ISSN 1744-1986

Technical Report N^{o} 2004/20

Synthesizer user interface design - lessons learned from a heuristic review

Allan Seago
Simon Holland
Paul Mulholland

11th June 2004

Department of Computing
Faculty of Mathematics and Computing
The Open University
Walton Hall,
Milton Keynes
MK7 6AA
United Kingdom

http://computing.open.ac.uk

Synthesizer user interface design – lessons learned from a heuristic review

Allan Seago
London Metropolitan University
Commercial Road
London E1 1LA
a.seago@londonmet.ac.uk

Simon Holland
Dept of Computing
The Open University
Milton Keynes MK7 6AA
s.holland@open.ac.uk

Paul Mulholland
KMI
The Open University
Milton Keynes MK7 6AA
p.mulholland@open.ac.uk

In this paper, we review the types of user interfaces used in electronic synthesizers, both hardware and software, and conduct a heuristic review on a number of representative examples. The process of building and editing sounds in modern commercial synthesizers requires a good understanding, both of the system architecture and of the synthesis engine itself, as the user/system dialog is conducted in system-specific terms, rather than in terms familiar to a musician. User interfaces of contemporary synthesizers may be procedural, presenting the user with a set of functional modules which generate/process sound, or alternatively, offer the means of creating/editing sound, using direct manipulation techniques. We conclude that neither style emerges as being better suited to the task of sound synthesis; in this domain, direct manipulation is limited in the degree of control afforded, while the form filling style characteristic of fixed architecture synthesizers is slow and laborious.

Keywords: User interfaces, Music, Synthesizers, Heuristic Review

1 Introduction

This paper analyses a number of electronic music synthesizer user interfaces from various HCI points of view, and identifies relevant taxonomies, design issues and previous research for user interface design in this specialized and demanding domain. Heuristic evaluations are presented for various representative user interfaces from different parts of the taxonomy using a common task, and lessons are drawn for future user interface design and for future research in this area.

The user interfaces of audio hardware and audio software generally, and of synthesizers in particular, have received relatively little study from the HCI perspective. A number of the more relevant and recent HCI studies in this area are outlined here. An analysis conducted on the working methods of composers working with Computer Music Systems (CMS) (Polfreman & Sapsford-Francis, 1995) identified a number of typical tasks, and concluded that CMS designers should consider wide variations in composers' knowledge, and in the types of composer they are designing for. Furthermore, it was recommended that, in order to be usable by composers with varying levels of skill, and compositional style, CMS systems should provide more than one level of interaction, 'hide' unwanted levels of complexity, and provide a knowledge based system (KBS) for details that a user does not wish to specify directly. A previous review of synthesizer user interface design (Ruffner & Coker, 1990) focused on the control surfaces of four contemporary instruments, and commented on the degree to which they conformed to design principles identified by Williges *et al* (1987). They concluded that the demands placed on the user by the interfaces made them far from ideal for the purpose - noting that, in general 'user interface principles have been, at best haphazardly applied' in the design of the synthesizer interface. The authors also suggested a number of issues that should drive future research in this area. Another more recent study (Fernandes & Holmes, 2002) has applied a heuristic evaluation to an electric guitar pre-amplifier interface.

The present paper examines a number of interaction styles, taking a number of commercial software and hardware synthesizer user interfaces (UI) as examples. In addition, a *heuristic evaluation*, derived from Nielsen and Molich, and described in Nielsen, 1994, is conducted on these interfaces.

2 Background

The range of tools and techniques available to the musician for the design and editing of sound is vast; however, the modern synthesizer is far from intuitive in its use. The UI design of hardware synthesizers is an illustration of a number of issues raised by Harold Thimbleby with respect to such devices as hand held calculators, televisions, microwave cookers etc (Thimbleby, 2001). He notes that the hand held calculator is a 'mature technology', with well defined requirements, and goes on to describe two models of calculator which look superficially very similar, but whose controls often do different things. Similarly, over the past fifteen years, control surface designs of commercially available synthesizers have to some extent converged, to the extent that we can consider this type of instrument to have acquired a generic interface (Pressing, 1992). As with calculators, however, the user cannot assume that similar looking buttons will perform the same function, and conversely, a given function could be conceivably performed by a number of different controls.

As Thimbleby says, the range of tasks performed by a calculator is well defined; however, that which must be performed by a synthesizer is considerably greater, and, as discussed in this paper, is not so easily defined. Nevertheless, the difficulty of negotiating the interface means that most users have contented themselves with making use of a bank of instantly available presets - evidence for this is largely anecdotal, but "allegedly, nine out of ten DX7s coming into workshops for servicing still had their factory presets intact". (Computer Music, 2004)

Over the last few years, hardware synthesizer functionality has increasingly been migrating into software (*Reaktor*, *Reason* etc), and it is useful to examine the effect this has had on the user interface. Although in theory freed from the constraints imposed by the hardware – the limited space available for controllers and data displays, in fact, software designers have sought to emulate their hardware counterparts not only in broad functionality – the synthesis of sounds - but also in the interface offered to the user; thus the user is presented on screen with a simulation of an analog synthesizer control surface, and manipulates software buttons, faders and rotary dials using the mouse.

3 'Real time' and 'fixed' synthesis

Pressing, 1992 describes the controls of the synthesizer interface as falling into two broad categories – those which govern 'real time' synthesis, and those which provide access to the parameters governing 'fixed' synthesis. 'Real time' synthesis controllers - pitch wheels, foot pedals and, of course, the keyboard - allow instant and dynamic modification of single scalar quantity aspects of existing sounds – pitch, filter cut-off frequency, volume etc. These controllers are very much designed and positioned on the control surface to meet the requirements of performance, and are to a great extent intuitive in their use (the effect that a controller has on the sound is instantly audible); they will not be considered further here.

The 'fixed synthesis' category of controllers, however, will be examined more closely. This component of the interface allows the design, programming and representation of basic sound objects, and typically requires an in-depth understanding of the internal architecture of the instrument, and the methods it uses to represent and to generate sound. The user is obliged to express directives for sound specification in system terminology, rather than in language derived from the user domain.

4 Synthesis as industrial control process

Software sound synthesis invites comparison between the types of interactions it affords, and those of other application domains, such as text processing, graphics and animation. However, the user interface of the typical commercial synthesizer, in both software and hardware, more closely resembles those interfaces which facilitate industrial control processes, which act as intermediary between the operator and the real world (Dix *et al*, 1998), and in which two levels of feedback are provided – firstly, from the instrumentation - meters, gauges etc – and secondly, from the equipment or processes being controlled (see fig 1).

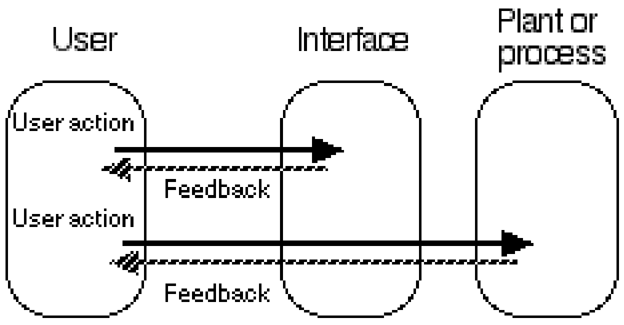

Fig 1: Adapted from Dix *et al* (1998) Human-Computer Interaction

A useful analogy is that of the flight control deck, where the pilot is provided with specific information on altitude, pitch, yaw, and speed from the instrumentation, but also is more directly afforded 'real world' information by the behaviour of the aeroplane itself. Similarly, the user of a typical synthesizer is able to determine the current 'state' of the sound being edited by the parameter values displayed in the LCD, but can also audition the effect of the editing process by pressing a key.

This is not the case with, for example, a desktop operating system, where the mouse dragging of a document icon from one folder to another is not accompanied by any perceptible 'real world' feedback - because the user is engaging with a "representation or model of reality" (Shneiderman, 1997), he/she takes it on trust that the action has actually happened. Similarly, the results of actions taken by the user of a WYSIWYG graphics or word processing program are visible on the screen - no confirmatory 'real world' feedback is provided until the document is printed. From the user's point of view, the interface and the objects of interest are one and the same (Dix *et al*, 1998).

Such a 'fusing' of the object of interest and its iconic representation is more problematic with sound synthesis systems, most obviously because the object of interest is non-visual. While an iconic representation can readily be found both for objects such as documents, drawing tools and geometric shapes, and for actions such as copy, delete, draw etc, sound does not lend itself to any form of consistent visual description which can be said to make "psychological sense" (Karkoschka, 1966) – that is to say, where its subjective and timbral, rather than purely acoustical and measurable characteristics are apparent. (The Western musical score captures only a limited range of sound attributes – pitch, for example; but it otherwise relies heavily on verbal directives to indicate subtleties of tone colour and articulation).

5 Visual representation

Visual representation of sonic information is usually in either the **time domain (**a plot of audio signal voltage with respect to time, e.g. fig 2, or the **frequency domain (**a plot of the relative amplitudes of the frequency components of a waveform.

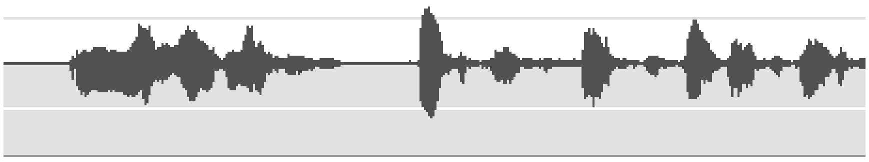

Fig 2: Time domain representation of sound

The interpretation of time domain plots is, to some extent, intuitively clear; this is a sound made up of sonic fragments, of varying degrees of loudness, punctuated by silence (this is a recording of normal speech). Zooming in on this display reveals details of its waveform.

In theory, this *output expression* of the system is capable of being used to formulate an *input expression* in a manner characteristic of direct manipulation systems (Dix *et al*, 1998) - in this case, by the provision of tools to 'draw' and 'edit' the desired waveform. (One of the first synthesizers which allowed the user to sample and edit sound directly, the Fairlight CMI, included a page on which waveforms could be drawn on screen using a light pen.) In practice, of course, an interface for 'designing' sounds in this way firstly assumes a static, non-varying waveform over time, and, secondly, is hampered by the lack of an obvious and intuitive mapping between the subjective and perceptual characteristics of the sound as a whole and its visual representation on screen. It presupposes that the user is able to specify minutely the waveform of the sound that he/she is trying to create.

This is also true of *frequency domain* representation, typically taking the form of a frequency spectrum plot derived from Fourier analysis of a waveform. The *core language* information which it provides on the frequency content of a waveform is too low-level to be useful as a general means of manipulating sonic attributes.

More generally, there is a considerable gulf between the *task language* and the *core language* (Dix *et al*, 1998). Fig 3 illustrates the point. The left hand column contains examples of terms which might be used by a musician to describe a given sound. Such terms are used to describe those attributes of sound - timbre, texture and articulation - which cannot be entirely captured by the conventions of the Western musical score and are often chosen for their perceived extra-musical analogies with other domains - colour and texture, for example - or for emotional associations. The right hand column, by contrast, describes objective and measurable quantities and attributes associated with sound. The problem is one of mapping one set of descriptors to the other.

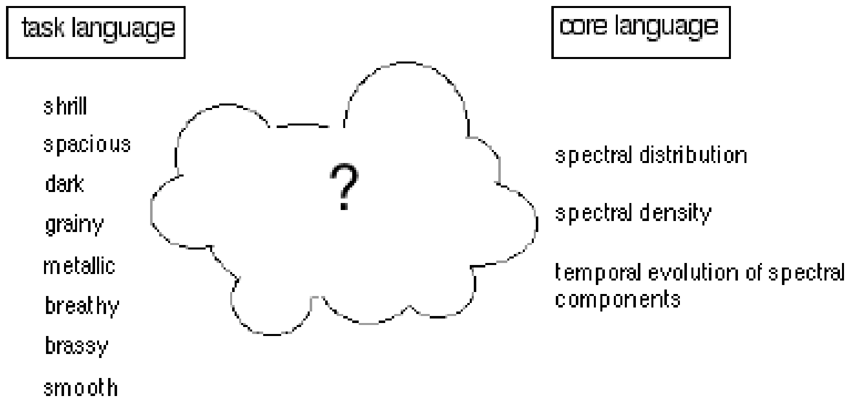

Fig 3: User and task languages

The problem of bridging the gulf between task and core language in sound synthesis has been approached in a number of ways – using techniques drawn from artificial intelligence (Miranda, 1995) and knowledge based systems (Ethington & Punch, 1994, Rolland & Pachet, 1996). Some of these approaches are briefly summarised here. A successful metaphor has been drawn from the control surfaces and sound generating mechanisms of acoustical music instruments. Physical modelling is the simulation in software of these mechanisms (Smith, 1992), the user being able to vary the software parameters of an emulated instrument – the length of its strings or bore, the size of its finger holes, the stiffness of its reed, the density of its soundboard etc – in order to create new and interesting sounds. Another approach has been to consider the multidimensional parameters of sound as the axes of a *timbral space*, and to devise means of structuring and negotiating that space (Vertegaal & Bonis, 1994).

6 User interface architecture types

Until the early 1980s, commercial synthesizer user interface design was primarily **indirect** and **procedural**, in which a given sound was defined in terms of the electronic modules required to generate it, and in which the interface elements presented to the user offered control over the settings of these modules. Two types of architecture can be discerned within this overall category, **fixed** and **user specified** (see fig. 4).

Fixed architectures present to the user an internal model of sound which is essentially static and tends to be hierarchical - sound as a tree stuctured assemblage of parameters. Typically, a given sound will consist of a number of 'voices', each of which in turn is made up of a number of waveforms, or 'elements'. For the user, the task of defining a sound is one of negotiating this structure, specifying parameters by a 'form filling' process. User specified architectures, by contrast, are essentially fluid and non-hierarchical. These two types will be examined in this study.

A second interaction style, which exploits the GUI interface, and which will also be examined, is that of **direct engagement** with visual representations of sound (usually, but not always, through direct manipulation techniques), as distinct from manipulating the means of generating that sound. Here, the user is able to work with a visual time or frequency domain representation of the sound, using tools analogous to those used in graphics packages.

Indirect, procedural	- fixed architecture
	- user specified architecture
Direct engagement	

Fig 4: User interface architecture types

7. Task analysis and heuristic evaluation

In this section, we will report on a task analysis and heuristic evaluation carried out on a small but broadly representative range of synthesizer user interfaces. The synthesizers surveyed here are discussed under the three interface categories noted above. For each synthesizer, a task analysis has been performed, the task in question being the creation of a simple 'sound object'; a sound whose time domain waveform is a sawtooth, whose frequency is 440 Hz, and whose overall amplitude envelope describes a long and smooth decay. Before considering the analysis further, a few comments are in order.

Firstly, it should be pointed out that many synthesizers do not fit solely into one or other of these categories, and make use of a variety of techniques to achieve the same end; indeed, one of the features of a good interface is precisely this kind of flexibility.

Secondly, all are very different in their capabilities, performance modes, and methods of sound generation, and in order to compare 'like with like', the chosen task is deliberately limited in scope.

Thirdly, the task, as defined here, is, to some extent, contrived. In a working situation, a user would be more likely to take an existing sound from the library available, and edit it, rather than generate a sound *ab initio*. However, given that the libraries available will differ (and will not exist in some cases), the evaluation necessarily needs to analyse the process from the beginning.

Five distinct phases in the generation process can be discerned; *initialisation*, *waveform selection*, *pitch selection*, *envelope selection*, and *save*. The first and last of these really only apply to synthesizers which are programmable; however, the second, third and fourth phases (which may occur in any order) are common to most, if not all architectures, and relate to timbre, pitch and loudness respectively. Pitch and loudness are quantities which are located on to one dimensional scales - high/low, loud/soft, and can be readily mapped onto controllers such as knobs ansd sliders (as mentioned above, these can be configured as real time synthesis controllers). Timbre, on the other hand is multi-dimensional, and as such is less easily captured and represented in a way that permits intuitive modification; and it is this aspect of synthesizer programming which has presented problems for interface design.

Each synthesizer is evaluated in terms of the number of steps required to complete the task outlined above. The raw data from each task analysis is not presented here, but is viewable on *http://www.lgu.ac.uk/~seago/Task_analysis.htm*.

Additionally, a heuristic evaluation technique, adapted from Nielsen, 1994*,* is used. This is a structured method for evaluating a user interface design, by considering it against a number of design principles and criteria, among them visiblity of system status, match between system and real world, user control and freedom. Normally, this would be done by several independent evaluators; in this paper, however, the author has used it to highlight some of the features of, and differences between various interaction styles.

7.1 User specified architectures - Reaktor

In this architecture model, sound is viewed as the output of a network of functional components - oscillators, filters, amplifier. A simple representation is given in fig 5.

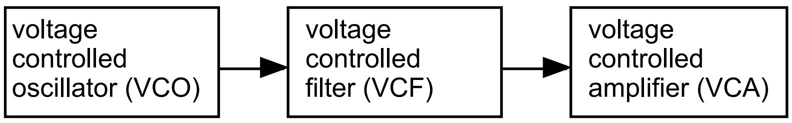
voltage controlled oscillator (VCO)
voltage controlled filter (VCF)
voltage controlled amplifier (VCA)

Fig 5: Basic oscillator/filter/amplifier configuration

The structure of this network, as stated previously, is fluid, and can become quite complex - in addition to the possibility of the output of any element being processed by one or more other elements, the parameters of each element - frequency, envelope, cut-off frequency - can be dynamically controlled by the output of another. Early subtractive synthesizers (Moog, ARP etc) come under this heading; with the basic components being linked by patch cords, the signal path is visible and immediately modifiable.

In practice, of course, the concept of sound as a non-hierarchical network of functional modules is limited, in the case of hardware synthesizers, by the number of hardware modules available. A software version, on the other hand, has no such restriction. One striking aspect of the oscillator/filter/amplifier model is the fact that it has survived the arrival of many other synthesis methods, and that its associated vocabulary has been appropriated and applied in software; it has in many respects become a *lingua franca* for audio synthesis.

Reaktor (Native Instruments, 2003) is a good example of a software synthesizer that, amongst other things emulates and mimics a modular subtractive synthesizer. The organising principle here is one in which individual software ‘synthesizers’, or *instruments* are grouped to form an *ensemble* (see fig 6).

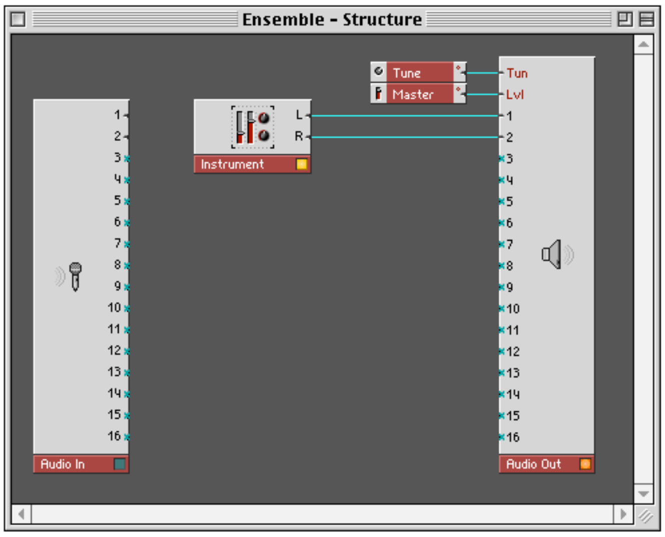

Fig 6: Grouped *Reaktor* instruments

Each instrument is made up of a number of modules, some of which may be drawn from the subtractive/FM synthesis domain (envelope generators, oscillators, etc); others may be themselves instruments. Connections between components are made by mouse-dragging, and in this way, a complex and fluid structure may be generated; one which is also recursive, in that instruments may be defined as assemblages of other instruments.

Four actions are required to initialise an instrument, two to select the waveform, and two for the pitch. The process of specifying the envelope, however, is rather more complicated, and most of the interaction is devoted to this area.

Note that the interaction involved in building an instrument is, on the face of it, one of direct manipulation; it is important to emphasise however, that the 'objects of interest' with which the user engages are not representations of the sound itself, but of the functional components required to create it.

Four actions are required to complete the voice initialisation phase, and two each to select the waveform and pitch. However, no less than fourteen actions are required to specify the required envelope.

Visually, the interface presents a clear and uncluttered view of the system. As in the hardware version, there is clear visibility of the system state at all times, and actions are reversible. The interaction is consistent throughout, (a given action will produce the same result in different contexts), and the GUI makes 'illegal' actions impossible. However, like the hardware version, the user is unable to aurally evaluate the success of his/her actions until a minimum number of connections have been made; up until this point, there will be no sound at all.

7.2 Fixed architectures - Yamaha SY35

As discussed above, the control surface of more recent hardware-based synthesizers have standardised in recent years: typically, it will consist of

- Performance controls - keyboard, pitch wheel, sliders, ribbon;
- Program or patch selection controls;
- Programming controls - allowing program parameters to be changed;
- Mode selection controls - play mode, edit mode, store mode, utility;
- Sound engine.

Limitations on the control surface space mean that a given control may have to be multi-functional, its exact usage at any given time being determined by the mode selection.

The model of sound on which the interface is built is one which has a static and essentially hierarchical structure, whose constituent parts are parameter settings defining waveforms, envelopes, filter cut-off frequencies, LFO frequencies etc, which can be modified or turned on and off. In this interface model, the task of defining or editing a sound involves the negotiation of this structure, incrementally modifying the sound by selecting and changing individual parameters. An example of such an interface is that of the Yamaha SY35.

A given instrument (or *multi*) consists of a number of voices, each of which comprises two or four waveforms (or *elements*). Parameters can be specified/modified at all levels of the structure; pressing the Edit/Utility button provides access to these parameters.

The initialisation phase requires four separate actions in all. The selection of the waveform involves cycling through the available options, using the +1/YES button, using an arrow button to locate the cursor on the correct field, and then setting the value of the selected field, using the +1/YES button again. The number of actions reqiured is at least nine, and may be more; specifying the amplitude envelope involves even more actions. Overall, the process involves far more user actions than was the case with Reaktor.

The modification of a sound can only be done incrementally. The LCD indicates no more than one parameter at a time; this means that there is no overall visibility of the system state. There is however, constant feedback available; the user is able to assess the effect of the change that s/he has made; actions are at all times reversible, and errors - ‘illegal actions’ - are impossible. Parameters are selected, and modifications effected in the same way throughout the structure.

It is interesting to consider this 'tree' structure negotiated by the user in the light of 'depth versus breadth' studies of menu structures in application software packages (Landauer & Nachbar, 1985, Norman & Chin, 1988, Kiger, 1984). Menu structures can be either 'deep', with a number of levels, or 'broad' with fewer levels, but there seems to be general agreement that users are better able to navigate 'broad' structures with no more (and preferably fewer) than four levels. The menu structure on the SY35 conforms to the 'broad' model.

7.3 Direct engagement

All the interfaces examined in the previous two sections are predicated on a model of sound as an assemblage of components which generate or modify sound; this assemblage, having been designed, is the engine which generates the required sound. The following section deals with direct engagement interfaces.

Direct manipulation is much more a feature of this mode of interaction, than was the case with *Reaktor*. The use of the term in this context requires some qualification, however. While the interaction retains some features of a direct manipulation interface, as defined in Shneiderman, 1997 (visibility of the objects of interest, incremental action at the interface, syntactic correctness of all actions), there are important limitations. Actions are not necessarily reversible - editing can be destructive (where at each edit point, the modified sound replaces the previous version) or non-destructive, where the user is able to back out of an edit step. Also, the degree of control afforded here is quite limited. The only aspect of the sound which lends itself to direct manipulation to any extent is that of amplitude (there is a clear intuitive connection to be made between the amplitude of the waveform, and its subjective loudness); but as discussed before, neither time-domain nor frequency domain waveform representation convey very much information on subjective sound colour. In short, the user still needs to formulate the directives to the system in system oriented terminology - amplitude envelopes of partials, formant frequency bands etc. Thus, the predominant feature of a direct manipulation interface - that the output expression of the object of interest can be used to formulate an input expression - applies only partially here.

In this section we review two examples of a DM synthesis interface - *Sound Sampler* and *Metasynth*.

7.3.1. Sound Sampler

Sound Sampler is a package by Alan Glenns, designed for the editing of short audio samples, and is, strictly speaking, not a synthesizer; the waveforms and signal processing facilities provided are too limited. However, it offers the user the ability, via a mouse, to directly manipulate the envelope of the sound – and, given its relative simplicity, this makes it a very useful example for illustrating several issues relevant to our concerns.

There is no initialisation phase as such; waveform and pitch parameters can be specified in a single dialog. A sound file is then generated, and displayed on screen; a further menu option makes available the following dialog (fig 7).

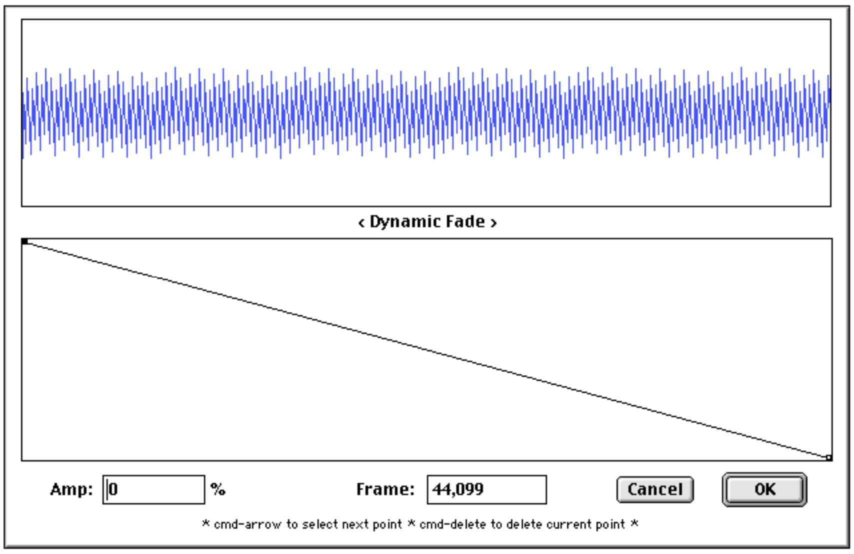
‹ Dynamic Fade ›
Amp: 0 %
Frame: 44,099
Cancel
OK
* cmd-arrow to select next point * cmd-delete to delete current point *

Fig 7: Modifying the amplitude – *Sound Sampler*

The user can then drag the ends of the horizontal line displayed below the waveform to specify amplitude; the waveform is then regenerated and redrawn.

The process of executing the test task is considerably shorter than was the case in the previous two examples - but this is to a great extent because the software itself is far less sophisticated. Waveform and pitch selection is done in one dialog - and envelope specification, using direct manipulation, is accomplished using far fewer user actions.

The task is completed using far fewer actions than was the case in the previous two examples (although the 'Select waveform' process may entail a number of actions to do with sample rate, and whether the sound is mono or stereo). In particular the 'Select envelope' phase is relatively speedily accomplished.

This is a very basic synthesizer, the simplicity of which is reflected in its screen design. However, the interaction described does exhibit the features of a good interface in that the system status (i.e. the current sound) is visible at all times, actions can be reversed, the GUI makes it difficult to make errors, and the menus make available actions visible.

7.3.2. Metasynth

Metasynth, by Eric Wenger of UI software, is a more comprehensive and sophisticated synthesizer which makes greater use of direct engagement. Notable amongst these is the Image Synth window which displays a frequency against time graph, providing the user with an image of the spectrum as it shifts in time; the amplitude of a spectral component at any moment is indicated by its brightness.

The user is provided with a palette of tools for editing this image; a new sound can be auditioned at any time by generating the sound depicted by the image. The frequency against time representation of the sound makes a perceptual mapping between the characteristics of the image and those of the sound it represents rather more apparent than is the case with a purely time domain representation.

In theory, the initial part of the test task - creating a sawtooth waveform - should not pose any great problems. The harmonics of the sound can be drawn into the window, and the amplitudes of each adjusted by the shading tools. It has to be said, however, that, in this particular program, this process is awkward and laborious.

The program does offer other ways of achieving the same end. The procedural synthesizer window enables the user to specify and graphically edit a waveform, and designate its pitch. In terms of the number of user actions required to accomplish this particular task (including the saving of the voice to a file), *Metasynth* is the most straightforward to use, requiring eleven mouse clicks in all.

As in other GUI-based synthesizers, the effect of an action is immediately apparent, both visually and aurally; actions can be reversed. The flexibility of the system, however, means that a bewildering array of controllers is presented to the user, which results in a rather cluttered appearance, and the mechanics of actually manipulating the controls is not intuitively clear. The overall envelope can be modified by the use of the buttons on the left hand panel.

8 Conclusions

In this paper, we have reviewed and classified a number of synthesizer user interfaces, focusing in particular on the 'fixed synthesis' task of manipulating timbre. A taxonomy of user interaction approaches for this domain has been identified. An important distinction is to be made between user interfaces which allow visual representations of sound to be manipulated more or less directly (usually, but not necessarily, by direct manipulation) and those interfaces that allow the manipulation of a structure or parameters of an architecture designed to generate sound. The term 'direct engagement' is proposed for the former approach. Direct engagement often involves direct manipulation; however, the defining characteristic is that the user engages with a visual representation of the sound, or some part of it. The second main approach, in which users either parameterise an existing architecture, or specify their own, has much in common with that governing industrial control processes.

For the purpose of exploring interface design issues in the domain, a heuristic evaluation of a representative group of hardware and software synthesizers has been carried out, in which the basis for comparison was the same task in each case. In terms of the number of user actions required, the interface of the Yamaha SY35 – a typical representative of the 'fixed architecture' type - emerges as the least well suited to the test task. In particular, the specification of the envelope of the sound is far more easily and intuitively achieved using graphic means, working on a direct representation of the sound in the time domain, than by typing in a sequence of numbers. It should be borne in mind, however, that the sound object created was very simple; a more complex sound would be less easy to create using the techniques characteristic of the direct engagement approach, because of the problems of representation. In fact, no one style emerges overall as being best suited to the task of sound synthesis; direct engagement is limited in the degree of control afforded, while the form filling style characteristic of fixed architecture synthesizers and generator manipulation in general is slow and laborious.

Areas which suggest themselves for possible further investigation include, firstly, the development of a common hardware interface. The user interfaces of most contemporary hardware synthesizers generally are of the 'fixed architecture' type. Given the degree of convergence that has already occurred (the generic interface already discussed), there is no reason why, in principle, a common interface could not be developed. Such an interface would further standardise the means of navigating the parameter structure and specifying parameter values, and offer a display which gave the user a more global picture of the current state of the instrument.

The other direction of potential research is the bridging of the gulf between the task and core languages of this particular domain. This is an approach that we are currently pursuing. However, we suggest that a fruitful line of enquiry would be the examination of the cognitive processes and working methods of users engaged in creating and editing sounds, in order to illuminate and guide the designing and building of a user interface which reflects and facilitates these processes.

References

Dix A., Finlay J., Abowd G. and Beale R., *Human-Computer Interaction* Prentice Hall, 1998

Ethington R. and Punch B., SeaWave: A System for Musical Timbre Description *Computer Music Journal* 18:1 pp 30-39, 1994

Fernandes G. and Holmes, C., Applying HCI to Music-Related Hardware, CHI 2002

Karkoschka E., *Notation in New Music*, Universal Edition, 1966

Kiger J. I., The depth/breadth trade-off in the design of menu driven user interfaces *International Journal of Man-Machine Studies*, 20, 1984

Landauer T. K. and Nachbar D. W., Selection from alphabetic and numeric menu trees using a touch screen: Breadth, depth and width *Proc. CHI'85, Human Factors in Computing Systems, ACM*, New York, 1985

Miranda E. R., An Artificial Intelligence Approach to Sound Design, *Computer Music Journal* 19(2): 59-75, MIT Press, 1995

Nielsen J., Heuristic Evaluation, in Nielsen, J and Mack R. L (eds), *Usability Inspection Methods*, Wiley, 1994

Norman K. L. and Chin J. P., The effect of tree structure on search in a hierarchical menu selection system, *Behaviour and Information Technology*, 7, 1988

Polfreman R, Sapsford-Francis J., A Human-Factors Approach to Computer Music Systems User-Interface Design Proceedings of the 1995 International Computer Music Conference ICMA, 1995

Pressing J., *Synthesizer Performance and Real-Time Techniques*, Oxford University Press, 1992

Rolland P-Y. and Pachet F., A Framework for Representing Knowledge about Synthesizer Programming, *Computer Music Journal* 20(3): 47-58 MIT Press, 1996

Ruffner J. W. and Coker G. W., A Comparative Evaluation of the Electronic Keyboard Synthesizer User Interface *Proceedings of the Human Factors Society 34th Annual Meeting* Human Factors Society, 1990

Shneiderman B., *Designing the User-Interface: Strategies for Effective Human-Computer Interaction* Reading, Massachusetts: Addison-Wesley, 1997

Smith J. O., Physical modeling using digital waveguides, *Computer Music Journal*, vol. 16, no. 4, pp. 74-91, 1992

Thimbleby H., The Computer Science of Everyday Things *Proceedings of the Australasian User Interface Conference,* 2001

Vertegaal R and Bonis E., ISEE: An Intuitive Sound Editing Environment *Computer Music Journal* 18(2): 21-29, MIT Press, 1994

Williges R., C Williges B. H. and Elkerton J., Software Interface Design, in Salvendy G (Ed) *Handbook of Human Factors* New York: John Wiley & Sons, 1987

The CM Guide to FM Synthesis, Computer Music, http://www.computermusic.co.uk/tutorial/fm/fm1.asp, accessed 19/1/2004

Native Instruments, www.native-instruments.com, accessed 4/12/03