources. He will also be able to select or deselect any of the available booking slots. The booking sheet will probably extend over more than one screen, and a user will be able to move from screen to screen, viewing different parts of the sheet. Usercode access, selection and deselection will all happen by simply selecting the slot. The state of the slot will determine the action of the system. The booking sheet of a particular user will be updated to reflect the booking changes

made by other users. If a user does not log on in the hour in which he has booked aresource, the resource will again become free for other users. Those freed resources must be displayed to other users involved in booking. When a user's booked time for a resource expires, this resource will become available to other users and will be reflected as such on any booking sheets. Any audio patches involving the resource will be unpatched and this unpatching displayed. The resources will be removed from the audio patch sheets. Any MIDI patches will be unpatched and this unpatching displayed. The resources will be removed from the MIDI patch sheets. The time that a user actually uses a resource as well as his workstation usage time will be recorded.

CURRENT USER - RICHARD WED 22/8/1990

TIMES - AM PM

RESOURCE	12-1	1-2	2-3	3-4	4-5	...	9-10	10-11	11-12
DX7	U	U					X		
S220									
TR505									
DEP-5									
TASC238									
.									
.									

U - already booked
X - booked by current user
Select open slot to book.
Select "X" slot to unbook.
Select "U" slot to see usercode of current user.

Figure 1 - Booking Sheet

The Audio Patcher/mixer Facility

A user can request the audio patcher/mixer facility. The audio patch facility will allow a user to route audio signals from one resource to another, mix the output of resources and to automate the mixing process. Only resources which have been booked by the user for the current time period will appear on the patch grid. An example patch grid is given in figure 2 below.

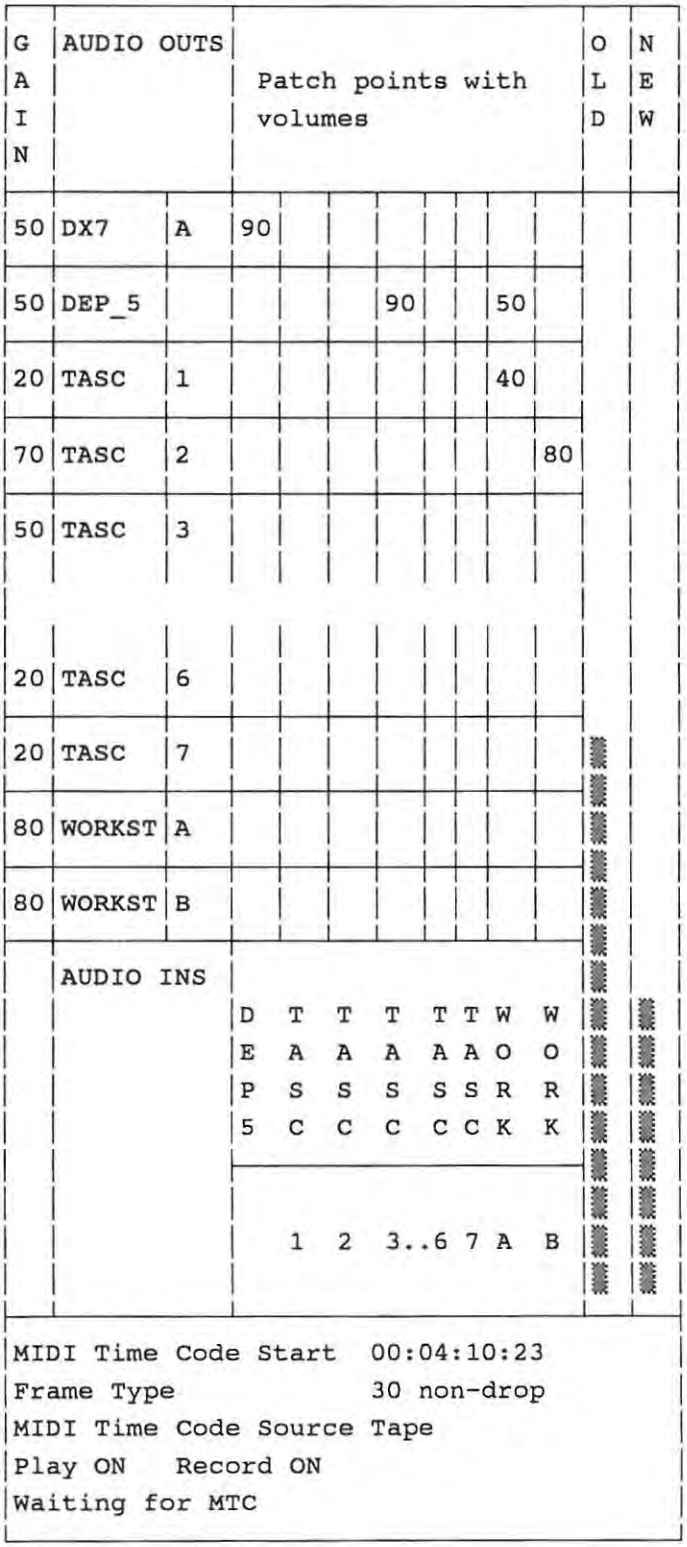

Figure 2 - Audio Patcher/mixer screen

A user can patch the OUT of one resource to the IN of another by selecting the appropriate

intersection block of the patch grid. The OUT of a resource may not be patched into the IN of the same resource. The system will ensure this.

Note that in figure 2 each of the patched points has a number within it. This number is an amplitude level and can range from 0-99. In figure 2, the output of the DX7 has been patched to the input of the DEP-5. 90% of the DX7's maximum output level will be routed to the DEP-5. 90% of the maximum output level of the DEP-5 is routed to track 3 of the multi-track Tascam-238. 50% of the maximum output level is routed to one of the user inputs (USERIA), after first being mixed with the output of track 1 of the TASCAM-238. The track 1 level has been reduced to 40% of its maximum. After patching, the amplitude level at the patch point will be set to 80%. At any time, any patch point can be selected. It will then be the 'current' patch. The amplitude of the currently selected patch can be moved up or down at any time.

The user has control over gain on each of the resource outputs. As for the amplitude values, a user can select and alter the gain levels.

Figure 2 shows the DX7 as having only one audio output. Similarly, the DEP-5 is being used in mono mode. The nature of the instruments and how they are connected to the audio patch bay, will be determined via a configuration program. This program will be described later in this document.

Every new booking slot period, the user might gain some resources and lose others. The audio patch screen will be altered to reflect this.

A user can request to record changing patch point input levels. This is not the case for gain levels. When a user requests to record, the patcher/mixer will wait until a MIDI Time Code message is received which is beyond the start frame value. As soon as this happens, any amplitude changes which the user makes to the current patch will be recorded. A new patch can be selected at any time and its changing amplitude levels recorded. When a new patch is selected, the first value it takes on is the value currently recorded, if there is one.

When a play request is made, the patcher/mixer will once again wait for the receipt of a MIDI Time Code message beyond the start frame. On receipt of the message, the patcher/mixer will move to the correct position in its "tracks" of mix level recordings. It will "play" back any currently recorded mix levels. For the currently selected patch, if amplitude levels are being changed, it will display the old levels and the new levels in bar graph form. The new levels will be 'played out'. If the currently selected patch is not being changed, its recorded values will be 'played', and there will be

no 'new' bar graph display. During play time, the changing patch point levels of all patches will be displayed as numbers.

Recording and playback can be started up at any time, and both can happen at the same time.

A user can specify the SMPTE start frame, the frame type, and also the MIDI port on the workstation from which MIDI Time Code messages will be received.

The MIDI Patch Facility

A user can request the MIDI patch facility. As will be seen from the hardware layout, a single MIDI output from the workstation will be sent via RS422, direct MIDI or wireless MIDI, directly to the MIDI patch bay, where it will be patched to the relevant resources. There will also be a MIDI input to the workstation from the patchbay. A typical patch bay screen is given in figure 3. Once again, only resources booked by the user will appear on this patch grid.

In figure 3, the MIDI output from the SMPTE/MIDI converter device is patched to the workstation user MIDI input. Only one device can have its MIDI output patched to the workstation. The MIDI output generated at the user's workstation is patched to go to the DEP-5 effects unit as well as the DX7 and S220. If a user were to use an instrument upload/download program to upload and download programs on the DX7, then the MIDI out of the DX7 would have to be patched to the user's MIDI in. If a user were to try and patch more than one input, there would be an error message.

The receiving channels of certain MIDI controllable devices can be changed. Those whose channels can be changed will be marked as such. Part/voice names are associated with each of the channels.

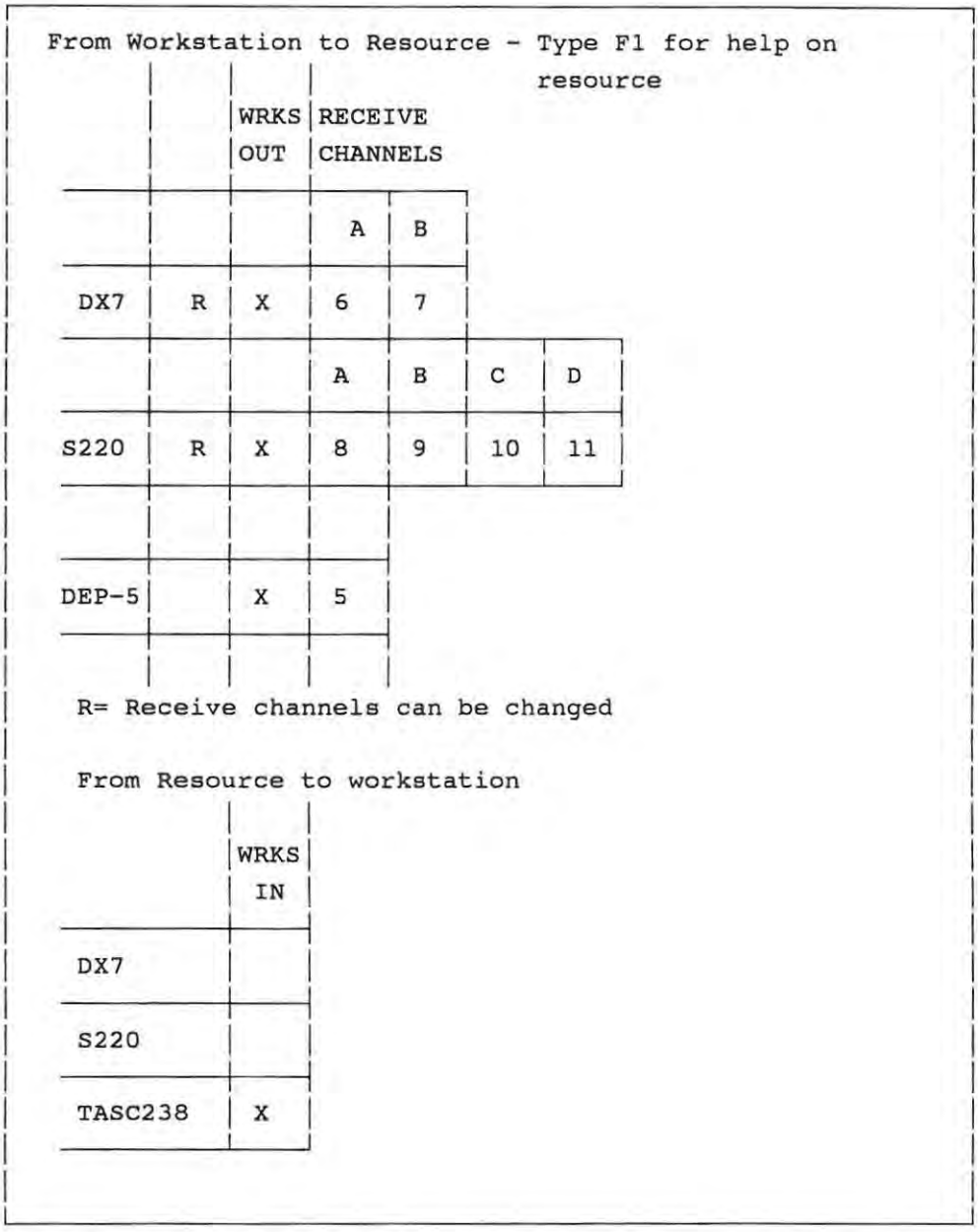

Figure 3 - MIDI patch screen

Both the audio and MIDI patch settings can be stored in a setup file, probably on a floppy disk. The setup file can be reloaded, thereby restoring the original setup. If instruments in the saved setup are not currently booked, then they, and their corresponding patches, will not appear on the patch screen. The system will utilize identical booked resources if the resource saved in the setup file is not booked. A user can request to book all the resources required for a particular setup.

Every new booking slot period, the user might gain some resources and lose others. The MIDI patch

screen will be altered to reflect this.

Configuration

Configuration will be performed at the server by the system manager. The manager must specify the network ID number and matrix position of each audio patch bay. He must specify the number of assignable patch nodes associated with each patch bay. He must specify the network number and matrix position of each equalization unit. All this information could be entered in the form of a patch matrix as shown below.

Gain Units Audio patchers

network ID	network ID nodes	network ID nodes
network ID	network ID nodes	network ID nodes

He can also specify the configuration of the MIDI patch bays in a similar manner. The audio and MIDI patch bays will have the same manufacturer ID numbers. They will be distinguished from each other via a network ID number which will be assignable via front panel switches.

The manager must specify the names of all the MIDI resources accessible to users. These are resources which could be booked by workstation users. The manager will select each resource from a list of resources provided by the system on request (MIDI resource types). A resource selected from this list may or may not be able to have its MIDI channels changed remotely. On selecting one of these resources, the manager will have to specify the system exclusive channel number (Device ID) of the resource, if it has one. A manager can delete a resource from the resource list or add a resource to the resource list at any time. A resource can only be deleted if it is not booked by a user.

Each of the resources could have multiple audio in's and out's. Current instruments usually have more than one audio output, while a multitrack tape recorder will typically have a number of audio ins and outs. One or more of these audio outs and ins could be used in a particular network configuration. The manager must specify the number of audio in's and out's utilized by each resource.

The manager must also specify whether there is a MIDI out for each resource.

The system must know how the audio ins and outs of the various resources connect to the audio patch bay nodes. It must also hold information about connections to the MIDI patch bay. This is the subject of a later section.

	Audio		MIDI	DEVICE ID
RESOURCE	Ins	Outs	Outs	ID no
DX7		2	X	5
CZ101		1	X	1
CZ101		1	X	2
DX21		1	X	12
S220		2	X	2
TR505		1	X	1
D110		2	X	8
DEP-5	2	2		9

Figure 4 - MIDI Resource Configuration

MIDI Resource Type Configuration

There can be many MIDI resources of the same type. For example, there may be many DX7's, each DX7 receiving on its own channel. Each type of resource has its own system exclusive messages for changing channels. Some resource types may not be able to have channels changed. These resource types, together with their system exclusive message formats, must be entered by the manager. When a manager requests to configure resource types, he will first be given a list of the current resource types. He will then be requested to type a new resource type name. Then he must enter the number of parts in the resource.

Resource name	No. of parts
CZ101	1
DX7	2
DX21	1
D110	9

Figure 10 - Resource list

For each part, he must enter a part name and a sysex expression. The sysex expression will be used to generate one or more sysex messages which will be instrumental in changing the channel on which the part resides. In the sysex expression in figure 11, R refers to the current receive channel number, N to the new channel number. S may also be used and refers to the current sysex number (Device ID).

D110 Part number	Part name (2 Digits)
1	P1
2	P2
3	P3
4	P4
5	P5
6	P6
7	P7
8	P8
9	R
Sysex expression - Part 3	
F0 + 41 + S + 16 + 12 + 06 + 00 + 18 + N + (S + 16 + 12 + 06 + 18 + N) + F7	

Figure 11 - Resource type specification

The manager will also be required to enter textual information regarding the resource. This information might include the correct MIDI patching of the instrument for channel change messages,

etc. In this way, a library of MIDI resource types is built up.

Music Resource Availability

The manager can request a display of resources available. A list will be produced showing which resources are available to be booked by users.

Resource Name	Available to be booked
DX7	X
D110	
TR505	

The manager can indicate that resources are available or unavailable to be booked. The 'X' alongside the DX7 in the above diagram indicates that the DX7 is unavailable for booking.

Configuration of Audio and MIDI Connections

AUDIO OUTS from device

NAME	NO	SFX
DX7	1.4	A
DX7	1.5	B
D110	1.1	L
.		
.		
DEP-5	1.7	B

AUDIO INS to device

NAME	NO	SFX
DEP-5	1.6	A
DEP-5	1.7	B

X.Y means row.connector for outs from device
X.Y means col.connector for ins to device
where rows and columns refer to patchbay matrix

Figure 5 - MIDI Resource lines

Configuration of the audio patch bay will simply require associating names with hardware connections

on the patch bay. On request, the manager will be presented with a layout of the patch bay containing resource names. A resource name entry will be provided for each audio in and out connection utilized on the resource. The first task of the manager will be to provide an appropriate suffix for each in and out connection. This suffix should relate to the actual in and out connector names on the instrument. His next task is to fill in the correct patch bay numbers alongside these named connections. A possible screen for the audio patch bay is presented in figure 5.

As far as the MIDI patch bay is concerned, the manager must initialize the channels of resources or voices of instruments. MIDI resources are shown with the parts to which channels can be assigned, labelled. The manager must associate channel numbers with these parts. The manager can request information regarding a resource type.

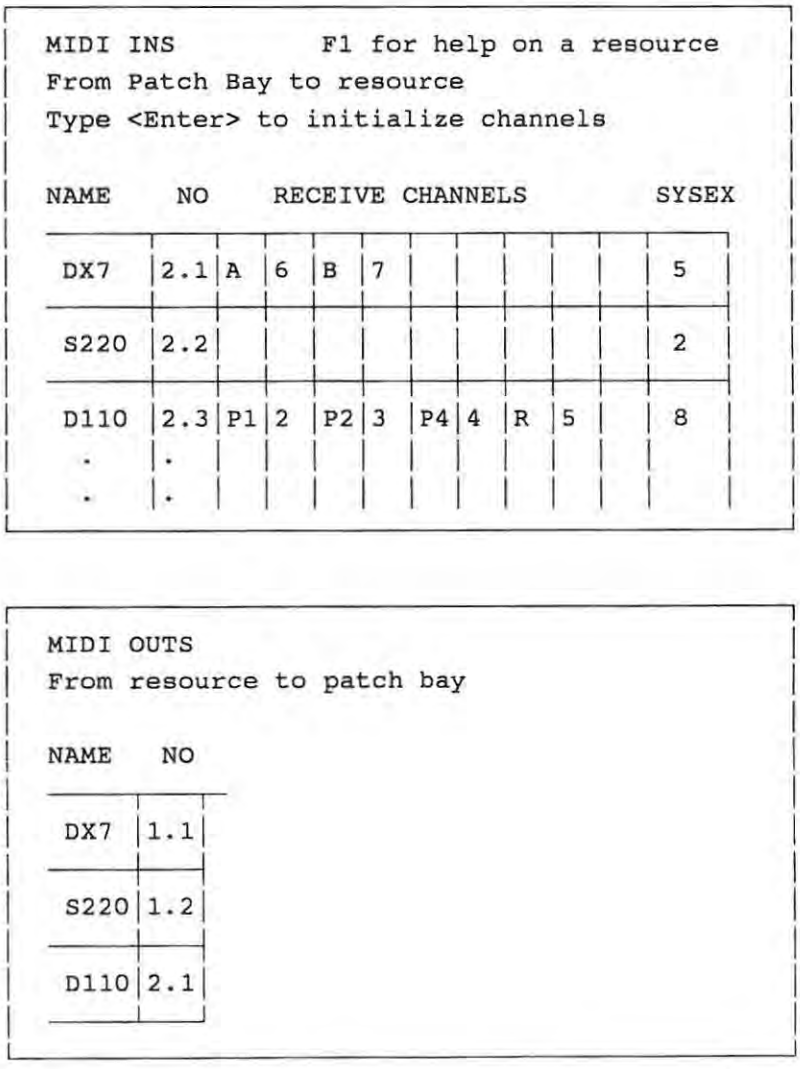

Figure 8 - MIDI Patch configuration screen

Configuration of Workstations

Each workstation has a numeric identifier associated with it. A manager will be required to type in the address of any new workstation added to the network. This might be the IP address or ethernet address, depending on the level of network support software. Also, for each workstation, he must specify the patch and connector numbers for the audio ins and outs as well as for the MIDI ins and outs.

		Audio in		Audio out			
ID	Ethernet Addr	L	R	L	R	MIDI in	MIDI out
1	02608C576871	1.8	1.7	1.8	1.7	1.4	1.6
2	02608C576872	1.9	1.1	1.9	1.1	1.5	1.5

Figure 7 - Workstation Configuration

The manager must type in the user code and initial password of any new user on the system.

Usercode	Password
Foss	Rich
Wilks	Ant

Figure 9 - Usercode table

The Audio Tape Control Facility

A user can request the audio tape facility. He must previously have booked a tape deck. A user will be able to have remote control over an audio tape deck via this facility. The screen will provide certain basic functionality to the user. The user will be able to fast rewind, fast forward, stop, play, pause and record. He will be able to select the track on which to record. This basic screen is shown in figure 10. A counter on the front panel will be continuously updated. Only tape decks which can provide this basic functionality remotely will be supported. Initially, only the TASCAM 238 will be supported.

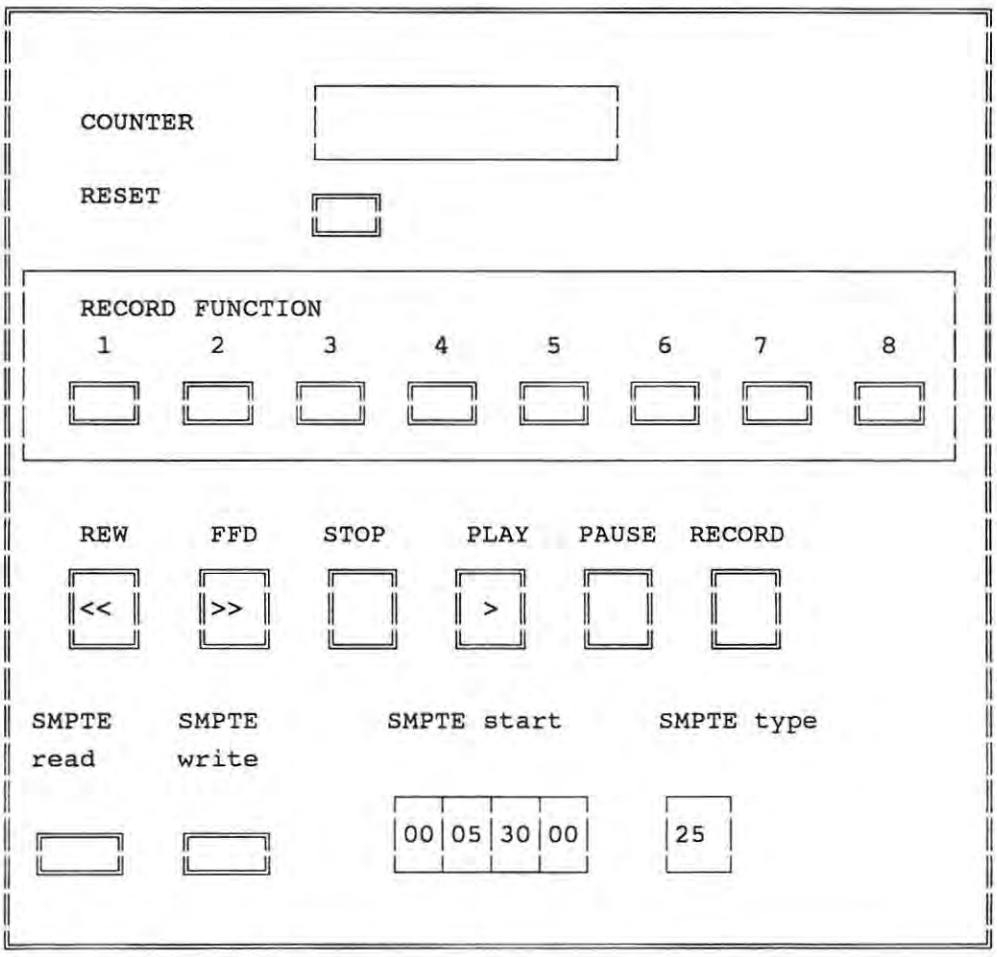

Figure 12 - TASCAM 238 front panel

The user can request to generate SMPTE sync or read SMPTE sync, or neither. Firstly, however,

he must specify a SMPTE start frame and frame type. If SMPTE reading is on, then MIDI Time code bytes will be sent to the workstation computer. The track to be striped with SMPTE will always be the outermost track of the tape.

The first task of the project will be to provide multitrack tape deck control, and in particular, control over the Tascam 238. Similar remote control over a VHS video cassette recorder will be possible in the future. In this case, synchronization code from an audio track of the video cassette will be used to provide the necessary timing for a workstation sequencer.

An eventual goal will be to slave the multi-track tape recorder to the video, with the sequencer synchronization signals coming from the multi-track.

Audio Tape Configuration

In the current system, only the TASCAM 238 tape deck will be supported. SMPTE will be read and generated via a Music Quest MIDI interface card within the server IBM PC. This Music Quest card will send MIDI time code bytes to the workstation computer which has control over the tape deck. The studio manager must specify the MIDI connection number which the MIDI out from the Music Quest card makes with the MIDI patch bay. He must also specify the number of audio ins and outs which are to be connected to the audio patch bay.

Tape name	MIDI out	Audio ins	Audio outs
TASC 238	1.4	7	7

Figure 13 - Tape server details

Having specified the number of audio ins and outs, the manager must specify how these audio ins and outs are connected to the audio patch bay.

AUDIO OUTS from tape			AUDIO INS to tape		
NAME	NO	SFX	NAME	NO	SFX
TASCAM 238	1.4	1	TASCAM 238	1.6	1
TASCAM 238	1.5	2	TASCAM 238	1.7	2
TASCAM 238	1.6	3	TASCAM 238	1.8	3
TASCAM 238	1.7	4	TASCAM 238	1.9	4
TASCAM 238	1.8	5	TASCAM 238	1.10	5
TASCAM 238	1.9	6	TASCAM 238	1.11	6
TASCAM 238	1.10	7	TASCAM 238	1.12	7

X.Y means row.connector for outs from device
X.Y means col.connector for ins to device

Figure 14 - Tape Resource lines

Recording studio use

The manager will, on request, be able to obtain a display of the studio usage of any particular user or of all the users. The studio usage will indicate the total number of hours spent at a workstation, as well as a breakdown of the number of hours spent using each musical resource in the studio. The manager will be able to request that the usage time on all resources for a particular user is zeroized. He will be able to do this for all the users as well.

Appendix C

Format of System Exclusive Messages for the Audio Patcher/Mixer Unit

Connect Patch and Change Patch Level messages are both 7 bytes long, whereas a Disconnect Patch is 6 bytes long since it doesn't require a patch level value.

Data byte	Explanation
SysEx Start	0xF0
Network Device	0x7D
Network Device ID	0x03
Audio Patch	0-15 (input connector)
Audio Patch	0-15 (output connector) + (0x10 or 0x20 or 0x40)*
Audio Patch Level	0-127 (not used for patch disconnections)
SysEx End	0xF7

* Connect Patch = 0x10
  Change Patch Level = 0x20
  Disconnect Patch = 0x40

Appendix D

Events and Responses for the Remote Studio Access Network

EVENT	RESPONSE
User requests to start up system at workstation	Prompt for usercode
User enters usercode	Check whether usercode is valid. Ask again if not, otherwise prompt for password. If user is already logged on, give error message.
User enters password	Check whether password is correct. Ask again if not, allow three tries. Record user as logged in. Display user commands.
User provides new password	Update the users password
User requests booking sheet	Display the booking sheet. Switch off MIDI transmission.
User requests to book a resource	Update booking sheet to reflect booking of instrument. Display update to all other users using booking sheet.
User requests usercode of user with booked resource	Display usercode.
User unbooks a resource	Update booking sheet to reflect unbooking of resource. Display change to all users with a booking sheet. If the booking slot is the current booking slot, then unpatch any audio patches made with this resource. Also unpatch any MIDI patches.
User requests to book all resources required for a particular setup	Change booking sheet to reflect the necessary bookings. Display these booked resources on the booking sheets of other users. Display any resourecs which couldn't be booked. Book resources which are identical to resources which could'nt be booked.
The next booking slot is signalled	Unbook the resources of any users not logged on and who have resources booked. Initialize the

	resource times for users using resources for the first time. Unpatch audio patches of users losing resources. Unpatch MIDI patches of users losing resources. Switch off MIDI transmission at the workstation side before unpatching. Display the changed patches to users. Update the booking sheet displays to reflect the loss of old time slot and gaining of a new one.
Resource timer ticks	Increment the resource times of all users currently using resources. Update the workstation time for the user.
User requests to load a setup	Pick up the audio and MIDI patching information from the setup and perform the necessary patches. Perform the necessary patch and gain level settings. Switch of MIDI transmission before doing MIDI patching. Instruments which are currently not booked will not be patched. Change channels of resources whose channels can be changed. Unbooked resources are displayed. If there are insufficient audio patch nodes, indicate this.
User requests to save a setup	Save the current audio and MIDI patching information in the setup
User requests the audio patching facility	Provide an audio patch grid comprising the resources which the user has booked, together with any patches and patch levels which the user has set up.
User requests to patch an audio patch	Make the patch connection and open a patch track. Set the audio level at the patch point to 80%. Do not allow the patch if the out of one resource is being patched to the IN of the same resource. Give a message if there are insufficient nodes.
User requests to unpatch an audio patch	Indicate on the patch grid that the unpatching has been performed. Send the appropriate unpatch message to the audio patcher/mixer.
User selects an audio patch	Display the audio level of the patch point on the 'old' bar graph. Make this patch the current patch.
User requests to alter patch levels at the selected patch point	Enable the reading of new patch levels. Signal that patch reading is taking place.
User requests to stop altering patch levels at the selected patch point	Disable the reading of new patch levels. Signal that patch reading is no longer taking place.

User increments the patch level at the selected patch point	Display the new level at the patch position and on the 'new' bar graph. Play this new level and store it as the value at the current patch in the Audio patches store.
User decrements the patch level at the selected patch point	Display the new level at the patch position and on the 'new' bar graph. Play this new level and store it as the value at the current patch in the the Audio patches store.
User selects a gain input block	Display the current gain value on the old bar graph
User requests to start entering gain levels	Enable the reading of gain levels for the selected gain
User increments the gain level	Send the new incremented gain level to the audio gain unit and indicate the new gain level on the patch screen
User decrements the gain level	Send the new decremented gain level to the audio gain unit and indicate the new gain level on the patch screen
User requests to stop entering gain levels	Disable the alteration of gain levels
User enters a SMPTE start value	Store the SMPTE start value for correct synchronization
User enters the SMPTE frame type	Store the SMPTE frame type for correct synchronization
User selects a particular source for MIDI Time Code	Start reading MIDI Time Code from this source
User requests to record level changes	Signal that recording is on.
User requests to stop recording level changes	Signal that recording is off
User requests to play back patch level tracks	Signal that playback is on.
User requests to stop playing back patch level tracks	Signal that playback is off
New MIDI time code arrives which is greater than the start frame	If in play mode, play all patch tracks levels at this time, except the current patch level. Display these levels on the screen at the patch points. Show the currently selected patch level on the 'old' bar graph.

	If the user has not requested to alter values at the current patch, play the current patch track level as well. If in record mode, and if the user is altering values at the current patch, record the current level in the currently selected patch track at the correct relative MTC time.
User requests MIDI patching	Provide a MIDI patch grid which indicates how users are patched to instruments and how instruments are patched to users. It also indicates the receive channels and transmit channels of the instruments.
User requests to patch from workstation to resource.	Switch off MIDI transmission to and from sequencer. Perform the MIDI patching if possible, otherwise give an error message. Indicate that the patch has been performed. Switch on MIDI transmission.
User requests to patch from resource to workstation.	Switch off MIDI transmission to and from sequencer. Perform the MIDI patching if possible, otherwise give an error message. Indicate that the patch has been performed. Switch on MIDI transmission.
User unpatches a MIDI patch from workstation to resource.	Switch off MIDI transmission to and from sequencer. Perform the necessary MIDI unpatching. Indicate that unpatching has been performed. Switch on MIDI transmission.
User unpatches a MIDI patch from resource to workstation	Switch off MIDI transmission to and from sequencer. Perform the necessary MIDI unpatching. Indicate that unpatching has been performed. Switch on MIDI transmission.
User requests information regarding a resource.	If the resource is not a library resource, indicate this, otherwise provide the information.
User changes a receive channel for a particular resource.	If it is a non-library resource, indicate that channels cannot be changed. Otherwise, send the appropriate channel change information to the resource.
	Check whether tape deck is booked. Display multitrack commands. Start up the tape timer.
User requests to control tape deck	Request tape deck mode status and tape deck counter value.
	Transmit this command to the tape deck in the

Tape timer ticks.	format it requires.
User gives a multitrack command.	Determine the current mode and display it. Determine the counter value and display it.
Tape issues mode status	Start the SMPTE card writing SMPTE code of the set frame type starting at the given origin.
Tape issues counter value.	Disable writing by the SMPTE card
User requests to write SMPTE	The SMPTE frame type is updated on the SMPTE card. The SMPTE start frame is initialized to zero. Send SMPTE start frame value to SMPTE card
User requests to stop SMPTE write User provides SMPTE frame type User provides SMPTE start frame	Stop transmission of MIDI data to and from sequencer. Update the audio patch info to reflect the patches that are now freed up. Request the audio patcher to return patched nodes to the free pool. Update the MIDI patcher info to reflect patches that are now freed up. Clear patches on the MIDI patcher. Update the booking information. Inform users with booking sheets of the freed resources. Record the workstation usage time.
User logs out	List the available configuration commands Store information regarding this new patch bay.
	If there are audio ins and outs still connected, give a 'patch bay in use' message. Otherwise delete record of patch bay. Store information regarding the unit.
	If there are audio outs still connected, give a 'unit in use' message. Otherwise, delete record of unit. Store information regarding this new patch bay.
Manager requests to start configuration	
Manager adds an audio patch bay with a particular network ID, matrix position and node number.	If there are MIDI ins and outs still connected, give a 'patch bay in use' message. Otherwise delete record of patch bay.

Manager deletes an audio patch bay with a particular matrix position.	
Manager adds a gain unit with a particular row number and network ID.	
Manager deletes a gain unit with a particular row number.	
Manager adds a MIDI patch bay with a particular matrix position and network ID.	
Manager deletes a MIDI patch bay with a particular matrix position.	
Manager requests resource list	Display resource list
Manager requests a resource type list	Display the list of resource types
Manager requests to add a resource from the resource type list (library resource) with a given number of audio ins and outs, and MIDI outs	Add the resource to the resource list if there are sufficient audio ins, audio outs, MIDI ins and MIDI outs. If not, give an error message showing the available connections
Manager requests to add a non-library resource with a given number of audio ins and outs, and MIDI outs	Add the non-library resource to the resource list, if there are sufficient audio ins, audio outs, MIDI ins and MIDI outs. If not, give an error message showing the available connections. Alter MIDI and audio in and out connection information to reflect these new ins and outs.
Manager deletes a resource	Delete the resource from the resource list. If the resource is currently booked, give a 'resource booked' message, which indicates when the resource can be deleted.
Manager enters sysex channel no (device ID) for a library resource	Update the library resource information with this channel number
Manager requests for music resource to be made available for booking	Alter Music Resources store to make music resource available.
Manager requests for music resource to be made unavailable for booking	Alter Music Resources store to make resource unavailable for booking

Manager requests audio configuration	Display audio configuration information - suffixes, patch bay numbers, and resource names for audio ins and outs.
Manager provides the necessary information for an audio out connection to the audio patch bay from a music resource.	Associate this patch bay information with the audio out. If there is already a connection at the requested point, give an error message. If the resource is already booked, give an error message.
Manager provides the necessary information for an audio in connection from the audio patch bay to a music resource.	Associate this patch bay information with the audio in. If there is already a connection at the requested point, give an error message. If the resource is already booked, give an error message.
Manager requests MIDI configuration	Display MIDI configuration information - resource names, patch bay numbers, part names and receive channels for MIDI ins.
Manager provides a patch bay row and connector number for the MIDI out from an instrument.	Associate this patch bay number with the MIDI out.
Manager provides a patch bay column and connector number for the MIDI in to an instrument.	Associate this patch bay number with the MIDI in.
Manager requests information regarding a library resource	Display resource specific information for a library resource.
Manager provides a receive channel number for a library resource part	Update the resource information for the library resource part.
Manager provides a receive channel number and part name for a non-library resource	Update the resource information for the non-library resource
Manager requests to configure resource types	List the current resource types
Manager adds a new resource	Add new resource to resource info
Manager enters the number of parts	Associate the number of parts with the library resource
Manager enters a part name for a part	Associate the name with the library resource part
Manager enters textual information regarding the resource	Associate the information with the library resource.
Manager enters sysex expression for a library part	Associate the sysex expression with the library resource part.

Manager requests workstation configuration	Display workstation list
Manager provides an ethernet address for a particular workstation	Associate this ethernet address with the workstation
Manager provides the audio patch bay row and connector numbers for the left or right audio out from the workstation.	Associate this information with the audio out.
Manager provides the audio patch bay column and connector numbers for the left or right audio in to the workstation.	Associate this information with the audio in.
Manager provides the MIDI patch bay row and connector numbers for the workstation MIDI out	Associate this information with the MIDI out
Manager provides the MIDI patch bay column and connector numbers for the workstation MIDI in	Associate this information with the MIDI in
Manager requests user configuration	Display user list
Manager enters a usercode and password for a new user	Update the user information with this new usercode and password
Manager requests a usage report for all users	Generate a report of resource usage for all users
Manager requests a usage report for a particular user	Generate a report of usage for the particular user.
Manager requests usage to be initialized for all users	Initialize usage of resources for all users
Manager requests usage to be initialized for a particular user	Initialize resource usage for the particular user
Manager requests tape server configuration	Display the name of the tape deck (always TASCAM 238 in this system), the MIDI patch bay connection from the output of the tape's SMPTE card, and the number of utilized audio ins and outs (both of these numbers will be 7 in this first system).
Manager specifies the number of audio ins required into the tape deck	Determine whether it is possible to connect this number of audio ins. Give an error message if not and adjust audio in connection information if it is.

Manager specifies the number of audio outs required from the tape deck	Determine whether it is possible to connect this number of audio outs. Give an error message if not and adjust audio out connection information if it is.
Manager provides MIDI patch bay row and connector number for SMPTE card's MIDI connection.	Associate patch bay row and connector with tape deck.
Manager provides the necessary information for an audio out connection to the audio patch bay from a tape.	Associate this patch bay information with the audio out.
Manager provides the necessary information for an audio in connection from the audio patch bay to a tape.	Associate this patch bay information with the audio in.
MIDI byte arrives from MIDI resource	Store the byte, signal that byte has been received, pass on byte to sequencer.
MIDI byte arrives from sequencer	Store the byte, signal that the byte has been received, pass on byte to the MIDI patch bay.

Appendix E

A Data Dictionary for the Remote Studio Access Network

Logging In

```
start_system_request    =     * A request to start up the system at a
                              workstation *

usercode prompt   =     * a message requesting usercode entry *

Users =     {User}

User        =     * A user at a workstation. *
            =     * @usercode + password + work_time + workstation_ID_ref*
                  + log_status

log_status  =     *[on|off]

usercode    =     * A string that uniquely identifies a User,
                there is no distinction between uppercase
                and lower case *
            =     string8

string8     =     {alphanumeric_character}8

alphanumeric_character = * [A..Z|a..z|0..9|_] *

already_logged_on_msg =       * a message to indicate that user is already
                              logged on *

password prompt   =     * a message requesting password entry *

password    =     * A string which allows a User entry into the system*
            =     string8

incorrect_password      =     *message to indicate incorrect password*

user_commands     =     * a list of commands available to the user *

current_user      =     * the usercode of the user curremtly working at the
                        workstation *
                  =     usercode

work_time   =     * Time spent by the User on the Workstation *
            =     * integer, units:hours *

Workstations      =     {workstation}

Workstation =     * A computer from which a user can access the network
                  music resources *
                  =     @workstation_ID + ethernet_address + MIDI_out_row +
                        MIDI_out_connector + MIDI_in_col +
                        MIDI_in_connector

ethernet_address  =     * 12 hex digits *

workstation_ID    =     *1..8*
```

```
MIDI_out_row        =     * The row number of the MIDI patcher bay to which
                          the resource MIDI out is connected *
                          =     * 1..2 *

MIDI_out_connector        =     * The connector on the bay to which the
                                resource MIDI out is connected *
                          =     connector_number
                          =     * 1..16 *

MIDI_in_col         =     * The column number of the MIDI patcher bay to
                          which the resource MIDI in is connected *
                          =     * 1..2 *

MIDI_in_connector =     * The connector on the bay to which the resource
                          MIDI in is connected *
                          =     connector_number
                          =     * 1..16 *

log_out_request     =     * a request to log out of the system*
```

Booking

```
booking_sheet_request     =     *A request for the booking sheet*

Bookings      =     {Booking}
Booking             =     *  An indication of the time allocation of resource
                          to user *
                          =     {@music_resource_ID-ref + @usercode-ref +
                          booking_day + booking_time}

booking_sheet       =     * An indication of all the resources, times and the
                          booking status of resources at the given times.
                          There are five days of booking slots always
                          available *
                          =     {music_resource_name + usercode + booking_day +
                          booking_time}

booking_day =     * An indication of the day on which the booking was
                    performed *

                    =     * day + month + year *

day           =     *1..31*
month =     *1..12*
year          =     *1990..2050*

booking_Time                      =     * 0..23 *

music_resource_ID =     * integer *

booking_request     =     * A request by the user to book a resource at a
                          particular time *
                          =     music_resource_ID + booking_day +
                                booking_time

resource_status_s1.2.3    =     [available|unavailable]

unbooking_request =     * A request by the user to unbook a resource at a
                          particular time *
                          =     music_resource_ID + booking_day +
                                booking_time
```

```
usercode_request   =     * A request for the usercode of a user who has
                         booked a resource at a particular time *
                         =     music_resource_ID + booking_day +
                               booking_time

own booking indicator    =     * an indication that a booking slot is owned
                               by the current user *

other booking indicator =      * an indication that a booking slot is owned
                               by another user *

deselect_indicator             =     * an indication that a booking slot is
                                     open *
                               =     music_resource_ID + booking_day +
                                     booking_time

audio unpatch indicator =      * an indication of the audio unpatching of a
                               music resource *

MIDI unpatch indicator   =     * an indication of the MIDI unpatching of a
                               music resource *

unbooked resources display     =     * an indication of all the resources in
                                     a setup that are not currently booked *

audio patcher request    =     * refer to 'Packet format specification for
                               audio patcher/mixer unit' *

MIDI patcher request     =     * refer to 'Packet format specification for
                               MIDI patch unit' *

Music_resources    =     * all the Music resources known to the system *
                         =     {Multitrack_tapes + MIDI_resources}

Multitrack_tapes   =     {Multitrack_tape}

Music_resource     =     * common data:  music_resource_name + audio_ins +
                         audio_outs + MIDI_out_row + MIDI_connector *
                   =     [Multitrack_tape | MIDI_resource]

music_resource_name      =     string8

Multitrack_tape    =     @multitrack_ID + Multitrack_type +
                         tape_port_address + SMPTE_port_address
multitrack_ID            =     *integer*
tape_port_address =      *integer*
SMPTE_port_address       =     *integer*

Multitrack_type    =     @Multitrack_type_name + mode_status_id +
                         mode_header_no + mode_tail_no + counter_status_id +
                         counter_header_no + counter_tail_no +
                         rewind_command_code + rewind_status_value +
                         rewind_status_request + fastfwd_command_code +
                         fastfwd_status_value + fastfwd_status_request +
                         stop_command_code + stop_status_value +
                         stop_status_request + play_command_code +
                         play_status_value + play_status_request +
                         pause_command_code + pause_status_value +
                         pause_status_request + recpause_command_code +
                         recpause_status_value + recpause_status_request +
                         record_command_code + record_status_value +
                         record_status_request + track1_command_code +
                         track2_command_code + track3_command_code +
                         track4_command_code + track5_command_code +
```

```
                        track6_command_code + track7_command_code

audio_ins    =      * 1..8 *
audio_outs   =      * 1..8 *

MIDI_resources      =      {MIDI_resource}

MIDI_resource       =      @MIDI_resource_ID + MIDI_in_col + MIDI_connector +
                           MIDI_resourceType_name-ref

MIDI_resource_part        =      MIDI_resource_ID + part_name +
                                 receive_channel

receive_channel     =      channel

channel             =      * 1..16 *

part_name    =      * the name of the MIDI resource part *
             =      string8

MIDI_resource_types       =      {MIDI_resource_type}

MIDI_resource_type        =      * The specification for a particular type of
                                 MIDI resource, eg. DX7. There can be many
                                 MIDI resources which are of this type*
                          =      @MIDI_resource_type_name + number_of_parts +
                                 MIDI_resource_info

MIDI_resource_part_type =        * the type specification for a part within a
                                 resource type *
                                 =      @resource_type_name + @part_name +
                                        sysex_expression

MIDI_resource_info        =      * information about the voices/parts in the
                                 resource and its channel assignments *
                                 =      {alphanumeric_character}500

number_of_parts     =      * the number of separate channel-assignable parts
                           in the resource *
                           =      * 1..16 *

sysex_expression    =      * a list of bytes and arithmetic expressions which
                           can be used by an interpreter to send out a system
                           exclusive message *
                           =      F0 + byte_string + F7
byte_string         =      expression | byte_string + expression
expression   =      * an expression incorporating hex digits, the receive
                    channel (R), sysex channel (S), new channel number (N),
                    hex digits, and arithmetic and logical operators *

expression   =      term | byte_expression + add_level_op + term
term                =      factor | term + mult_level_op + factor
factor              =      number | '(' expression ')'
number              =      byte | R | S | N
byte                =      hex_digit + hex_digit

Resource_time       =      * The amount of time spent by a user on a
                           particular resource *
                    =      @music_resource_ID_ref + @usercode_ref + time_used
time_used           =      * 0..744 *
```

Audio/MIDI patching

```
start_up_audio     =      *A request to start audio patching*

audio_patch_info   =      * information from which the user will do patching
                          *
                          =      {audio_out_info} + {audio_in_info}  +
                                 {audio_patching_info}

audio_out_info             =      {(music_resource_name + audio_out_name)  +
                                  gain_ampl}

audio_in_info              =      {(music_resource_name + audio_in_name)}

audio_patching_info        =      {[music_resource_name + audio_out_name |
                                  "workstation" + audio_out_side] +
                                  [music_resource_name + audio_in_name |
                                  "workstation" + audio_in_side] +
                                  patch_volume}
patch_volume               =      *the volume at a particular patch point*
                          =      *0..127*

audio_patch_info_request_w1.4.2      =      *a request for audio patching
                                            info*
                                            =      current_user

audio_patch_info_s1.4.2 =      * audio patching info sent to workstation *

Audio_patcher_mixer        =      @matrix_row + @matrix_col + network_ID +
                                  total_nodes + nodes_left
matrix_row          =      * the row position in the audio patch matrix*
                          =      *1..2*
matrix_col          =      *the column position in the audio patch matrix*
                          =      *1..2*
network_ID          =      * 0..127 *
total_nodes         =      * the total number of patch nodes in the patch bay
                          *
                          =      * 1..64 *
nodes_left          =      * the patch nodes left over for use *
                          =      * 1..64 *

Audio_outs  =      { Workstation_audio_outs + Resource_audio_outs}

Audio_out   =      * This is an output connection from the audio out of a
                   music resource or workstation to the audio patch bay *
                   common data:  *  gain + connection_indicator *
            =      [Workstation_audio_out | Resource_audio_out]

connection_indicator =  * an indication of whether there is a connection at
                        this connection point or not *

patch_bay_connector        =      * a connection point on a patch bay *
                          =      * 1..16 *

Workstation_audio_out      =      matrix_row + audio_out_connector +
                                  @workstation_ID_ref + @audio_out_side

audio_out_connector        =      patch_bay_connector

audio_out_side      =      left_right
left_right  =      * an indication of the left or right audio channel *
                   =      * [left|right] *
```

```
Resource_audio_out      =     matrix_row + audio_out_connector +
                              @music_resource_ID_ref + @audio_out_name
Audio_ins    =     {Workstation_audio_ins + Resource_audio_ins}

Audio_in     =     * This is an input connection to a music resource or
                   workstation from the patch bay *
                   common data:  * *
             =     [Workstation_audio_in | Resource_audio_in]

Workstation_audio_in     =     matrix_col + audio_in_connector +
                               @workstation_ID_ref + @audio_in_side
audio_in_side       =     left_right

Resource_audio_in =     Matrix_col + audio_in_connector +
                        @music_resource_ID_ref + @audio_in_name

Audio_gain_unit     =     * This is a unit with 16 inputs which plugs into an
                          audio patcher/mixer. It allows for gain control on
                          all 16 inputs *

                          =     @matrix_row + network_ID

Audio_patch =     * A patch connection of an Audio out of the audio patcher
                  to an Audio input of the audio patcher. By default, the
                  overall level is set 80% full. *
                        =     @matrix_row + @audio_out_connector +
                              @audio_col + @audio_in_connector +
                              patch_volume + usercode_ref

current_patch       =     *The currently selected patch*
                    =     matrix_row + audio_out_connector + audio_col +
                          audio_in_connector

audio_patch_request       =     * A request by the user for an audio patch to
                                be made. The initial volume at the patch
                                point will be 80% of maximum *
                          =     [(music_resource_name + music_resource_ID +
                                audio_out_name) ┼ ("workstation" +
                                audio_out_side)] + [(music_resource_name +
                                music_resource_ID + audio_in_name) ┼
                                ("workstation" + audio_in_side)]

audio_patch_request_w1.4.3.1  =     audio_patch_request

audio_patch_indicator     =     * a display of an audio patch *
                                =     [(music_resource_name + audio_out_name
                                      + music_resource_ID)|("workstation" +
                                      audio_out_side)] +
                                      [(music_resource_name +
                                      music_resource_ID +
                                      audio_in_name)┼("workstation" +
                                      audio_in_side)] + patch_on_indicator
patch_on_indicator        =     "X"
patch_off_indicator       =     " "

( The assumption here is that the user/user interface carries knowledge of
the actual patch connections )

audio_unpatch_request     =     * A request by the user for an audio unpatch
                                to be made *
                          =     [music_resource_name + music_resource_ID +
                                audio_out_name ┼ "workstation" +
                                audio_out_side] + [music_resource_name +
                                music_resource_ID + audio_in_name |
```

```
                                "workstation" + audio_in_side]

audio_unpatch_indicator =       * a display of an audio unpatch *
                        =       [(music_resource_name +
                                music_resource_ID +
                                audio_out_name)|("workstation" +
                                audio_out_side)] +
                                [(music_resource_name +
                                music_resource_ID +
                                music_resource_in_name)|("workstation"
                                + audio_in_side] + patch_off_indicator

audio_patch_selection   =       * a request by the user to make a particular
                                patch the curent patch *
                        =       [(music_resource_name +
                                music_resource_ID +
                                audio_out_name)|("workstation" +
                                audio_out_side)] +
                                [(music_resource_name +
                                music_resource_ID +
                                audio_in_name)|("workstation" +
                                audio_in_side)]

audio_patch_selection_w1.4.4.1          =       audio_patch_selection
current_patch_s1.4.4.1  =       current_patch + patch_volume

'old' bar graph display of current patch  =     * a display of the current
                                                patch level in bar graph
                                                form *

start entering patch levels     =       *a request to start the entry of new
                                        patch level values at the current patch
                                        point*
stop entering patch levels      =       *a request to stop entering new patch
                                        level values at the current patch
                                        point*

patch_volume_level              =       *a new patch volume level*
                                =       amplitude

patch_volume_increment  =       * a value indicating by how much the value of
                                the current patch volume must be incremented.
                                There will be a limit to the incrementing of
                                the current patch value. *
                                =       *0..127*

patch_volume_increment_w1.4.4.3         =       patch_volume_increment +
                                                current_patch

patch_volume_decrement  =       * a value indicating by how much the value of
                                the current patch volume must be decremented.
                                There will be a limit to the decrementing of
                                the current patch value. *
                                =       *0..127*

patch_volume_decrement_w1.4.4.3         =       patch_volume_decrement +
                                                current_patch

current_patch_volume_s1.4.4.2 =         patch_volume

'new' bar graph display of current patch  =     *A display of the newly
                                                entered current patch value
                                                in bar graph form*
```

```
current_patch_level_display   =     *a display of the patch volume at the
                                    currently selected patch point*

patch_level_entry_on    =     * an event store indicating that new patch
                              levels are being entered*
patch_level_entry_off   =     * an event store indicating that patch levels
                              are not currently being entered *

gain_selection          =     * a request for a the gain on a particular
                              audio out to be the current gain*
                        =     [music_resource_name + music_resource_ID +
                              audio_out_name | "workstation" +
                              audio_out_side]

current_gain      =     *an indication of the current gain*
                  =     matrix_row + audio_out_connector

current_gain_s1.4.5.1   =     current_gain + gain_level

'old'_graph_display_of_current_gain =     *a display of the value of the
                                          current gain*
start_entering_gain_levels    =     **
stop_entering_gain_levels     =     **

gain_level_increment    =     * a value indicating by how much the
                              currently selected gain must be incremented.
                              There is a limit to how much the value can be
                              incremented *
                              =     *1..127*

gain_level_increment_w1.4.5.3 =     gain_level_increment + current_gain

gain_level_decrement    =     * a value indicating by how much the
                              currently selected gain must be decremented.
                              There is a limit to how much the value can be
                              decremented *
                              =     *1..127*

gain_level_decrement_w1.4.5.3 =     gain_level_decrement + current_gain

MTC_source        =     *an indication of the port from which MIDI Time
                        Code data must be read*
                        =     *[computer|studio]*

SMPTE_frame =     * A complete SMPTE frame *
            =     hours + minutes + seconds + frames
hours       =     * 0..23 *
minutes     =     * 0..59 *
seconds     =     * 0..59 *
frames      =     * 0..30 *

SMPTE_frame_type  =     *[24|25|30|30D]*

SMPTE_start       =     *the first SMPTE time on tape*
                        =     SMPTE_frame


new_MTC     =     * an indication that a new MIDI time code message has
                  arrived and that it is beyond the start frame*

Relative_time     =     * the difference between the current SMPTE time
                        (provided by the most recent MTC message) and the
                        start frame entered by the user, in terms of
                        frames. *
```

```
MIDI_byte_from_music_resource =     * storage for MIDI bytes that have come
                                    in to the workstation from a Music
                                    Resource such as a multitrack tape
                                    recorder *

Music_recorder_MIDI_byte      =     * an event which indicates that a byte
                                    has come in from a Music Resource such
                                    as a multitrack tape recorder *

MIDI_bytes_from_sequencer     =     * storage for MIDI bytes that have come
                                    in to the workstation from a sequencer
                                    *

Sequencer_MIDI_byte      =     * an event which indicates that a byte has
                               come in to the workstation from a sequencer *

Recording on/off  =     * requests to start or stop recording *

Audio_Patch_event =     * the patch level of a patch at a particular SMPTE
                        time*
                        =     @matrix_row + @audio_out_connector +
                              @audio_col + @audio_in_connector +
                              @relative_time + patch_volume

Audio_Patch_event_w1.4.7.1    =     Audio_Patch_event

Patch_track =     {Audio_Patch_event}
Patch_tracks      =     {Audio_Patch_track}

Playback_on/off   =     *requests to start or stop playback*

non_current_track_playback_request_w1.4.7.4     =     current_patch +
                                                      relative_time +
                                                      current_user

audio_patch_display_of_non_current_tracks =     *a display of the patch
                                                volumes of the patch events
                                                within the non-current
                                                patch tracks*


current_track_playback_request_w1.4.7.7   =     current_patch +
                                                relative_time +
                                                current_user

audio_patch_display_of_current_track      =     *a display of the patch
                                                volume of the patch events
                                                within the current patch
                                                track*

patch_volume_level_request    =     * a request for the percentage of the
                                    outgoing audio signal which will be
                                    transmitted to the audio in of a
                                    resource or workstation at a particular
                                    patch point(input by the user) *
                              =     [music_resource_name + music_resource_ID +
                                    audio_out_name | "workstation" +
                                    audio_out_side] + [music_resource_name +
                                    music_resource_ID + audio_in_name |
                                    "workstation" + audio_in_side] + amplitude

patch_volume_level_indicator  =     * an indication of the percentage of
                                    the outgoing audio signal which will be
                                    transmitted to the audio in of a
```

```
                                    resource or workstation at a particular
                                    patch point *
                        =       [music_resource_name + music_resource_ID +
                                audio_out_name | "workstation" +
                                audio_out_side] + [music_resource_name +
                                music_resource_ID + audio_in_name |
                                "workstation" + audio_in_side] + amplitude

patch_status        =       * An indication of patching or unpatching *
                    =       [on|off]
volume              =       *0..127*

gain_level  =       * An audio equalizer message which indicates the gain at
                    a particular audio out *
                    =       network_ID + audio_out_connector + gain_indicator +
                            equalizer_level
gain_indicator      =       * An indicator that the next parameter is the gain
                            value *

MIDI_patching_request   =       *A request to do MIDI patching*
                                =       MIDI_patching_request_indicator +
                                        ethernet_address
MIDI_patching_request_indicator         =       'M'

resource_workstation_patches    =       * all the resources which are patched
                                        to the workstation *
                                        =       { music_resource_ID }

music_resources_with_MIDI_outs          =       * all resources booked by user
                                                with MIDI outs *
                                                =       {music_resource_name +
                                                        music_resource_ID}

music_resources_with_MIDI_ins           =       * all resources booked by user
                                                with MIDI ins *
                                                =       {music_resource_name +
                                                        music_resource_ID}

workstation_resource_patches    =       * all the resources to which the
                                        workstation is patched *
                                        =       {music_resource_ID}

MIDI_resource_channel_info      =       * the current receive channel numbers
                                        of all the parts of all the user's
                                        resources *
                                        =       { MIDI_resource_ID + {part_name +
                                                receive_channel}}

MIDI_resource_channel_info_request_w1.5.1.3     =       current_user +
                                                        MIDI_resource_channel
                                                        _info

MIDI_patcher        =       * performs patching of MIDI outs to MIDI ins *
                    =       @matrix_row + @matrix_col + network_ID

MIDI_from_ws_patch_request=     * A request for a MIDI patch from a
                                workstation to a resource *
                            =       MIDI_resource_ID

MIDI_to_ws_patch_request        =       * a request for a MIDI patch from a
                                        resource to a workstation *
                            =       MIDI_resource_ID

MIDI_to_ws_patch_request_w1.5.2.1       =       MIDI_to_ws_patch_request +
```

```
                                        workstation_id

MIDI_from_ws_patch_indicator  =     * An indicator of a MIDI patch from a
                                    workstation to a resource *
                              =     MIDI_resource_ID + MIDI_resource_name +
                              patch_on_indicator

MIDI_to_ws_patch_indicator    =     * An indicator of a MIDI patch from a
                                    resource to a workstation *
                              =     MIDI_resource_ID + MIDI_resource_name +
                              patch_on_indicator

MIDI_from_ws_patch_request=   * A request for a MIDI patch from a
                              workstation to a resource *
                              =     MIDI_resource_ID

MIDI_from_ws_patch_request_w1.5.2.2 =     MIDI_resource_ID + workstation_ID

MIDI_to_ws_unpatch_request    =     * a request for a MIDI unpatch from a
                                    resource to a workstation *
                                    =     MIDI_resource_ID

MIDI_to_ws_unpatch_request_w1.5.3.1 =     MIDI_resource_ID + workstation_ID

MIDI_from_ws_unpatch_indicator      =     * An indicator of a MIDI unpatch
                                          from a workstation to a resource
                                          *
                              =     MIDI_resource_ID + MIDI_resource_name +
                              patch_off_indicator

MIDI_to_ws_unpatch_indicator  =     * An indicator of a MIDI unpatch from a
                                    resource to a workstation *
                              =     MIDI_resource_ID + MIDI_resource_name +
                              patch_off_indicator

MIDI_patcher_patch            =     * a message to a specific MIDI patcher
                                    to patch a MIDI out to a MIDI in *
                              =     MIDI_patcher_network_ID +
                                    MIDI_out_connector + MIDI_in_connector
                                    + patch_status

receive_channel_change  =     * The MIDI system exclusive message which
                              will perform the actual receive channel
                              change for a particular voice of the
                              synthesizer *
                              =     F0 + {byte} + F7

receive_channel_request =     * a request to associate a channel number
                              with a MIDI resource part *
                              =     library_resource_ID + part_name +
                                    channel

Music_resource_info_request   =     * a request for resource specific info
                                    regarding voices and channels on a
                                    particular music resource. It will only
                                    be possible to request this info for a
                                    device in the library of resources. *
                                    =     music_resource_name

Music_resource_info     =     * a page of text describing the voices used
                              and channels assigned in a particular music
                              device *
```

Setups

```
load_Setup_Request                  =     * A request to load a setup which
                                          will comprise information to
                                          correctly configure the audio and
                                          MIDI patch bays *
                                    =     setup_name
load_setup_request_s1.3.1.1   =     *a setup request from workstation to
                                    server*
                                    =     usercode + setup_name

save_Setup_request                  =     * A request to save a setup *
                                    =     setup_name + ethernet_address +
                                          save_setup_indicator
save_setup_indicator          =     'S'

Setups       =     {Setup}
Setup =     *this gives information required to book resources, to perform
            the required audio and MIDI patching of those resources, and to
            change channels correctly*
            =     @usercode + @setup_name + {resource_ID} +
                  {audio_setup_patch} + {MIDI_patch_to_ws} +
                  {MIDI_patch_from_ws} + {MIDI_resource_part}

Current_setup     =     *Same format as Setup, but is temporarily used to
                        contain the latest setup info. This may differ from
                        the Setups store since certain resources may not
                        have been booked*

audio_setup_patch =     * information about the instrument/workstation
                        patch connections. This is not done at the level of
                        hardware connections in case the configuration
                        changes *

                        =     audio_out_member + audio_in_member

audio_out_member  =     ['w' + audio_out_side|music_resource_ID +
                        audio_out_name]

audio_in_name             =     ['w' + audio_in_side|music_resource_ID +
                                audio_in_name]
unbooked_resource_display       =     *A display of resources which are
                                      needed for setup but are not currently
                                      booked*
                                      =     {resource_name}

unbooked_resources        =     *A list of resource ID's of resources which
                                are in the selected setup but which have not
                                been booked*
                          =     {resource_ID}

unbooked_resources_w1.3.1.1     =     unbooked_resources

Patch_Server      =     * The patch server which, amongst other things, may
                        be responsible for tape transport control. A MIDI
                        line runs from the server to the central patch bay,
                        allowing for the flow of synchronization
                        information to the workstations. *
            =     ethernet_address
```

Tape Control

```
multitrack_control_request      =     **

multitrack_control_request_w1.6.1   =     multitrack_control_request +
                                          current_user

multi_track_not_booked          =     *indication that multitrack is not
                                      booked*

enable_tape_timer_w1.6.1                  =     *message to start tape
                                                timer ticking for update of
                                                tape status*

tape_user                 =     * the current user of the multitrack *
                          =     usercode

exit_multitrack_request         =    **

exit_multitrack_request_w1.6.2        =     exit_multitrack_request +
                                            current_user

multitrack_commands       =     * a display of all the multitrack tape
                                commands *

incorrect_frame_type      * a message indicating that the start frame is of
                          the incorrect frame type *

multitrack_tape_command =     * A command to perform a specific function on
                              the multitrack tape recorder eg. record, play
                              *
                              =     multitrack_command_indicator +
                                    [record|pause|stop|play|rewind|fastforw
                                    ard|track1_rec|track2_rec|track3_rec|tr
                                    ack4_rec|track4_rec|track5_rec|track6_r
                                    ec|track7_rec]

multitrack_command_code =     * A multitrack command coded in a form that
                              the tape deck can understand *
                              =     1{alphanumeric}3

SMPTE_write_request       =     * will cause SMPTE to be written to track 8
                              of the multitrack *

SMPTE_stop_write_request        =     * will cause SMPTE to stop being
                                      written to track 8 of the multitrack *

mode_status_request             =     *a request for the tape deck to tell
                                      what its current mode is*
                                =     {alphanumeric}
counter_value_request     =     *a request for the tape deck to tell what its
                              current counter value is*
                              =     {alphanumeric}
tape_mode_status          =     *an indication from the tape deck of what its
                              current mode is*
                              =     {alphanumeric}
tape_counter_value              =     *an indication from the tape deck of
                                      what its current counter value is*
                                =     {alphanumeric}
rewind_on                       =     *an indication on the display that the
                                      tape is rewinding*
fast_forward_on           =     *as above but for fast forward*
play_on                         =     *as above but for play*
stop_on                         =     *as above but for stop*
record/pause_on           =     *as above but for record/pause*
```

```
record_on                       =     *as above but for record*
pause_on                        =     *as above but for pause*
```

Configuration

```
Configuration_request           =     **

Configuration_command_list      =     * a list of configuration commands
                                      which allow the manager to fully
                                      configure the music network *
```

Configuration of Patch Bays

```
audio_patcher_add_msg   =     * all the information required to configure
                              an audio patch bay into the system *
                              =     network_ID + total_nodes +
                                    matrix_position

audio_patcher_del_msg   =     * information required to de-configure an
                              audio patch bay from the system *
                              =     matrix_position

MIDI_patcher_add_msg    =     * the information required to configure a
                              MIDI patch bay into the system *
                              =     network_ID + matrix_position

MIDI_patcher_del_msg    =     * the information required to de-configure a
                              MIDI patch bay from the system *
                              =     matrix_position

equalizer_add_msg =     *all the information required to configure an
                        equalizer into the system*
                        =     network_ID + matrix_row

equalizer_del_msg =     * the information required to de-configure an
                        equalizer from the system*
                        =     matrix_row
```

Configuration of Resources

```
Resource_list_request   =     * a request to see all the available
                              resources in the system *

resource_list     =     * a list of all the available resources *
                  =     {music_resource_name}

MIDI_resource_type_list_request     =     * a request to see all the
                                          available MIDI resource types in
                                          the system *

MIDI_resource_type_list =     {MIDI_resource_type_name}

MIDI_resource_type_add_msg    =     * the name of a MIDI_resource_type to
                                    add to the current resources*
                              =     MIDI_resource_type_name

resource_delete_msg     =     * the name of a resource to be deleted from
                              the current resources *
```

```
                          =     resource_name

Resource_in_use_message   =     *indicates that resource cannot be deleted*

MIDI_resource_device_ID   =     * a message indicating the device ID of a
                                MIDI resource *
                          =     music_resource_ID + device_ID
device_ID                 =     * 1..32 *

resource_audio_ins        =     * the number of audio ins on the resource *
                          =     music_resource_ID + audio_in_no
audio_in_no       =     * 1..16 *

insufficient_audio_ins    =     * a message indicating insufficient audio ins
                                on the patcher/mixer *

resource_audio_outs       =     * the number of audio outs on the resource *
                          =     music_resource_ID + audio_out_no
audio_out_no              =     * 1..16 *

insufficient_audio_outs   =     * a message indicating insufficient audio
                                outs on the patcher/mixer *

resource_MIDI_out         =     * indication of MIDI out on resource *
                          =     * [present|not present] *

insufficient_MIDI_outs    =     *A message indicating insufficient MIDI outs
                                on mixer*
Audio_configuration_request   =     **

Audio_in_list     =     * Indication of all possible audio ins and their
                        names and patch bay numbers *
                  =     {music_resource_name + music_resource_ID +
                        audio_in_name + matrix_col + connector}

Audio_out_list    =     * Indication of all possible audio outs and their
                        names and patch bay numbers *
                  =     {music_resource_name + music_resource_ID +
                        audio_out_name + matrix_row + connector}

audio_out_connection      =     * the resource ID and name of an audio out
                                from a music resource and its associated
                                connector *
                          =     matrix_row + audio_out_connector +
                                music_resource_ID + audio_out_name
audio_out_name            =     string3
connection_used   =     *A message indicating that the required connection
                        is already in use*
audio_in_connection       =     * the resource ID and name of an audio in to
                                a music resource and its associated connector
                                *
                  =     matrix_col + audio_in_connector + music_resource_ID
                        + audio_in_name
audio_in_name     =     string3

MIDI_configuration_request    =     **

MIDI_in_list      =     * Indication of all possible MIDI ins, their
                        resource names, patch bay numbers and current
                        receive channel assignments *
                  =     {MIDI_resource_name + MIDI_resource_ID + MIDI_col +
                        connector + {part_name + receive_channel}}

MIDI_out_list     =     * Indication of all possible MIDI outs, their
```

```
                    resource names and patch bay numbers *
                =   {MIDI_resource_name + MIDI_resource_ID + MIDI_row +
                    connector}

MIDI_in_connection      =   * the ID of a resource with a MIDI in
                            together with its connector *
                =   MIDI_col + MIDI_in_connector + MIDI_resource_ID

MIDI_out_connection     =   * the ID of a resource with a MIDI out
                            together with its connector *
                =   MIDI_row + MIDI_out_connector + MIDI_resource_ID
```

Setting up resource parts

```
MIDI_resource_part_channel      =   *a receive channel number for a MIDI
                                    resource part to receive on*
                                    =   MIDI_resource_ID + part_name +
                                        receive_channel

MIDI_resource_type_info_request     =   *a request for information
                                        regarding the parts and channel
                                        setting of a MIDI resource type*
                                    =   MIDI_resource_type_name

MIDI_resource_type_info =   *information regarding the parts and channel
                            setting of a MIDI_resource_type*
```

Configuration of Workstations and Server

```
workstation_configuration_request   =   **

workstation_list    =   * a list of all the workstations in the system with
                        their ethernet addresses and patch connections *
                        =   {workstation_ID + ethernet_address +
                            matrix_col + left_audio_in_connector +
                            matrix_col + right_audio_in_connector +
                            matrix_row + left_audio_out_connector +
                            matrix_row + right_audio_out_connector +
                            matrix_col + MIDI_in_connector + matrix_row +
                            MIDI_out_connector}

workstation_audio_in_left       =   * the patch bay connector for the left
                                    audio lead to a workstation *
                                    =   workstation_ID + matrix_col +
                                        audio_in_connector
workstation_audio_in_right      =   * the patch bay connector for the right
                                    audio lead to a workstation *
                                    =   workstation_ID + matrix_col +
                                        audio_in_connector
workstation_audio_out_left      =   * the patch bay connector for the left
                                    audio lead from a workstation *
                                    =   workstation_ID + matrix_row +
                                        audio_out_connector
workstation_audio_out_right     =   * the patch bay connector for the right
                                    audio lead from a workstation *
                                    =   workstation_ID + matrix_row +
                                        audio_out_connector

workstation_MIDI_in     =   * the patch bay connector to the  MIDI in of
                            the workstation *
                        =   workstation_ID + matrix_col +
                            MIDI_in_connector
workstation_MIDI_out    =   * the patch bay connector to the  MIDI out of
```

```
                        the workstation *
                =       workstation_ID + matrix_row +
                        MIDI_out_connector

workstation_ethernet_address  =     * the ethernet address associated with
                                    the ethernet card in a particular
                                    workstation *
                                    =     workstation_ID + ethernet_address

server_ethernet_address       =     ethernet_address
```

Audio Tape Configuration

```
tape_configuration_request    =     **

tape_audio_in           =     * A connection from audio patch bay to tape
                                in *
                        =     matrix_column + audio_out_connector +
                              multitrack_name + tape_audio_in_no

tape_audio_out          =     * A connection from tape out to audio patch
                              bay *
                        =     matrix_row + audio_in_connector +
                              multitrack_name + tape_audio_out_no

tape_MIDI_out           =     * A connection from the MTC out of a tape or
                              its associated SMPTE/MTC card to the MIDI
                              patch bay *
                        =     multitrack_name + matrix_row +
                              MIDI_in_connector
```

User configuration

```
usercode_configuration_request      =     **

usercode_list     =     * a list of all the users and their associated
                        passwords *
                  =     {usercode + password}


new_password            =     **
                        =     usercode + password

single_user_report_request    =     * A request for a report on the usage
                                    of studio resources by a single user *
                                    =     usercode

all_user_report_request =     * A request for a report on the usage of
                              studio resources by all users *

single_user_init_request      =     * A request to initialize resource
                                    usage times for a single user to zero *
                              =     usercode
all_user_init_request   =     * A request to initialize resource usage
                              times for all users to zero *

all_user_usage_report   =     *an indication of the usage time of
                              workstations and resources by the users*
                  =     {usercode + workstation_time + {Music_resource_name
                        + music_resource_time}}
```

```
single_user_usage_report      =     *an indication of the usage time of
                                    workstations and resources by a single
                                    user*
                              =     usercode + workstation_time _
                                    {music_resource_name +
                                    music_resource_time}
```

MIDI_resource_type Configuration

```
MIDI_resource_type_configuration_request  =     **

MIDI_resource_type_name =     *name of a new MIDI_resource_type*
                              =     string8

MIDI_resource_type_parts_no   =     * the number of voices/parts associated
                                    with a MIDI resource type *
                                    =     *1..16*

MIDI_resource_part_type_chan_sysex  =     * a string which, when
                                          interpreted, will result in a
                                          channel change sysex expression
                                          being sent to an instrument *
                                          =     {sysex_expression,} +
                                          sysex_expression
sysex_expression        =     [hex_no | device_ID | rec_chan | new_rec_chan
                              | arith_expression]
hex_no            =     *2-digit hex no*
device_ID         =     'I'
rec_chan          =     'R'
new_rec_chan      =     'N'
arith_expression        =     *an expression comprising sysex_expressions
                              as operands and the operators +,-,&,||,^,!,
                              together with brackets*

MIDI_resource_type_info =     * a description of the resource for the
                              user's benefit - how the parts are assigned
                              to channels, etc.*
                              =     MIDI_resource_type_name + text_info
```

Appendix F

Audio Patching/Mixing Transforms

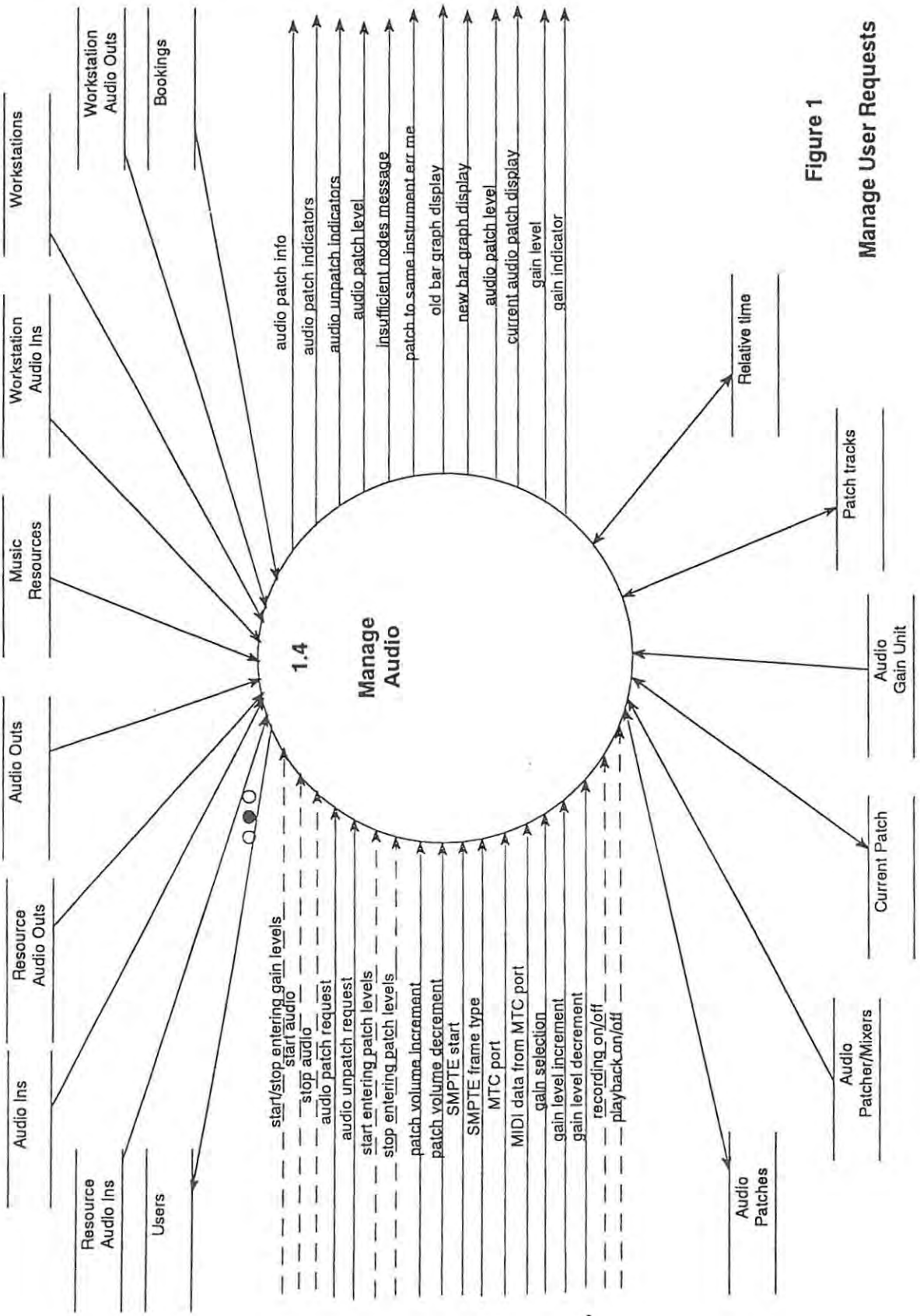

Figure 1

Manage User Requests

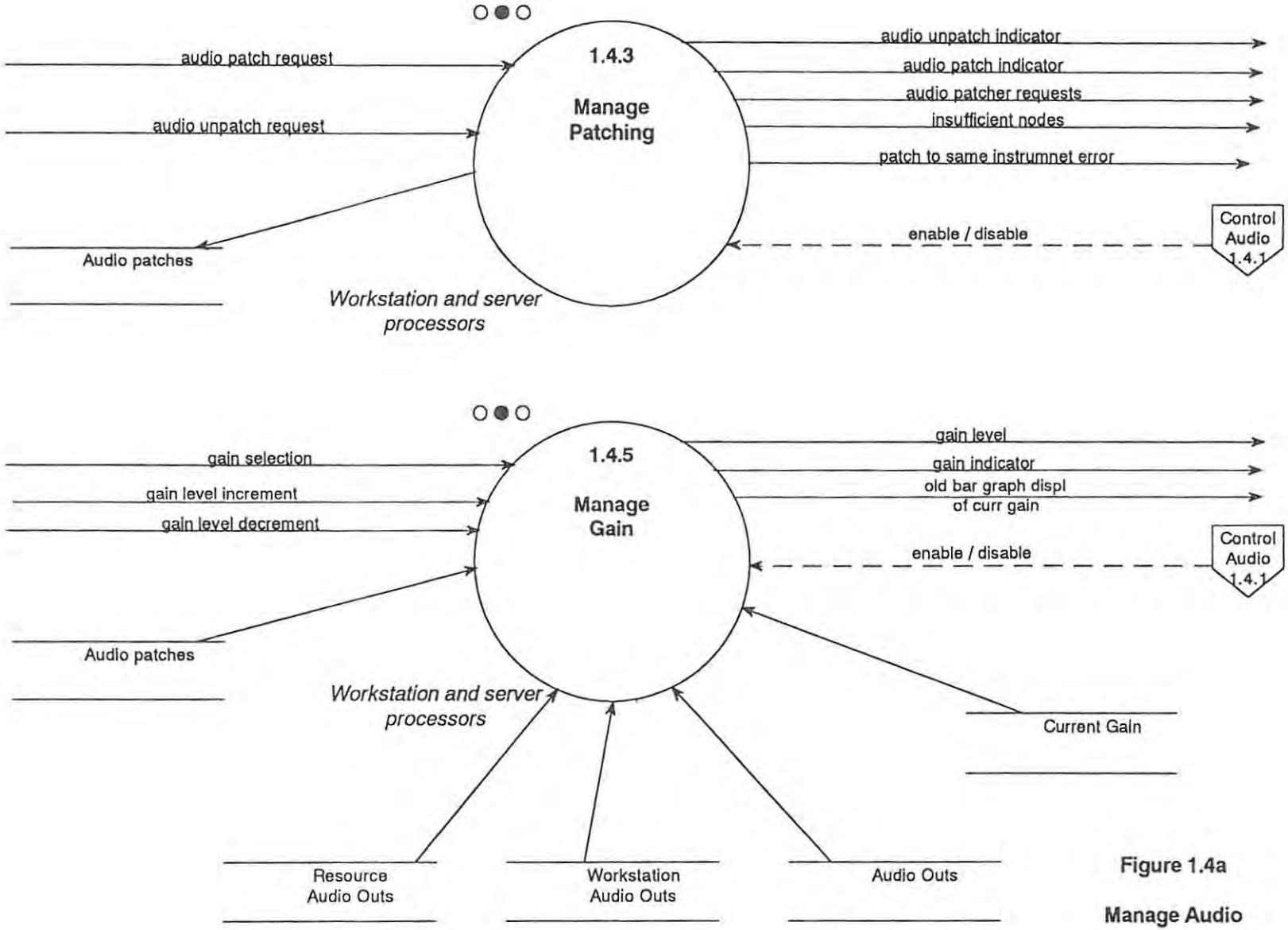

Figure 1.4a

Manage Audio

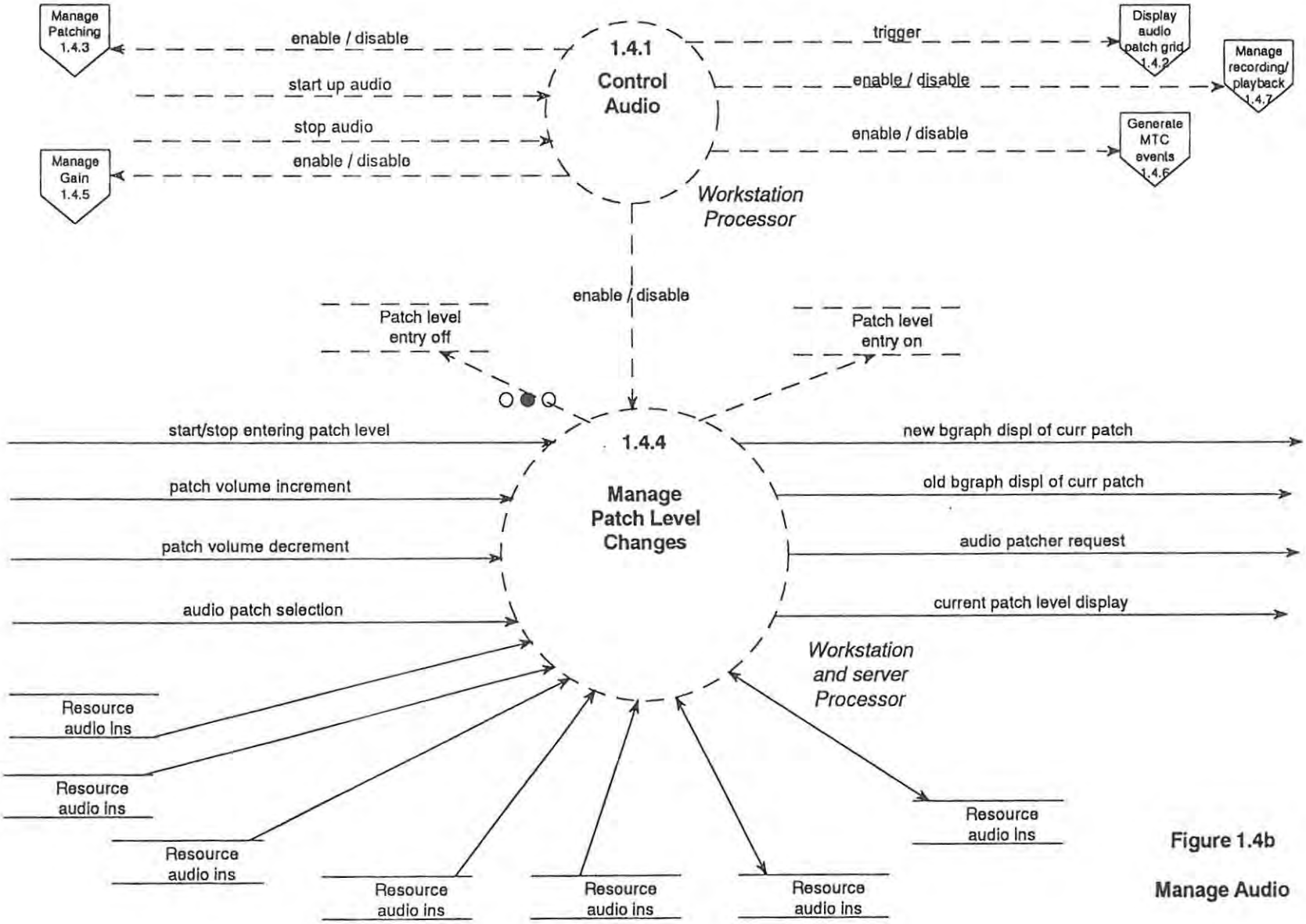

Figure 1.4b

Manage Audio

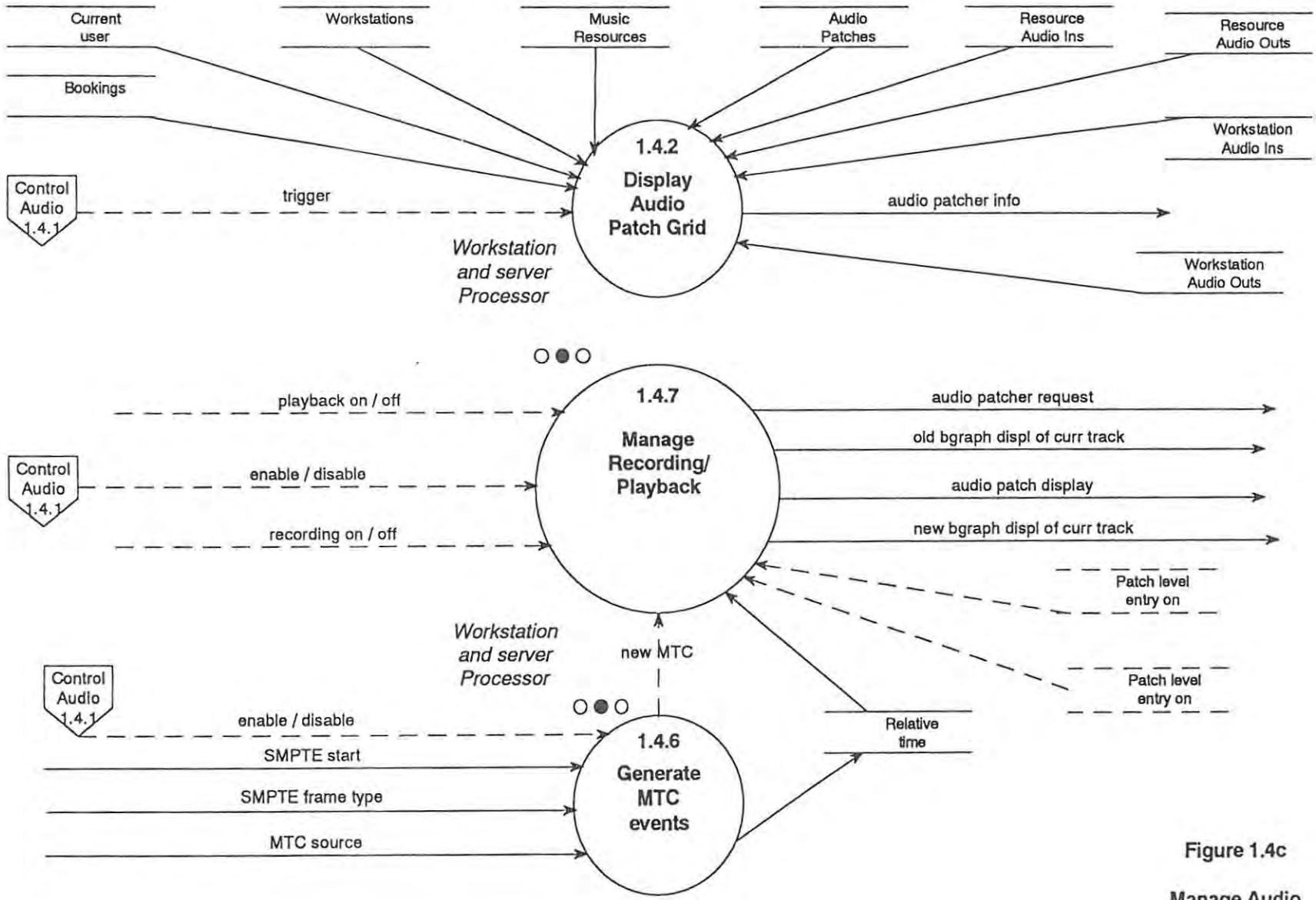

Figure 1.4c

Manage Audio

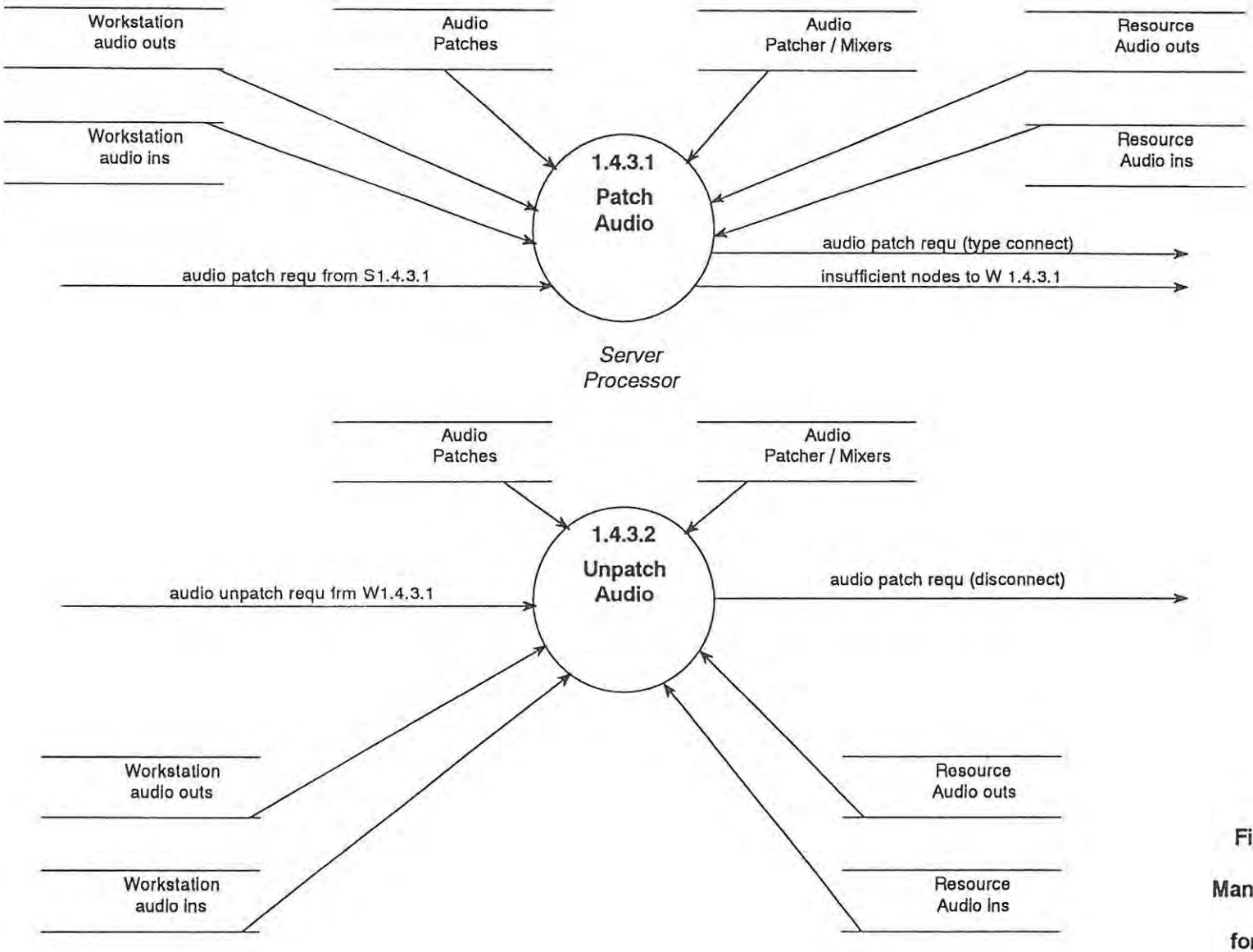

Figure 1.4.3a

Manage Patching

for SERVERS

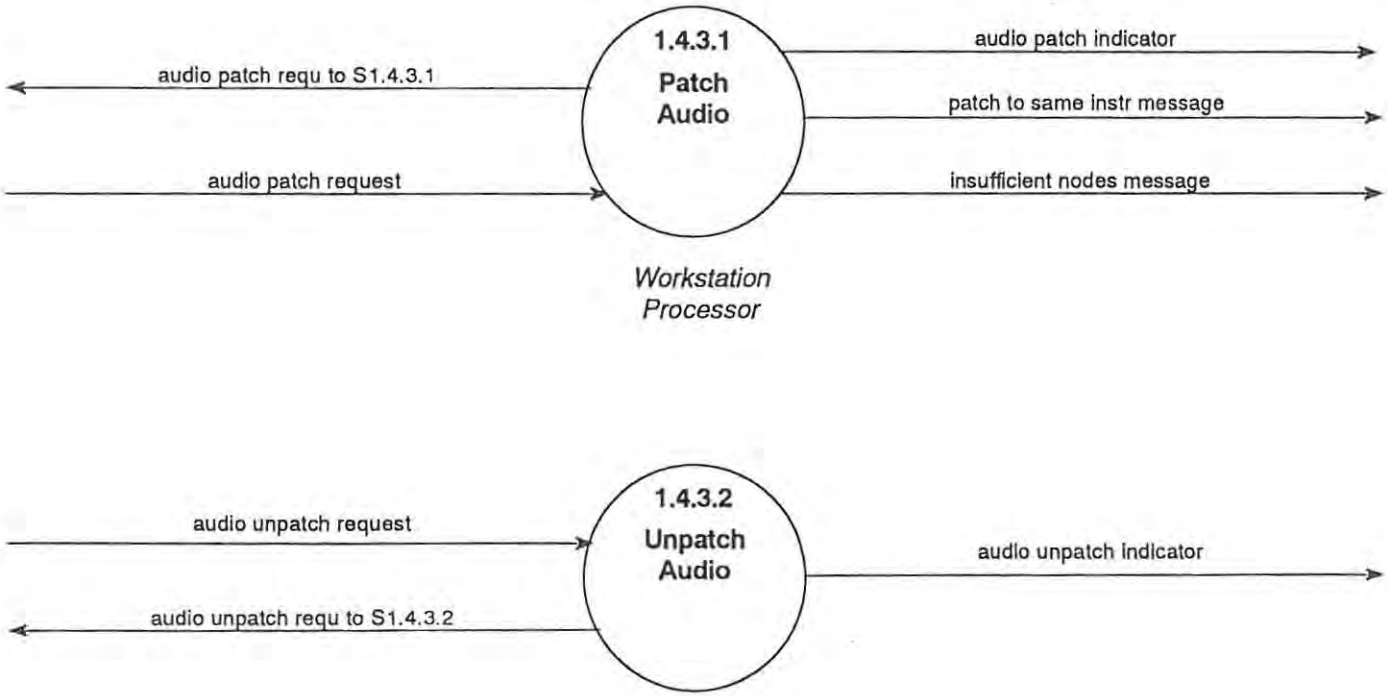

Figure 1.4.3b

Manage Patching

for WORKSTATIONS

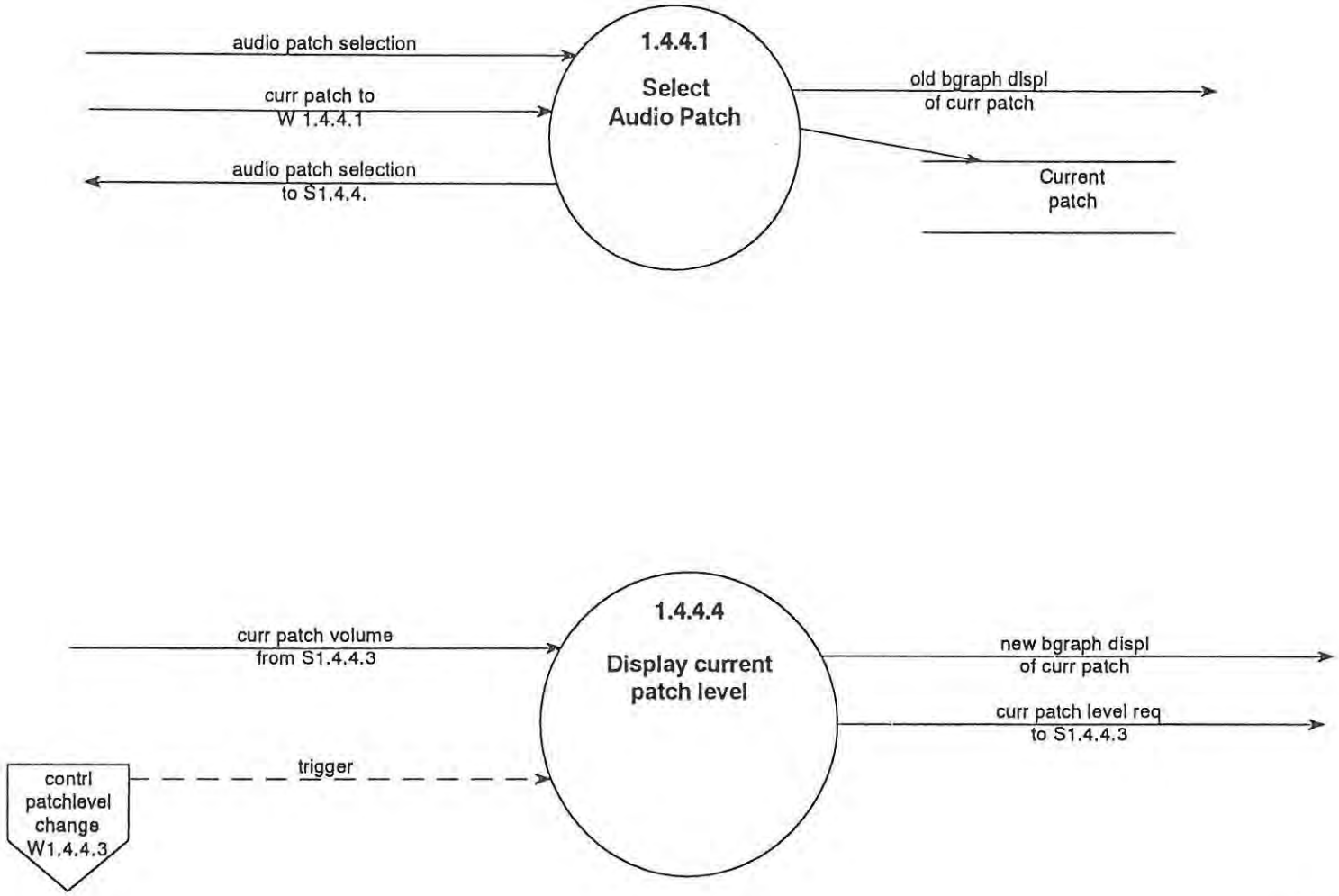

Figure 1.4.4a

Manage Patch Level Changes

for WORKSTATIONS

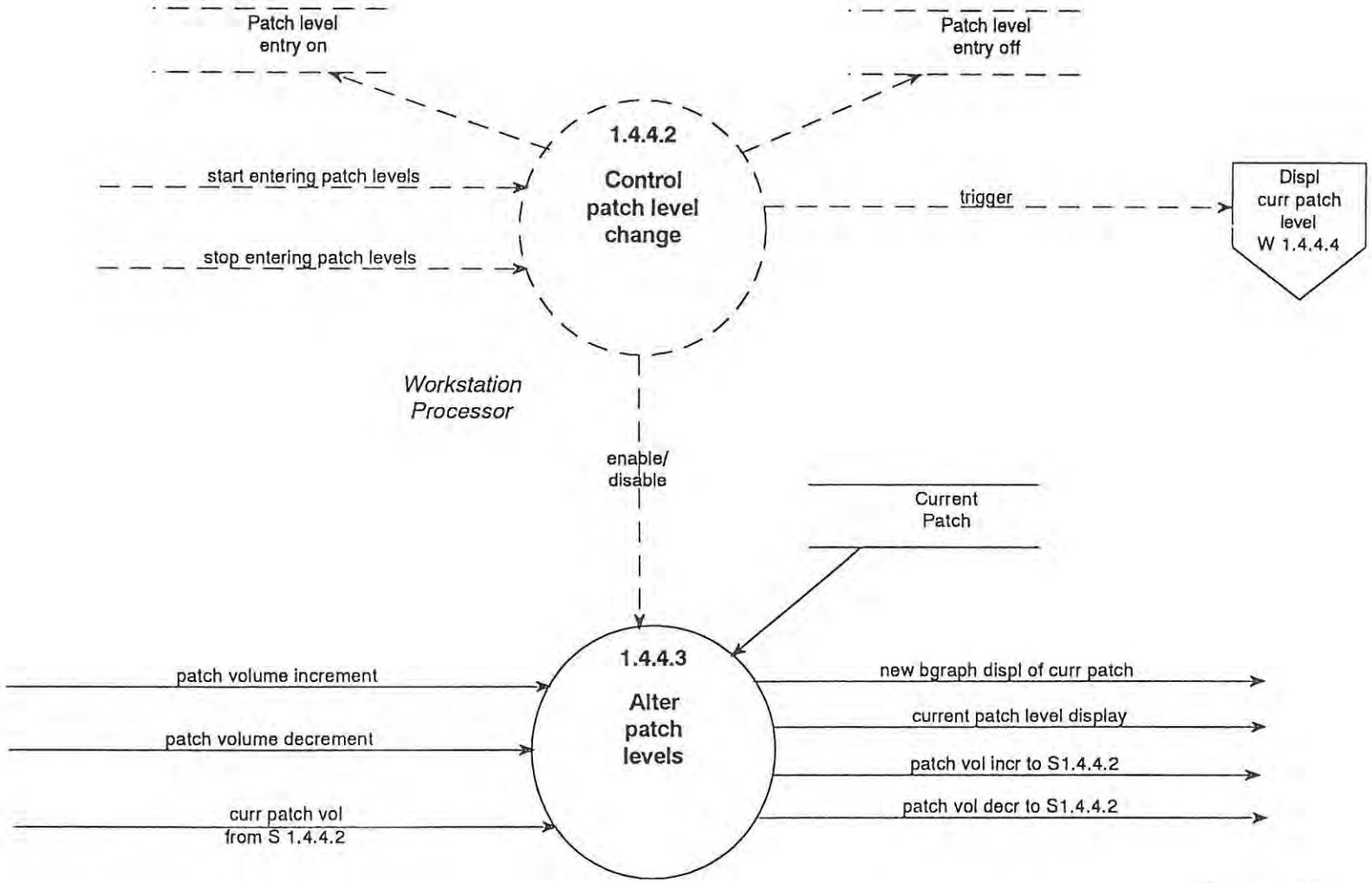

Figure 1.4.4b

Manage Patch Level Changes

for WORKSTATIONS

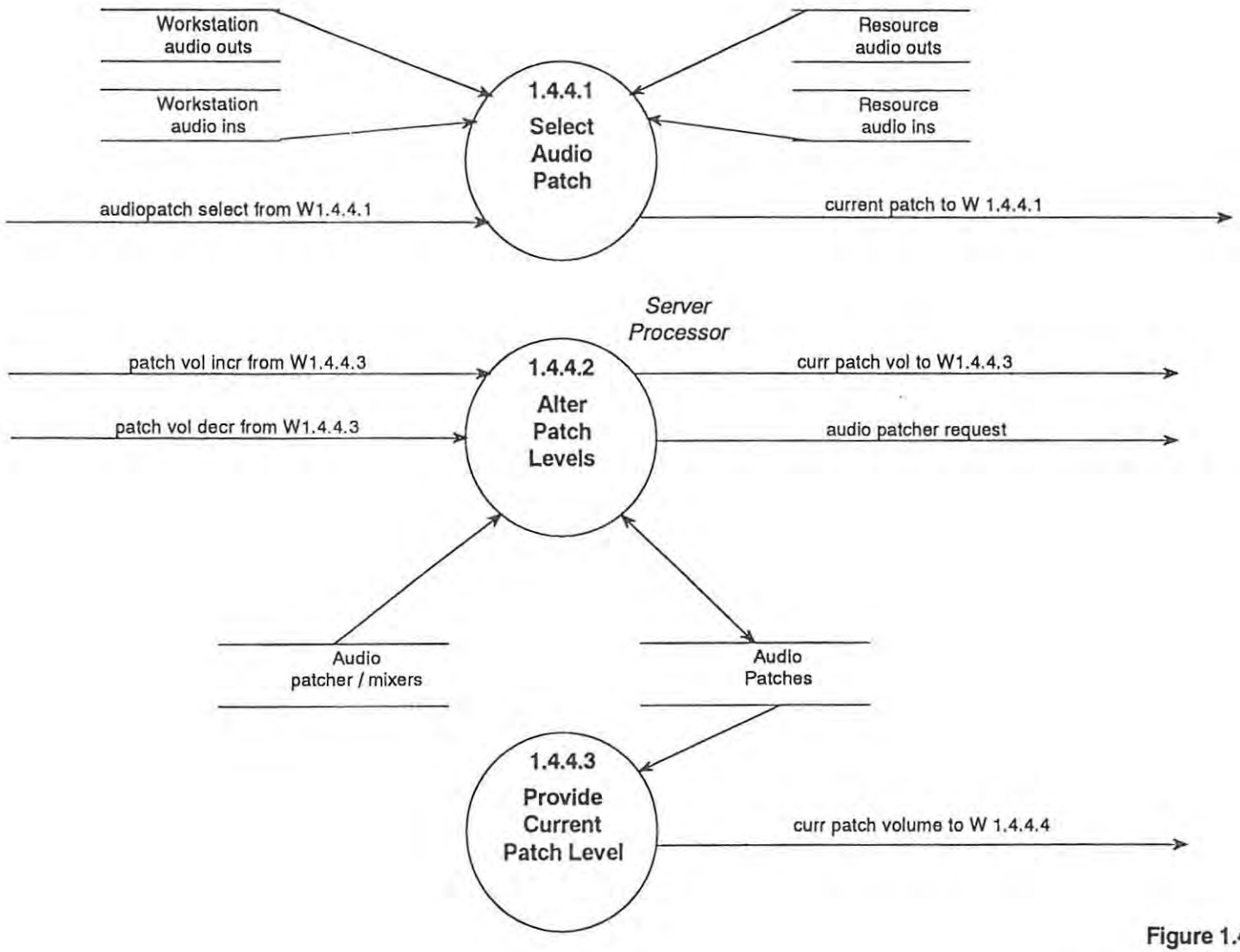

Figure 1.4.4c

Manage Patch Level Changes

for SERVERS

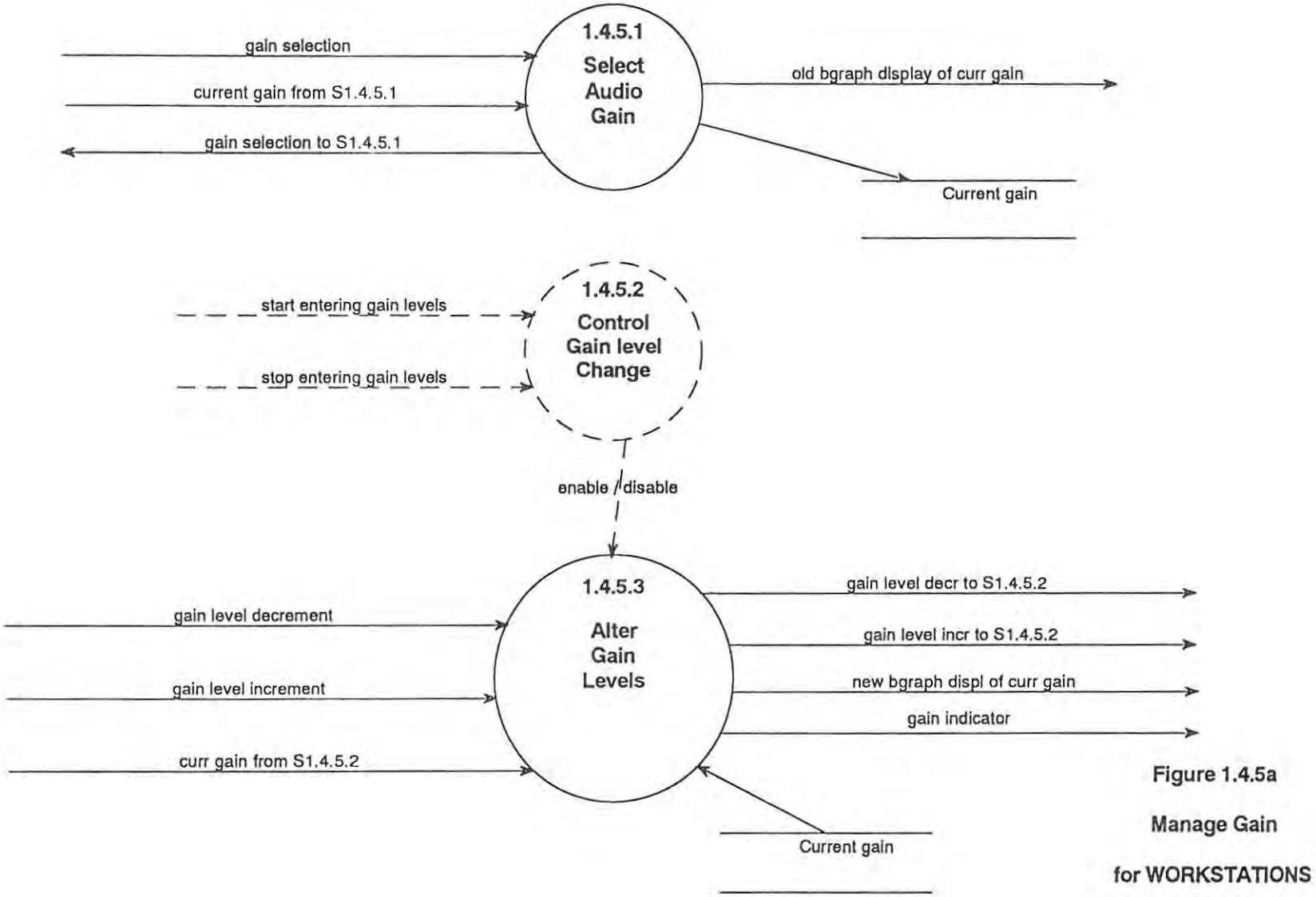

Figure 1.4.5a

Manage Gain

for WORKSTATIONS

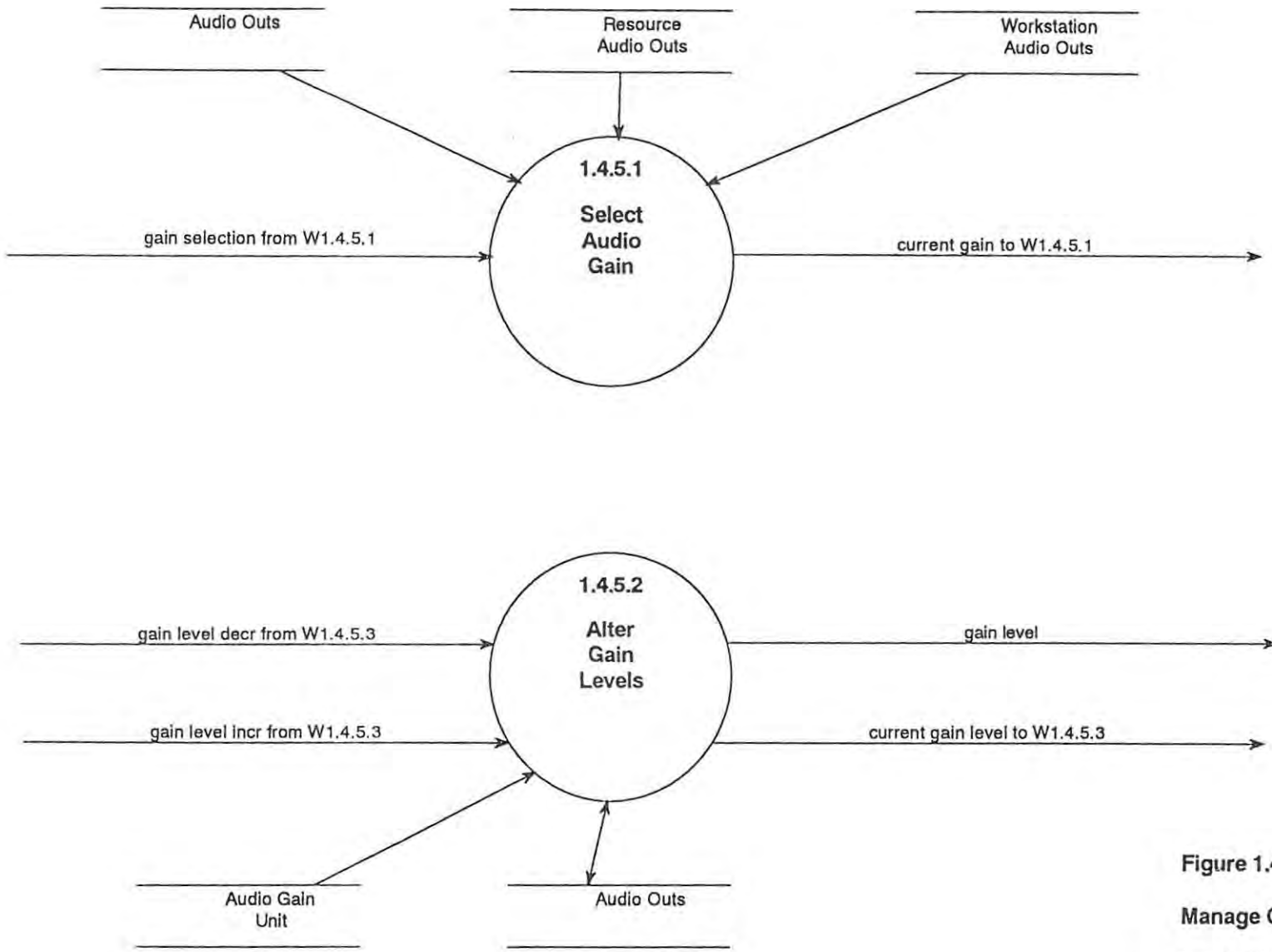

Figure 1.4.5b

Manage Gain

for SERVERS

Figure 1.4.6

Generate MTC events

for WORKSTATIONS

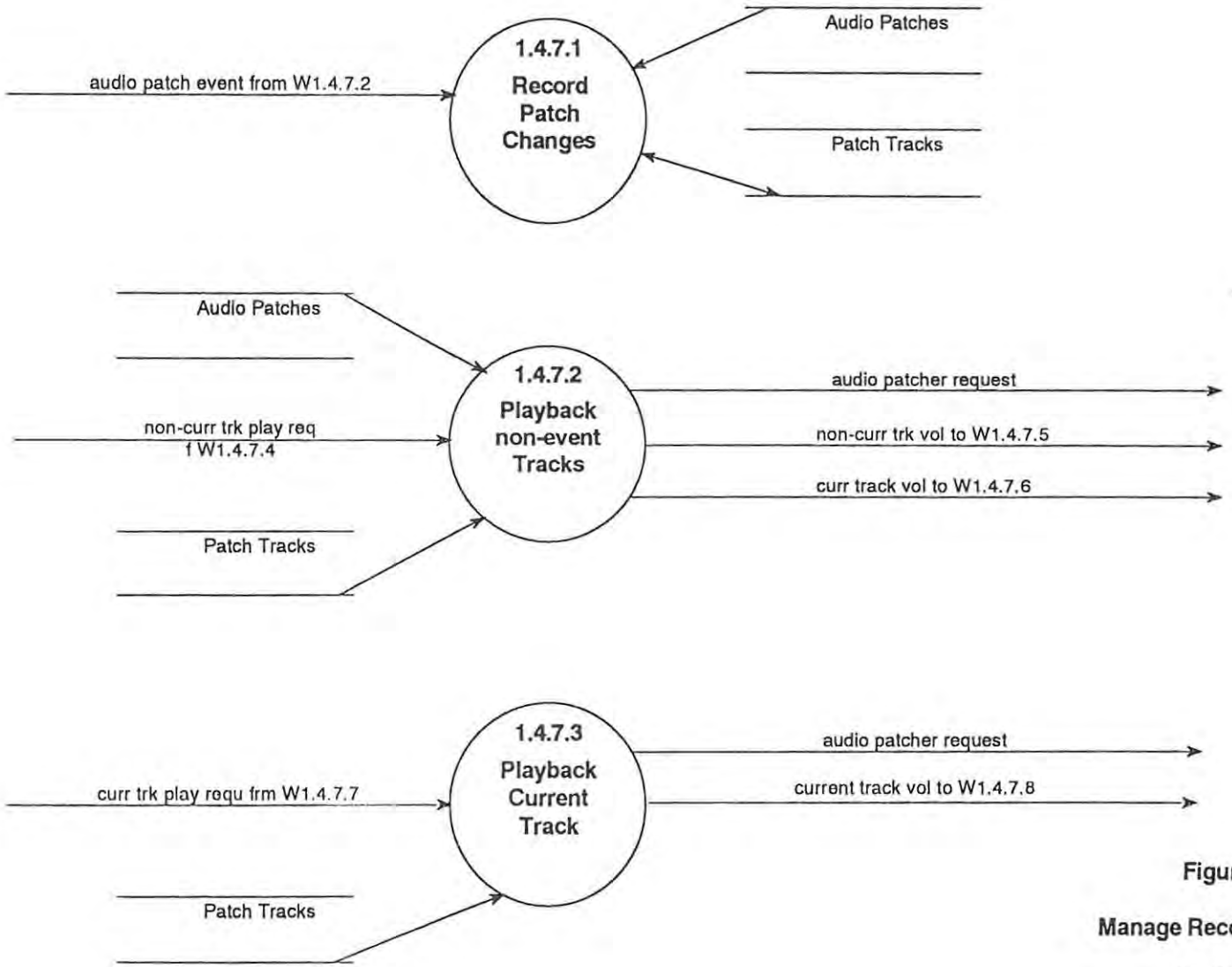

Figure 1.4.7a

Manage Recording/Playback

for SERVERS

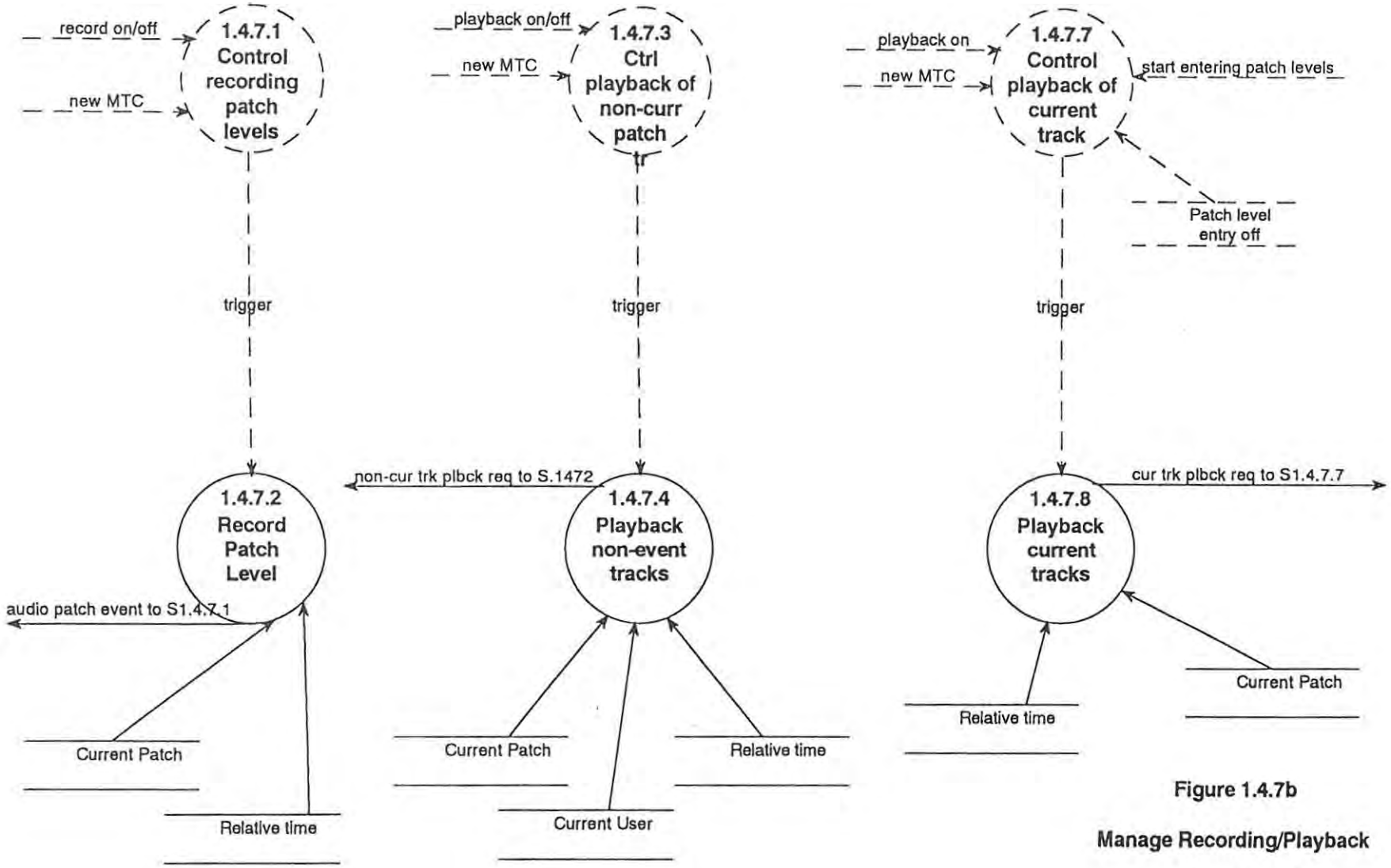

Figure 1.4.7b

Manage Recording/Playback

for WORKSTATIONS

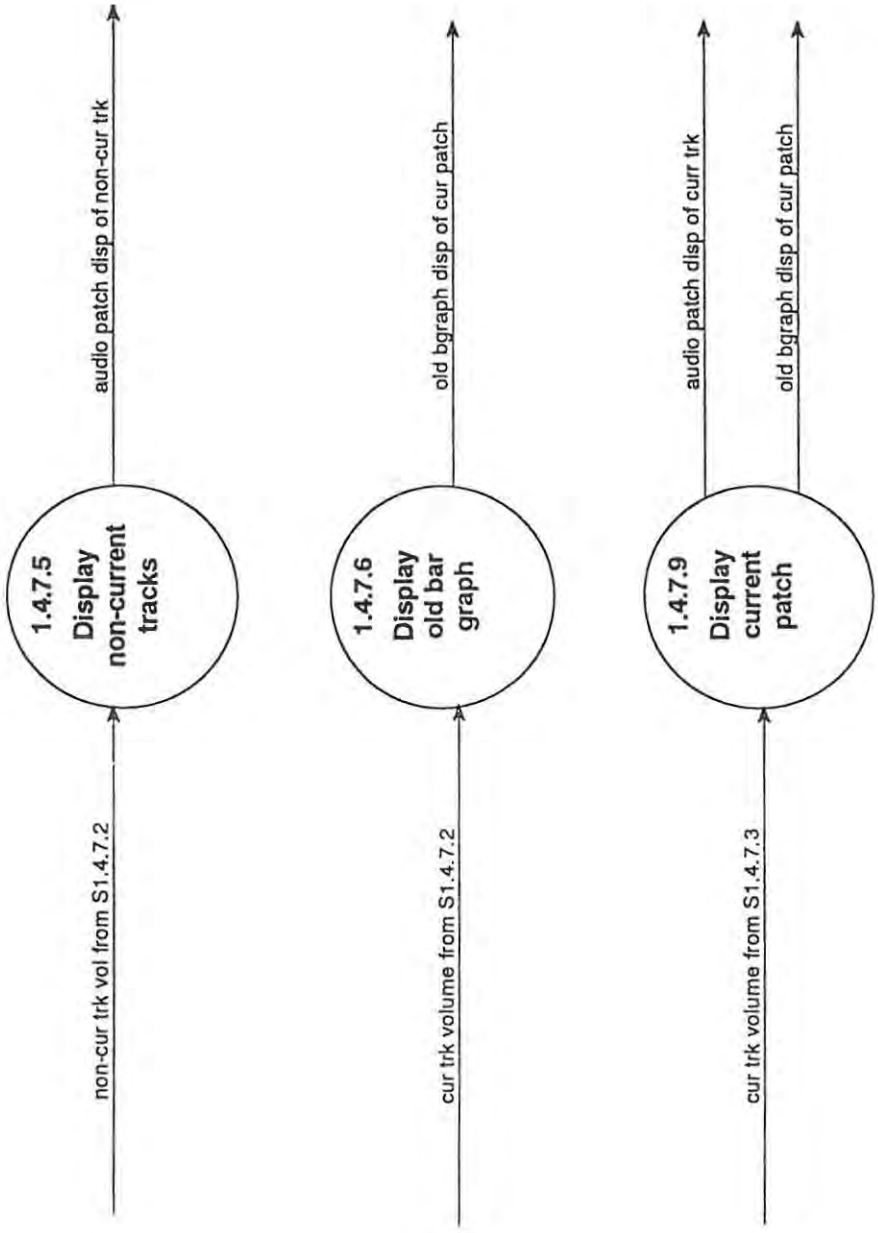

Figure 1.4.7c

Manage Recording/Playback

for WORKSTATIONS