DIVISION OF COMPUTER SCIENCE

User Interface Design and Computer Music Systems

Richard Polfreman,
John Sapsford-Francis
John Lewis
Howard Burrell

Technical Report No 215

January 1995

User Interface Design and Computer Music Systems

Richard Polfreman, John Sapsford-Francis, Dr. John Lewis, Prof. Howard Burrell

Key Words: **User Interface Design, Computer Music, Task Analysis, Software Synthesis.**

Abstract: **This report describes preliminary work carried out as part of a PhD research project into the design of user interfaces for computer music systems (incorporating software synthesis systems). It describes some of the features of computer music systems, problems with current user interfaces and a possible approach to developing solutions.**

Introduction:

This PhD research project has been largely motivated by personal experience of computer music systems and user interface programming. The project seeks to help reduce the divisions that currently exist between composers according to the technology that they use and to encourage a wider use of Computer Music Systems (to a level comparable with other uses of computers in music). These aims are to be achieved by the application of various techniques from the area of Human Computer Interaction, producing criteria for both evaluating existing systems and designing new prototype interfaces. Such prototypes need high levels of usability and learnability for composers who are not necessarily expert in computer technology or sound synthesis. Major research areas the project will involve include compositional strategies, music representations and notations, and user interface design methodologies and techniques.

Computer Music Systems (CMSs) have been in existence for nearly 40 years but have been widely ignored by professional composers/musicians outside the 'avant garde' and outside academic/research establishments. There are perhaps two main problems associated with computer music systems in the past. One has been the processing power required to synthesise musical sound. This has meant extended processing times - preventing CMSs from competing with the immediacy of electronic synthesis - or prohibitively expensive computers for rapid processing. However, with the advent of RISC based PC's, the possibility of real time synthesis on affordable computers is not far away (in fact a new "native" version of the Csound program on Power Macintosh is capable of real time synthesis).

The second problem has been the user interface. CMSs originated on mainframes and in academic/research establishments. When porting theseto PC's, little use has been made of the WIMP (Windows, Icons, Menus, Pointers) environment. Even new CMSs designed to work in the WIMP environment have generally retained either a partial command line language or close links to a particular programming language (which has helped restrict the use of such systems to composer-programmers). The setting of control parameters within these systems is not obvious and so use has also been limited to those with detailed knowledge of synthesis methods and models.

The user interface is the problem area under investigation by this research project in its attempts to provide criteria for, and prototypes of, highly learnable and usable systems and to assess the application of existing user interface design methods specifically to computer music systems.

Contents:

Figures:

1. Computer Music Systems:

1.1 Definition:

For this project a computer music system is defined as follows:

"A Computer Music System is a computer hardware/software package for the synthesis and control of musical sound."

The important feature of this definition is the *synthesis* element. There are various music applications of computer technology that are widely used by a great number of musicians/composers including sequencing - the control of external hardware synthesisers/ samplers/ effects by computer software (usually via the MIDI protocol) - and digital audio recording and editing. This definition is comparable with the common term in computer music of Software Synthesis Systems except we are explicitly including the hardware in a complete Computer Music System - including any additional hardware such as processor boards that are often necessary to improve synthesis times. Software synthesis has been left behind in terms of wide usage in comparison with other music applications of computers (despite its longer history) and has been slow to catch up. One of the limitations has been that computer synthesis is very processor intensive and so on small computers prohibitively slow (minutes to hours to synthesise 1 second of sound). However PCs are getting ever faster and now can synthesise at a reasonable speed (especially when utilising additional hardware components such as 'sound cards') and at real time on new RISC based machines.

Another limitation of CMSs are the user interfaces that they generally possess, which are often command language based or have close links with computer languages (that must therefore be learnt in order to make use of the system) or are just far too complex for non computer/synthesis specialists to use effectively. One of the reasons for this complexity may be that most computer synthesis programs have originated on large computers in academic/research establishments with little commercial pressure to provide user friendly packages. The commercial world has been slow to take up computer synthesis - due to the processing power required and the complexity of the synthesis models making it difficult to produce user friendly CMSs that synthesise sound speedily.

The advantage of using CMSs is that they are more flexible than hardware systems, and in fact can potentially emulate any electronic synthesiser or acoustic instrument ever made while extending beyond their limitations. CMSs can also be readily updated or otherwise modified (e.g. customisable systems) and easily linked into multi-media set-ups.

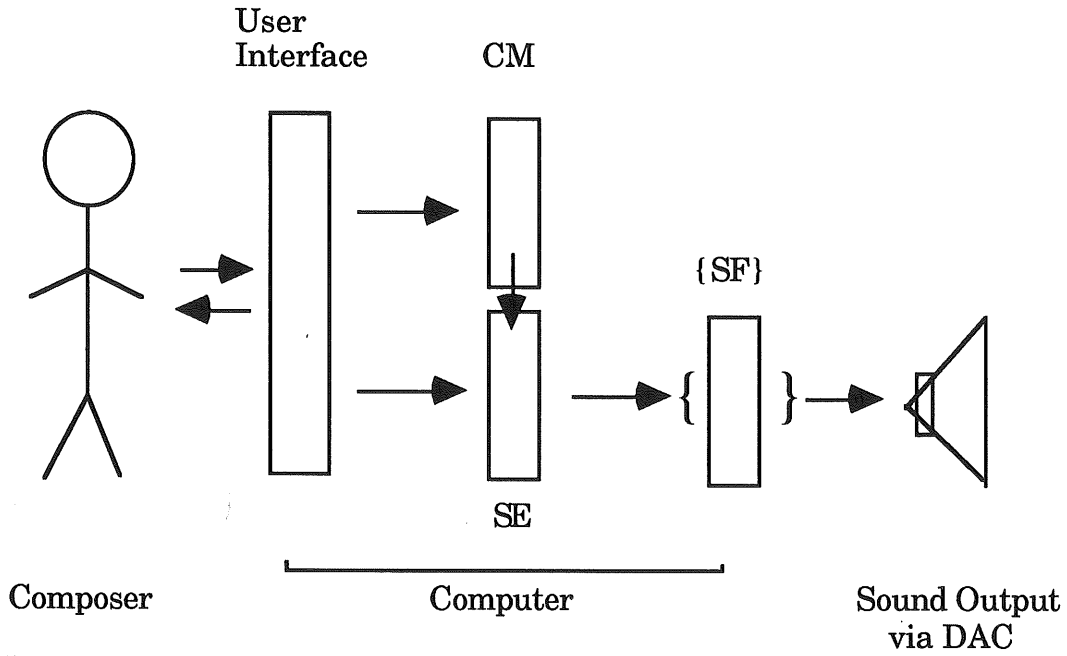
1.2 Structure:

The software structure of a computer music system can conceptually be broken down into three main elements - the User Interface, a Control Mechanism and a Synthesis Engine (see Fig.1.2.1). The composer interacts with the user interface in order to configure the synthesis engine (which contains the mathematical algorithms used to generate or synthesise the sound data) and schedule the control mechanism (which applies time varying performance parameters to the synthesis engine in order to produce musical sound). This result can be stored in a soundfile which can be output through a Digital-to-Analogue Converter (DAC) at a later time (or in real time sent directly to the DAC) and on to a sound output system.

If we compare a computer music system with a more common music situation (a music score, musician and musical instrument) the Synthesis Engine may correspond to the musical instrument and the Control Mechanism to the music score. The 'musician' is then shared between the two in a flexible manner - sometimes a complex synthesis process has control data built into the SE part with a simple control parameter schedule, whilst at other times the SE set up is simple with complex control data. Often the same end result can be achieved either way. This flexible division between the SE and the CM permits many different approaches to creative composition with computer music systems.

In practice the composer may use several software packages to produce a single composition or composition segment. For example, a composer may use a MIDI sequencer (or MIDI tool such as Patchwork or Common Music) to generate MIDI information which is stored in a file. This file is then read by Csound during a synthesis to create a soundfile. The soundfile can then be edited using a sound editing tool such as Sound Designer II (Digidesign) and then perhaps processed by a Csound reverb program or similar.

Fig. 1.2.1. Computer Music System.



CM = Control Mechanism
SE = Synthesis Engine
SF = Soundfile
DAC = Digital to Analogue Converter

Note: the synthesis engine can often use soundfiles, or input from an ADC in real time, as audio sources for processing - not shown here (for simplicity).

1.3 Synthesis Models:

In order to synthesise 'CD quality' sound on a computer we must specify 44100 16-Bit numbers for each second of sound produced. In complex timbres it is not clear what numbers we need to specify in order to achieve the sound we wish to generate. To get round this data production problem we use a synthesis model to allow us to produce the sounds we require and with less parameter setting. The variety of synthesis models used in computer music systems can be placed into two main classes, Digital Signal Processing (DSP) Models and Physical Models.

The DSP model uses mathematical algorithms to simulate electronic devices such as oscillators, filters, envelopes, spectral analysers, reverberators, delay lines etc. These can be 'connected' together in order to 'build' a synthesis process - a virtual electronic synthesiser. Soundfiles can also be used as source sounds that are modified by DSP components. When triggered by the control mechanism, the synthesis

engine calculates the resulting signal which is then passed to a soundfile or DAC. Various synthesis techniques can be simulated including Additive Synthesis, Subtractive Synthesis, Amplitude Modulation, Frequency Modulation, Wave Shaping, Linear Predictive Coding, Phase Vocoding, Granular Synthesis etc. Most hardware synthesisers utilise a subset of DSP techniques in a predefined configuration and it is the most widely used synthesis model in CMSs. A problem with this model is the user has to be familiar with DSP techniques to create a meaningful synthesis set up and even then it is not always clear what control values are needed to give the required output.

Physical Models use mathematical techniques to simulate the vibrations produced when exciting physical objects (instruments) and hence the sound. Some models are developed specific to a particular musical instrument, while others can simulate component objects and the result when connected together. For example modal modelling (simulating each mode of vibration with a simple harmonic oscillator, the relative amplitudes of the different modes controlled by the level of excitation and connections) to simulate vibrating tubes, bells, strings, membranes etc. These can be assembled together to produce an instrument using a variety of 'connections', and excited in different ways - 'striking', 'plucking', 'blowing reed' 'bowing' etc. Thus a virtual acoustic instrument is constructed and 'played'. A possible advantage of physical models is that most musicians/composers are familiar with the timbres produced by physical objects and so conceptually it is easier to create synthesis set ups than perhaps with DSP models (depending on implementation). A problem with physical models is that they are potentially as difficult to play as 'real' instruments and so setting of control parameters is again difficult.

A related consideration of physical models is that of the stability of solutions. There is often more than one solution to the modelling equations for a particular parameter set for some types of physical model. This can be considered as a positive effect in terms of capturing the richness of a real instrument but on the other hand can make control of the model for musical purposes difficult. In the "real" situation it is often the skill of the musician that enables stable or periodic timbres to be produced (and controlled use of instability). This skill implies a complex knowledge base and inference mechanism, incorporating aural and tactile feedback from the instrument, that is built up through years of practice. Use of feedback loops and filters have been proposed as methods for limiting the output of physical models to single stable solutions - both periodic and persistent (Rodet,1994).

Systems can be a hybrid of the two model classes allowing physical models to be modified by reverberators/filters etc. or less obviously using DSP generated sound to excite physical objects.

For both Physical and DSP models, the setting of control values is problematic because their controls are not music parameters.

In digital signal processing models we are required to set signal parameters such as frequencies (Hz), amplitudes (dB), filter cutoff frequencies(Hz),attack times(s), multiplication factors etc. which do not always correlate well with music parameters such as pitch (note values), loudness (ppp-fff), brightness etc. Most music parameters tend to be more descriptive (qualitative) than absolute (quantitative).

In a physical model we specify values describing the physical parameters of the instrument - string tensions(N) and densities(kg/m^3), tube lengths(m) and diameters(m), membrane tensions etc. and also the physical properties of excitations such as lip pressures(N/m^2), wind speeds(m/s), strike velocities(m/s) and positions(x,y,z), bow pressures(N/m^2) and velocities(m/s) etc.

Mapping from music parameters to physical or signal parameters is not easy and requires expert knowledge. An interface system that can carry out this mapping (via a knowledge data base, neural net or other mechanism) can ease the work/knowledge load considerably for the composer/musician. However, if the system allows the creation of arbitrary instruments, automated mapping of parameters is difficult. In an idealised system a program could 'learn' how to play any instrument created (i.e. how to map music parameters to model parameters - requiring quantification of music parameters), but this may not be a practical reality. An alternative is to encourage the re-use of predefined mappings for predefined instruments by non-expert users, while providing experts with facilities to create their own mappings, or to set synthesis parameters directly, that will give these users access to the details they require.

1.4 Control Protocols:

MIDI (Musical Instrument Digital Interface) originated in the early nineteen eighties to allow electronic musical devices to communicate with one another, across manufacturers, in a standardised format. The development of personal computers lead to MIDI software controllers to manipulate these electronic devices. The popularity of MIDI has led to its incorporation into CMSs to control synthesis processes. Problems with MIDI include its limited speed (and bandwidth) and the lack of information about synthesis that it carries - MIDI effectively only controls performance keys/buttons/sliders remotely. MIDI controls (noteon, noteoff, volume, pitchbend, controller etc.) can be mapped onto control parameters of a synthesis engine (e.g. carrier to modulator ratio in FM synthesis) via the Control Mechanism, but are really not adequate for this purpose (bandwidth is poor and it is difficult to allocate controls to a single event since many are channel specific). MIDI was not designed to control synthesis in this way.
ZIPI (Zeta Instrumental Processor Interface) is a potential replacement for MIDI (developed in collaboration with CNMAT, Berkeley) incorporating far greater control, at greater speed - including sending real time digital audio signals. Each note has slots for pitch, brightness, loudness, (3D) timbral space location, (3D) physical space location, noise/pitched ratio, even/odd harmonic balance etc. compared with a MIDI note that has pitch and velocity (and channel) only.

Currently using MIDI can help make a system widely usable but places severe limitations on the control available (and is only really suitable for keyboard control of sound) - ZIPI is well suited for use in CMSs and its Music Parameter Description Language is only one layer - there will be other layers for Virtual Reality, data dumping, MIDI (preserving some backwards compatibility), mixer automation etc. ZIPI is not yet fully implemented but is a promising future control system.

Looking at ZIPI note parameters we see that they are close to perceptual parameters (since they are to be applied to arbitrary synthesis set-ups) and so once again the problem of mapping these to synthesis parameters is apparent. The designers of the system imply that manufacturers should provide default mappings while allowing access to user definable maps. This is perhaps what a computer music system should provide - libraries of suitable maps with user definable mappings available to those who require them.

1.5. Current Systems:

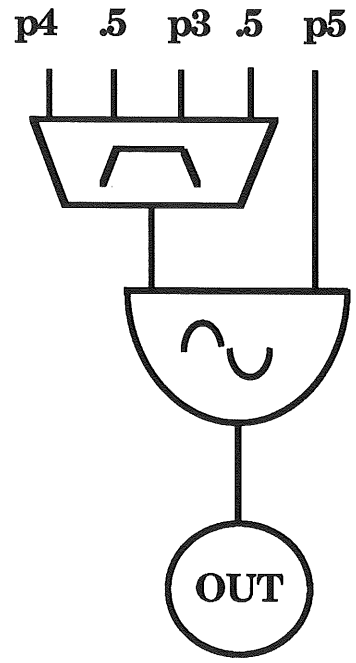
The current state of computer music systems can be summarised in a brief description of some relevant existing computer music software for Macintosh computer systems, which are becoming perhaps the standard PC for computer synthesis of sound (several of these programs are available on other platforms also). These descriptions highlight some the key problems in CMS user interfaces and some of the solutions attempted - with various degrees of success.

Csound (etc.)

Csound belongs to a family of software synthesis systems developed from the MusicN series of programs initiated by Max Matthews at the Bell Labs in America. They are all closely related, versions for the Macintosh including Barry Vercoe's latest MIT Csound and an experimental version of MusicV with graphical orchestra interface by Simone Bettini. Csound et al. are the classic DSP model implementation for CMSs. The user must provide the program with two text files - an *Orchestra* and a *Score* . This corresponds to the SE/CM division in our CMS structure. The user must learn the Csound orchestra language to set up a synthesis process and then also learn the score language - essentially a list of events with parameters. The Csound program then compiles these files (reporting errors that the user must debug) and produces a sound file or real time performance. On a standard Macintosh real time synthesis is not possible, although a RISC based Macintosh (Power Macintosh) is able to do some with a version in native code (i.e. optimised for the new processor). Csound incorporates a wide range DSP elements useful for music purposes.
The Csound interface itself contains various file selectors that allow the user to specify the input text files, folder or directory selectors to tell Csound where to look for input soundfiles and where to write output soundfiles and a command line input for Csound instructions such as running a spectral analysis of a soundfile. User feedback on compiler progress is in the form of reported syntax errors in the source files, graphic displays of generated waveforms and score events listed as each one is processed.

Fig. 1.5.1 Csound Example -

(a) Graphic Orc Representation



(b) Csound Orc code

```
sr = 44100          ; sample rate
kr = 4410           ; control rate
ksmps = 10          ; sr/kr
nchnls = 1          ; audio channels


instr 1

;p4 = amplitude
;p5 = fundamental
;k1 = envelope
;a1 = oscillator
   k1   linen  p4, 0.5, p3, 0.5
   a1   oscil  k1, p5, 1
        out    a1

endin
```

(c) Csound Score Code

```
;function table
;f1  0  8192  10  1 1 1 1 1 1 1 1 1 1

;                    p3    p4     p5
;instr   start       dur   amp    fund

i1          0        1    8000   440
i1          1        .    10000  220
i1          2        .    15000  110
i1          3        .    8000   440
i1          .        .    .      220
i1          .        .    .      110

e
```

(d) Summary

The graphic representation shows an envelope generator controlling the amplitude of an oscillator which is output as an audio signal. p3 ,p4 and p5 are parameters to be read from the score.

In the Orc code sample rate etc. are first initialised. Linen creates the envelope, amplitude p4, duration p3, attack and decay both 0.5 seconds and its output k1. Oscil creates the oscillator, frequency p5, amplitude k1, signal from f1 and output a1. Out sends a1 to a soundfile on compiling. Instr1 and endin enclose the instrument.

In the Score code f1 is a function table created by gen10 at time 0, size 8192. gen10 creates a fundamental and series of harmonics - here 10 harmonics all the same amplitude as the fundamental. This is the waveform to be output by the oscillator. There then follows an event list of notes to be performed with various start times, durations, amplitudes and frequencies. e ends the score.

The Csound language is not difficult to learn for programmers, but for non-programmers the mere fact that it uses a text based programming language is problematic (users are forced into a programming 'way of thinking' which may not correlate with a musical one). Another problem is that it is a low level language with few high level music structures and no methods for abstraction e.g. there is no 'chord' structure, only 'events' (notes) in the score. A variety of people are working/ have worked on Graphical User Interfaces (GUIs) for Csound but none seem to be fully implemented on PCs as yet. There are score translators for Csound for music scores to be used for control and MIDI implementation has been incorporated allowing a standard MIDIfile to be used to control elements of synthesis. However these are only partial solutions and may not be sufficient for specifying detailed Csound synthesis. The source code is available for Csound allowing modifications (new DSP implementations) and perhaps a new user interface to be constructed. A simple Csound example is shown in fig. 1.5.1. above.

Simone Bettini's MusicV implementation has a graphical orchestra compiler based on the block diagram representation used in literature etc. It allows the user to build an orchestra by direct manipulation of DSP components, graphically connecting them together, and compiling an orchestra file from it. This orchestra is then fed to the MusicV compiler together with a score (text based) and used to synthesise a soundfile. Again this is only a partial solution - the score language must still be learnt. Also detailed knowledge of signal processing is required in order to create a meaningful synthesis. The interface does not allow illegal connections and warns if connections are missing - thus assisting some of the debugging problems that can occur - but is not specific about where in the synthesis the fault lies.

New Csound GUIs are likely to be developed for some time to come and so there may be interesting developments in this area - especially with the new real time capabilities.

Patchwork™ (Ircam)

Patchwork is essentially tool for creating Lisp programs graphically on the Macintosh, which is provided with libraries of functions for music applications. The composer can graphically construct patches that produce musical output, either control output (MIDI, Csound Score) or sound output (Chant fof synthesis) - usually via algorithmic compositional techniques. Problems with the interface include the fact that graphical objects are not always obvious in what they do, on line documentation is incomplete, only Csound scores are produced not orchestras, the composer is forced into an algorithmic style of composition (i.e. it is not easy to write music scores directly - but that is not what the tool is intended for).

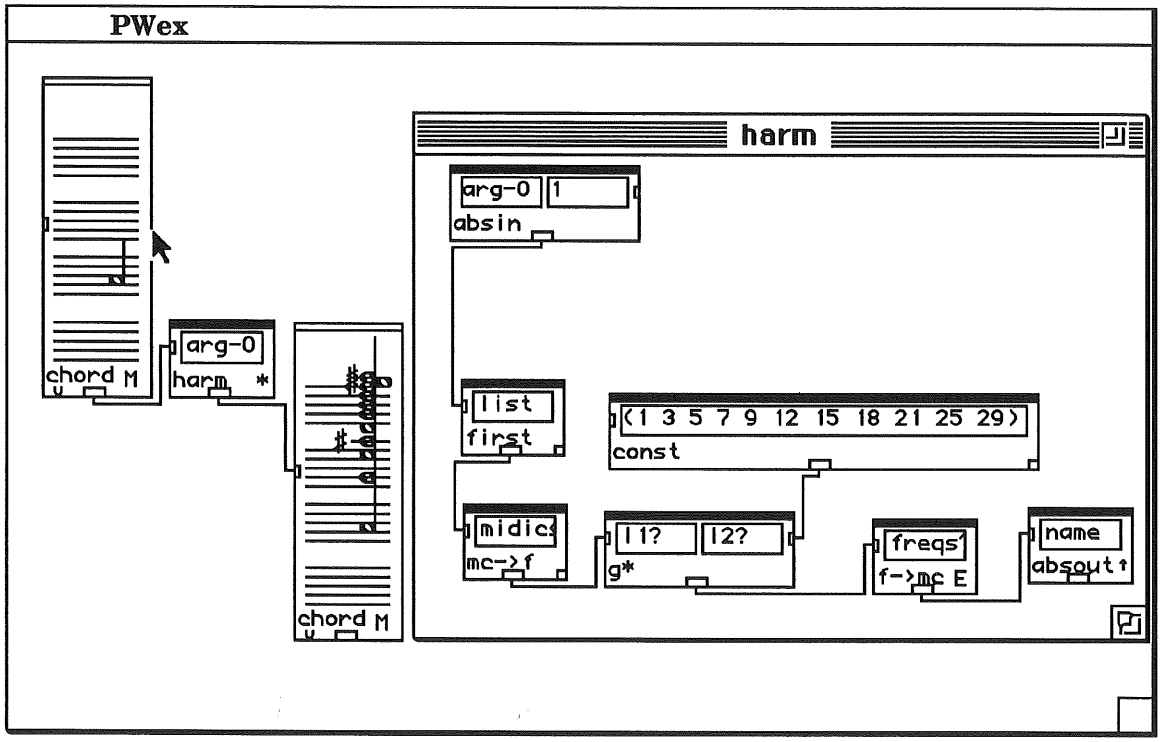
A major drawback is that Patchwork is Lisp embedded and so the user needs to learn Lisp in order to make full use of the system - especially debugging since errors are Lisp errors. In fact Patchwork can be used purely to construct Lisp programs and in this sense contains much excess functionality for musicians who are not computer programmers.

A key tool in Patchwork is that of abstraction - whereby the user can create a new patchwork object that contains within it a group of connected patchwork objects. Thus complexity can be hidden and reuse of others work can be straightforward. This idea is very powerful and can lead to multi-level systems providing beginners with simple objects and experts with the ability to access details within them (see fig. 1.5.2. below).

Common Music / Common Lisp Music / Common Music Notation

These are Common Lisp programs dedicated to manipulation of music data designed at CCRMA by Heinrich Taube. Common Music is again a program for algorithmic composition producing control output in the form of MIDI, CLM (Common Lisp Music) and Music Kit data. Common Lisp Music allows sound synthesis (DSP model) to be carried out and Common Music Notation assists score production. All three use the Lisp environment as their interface although the development of GUI's has been considered. Being Common Lisp programs means that they can be used across a variety of platforms (Macintosh, NeXT, Unix), but also require the user to be Lisp knowledgeable.

Fig. 1.5.2. Patchwork Example
Window 'PWex ' shows a single note input to the 'harm' object which outputs values producing a chord using the input note as the fundamental with various added harmonics (rounded to the nearest semitone). The window 'harm' shows the contents of the abstracted object harm - this object can be used without knowing its detailed contents i.e. complexity can be hidden from the user.



Mosaic / Modalys (Ircam)

Mosaic is a modal modelling system (physical model) currently under development at IRCAM and uses a Scheme (Lisp dialect) interpreter as its interface. The user can construct virtual instruments and set up performances but the processing is slow and performance control is difficult - requiring a knowledge base of performance data to make effective use of it. Instrument construction is at a low level requiring the user to specify in detail the physical properties of the components (e.g. string tensions, radii and densities). Mosaic is MIDI compatible and has interesting facilities such as that for 'morphing' instruments i.e. the physical properties of the instrument (e.g. dimensions) can change during performance. This system awaits a GUI or other improved interface.
A new Power Macintosh version is available with enhanced processing speed.

Kyma™ (Symbolic Sound Corp.)

Kyma is a graphical object oriented system for music composition that runs on Macintosh computers using additional hardware in the form of the Capybara processor box (containing up to 9 Motorola DSP56001 DSP chips) to handle synthesis/signal processing operations. Kyma is based on the Smalltalk-80 object oriented environment and incorporates many compositional styles into sound objects which have iconic representations for direct manipulation by the composer. A sound object produces a stream of sound data when evaluated (played) - irrespective of the synthesis method used to create the data, allowing for example FM synthesis, additive synthesis, processed sound samples etc. to represented at the same level (i.e. can be manipulated in similar ways) with icon designs indicating the source for each. Even scores are sound objects - Kyma is essentially a one level system with a hierarchical structure within each sound object. Pre-defined sound objects are provided which the user can use in their own sonic designs. Control of these sound objects can be very complex, both in terms of what is achieved and how it is achieved.

The system is very powerful and also very complex. The additional hardware allows a significant amount of real time synthesis to be carried out giving the composer immediate feedback on the work s/he is doing but the complexity of the system is still a large problem for the inexpert user to overcome. Abstraction of complex set-ups is available which can help hide some of this complexity, but detailed knowledge of DSP techniques is a necessity for effective use of the system. Learning of the Smalltalk is necessary for the more detailed work including music scoring.

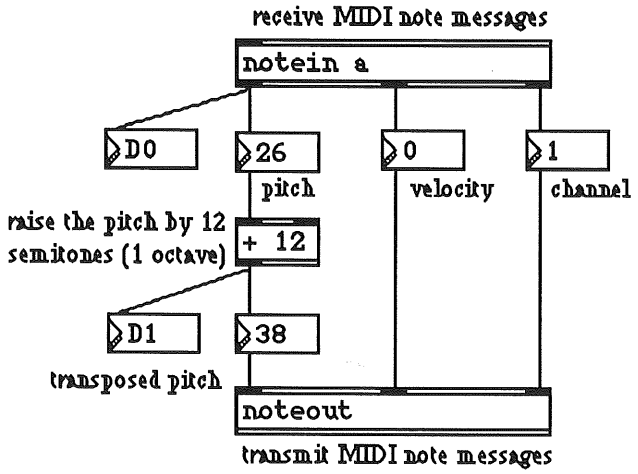
Max™ (OPCODE/Ircam)

Max is a graphical programming language for handling data streams - in this case MIDI although with additional hardware and software objects it can be used for controlling synthesis processes, QuickTime movies, CD players, video disc players etc.. It is used most often for interactive performances where complex streams of MIDI data are triggered by various controllers.

Max has been criticised for not being musical (other than handling MIDI data), little symbolic value to most graphical objects, and "spaghetti like" patches resulting from sophisticated set-ups. Max can incorporate C language modules for data manipulation, and also supports abstraction similar to that of patchwork.

There are some experimental programs that allow synthesis objects to run with Max on the Macintosh - implementing a DSP model (Unison).
A simple Max example is shown in fig. 1.5.3. below.

Fig. 1.5.3. Max Example



The notein object receives MIDI note data in real time from a MIDI interface connected to the computer. It splits the notes into their components - pitch, velocity and channel which are then displayed. 12 is added to the pitch or note value to transpose incoming notes by one octave. The transposed pitch is displayed and used in reconstructing an output note with the noteout object. This is output to the MIDI interface also in real time.

TurboSynth™ (Digidesign)

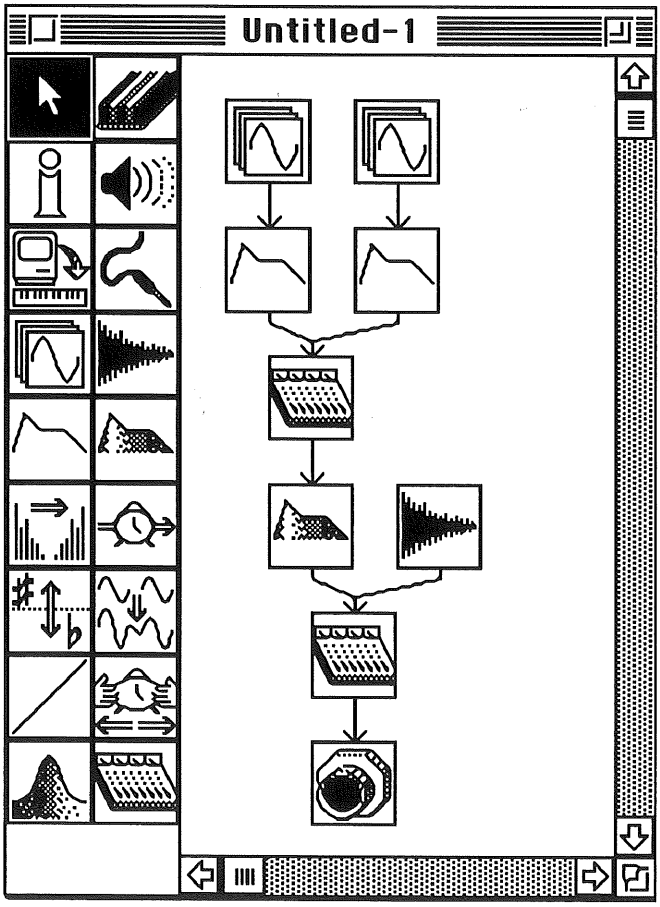
TurboSynth is a purely commercial package (the others are mainly research/academic in origin) based on the DSP model that allows the user to connect graphically a variety of DSP components to set up a synthesis. This then compiles to a sound file on a SampleCell II card in the Mac and can be played via MIDI. This is similar to a graphical Csound orchestra compiled with a single score event (one note) and the sample note played by MIDI. This is less powerful than Csound since in Csound we can vary the orchestras input parameters continuously - with Turbosynth we would have to repeatedly change the

'orc' , synthesise and then join the soundfiles together. TurboSynth also has fewer DSP components and less detailed control over them. The compromise in power does mean that sound generation by TurboSynth is very fast, and its user interface is far more usable/learnable than Csound. A good point to note about Turbosynth is the use of meaningful symbols for DSP components and tools. An example patch is shown in fig. 1.5.4. below.

SeaWave

SeaWave is a computer music system that uses additive synthesis techniques in a manner that attempts to solve the problems of complex parameter setting. The user interface contains two levels for editing timbres. One is a high level mode where the user can define the sound using descriptive (perceptual) parameters, the other low level mode allows more precise setting of partial amplitudes. The high level mode allows the user to create a new timbre without learning the details of additive synthesis techniques while the low level mode allows those with a knowledge of additive synthesis to apply that knowledge in sound creation. This permits both 'experts' and 'novices' to make use of the same system and should therefore be of value to a wide range of users. The high level description parameters include terms such as bowed, plucked, hammered, struck, strummed etc. for the attack part of the sound and terms such as grating, harsh, hollow, windy, bright, bouncy, dark etc. in different classes of 'presence' to describe the more general qualities of the sound. Currently this system either generate standard soundfile output which can be played via various software packages or dumped to a sampler - similar to TurboSynth in that it produces one sound event from a particular synthesis - or it can produce a Csound function table that can be used in a Csound score for producing a complete piece (or part of a piece).

Fig. 1.5.4. Turbosynth Example.



The turbosynth window shows two waveforms enveloped and mixed together before being passed though a filter. This signal is then mixed with a soundfile to produce a new sound sample. The left of the window shows a palette of DSP devices and tools that can be used in turbosynth. This synthesis set up can be created in a few minutes by a non expert - a marked gain in ease of use over Csound and its relatives, and a reduction in power of the system also. Opening the DSP objects allows their parameters to be set - e.g. the waveform to be generated by an oscillator object.

SeaWave is interesting in that it is one of few systems that provide a (pre-defined) mapping of perceptual or music parameters onto synthesis parameters and it also provides a two level system for sound synthesis. Limitations are that it does not include a scoring system for producing complete instrument parts (like TurboSynth) and only employs one synthesis technique (additive synthesis) which is not necessarily the best technique for a generating a particular desired timbre. However, it represents a considerable advance in usability over other implementations of software synthesis for both those with limited knowledge of additive synthesis and those who are more expert, and is an exciting development.

Summary

We have described in brief some of the key CMSs available for PCs, some of which we have here (Csound, Max, Patchwork, Common Music) others we hope to obtain or at least gain access to in order to explore them more thoroughly. We have not described here the plethora of sequencing, notating and audio editing systems available which do not directly support synthesis.

Csound is one of the more complete CMSs, using a DSP model without requiring any additional hardware. Others such as Max, Patchwork are Common Music are really control mechanisms that can be used to control synthesis either through additional software or by driving external synthesisers (using MIDI), but do not fall into the category of simple sequencers due to their more flexible/powerful nature.

Kyma is a very powerful system that can carry out real time synthesis but it requires additional (expensive) hardware and has a complex user interface - even though it uses mainly graphics driven control. Kyma is however far more usable than Csound for non-programmers - the graphics are easier to deal with than the text language of Csound and much faster processing gives more immediate feedback on users work (other than the Power Macintosh Csound).

Abstraction as available in Patchwork, Max and Kyma is a powerful tool for hiding low level complexity and encouraging re-use of others work both of which can help the non-expert user.

TurboSynth is essentially a highly simplified equivalent of Kyma (or Csound with a graphic interface) that provides facilities for timbre creation but not music scoring. Again knowledge of DSP techniques is required to create and manipulate sound but the level of knowledge required is less than for the other systems.

Mosaic is one of the few physical modelling systems currently available and is more of a research tool in its present state. Its advantage over other physical modelling systems is its flexibility i.e. it is not limited to modelling one type of acoustic instrument and allows the user to construct arbitrary acoustic situations from the available components (and indeed new components can be created). However this leads to extended processing times on PCs. This will be improved with the native RISC processor version. Although providing a conceptual framework within which most composers are perhaps more familiar than with DSP models, the current instrument creation and control systems are at such low level as to make the system equally difficult to use.

SeaWave demonstrates the feasibility of using maps from descriptive (music) control parameters to synthesis parameters in order to create desired sounds. The system itself is a research project still under development and does not provide a scoring system for creating complete pieces of music. However it allows users with little or no knowledge of DSP techniques to use them in sound synthesis while allowing users with that knowledge access to DSP parameters.

If we compare DSP model implementations such as Csound to electronic synthesisers over a number of years, Csound may correspond to an era when electronic synthesisers consisted of individual components patched together with a mass of cables requiring the user to be more of an electronic engineer than composer. At the time of the early MusicN languages electronic sound synthesis was done in this way, but while electronic synthesisers have advanced considerably since then (in terms of ease of use), software synthesis systems have lagged behind.

2. HCI - TA

2.1 Definition:

The problem domain of the discipline of Human Computer Interaction has been stated as: "the design of humans and computers interacting to perform work effectively" (Long & Dowell, 1989), and includes hardware design, training methods etc. as well as software design (including the user interface). User interface design is the key area for this project, taking methods from HCI that have been applied to ensure highly usable/learnable interfaces that allow the users to perform their tasks as easily, effectively and efficiently as possible. The Macintosh computer has a very highly regarded user interface to its operating system - key points being direct manipulation of graphical objects, use of metaphors and consistent behaviours within and between applications. Together with its availability (and price range) and support for other music applications by a variety of companies, the interface makes the Macintosh one of the most suitable computers for usable computer music systems (although IBM PCs running WIMP environments are perhaps catching up, and the choice between systems is becoming more to do with personal preference than strict usability).

2.2 Users:

In order to provide an effective user interface, the designer must take into account the intended users, their environments, their usage patterns and tasks that they wish to carry out. In computer music most systems have been designed by academics/ researchers for academics/researchers. The intended users for computer music systems for this project are composers generally and as preliminary breakdown they are placed into three classes according to technology they employ in the composition process: *Computer Composer*, *Electronic Composer* and *Acoustic Composer*. Thus Computer Composer generates sound during composition on a computer, Electronic Composer on synthesisers/samplers and Acoustic Composer on acoustic (or electro-acoustic) instruments or does not use sound at all during composition. Many composers may (correctly) regard themselves as members of more than one group.

The classification does not restrict technology for *performance* - a composer may use a computer and synthesisers when composing a piece for philharmonic orchestra and would be regarded here as an *electronic* composer. A composer who uses a computer simply as a typewriter for CMN scores (with no sound feedback) is here an Acoustic composer even if the instruments that are used in performance are not necessarily acoustic.

There are other classification criteria that may be useful in defining users, such as musical styles, since these may affect the composition tasks involved. However, stylistic differences are more subtle and need to be addressed at a later stage of the project (during the task analysis process). Currently the computer composer classification is closely linked to limited range of compositional styles, but this is changing. This project hopes to contribute to a widening of the computer composer category to encompass currently acoustic and electronic composers and also greater range of musical styles.

A projected user analysis, showing some of the relevant knowledge areas that the three composer types may be expected to possess, is shown in Fig. 2.2.1. below. This is purely speculative at the moment and will be verified according to information obtained from a range of composers via questionnaires, interviews etc.

For the moment it is assumed that all *intended* users will have a working knowledge of the Macintosh interface and style, since this is necessary to operate the computer itself. This also assumes that the Mac interface is a good interface for non-computer programmers and for computer music. This assumption will be maintained unless evidence to the contrary is found (the Macintosh interface has been widely praised as a simple to use, user friendly system - key points being use of metaphors, consistency of behaviour within and between applications and direct manipulation of interface objects).

Fig. 2.2.1. Composers and Knowledge Domains.

Domain \ Composer	Computer	Electronic	Acoustic
Music Theory	•	•	•
Prog. Lang.	•		
Synth. Methods	•	•	
Basic Maths	•		
Seq. Software/MIDI	•	•	
DSP	•		
Com. Lang.	•		
Musical Knowledge	•	•	•
Basic Computer Operation (Assumed)	•	•	•

A more detailed description of the knowledge domains in the projected user analysis follows:

• Music Theory is the area of traditional music theory - scales, keys, chords etc. and will vary according to which music traditions are being followed - and also more modern concepts such as atonal algorithmic composition techniques. Composers are likely to have at least a working knowledge of music theory relevant to their compositional style(s) independant of the composer types identified (as far as these are independant of musical style).

• Programming Languages includes languages such as Basic, C++, Lisp, Pascal etc. and the programming and design skills required to use them effectively. Generally computer composers have experience of computer programming in one or more languages, and often have written their own compositional software tools. Computer programming skills would not be expected of electronic or acoustic composers generally.

• Synthesis Methods is knowledge of parameter setting appropriate to synthesis set ups in order to create the desired sonic effect, e.g. filter cutoff values, envelope shapes, wave shapes, FM frequency values etc. Such knowledge is often required of computer composers whose synthesis tools usually require the user to set almost all the synthesis parameters involved. Electronic composers will generally have some knowledge in this area since at least some parameters of this type are editable on most electronic synthesisers. However, over the past 15 years or so there has been a move away from programming synthesisers to using pre-programmed set ups, or making small changes to these preset sounds. This has allowed composers with little knowledge of synthesis methods to make use of electronic synthesisers.

• Basic Maths is relevant knowledge required for a mathematical understanding of synthesis methods, DSP, algorithmic techniques etc. Generally we would expect a computer composer to have some mathematical foundation, more so than the other composer categories.

• Sequencing Software/MIDI is the knowledge at the heart of electronic composer's world and is practical knowledge of MIDI set ups and use of MIDI sequencing packages for control of sound modules, effects units, mixers etc. Computer composer has usually made some use of software sequencers and MIDI

since most CMSs support some MIDI control of parameters. Acoustic composer may never have utilised these tools.

• Digital Signal Processing is a deeper knowledge than that implied by synthesis methods including an understanding of the underlying models and how a synthesis process can be constructed. This knowledge is necessary for computer composers since the majority of CMSs are programmable at this level. Electronic composer need not have this level of understanding due to modern digital synthesisers providing preset configurations and settings. (In the early days of electronic synthesis users had to manually patch together DSP components and so would have needed knowledge in this area). Acoustic composer would not be expected to have knowledge in this domain.

• Command languages include computer music languages such as Csound, MusicN, Cmusic, Common Music etc. and how to use them effectively. Again this is required of computer composer but not of the others.

• Musical knowledge is knowledge gained from the experience of writing, listening to and performing music. As such musical knowledge will vary greatly according to music culture, music style and experience/ education. All composers will possess this kind of knowledge.

It should be said here that the predicted user analysis is really a snapshot of a dynamic situation - technology is changing all the time and so is the knowledge required of the user of the technology. Also music education of composers is changing with increased use of technology in music education providing greater exposure to electronic and computer technologies.

The predicted user analysis suggests that computer composer has knowledge in all the areas required to make expert use of current computer music systems (by definition) and so is to some extent well served by such systems. Such users should be a useful source of information regarding evaluation of current systems, further requirements of such systems, task analysis of expert use of CMSs and expert knowledge to be incorporated in Knowledge Based Systems.

Acoustic composer has the least knowledge required to operate computer music systems. We are interested in acoustic composer as a potential user who is unable to make use of current systems. Task analysis of acoustic composers during composition should provide a model of task knowledge to be supported effectively by any software aimed at this user category.

Electronic composer is the mid range category with some knowledge applicable to current systems, but not enough to make effective use of them. Task analysis of these composers should again provide a model to be supported by software aimed at this category of users.

Computer composer may require a user interface that allows access to the details of complex synthesis processes, while automating some tasks regarded as tedious.

Acoustic composer requires the most immediately usable user interface and probably needs to be shielded from the complexity of computer based synthesis tasks. A good system will encourage acoustic composer to delve further into such complexities as s/he becomes more familiar with the environment. This should allow exploratory behaviour in the use of the system. ' Technophobia' amongst such composers may be a problem, or lack of interest in / knowledge about what computer music systems can (potentially) offer.

Electronic composer should be able to apply his/her existing knowledge of synthesis to the system and like acoustic composer be encouraged (but not forced) to explore further into synthesis process.

At a high level the compositional task knowledge across the three composer types may not be that different (although all musical task knowledge is likely to be idiosyncratic). It is likely to be more dependant upon the composers music training or cultural background. Comparison of task models resulting from composers in different areas might reveal much about where the musical task knowledge is changed by the task environment, and where support for acoustic/electronic composer is lacking in current computer music systems. In this project it is hoped to involve composers from all three areas

(although some will be experts in two or even all three areas) as task performers in a Task Analysis process, in order to build composition task models across technologies and systems.

How do these ideas about users help us in user interface design? Firstly they make us aware that there are potential users with considerably different levels of relevant knowledge. If we try to design a system that serves all of these users well we must involve composers from each group in the design process (following a user-centred design strategy). We might consider interface designs with a multilevel approach whereby a top level provides Acoustic Composers with preset values for ready made instruments using music or perceptual control parameters. The next level would provide access to reprogram the preset values still utilising ready made instruments and perhaps using MIDI as the control protocol. This level would be familiar to Electronic Composers. The next level would provide access to instrument construction for Computer Composers to use their detailed knowledge in creating the required synthesis configurations and use synthesis control parameters directly. Each level would be transparent until entered and have its own range of specific tools and editors. In this way novices can start at a simple level and work downwards as they please to more complex situations. Also reuse of others work via distribution of patches or instruments can be encouraged. At all levels the system should be designed to support user tasks effectively and efficiently.

2.3 Task Analysis

Task Analysis (TA) covers a variety of techniques and models aimed at telling us something about how work is carried out. Early TA methods try to capture the perceptual/motor nature of tasks - the later methods tend to capture the cognitive nature also. Task Analysis is a process that was originally developed for improving the training or retraining of employees, but more recently has been applied to the design and evaluation of interactive computer systems. TA has an important part to play in requirements capture (ensuring that users' tasks are fully supported), design evaluation (examining demands made by the system on users), user evaluation (ensuring a variety of representative tasks are tested), documentation, training and user support (Carey, Stammers,& Astley, 1989). It can also be an initial step in Knowledge Acquisition for the construction of Knowledge Based Systems.

To our knowledge a formal generalised task analysis of music composition at a high level has not been done before (see section on Cognitive Musicology for notes on task analysis of music composition that has been carried out) in spite of its importance in the various stages of software construction. In this work we will attempt to apply TA methods to music composition to see if a useful task model can be constructed in this way and utilised in user interface design and evaluation.

Foreseeable difficulties are that of the idiosyncratic nature of music composition that may cause problems when trying to construct a generalised task model, and also the fact that composition is an ill structured problem in the sense that a well defined ordering and structuring of the whole task within a hierarchical structure is unlikely to be possible, although there may be tasks with reasonably well defined substructures and collections of tasks that have certain relations between them. The task analysis method used must allow for this kind of 'loose' arrangement of tasks and not try to restrict the ordering and relations between tasks into a fixed pattern. (We might consider a reasonably well defined task structure but with the proviso that the task performer can 'jump' from one point to another within the structure without necessarily completing the current task before the jump). This kind of flexibility is required for modelling creative design tasks in general, not just music composition.

Various TA methods exist including **Hierarchical Task Analysis** (Annett et al. 1971), **Command Language Grammar** (Moran, 1981), **Task Action Grammar** (Payne & Green, 1986), **Goals, Operators, Methods and Selection rules** (Card et al. 1983), **Cognitive Complexity Theory** (Kieras & Polson, 1985), **Task Analysis for Knowledge Description** (Johnson et al. 1984) and **Knowledge Analysis of Tasks** (Johnson & Johnson, 1989). The techniques vary in what stages of software design they can be applied to (early - late), what they tell us about the tasks and indeed what exactly constitutes a 'task'.

In order to decide on an appropriate TA method it is important to be clear as to the purpose of the analysis. This is also important once a TA method has been selected as it can influence the data

collection methods, grain of analysis required, and the results presentation format. The aims of this project have been initially defined as:

> (a) To determine the user interface requirements for Computer Music Systems in order to provide criteria for this aspect of software design and to evaluate critically existing computer music software according to these criteria.
>
> (b) To assess the applicability of task analysis techniques to music composition and to explore alternative presentations of the tasks and methods associated with computer music to the user.
>
> (c) To develop user interface design solutions improving on current usability/learnability levels, providing abstraction levels close to the conceptual framework of the musician/composer.

Thus the main purpose of the TA is that of accurate (cognitive) user requirements capture for computer music system user interfaces which is then to be used to evaluate current software and assist design of new user interfaces. The project also attempts to measure the success of applying TA to music composition tasks - to examine whether music composition can successfully be decomposed into a task structure (containing goals, subgoals, plans, objects etc.) via application of TA techniques.

The TA method chosen should therefore be **system independent** (as we will be looking at composition processes carried out across technologies and systems), **appropriate to early analysis** (i.e. before any new interface system has been designed), **capturing cognitive elements of tasks** (music composition can be a highly internal process with few external/physical signs of how the process is being carried out), as well as being able to provide **criteria for design and evaluation of software systems.**

Having examined a variety of TA techniques it is proposed as a starting point to base this work on the KAT methodology for collecting task knowledge data, analysing the data in terms of TKS elements and producing a generalised TKS (Johnson, 1992). There are a number of reasons for doing so. Firstly KAT fulfils the criteria given above. Secondly the complete process of TA is outlined and well documented, including combining task elements from multiple descriptions of different users to produce generic descriptions - unlike HTA (Carey, Stammers & Astley, 1989) and methods of mapping a generalised TKS onto software design, Waddington & Johnson, 1989 (although not yet methods of evaluating existing designs, although Johnson is working on this). Also there are perhaps some parallels with Laske's model of musical activity describe later in this report. Further the relations between the elements that make up a complete task are various and include unordered relations allowing for some of the ill structured nature of composition mentioned earlier i.e. the task analysis does not necessarily produce a system with one path to achieving a particular goal and requirements that a task be done at a particular position within the overall task structure (although this can be fixed if required).

2.4 TKS/KAT:

What follows is a summary of TKS and KAT taken mainly from Johnson, 1992.

Johnson describes a task as:

'.... an activity that is undertaken by one or more agents to bring about a change of state in a given domain.'

He groups tasks in terms of 'roles' and 'jobs' - a job defined by a set of roles a person is expected to adopt, each role having a set of associated tasks.

' Task knowledge structures represent the knowledge people possess about tasks they have previously learned and performed in a given domain. Task knowledge is assumed to include knowledge of goals, procedures, actions, and objects' . As such a TKS is a description of task knowledge assumed to be held

by users in long term memory which is activated when carrying out the associated task. Johnson's summary definitions of TKS elements follow:

Goal: a state to be achieved, the purpose for which a task is undertaken.

Subgoal: an intermediate state in the path of achieving a goal.

Plan: the result of problem solving activity which identifies the path to achieving a goal in terms of goals and subgoals.

Procedure: an executable behaviour consisting of actions and objects achieving a goal or subgoal (i.e. carrying out a task).

Action: an operation that is performed as part of a procedure.

Object: an entity within the domain (informational, conceptual or physical) possessing a set of defining properties and associated with actions within procedures.

TKS also identifies *representativeness* and *centrality* of TKS elements. Central elements are those upon which the success or failure of a task critically depends. Representativeness describes the typicality of an instance of a class - a typical string instrument object might be a violin which is therefore representative of its class. Representativeness is a matter of degree, not all or nothing. These properties are important when combining TKS data across task performers.

A complete TKS model has three main components; a **goal oriented structure**, a **procedural substructure**, and a **taxonomic substructure**. The separate structures represent different aspects of task knowledge and so contain different classes of TKS elements.

The goal structure contains goals, subgoals and their relations (it therefore implies plans). Relations between goals/subgoals include hierarchical relations (goal decomposed into subgoals) and control relations - states for goals to be achieved including sequential, parallel, unordered and optional relations). The goal structure can be represented graphically via tree structures and transition networks or textually via a functional language, pseudocode or frames. Goals are described in terms of activities and objects and their relations.

The procedural substructure contains procedures in terms of the TKS actions and objects and relations between them - sequential, parallel, iterative, conditional etc. Each procedure has pre- and post-conditions determining the context that must exist prior to execution and that which will result from execution. The procedural substructure can be represented in the form of pseudocode, production system or frames.

The taxonomic substructure contains detailed information about objects - the class structure, attributes, etc. Task attributes include centrality, representativeness, and the actions and procedures associated with the object. Objects can be
represented by frames or through an object oriented language.

Knowledge Analysis of Tasks is a method of task analysis developed from TKS theory and consists of three parts: data collection, data analysis, task domain modelling. Three criteria for sampling are emphasised: People & Roles, Organisations, and Technology; the constraints of each should be thought out before the TA begins in earnest. Our people are composers who may regard composition as having separate roles within composition - instrument constructor (on computer) or configurator, score arranger, etc. Organisations are perhaps not relevant here, but technology is - we are trying to involve various technologies in the work.

Four KAT guidelines are suggested: **Identify the purpose of the analysis, check the analysis with task performers, analyse more than one task and performer, use more than one knowledge gathering technique.**

Johnson lists several knowledge gathering techniques he deems suitable for identifying the various TKS elements; the most common ones are summarised below (knowledge gathering is reviewed later as a separate issue):

Objects and Actions:	textbooks, manuals and other documentation. structured interviews questionnaires direct observation concurrent or retrospective protocols
Procedures:	direct observation concurrent protocols
Goals and Subgoals:	structured interview retrospective protocols direct observation questionnaires constructing tree diagrams

As data is gathered and analysed, the TKS elements are built up and validated by returning to task performers and obtaining further information from them in an iterative manner. A second stage of analysis is to combine information across task performers to produce a list of generalised TKS elements. This involves determining the centrality and representativeness of the TKS elements. Methods for doing so are suggested including; frequency counting (using thresholds for generic elements), rating scales, card sorting, recall of task components, grouping of 'alike' elements and labelling the groups. Validation of generic elements with task performers is again necessary.

Finally the results of the KAT process are combined to produce a generalised TKS model (Generalised Task Model, GTM) containing the goal, procedural and taxonomic substructures described above. This model can then be transformed in different ways to produce a model of the user interface (still device independent) Specific Task Model and further to a device Specific Interface Model, details in Johnson, 1992.

2.5 Knowledge Elicitation:

Having selected the TA methodology perhaps most suited to this work, the problem remains of how to elicit the musical task knowledge required in order to construct the task model. Johnson suggests some knowledge elicitation techniques to be employed in the TA process and Cordingley (Cordingley, 1989) describes several of these techniques in more detail (as well as others). Each method has its own merits and problems when attempting to elicit what is usually tacit knowledge from task performers - "...the kind of musical knowledge that, if implemented, would improve computer music tools is often not public or even shared among experts, but personal, idiosyncratic knowledge .... the elicitation of personal knowledge, and of action knowledge, still awaits a methodology, and easy to use, interactive support tools." (Laske, 1992).

Cordingley, 1989, gives a working definition of Knowledge Elicitation (KEL) as follows:

Knowledge elicitation is those activities undertaken by a person , the knowledge elicitor, to
- obtain material from any relevant source
- analyse and interpret that material
- put in a pre-encoded form which while useful to those who will encode the knowledge in the KBS (knowledge based system) language, also allows it to be scrutinised by all parties interested in KBS development.

For 'KBS language' we can perhaps substitute 'Task Model' and for 'KBS development' , 'task analysis process', since, "It is surely clear that the problems confronting those using task analysis in HCI are identical to those confronting the knowledge engineer in knowledge elicitation..." (Diaper, 1989).

Since the knowledge we seek is 'personal' and 'idiosyncratic' it is clear that the knowledge sources needed will to a large extent be human task performers.
Cordingley (Cordingley, 1989) provides a review of common knowledge elicitation techniques and advice on selecting the techniques for particular elicitation tasks. Some relevant methods are described briefly below together with problems we might expect to come across with each technique.

An initial approach to KEL is often the use of questionnaires. These allow a large group of human knowledge sources to be approached - larger than if the elicitor were to interview them individually - and can provide a first contact with knowledge elicitors. Looking at questionnaire results the elicitor can then select appropriate sources to go to for further elicitation. Problems with questionnaires are that the elicitor and provider cannot clarify each others questions and answers respectively, the amount of information on a questionnaire has to be limited in order that the time taken to complete it is not excessive, the size of spaces left for answers will affect the detail supplied and so on (i.e. care must be taken). Also the inflexibility of the questionnaire format is again limiting - in an interview the elicitor can take up points made, investigate new avenues, adjust to the elicitors personality etc. which is not possible with a questionnaire. As such the questionnaire is suited to broad surveys of information lacking in real depth but a useful starting point.

Questionnaires can then be followed up with interviews, perhaps picking up directly on points arising from information given in questionnaires. Interviews can be recorded in a variety of ways (note taking, video, audio etc.) and can be structured/unstructured and focused/unfocused (Cordingley, 1989). Structure refers the fixed/planned nature of questions asked, focus refers to scope of the subject matter covered. Interviews can require a large amount of preparation including decisions about where to hold it, what question types to ask (La France's six question types, Cordingley, 1989), the amount of structure and focus, suitable recording method etc. There is also the problem of how to (and time taken to) analyse the recorded information in what ever form it has been recorded in. For task analysis it is usually desirable (although not always possible) to interview knowledge providers in their usual task environment which can help recall of task processes, providers can perhaps give examples of tools or techniques they use, and the elicitor can note details of the arrangement of the physical task environment. A key problem with interviewing techniques is that the knowledge elicited is only knowledge that the source is able to verbalise - often difficult for experts who tend to forget verbalised approaches and 'just do it', i.e. have highly compiled task knowledge.

Another common elicitation technique is that of protocols - either concurrent, requiring the knowledge provider to give a verbal explanation of what they are doing during task performance, or retrospective, where immediately after task completion the performer perhaps provides a commentary to a video recording of their task performance. Other variations on this theme are possible such as interrupted protocols where the user is interrupted every time period and asked what they are doing. A key problem with concurrent protocols is that of interference - the fact that having to give a concurrent verbal report interferes with task performance quite drastically. Erricson and Simon, 1986 looked at this problem of interference and found that creative/artistic tasks (e.g. music composition) are possibly the most affected by interference. Retrospective protocols need to be carried out as soon as possible after task completion and need some record of the task performance to reduce post hoc rationalisation and hiding of errors made etc. and also to assist accurate recall of the task performance.

Task observation is a another technique where the knowledge elicitor simply observes task performance (or perhaps videos performance and studies the video) and notes exactly what the task performer is doing - perhaps in terms of the order of operations, where mistakes are made and how frequently, recurring operations etc. While this may get at information on low level goals/procedures it may not reveal much of the high level strategies or plans involved since we do not know how the individual operations fit into any particular pattern. However used in conjunction with retrospective protocols it can fill out the low level detail and help verify orders of events and where mistakes occurred etc.

Other techniques include Kelly's Personal Constructs, sorting, teachback, laddering, role play, simulations etc. suited to differing stages of analysis (see section on KAT above).

2.6 Task Observation/Recording:

Task observation is important in TA/KEL and is used in a variety of ways - for generating protocols, to be analysed as a KEL technique in its own right or simply as a record of what happened during task performance. Protocols made during task performance tend to interfere greatly with performance and retrospective protocols without a record of performance are difficult as the performer cannot remember all the detail of what occurred (and may not admit to mistakes). Computer capture of task performance is possible with computer based tasks. For other types of task video recordings can be used and played back to performers to assist recall. We have been examining ways of recording users interactions with computers on the Macintosh - without building purpose built music software with in built capture tools. The main reason for this is that computer based tasks can help project the internal workings of music composition (strategies/techniques) onto an external system assisting observation and analysis, without limiting us to one type of composer, since the three composer classes identified can all use computers in composition although in different ways - simple score writing, MIDI sequencing or computer synthesis or some combination.

AppleScript (AS) is a high level language for the Macintosh for programming macros that can be run or stored as mini applications. Using this language a small application can be created that runs in the background and automatically saves documents that are open in a given application every definable time period. This is similar to the autobackup facility provided by some applications, except that this application saves each time to a new file. This means that we have a record of the state of the 'artefact' (music score, sound files being edited etc.) at different stages throughout the task. This can then be presented sequentially to the user in a retrospective protocol. The problem with AS is that only certain applications respond to the necessary AS commands (AS uses AppleEvents, only a certain small set of which all Macintosh applications have to respond to). Also each application to be used must be incorporated into the applescript program before running - i.e. the program must know which software packages to backup from. Unfortunately we have not found any music software packages that are fully 'scriptable' and so work with this application. A Csound task could perhaps be recorded in this way since there is a scriptable text editor provided with AS.

QuicKeys is another macro provider in the form of a control panel and inits. One possible use of QuicKeys is its real time recording facility that allows the elicitor to record the user inputs (keyboard and mouse) to the machine and play them back later. At face value this is ideal as the session is recorded as it happens on the machine and requires no intrusive equipment (video camera etc.) but there are **problems**. One is that the playback will only work on the same machine it was recorded on and in the same state as it was before recording began i.e. if extra files are present files can be in different places in selections and so different files are selected leading to different results. If windows re-open in their last open position and were moved during task performance then mouse button presses are made in the wrong places. Another problem is that QuicKeys does not record MIDI input via Apple Midi Driver and so composers using a MIDI controller to input data will not record this into the QuicKeys protocol. Also the variability in saving/loading/deleting times on the Macintosh sometimes leads to replay errors for fast users. Also there is no editing facility to remove dead time for example, or variable speed playback. These problems make QuicKeys in this mode unsuitable for this work at the moment although other task observation work may find this useful.

A second use of QuicKeys is to create a macro that triggers the Apple screen 'snapshot' function which captures the screen to a picture file which can be viewed later using a variety of applications. This can be set to fire every definable time period say 5 minutes and a record built up of what the user had on the screen at various times. This is more hit and miss than the real time recording but has fewer practical problems of carrying it out. These snapshots can be printed out or shown on a second computer and used in aiding memory in retrospective protocols etc. This can be done on any Mac with QuicKeys installed and shown on almost any Mac. For showing a Mac with a larger screen may be necessary to see the complete screen clearly. The question still to be answered is, does the capture of the screen assist a retrospective protocol effectively?

Another possibility is to record screen output directly onto video tape. This can be done with some Macintosh computers directly AV machines, or via a third party video card or external hardware such as

a Video Logic's Mediator. This requires taking a video recorder into the composing environment which may or may not be possible/desirable, and the quality of video obtained is variable depending on the hardware involved (and at best is usually poor).

All these methods will only capture use of the computer and not use of external hardware (e.g. MIDI synthesisers) and some will not capture aural feedback used in composition. Therefore it will be necessary to set up a video camera to capture the behaviour of the task performer and use computer capture to get at the details of what is happening (it is very difficult to set up a camera that can see what is happening on the computer and capture use of other equipment at the same time). It would be problematic to use videos of both the screen output and the task performer at the same time in a retrospective protocol, however the video of screen output may be studied at a later date to help add detail to the retrospective protocol analysis and verify what actually happened during the task performance.

A further remark to make with regard to the computer capture of task performances is the problem of keyboard shortcuts. These are key presses that invoke commands from menus. With a purely visual capture method it is not always possible to know which command has been keyed since the only indicator may be the menu title flashing. These shortcuts are important to expert users of a system since they considerably reduce the time it takes to invoke a command. Therefore when using a visual capture method (such as video output of the screen information) it may be necessary to request that the task performer refrains from using shortcuts and uses the mouse to select menu commands as much as possible.

3. Cognitive Musicology:

3.1 Definition:

'Cognitive Musicology has as its goal the modelling of musical knowledge in its many forms.' (Laske 1992) - i.e. Listening, Composing and Performance. The relevant area of CMgy for this project is therefore its compositional aspect. CMgy has been driven chiefly by those in the field of Artificial Intelligence (AI) and Music attempting to create a computer system that is either a composer, listener and performer (or combination). At first glance CMgy appears to be exactly the area this project is currently concerned with (i.e. modelling of compositional tasks) but work in the field has had a different approach to analysis of compositional task knowledge than the type of task analysis generally used for user interface design.

3.2 Laske Model:

Otto Laske (perhaps the key person involved in defining CMgy) describes musical activity in terms of five levels (Fig. 3.2.1. , Laske, 1992) : the highest, an **Action Level**, that designs a **Planning Level**, that controls a **Task Level**, that applies the contents of a **Domain Level**, to a **Task Environment Level**. The **TEL** contains physical tools, historical conventions, cultural influence etc. and as such includes the computer music system (and its user interface). The **DL** contains domain specific knowledge (competence, declarative knowledge) that is supported by the **TEL** and used by strategies in the **TL** to select the **TEL**. The use of task structure knowledge contained in the **TL** (performance, procedural knowledge) is controlled by the **PL** containing knowledge of goal synthesis, task sequencing (performance, procedural). The **AL** describes the whole musical activity and is a meta-level for the subordinate levels.

In keeping with Laske's model, this project is attempting to improve one aspect of the **TEL** (the computer-composer interface), perhaps via reducing the burden of **DL** and **TL** knowledge introduced when using a computer music system, by eliciting and analysing knowledge from the **PL** and **TL** (both

forms of task knowledge) of composers from a variety of music backgrounds. By looking at task knowledge composers already possess, and task knowledge required for using current systems it is hoped to reduce any inconsistencies that may be forthcoming via designing a more appropriate user interface.

We can make a comparison between Laske's levels and Johnson's TKS substructures (see Fig.3.2.2) : Planning Level with Goal Substructure, Task Level with Procedural Substructure, Domain and Task Environment levels with Taxonomic Substructure (although Actions are perhaps part of the Task Level, Objects part of the Domain and Task Environment Levels). There is not a direct correspondence between the models (e.g. Laske's Task Environment level implies a much larger domain than Johnson's Taxonomic substructure, Johnson does not explicitly separate declarative and procedural knowledge) but there are relevant similarities.

Fig.3.2.1. Musical Activity (from Laske, 1992)

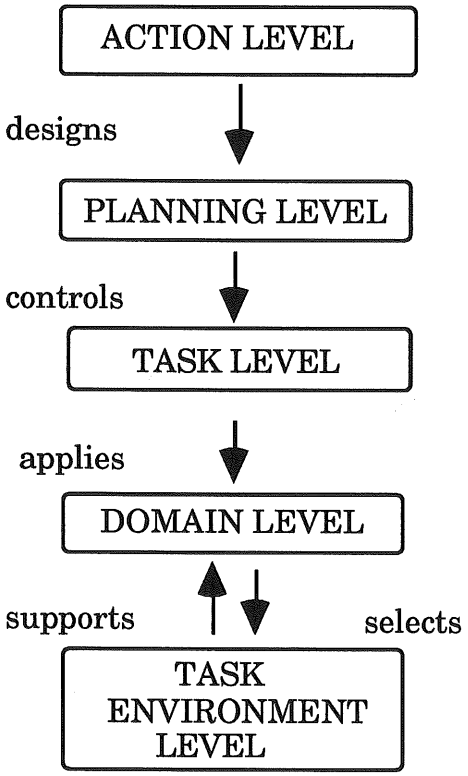
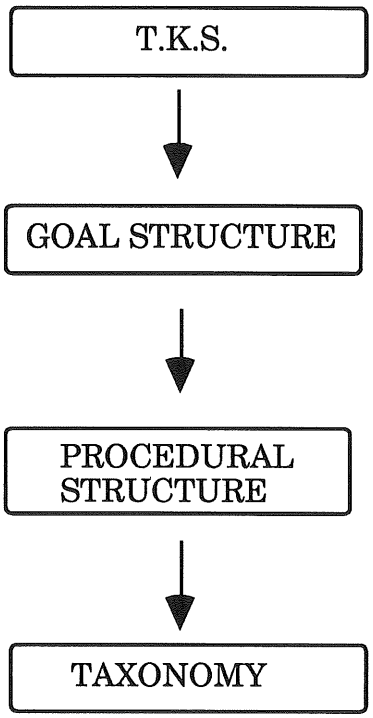


Fig.3.2.2. TKS Hierarchy



3.3 Cognitive Musicology and Composition:

Recent CMgy work related to composition has involved attempting to build models of music composition into Knowledge Based Systems in order to measure the effectiveness of the model i.e. by building a system and seeing whether it can produce the desired musical results. The concern of this project is to develop a model of the strategic paths taken by the composer in creating a composition and the tools required to achieve the composition goal - what s/he does. Cognitive musicology generally examines why or how particular music composition task decisions are taken and reducing composition to a set of rules and inferences (a knowledge base and inference mechanism). Some work in this area may have relevance in its attempts to capture musical task knowledge, even if the type of knowledge elicited is slightly different and the use to which the information is put is also different.

Laske (Laske, 1977) is one of the few researchers in the field to have looked at task analysis of musical processes - listening, composing and performance. In defining the area of cognitive musicology or

psychomusicology he has often been concerned with the complex memory systems involved and producing sufficiency analyses of tasks i.e. writing a conceptual hypothesis of the task in the form of a computer program which, if capable of solving the music composition problem, is said to be a sufficiency analysis. This analysis is not necessarily psychologically valid and would need to be tested - the program may achieve the task goal in a wholly different manner to the human task performer.
This project is more concerned with what Laske refers to as empirical task analysis rather than conceptual task analysis with which much of his work deals. A detailed account of the contents of any 'knowledge base' utilised in various composition tasks and how this interacts with other 'memory systems' is not the information required at this stage of the user interface design process, and it would be beyond the brief of this project to attempt the elicitation of all such information. Laske has limited himself to examining specific (given) tasks working within specific task environments and attempted to look at the cognitive processes involved in great detail. Rather for the purposes of this project, at this stage, it is more useful to look (initially) at a wide range of composition tasks and environments (time permitting) and identify any generic groups of tasks and task (TKS) elements that can be usefully brought to bear on user interface design and evaluation. Focus of the analysis may later be concentrated in some specific problem areas but this will need to be directed via the initial results of a more wide ranging preliminary investigation.

Laske's work has concentrated on eliciting non-verbal musical knowledge via two main computer systems - OBSERVER and PRECOMP (Laske, 1992) - setting specific music tasks such as creating a short melody line or CMN score production, and generating protocols recording the (musical) actions of the composer. These protocols are then analysed by the computer to elicit the required information. This information is then intended to be used to attempt to produce an automated composition system - this was not completed for Observer - and to build a knowledge model of music composition. This is very fine grained work on small tasks with specific systems. Laske has used purpose built command line driven software that incorporates the sound synthesis tools, protocol generator, protocol analysis and automated composition processes. Some kind of model is already built into the system in that the composer can only carry out the task in a way defined by the music task environment. For our work we would like to examine a variety of task environments and compositional styles and look at the more high level planning and operational aspects of music composition.

4. Current Situation:

4.1 Position now:

A questionnaire has been designed and will be sent out to as many composers across the board as is feasible, to gain a broad picture of compositional environments, methods, and problem areas and also find composers willing to partake in elicitation experiments etc.
We are carrying out some pilots studies with short composition tasks and different capturing methods etc.
We are also exploring some of the latest CMSs and computer music techniques/models and interfaces.

4.2 Future work:

Future work to be carried out includes:

Analysis of questionnaires to provide more detailed information on prospective users and their task environments.

Longer elicitation studies using methods found to be successful in the pilot studies and adapting to other situations etc.

Designing other knowledge elicitation experiments (such as giving a new system to user with minimal information and monitoring their queries of how to do various processes - and providing answers) and elicitations not using computer systems.

Evaluations of some current systems according the task models constructed.

The area of music representations and notations is still to be explored (it is difficult to notate some computer music processes with CMN) - how musical objects are presented to the user and how they can be manipulated, what use can be made of graphical scoring techniques etc.

Looking at knowledge based systems - for perhaps building 'intelligent' instruments that incorporate a partial 'performer' into the device such that the composer can set music parameters rather than mathematical/physical ones, or whether pre-defined mappings are sufficient (which can be edited to suit the users needs).

Prototyping designs based on the Task models developed and experimenting with representations/notations etc.

User evaluations of results.
Etc.

4.3 Summary:

The problems of user interface design are complicated in this case by the idiosyncratic and ill-structured problem nature of music composition, together with the wide range of sound synthesis methods and music styles possible with computer music systems. It is hoped that the application of task analysis methods can produce, if not a single generic task model then a group of tasks models, that can be used to specify the probable user demands on a computer music system. These results can then be used to construct a user interface structure optimised for music composition of various styles and methods without requiring expert knowledge of computer languages or synthesis models. These results can also be used in evaluating the current state of computer music systems - highlighting both good and bad features of these systems. Investigations into music representations and notations should then be carried out in order to flesh out the interface design into an effective working program that is then evaluated by users and modified accordingly.

It is not possible to design a system interface that will satisfy all users all of the time, but attempting to produce one that caters for the majority of user tasks in a manner that fits common working practices it is hoped that high levels of both learnability and usability can be achieved resulting in a wider use of computer music systems.

Bibliography

Annett,J. et al. (1971) Task Analysis, Training Information Paper No. 6, HMSO

Card, S.K. Moran, T.P. & Newell,A. (1983) The psychology of human computer interaction, Laurence Erlbaum Associates.

Carey, Stammers & Astley (1989) Task Analysis for Human-Computer Interaction , Ch. 2. *Ed.* Diaper, D Ellis Horwood.

Cordingley,E.S. (1989) Knowledge Elicitation: principles, techniques and applications, Ch.2. *Ed.* Diaper, D. Ellis Horwood.

Diaper,D. (1989) Knowledge Elicitation: principles, techniques and applications, Ch.1. *Ed.* Diaper, D. Ellis Horwood.

Dobrian,C. & Zicarelli,D (1991) The Max User's Manual Opcode Systems.

Ericsson,K. & Simon,H. (1985) Protocol Analysis: verbal reports as data, MIT Press.

Ethington,R. & Punch,B. (1994) SeaWave: A System for Musical Timbre Description, Computer Music Journal, 18:1 pp30-39 MIT Press.

Johnson,P. et al. (1984) Tasks, skills and knowledge: task analysis for knowledge based descriptions *in* INTERACT '84 - Proceedings of the First IFIP Conference on Human-Computer Interaction pp 23-27, *Ed* Shackel,B. Elsevier.

Johnson,P. et al. (1988) Task related knowledge structures: analysis, modelling and application, *in* People and Computers IV, *Eds.* Jones,D.M. & Winder,R. CUP.

Johnson,P. (1992) Human Computer Interaction: Psychology, Task Analysis and Software Engineering, Magraw-Hill.

Kieras,D.E. & Polson,P.G. (1985) An approach to the formal analysis of user complexity, International Journal of Man-Machine Studies, 22, pp365-394.

Laske, O (1977) Music, Memory and Thought: Explorations in Cognitive Musicology. University Research Press Michigan

Laske, O (1992) Artificial Intelligence and Music: A Cornerstone of Cognitive Musicology *in* Understanding Music with AI *Eds.* Balaban, Ebcioglu & Laske, MIT Press.

Long & Dowell (1989) Conceptions of the Discipline of HCI: Craft, Applied Science and Engineering *in* People and Computers V *Eds.* Sutcliffe,A. & Macaulay,L. CUP.

Malt, M. (1993) Patchwork™ Introduction (user manual) IRCAM.

Moran, T.P. (1981) Command Language Grammar: a representation for the user interface of interactive computer systems, International Journal for Man-Machine Studies 15, pp3-50.

Morrison,J.D. & Adrien, J-M. (1993) MOSAIC: A Framework for Modal Synthesis, Computer Music Journal 17: 1, pp45-56, MIT Press.

Payne,S.J. & Green,T.R.G. (1986) Task-action Grammars: a model of the mental representation of task languages, Human Computer Interaction 2, 93-133.

Roads,C. et.al. (1993) Editors Notes (a Max discussion), Computer Music Journal 17:? , pp3-11, MIT Press.

Rodet, X. (1994) Stability/Instability of Periodic Solutions and Chaos in Physical Models of Musical Instruments, Proceedings of the 1994 ICMC, pp386-393, ICMA

Taube,H. (1991) Common Music: A Music Composition Language in Common Lisp and CLOS, Computer Music Journal 15:2 pp21-32, MIT Press.

Vercoe, Barry (1992) Csound: A Manual for the Audio Processing System and Supporting Programs with Tutorials. MIT.

Acronyms:

AI	Artificial Intelligence
AM	Amplitude Modulation
AS	AppleScript
CCRMA	Centre for Computer Research in Music and Acoustics
CLM	Common Lisp Music
CM	Common Music
CMgy	Cognitive Musicology
CMN	Common Music Notation
CMS	Computer Music System
CNMAT	Center for New Music and Ausio Technologies
DAC	Digital-to-Analogue Converter
DSP	Digital Signal Processing
FM	Frequency Modulation
FOF	Fonctions d'ondes Formantiques
GTM	Generalised Task Model
HCI	Human Computer Interaction
HTA	Hierarchical Task analysis
IRCAM	Institut de Recherche et Coordination Acoustique/Musique
KAT	Knowledge Analysis of Tasks
KBS	Knowledge Based System
KEL	Knowledge Elicitation
MIDI	Musical Instrument Digital Interface
MIT	Massachusetts Institute of Technology
PC	Personal Computer
PL	Planning Level
RISC	Reduced Instruction Set Computer
TA	Task Analysis
TEL	Task Environment Level
TKS	Task Knowledge Structure
TL	Task Level
ZIPI	Zeta Instrumental Processor Interface