A CRITICAL ANALYSIS OF SYNTHESIZER USER INTERFACES FOR TIMBRE

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

ABSTRACT

In this paper, we review and analyse some categories of user interface for hardware and software electronic music synthesizers. Problems with the user specification and modification of timbre are discussed. Three principal types of user interface for controlling timbre are distinguished. A problem common to all three categories is identified: that the core language of each category has no well-defined mapping onto the task languages of subjective timbre categories used by musicians.

Keywords

Music, Synthesis, Synthesizers, Timbre, Semantic Directness, Usability.

1. INTRODUCTION

This paper analyses representative user interfaces for specifying and controlling timbre in electronic music synthesizers. Relevant taxonomies, design issues and problems for interface design in this domain are identified. We characterise an underlying problem for all categories of interface analysed. Some possible future directions for addressing the problems are proposed.

The user interfaces of audio hardware and software generally, and of music synthesizers in particular, have received relatively little study within HCI. An analysis conducted on the working methods of composers working with Computer Music Systems (CMS) [7] identified various typical tasks, and concluded that CMS designers must allow for wide variations in composers' knowledge and skill and wide individual variation in the types of composer they are designing for. Recommendations included: providing more than one level of interaction; hiding unwanted levels of complexity; and employing knowledge based systems (KBS) to manage details that a user does not wish to specify directly. A previous critique of synthesizer user interface design [10] 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* [15]. It was concluded that the demands placed on the user by the interfaces meant that they were far from ideal for the purpose: noting that, in general, 'user interface principles have been, at best haphazardly applied'. The authors also suggested issues that should drive future research in this area. Another more recent related study [4] has applied a heuristic evaluation to an electric guitar pre-amplifier interface. The present paper examines a number of categories of user interface for controlling timbre, taking commercial software and hardware synthesizers as examples.

This paper appeared as: Seago, Allan; Holland, Simon and Mulholland, Paul (2004). A Critical Analysis of Synthesizer User Interfaces for Timbre. In: Dearden, Andy and Watt, Leon Eds. Proceedings of the XVIII British HCI Group Annual Conference HCI 2004. Bristol, UK: Research Press International, pp. 105–108.

2. BACKGROUND

While the range of tools and techniques available to the musician for the design and editing of sound is very large, usability in modern synthesizers is generally poor [5,8,3]. Thimbleby's example of the design of electronic calculators [14] is of relevance here. He notes that the hand held calculator is a 'mature technology', with well defined requirements, but 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 degree converged, to the extent that we can consider the instrument to have acquired a generic interface [8]. However, one cannot assume that similar looking buttons will perform the same function. Conversely, a given function could be performed by diverse different controls.

The range of tasks that must be performed by a synthesizer is both broader and less easily defined than the range performed by a calculator. Poor usability has led to a situation where most users seem to have limited their choices of timbre to selections from a bank of preset timbres - evidence for this is largely anecdotal, but "allegedly, nine out of ten DX7s coming into workshops for servicing still had their factory presets intact" [1].

Over the last few years, hardware synthesizer functionality has increasingly been migrating into software (*Reaktor*, *Reason* etc). This development has potentially freed designers from the constraints imposed by hardware

limitations: particularly from the limited space available for controllers and displays, but also from cost constraints of hardware controls. Yet, software designers have sought to emulate hardware synthesizers not only in models of synthesis – how the sounds are generated - but also in the user interface. Thus, the user is presented on screen with a simulation of a synthesizer hardware control surface, and must control it via virtual buttons, faders and rotary dials that mimic the hardware they have replaced. For many users, this has the virtue of familiarity; but it tends to impose unnecessary usability problems.

Pressing [8] describes the controls of the synthesizer user 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, such as pitch wheels, foot pedals and the keyboard, allow instant and dynamic modification of single scalar aspects of existing sounds: pitch, filter frequency, volume etc. These controllers are designed and positioned on the control surface to meet the requirements of real-time performance, and it is relatively easy for users to understand their use: the effect that a controller has on the sound is instantly audible. Real time controls will not be considered further here.

The part of the interface devoted to 'fixed synthesis is the focus of the current study. In fact, as we will see in the next section, the term 'fixed' is something of a misnomer, since in many cases, the control of timbre is achieved by wide-ranging modifications of this element. A more suitable term might be 'relatively fixed'; however we will retain Pressing's terminology, while noting any resulting ambiguities.

The 'fixed synthesis' component of the interface allows the design and programming of sound objects. Its informed use typically requires an in-depth understanding of the internal architecture of the instrument, and the methods used to represent and to generate sound. Thus, under most current systems, the user is obliged to express directives for sound specification in system terminology, rather than in language derived from the user domain.

3. TASK AND CORE LANGUAGES

There is a considerable gulf between the *task languages* and the *core languages* [2] in synthesizer interfaces. *Task language* terms like *shrill*, *spacious*, *dark*, *grainy* etc are among those typically used by musicians to describe those attributes of sound - timbre, texture and articulation - which cannot be captured well by conventional musical notation. These terms are often chosen for their perceived analogies with other domains: colour and texture, for example, or for emotional associations. The vocabulary of the *core languages*, by contrast, refers to objective and measurable quantities associated with sound, such as spectral distribution and density, and their evolution over time. The problem is to map one set of descriptors onto the other. The bridging of the gulf between task and core language in sound synthesis user interfaces has been approached in diverse ways: using techniques from artificial intelligence [5], knowledge based systems [3,9] and by the embodiment of metaphors derived from acoustical mechanisms [13].

4. USER INTERFACE ARCHITECTURES

In this section, we will describe the three most common core languages used in controlling timbre in synthesizers. In approximate order of the complexity of associated user interface issues, (though not necessarily their complexity from other perspectives) they are as follows.

- Parameter selection in a fixed architecture,
- Architecture specification and configuration,
- Direct specification of physical characteristics of sound

For purposes of exposition, and reflecting historical trends, it is useful to begin with the second of these approaches first: Architecture Specification and Configuration, also known as User Specified Architecture. This approach to specifying timbre has its origin in the interfaces of early synthesizers, such as the Arp, Moog and EMS. In such early synthesizers, a given sound was defined in terms of the configuration of electronic modules required to generate it. The hardware interface offered total control over the choice, interconnection, and settings of these modules via physical plugboards. Modern versions of this idea use GUI based interfaces to accomplish similar ends.

The approach appearing first in the list above (Parameter Selection, also known as 'Fixed Architecture') came next historically. This approach effectively froze selected configurations of modules and simply allowed the user to vary the values of parameters controlling these modules. Different synthesizers may use quite different sound synthesis modules from each other, but the principle remains the same. Thus, fixed architectures present to the user an internal model of sound which is essentially a tree or graph structured assemblage of parameters. For the user, the task of defining a sound is one of traversing this structure, specifying parameters e.g. by a 'form filling' process. The earlier mentioned user specified architectures, by contrast, are essentially fluid and non-hierarchical. We will revisit both types below.

Finally, the third category of user interface for timbre control in synthesizers is Direct Specification. This was first widely introduced commercially in early Fairlight synthesizers. This category allows the user, in principle, to specify sound directly by, for example, drawing or modifying a waveform on the screen. This category will be described in much greater detail below.

In the next three subsections, will consider each of the three categories in more detail, describing modern interfaces from each category. We will draw on a series of user tests comparing the categories [11].

4.1 Fixed Architecture

As noted above, the 'fixed synthesis' control surfaces of more recent hardware-based synthesizers (recall that 'fixed synthesis' does not mean 'fixed architecture') have standardised in recent years. Typically, there are selection controls for preformatted sounds (known as 'programs' or 'patches'), programming controls (to change program parameters) and mode selection controls (play, edit, etc). Limitations on control surface space mean that controls may be multi-functional: their usage at any given time will be determined by the mode currently selected.

The model of sound generation used in interfaces of this category has a static and hierarchical structure, whose constituent parts are parameter settings defining waveforms, envelopes, filter cut-off frequencies, etc. The task of defining or editing a sound involves the traversal 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. The LCD indicates no more than one parameter at a time, providing no overall visibility of the system state. However, since all parameters have default values, instant feedback is available simply by listening to the current sound; the user is able to assess the effect of the changes made; actions are at all times reversible, and errors or 'illegal actions' are impossible. Parameters are selected, and modifications effected, in the same way throughout the structure.

4.2 User Specified Architecture

In this architecture, sound is viewed as the output of a network of functional components - oscillators, filters, and amplifiers. The structure of this network is fluid, and can become quite complex. The output of any element may be processed by one or more other elements. However, even greater fluidity comes from the fact that the parameters of each element, frequency, envelope and cut-off frequency, etc, can be dynamically controlled by the output of any other element. As already noted, early subtractive synthesizers were in this category; the basic components were linked by physical patch cords, and the signal path was visible and immediately modifiable.

In hardware synthesizers, the range of sound that can be produced is limited by the number of hardware modules available. Software versions, however, in important respects, have no such restrictions. One striking aspect of the oscillator/filter/amplifier synthesis model associated with subtractive synthesis 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. (In the user study reported in [11], a number of users were clearly confused by the apparent absence of these modules in an interface which simply named them differently).

Reaktor [6] is a good example of a synthesizer that emulates and mimics in software a modular subtractive synthesizer. Each instrument is made up of a number of modules drawn from the subtractive/FM synthesis domain (envelope generators, oscillators, etc). Connections between components are made by mouse dragging. In this way, a complex and fluid structure may be generated recursively, in the sense that instruments may be defined as assemblages of other instruments. The interaction style used to build an instrument is 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. As in the hardware equivalents, 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 DM style makes 'illegal' actions impossible. However, as with the hardware equivalents, the user is inherently 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.

4.3 Direct Specification

All the user 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 interfaces that allow the desired sound to be specified more directly.

Visual representation of sonic information is usually in either the **time domain** (essentially a plot of the waveform), or the **frequency domain (**a plot of the relative amplitudes of the frequency components of a waveform). The interpretation of time domain plots is, to some extent, intuitively clear. In principle, this *output expression* of the system is capable of being used to formulate an *input expression* in a manner characteristic of direct manipulation systems [2] - in this case, by the provision of tools to 'draw' and 'edit' the desired waveform. However, a user interface for 'designing' sounds in any detail in this way is hampered by the lack of any human understandable mapping between the subjective and perceptual characteristics of the sound in any detail and its visual representation on screen. In practice, no user is able to specify finely the waveform of imagined sounds in general. In other words, there is no *semantic directness* for the purpose of specifying any but the most crudely characterized sounds. The gap between core language and task language is just as wide as in the first two categories. To make the discussion more concrete, we will consider a system of this category as studied in [11].

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 illustrates our concerns well, and offers the user the ability, to directly manipulate the envelope of the sound, by dragging the ends

of the horizontal line displayed below the waveform to specify amplitude; the waveform is then regenerated and redrawn. This interaction exhibits 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.

As with *Reaktor*, this is a direct manipulation interface. The use of the term requires some qualification, however. Specifically, while the interaction in Sound Sampler retains some features of direct manipulation (visibility of the objects of interest, incremental action at the interface, syntactic correctness of all actions), there are important restrictions. Actions are not necessarily reversible: editing may be destructive (at each edit point, the modified sound replaces the previous version). Also, the degree of control afforded is quite limited. As noted earlier in outline, 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 between the amplitude of the waveform on the display, and its subjective loudness; but as indicated before, conventional waveform representations do not convey very much information on subjective sound colour. Thus, the user still needs to formulate the directives to the system in system-oriented terminology: amplitude envelopes, formant frequency bands etc. Thus, a characteristic 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.

Comments from users who were asked, in a series of user tests [11], to compare the interface of a 'Fixed Architecture' synthesizer with that of one which incorporated elements of Direct Specification revealed a unanimous preference for the latter.

4.4 Other Types of User Interface

The taxonomy of user interfaces for timbral control in synthesizers identified above is not exhaustive. However, the main other kinds of interface add little, if anything, of principle to our argument. One such category, noted earlier, controls a kind of synthesis called physical modelling [3]. This involves simulating, in software, physical systems such as stretched strings. Although the mental model of synthesis is quite different from those we have considered, from an interaction perspective, the resultant user interfaces are generally just examples of the parameter selection interfaces of section 4.1, or variations of those discussed in section 4.2. In any case, the vast majority of users do not have the specialized knowledge to be able to map from physical systems to timbre, consequently the arguments of previous sections apply with similar force.

5. CONCLUSIONS

In this paper, we have analysed various user interfaces for synthesizer timbre and identified a taxonomy of common user interface types for this domain. A distinction is made between user interfaces which allow visual representations of sound to be manipulated more or less directly and those that allow the manipulation of an architectural structure, or the parameters of such an architecture, which generates the sound. None of the core languages involved have been found to map appropriately to the task language of the musician.

Further work will look at how the chasm between the musician's task language and the available approaches can be bridged. Issues to be addressed in further work include:

- Empirical studies of timbre perception,
- Evolutionary design user interfaces for timbre,
- Empirical studies of how musicians describe timbres.

Other areas which suggest themselves for possible further investigation include, firstly, the development of a 'lingua franca' common 'fixed architecture' hardware interface: given the degree of convergence that has already occurred, this would appear to be feasible. More generally, we propose the examination of the cognitive processes and working methods of musicians engaged in creating and editing sounds, in order to guide the design of user interfaces which reflect and facilitate these processes. Any adequate solution will need to address the gulf between task and core language analysed above.

6. REFERENCES

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

[2] Dix A., Finlay J., Abowd G. and Beale R. (1998). Human-Computer Interaction. Prentice Hall.

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

[4] Fernandes G. and Holmes, C. (2002) Applying HCI to Music-Related Hardware. CHI 2002.

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

[6] Native Instruments, www.native-instruments.com

[7] Polfreman R and Sapsford-Francis J. (1995) A Human-Factors Approach to Computer Music Systems User-Interface Design. Proc. of the 1995 International Computer Music Conference ICMA.

[8] Pressing J. (1992) Synthesizer Performance and Real-Time Techniques. Oxford University Press.

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

[10] Ruffner J. W. and Coker G. W. (1990) A Comparative Evaluation of the Electronic Keyboard Synthesizer User Interface, Proc. 34th Annual Meeting Human Factors Society.

[11] Seago A. (2004). Analysis of the synthesizer user interface: cognitive walkthrough and user tests. TR2004/15, Dept of Computing, Open University.

[12] Shneiderman B. (1997). Designing the User-Interface: Strategies for Effective Human-Computer Interaction. Reading, Mass: Addison-Wesley.

[13] Smith J. O. (1992). Physical modeling using digital waveguides. Computer Music Journal, vol. 16 no. 4, 74-91.

[14] Thimbleby H. (2001). The Computer Science of Everyday Things. Proceedings of the Australasian User Interface Conference.

[15] Williges R., C Williges B. H. and Elkerton J. (1987). Software Interface Design. In Salvendy G (ed) Handbook of Human Factors. New York: Wiley.