Cognitive Dimensions and Musical Notation Systems

Alan F. Blackwell
Computer Laboratory
Cambridge University
Alan.Blackwell@cl.cam.ac.uk

Thomas R.G. Green
Computer Based Learning Unit
University of Leeds
Thomas.Green@ndirect.co.uk

Douglas J.E. Nunn
Audio and Music Technology
Anglia Polytechnic University
D.J.E.Nunn@anglia.ac.uk

Introduction

This paper approaches music notation as one example of a more general class of notation systems. Our work on notation systems was originally motivated by the need to understand the factors affecting the usability of programming languages (Blackwell 1996, Green & Petre 1996). Since then we have considered an increasingly wide range of usability problems, some of which would not normally be considered as notations, let alone programming languages (Green & Blackwell 1998, Blackwell & Green 1999). A recent study of users of a music typesetting package (Blackwell & Green 2000) has convinced us that music notation does indeed share the characteristics of the other notations that we have investigated. This paper presents our case for a new approach to the usability of music notation, using the results of that study as supporting evidence.

Notational systems: definition

Our research is concerned with the generic class of *notational systems*. A notational system includes two main elements: a notation, and an *environment* (tools, editors) for manipulating that notation. *Notations* are information-carrying representations. They usually have a set of rules for constructing them and interpreting them. We recognise many kinds, including mathematical expressions, diagrams, maps, timetables, human languages and of course music notation.

The *environment* for manipulating a notation can be a computer editor, but also includes ordinary physical tools or materials. Pencil and paper, along with many accessories (erasers, scissors, paste, rulers …) are a familiar environment that can be used to manipulate a wide range of notations. Even pencil and paper notational systems have many interesting properties – for example the creative options available to a composer may be significantly altered if he or she has left home without an eraser. The effect of computer editors is far more interesting. In some cases, a notation and an editor for creating it evolve together. In other cases, the notation is given by cultural context, and the challenge is to create an editor that will assist users to manipulate the notation effectively. Both phenomena have been observed in the development of music notation software. Both may have radical effects on the potential offered to musicians by the software. The analysis of these effects might broadly come under the heading of “usability”, though we really go much further than traditional approaches to assessing usability.

Assessing the usability of an editor

There are clearly a huge range of musical technologies that fall within our definition of notational systems. Not only conventional staff notation and typesetting software, but chord names, tablature, sequencer charts, MIDI patch controls, waveform editors, and the playlists in an MP3 player are all musical notations. They are aimed at very different classes of user, but all of them can be used for making music. How can users choose a notational system that meets their needs, and how can designers create new alternatives or improve the systems that they build?

Conventional evaluation techniques in Human Computer Interaction (HCI) are aimed at clearly-defined tasks, such as ‘delete a word’. Faced with music, they fall short. Composition is not a clearly-defined task; it requires sketching out possibilities, additions, deletions, firming up sketchy ideas, building up skeletons and so on. Moreover, staff notation, like a programming language, contains many subtleties. Frequently, one passage must echo another while being different (a theme returning in a modified form); a musical structure must be adhered to; dependencies between the parts (harmonic or rhythmic) must be observed; and so on.

Presented at ICMC 2000, Berlin: Workshop on Notation and Music Information Retrieval in the Computer Age

In these respects, music notation has much in common with a programming language. Most HCI methods devised for evaluating word-processors fall short when it comes to evaluating programming environments in exactly the same way that they fall short in evaluating music editors. You wouldn't assess the programmer's editor EMACS in the same way as you'd assess a simple word-processor, because it has a different purpose and it works on different material.

However, there is one HCI evaluation method that is specifically intended to evaluate systems such as programming languages; that is, systems in which creative design is an important activity and in which complex structures are being built and modified. That system is called the 'Cognitive Dimensions' framework. First described by Green (1989) in rather general terms, it was applied to a sample of visual programming environments by Green and Petre (1995) and has subsequently been applied to a wide range of usability problems (Green & Blackwell 1998). We describe here how that system can be applied to musical editing systems.

Cognitive Dimensions in a Nutshell

Aim: Cognitive Dimensions (CDs) do not offer highly detailed analyses, predictions of time taken for simple tasks, etc. Instead, they offer quick analyses that relate the typical user's activity, the structure of the notation, and the design of the editing system. These analyses can reveal mismatches which can (and frequently do) provoke designers into further thought about improvements, as well as providing a vocabulary with which users can identify and express usability concerns. The motto is: quick but provocative.

Procedure: The first step is to consider the typical user's activities at a generic level. We distinguish 6 activities, among them: transcription (e.g. writing out a score in a different key); search (e.g. finding all the occurrences of a theme or motif); and creative design. Each of these activities has a desirable profile of cognitive dimensions, which we consider next.

Cognitive Dimensions: The dimensions are the nub of this approach but we shall not go through them in detail. Each dimension relates to the notation (programming language, staff notation, lute tablature, or whatever) *as operated on by the editor*. The same notation may appear to have very different usability characteristics when viewed in one editor or another.

To get an idea of how they work, here are some example dimensions (A full description, with multiple examples and applications, is provided in Green & Blackwell 1998):

> **Viscosity** is the work required in making a local change to the notation. Viscosity depends on the notation: usually easy in a programming language like Basic (at a cost), sometimes hard in Modula (depending on the change). Viscosity also depends on the editor: smart editors like EMACS can reduce viscosity, but at a cost in abstractions (see below).
>
> **Hidden dependencies** are where changing one thing surreptitiously changes another. This is widespread in textual programming languages because expressions import values, types, data structures, etc. Graphical programming languages can use data flow lines to make dependencies explicit, but at a cost in diffuseness (see below).
>
> **Premature commitment** occurs when users have to make their minds up too early: what identifiers does this program need? Frequently a problem when writing a program with pen and paper. Text editors reduce viscosity (they allow afterthoughts more easily than paper does) so premature commitment is less of a problem.

Diffuseness is a measure of how much or little can be said in a few word or symbols. If a notation is too terse, it encourages slips of the pen; if too diffuse, it takes too long to get anything done. Diffuseness can be reduced by abstractions (see below).

```
000230-GENERATE-KEY.
       *
       *  ** NUMBERS 214031, 2531017 AND 999999937 ARE PRIME **
       COMPUTE RNG-SEED = RNG-SEED * 214031 + 2531017.
       DIVIDE RNG-SEED BY 999999937 GIVING RNG-TEMP REMAINDER RNG-SEED.
       MOVE RNG-SEED TO ST-ENT-FLD1(ST-I).
000230-EXIT.
       EXIT.
```

Extract of Cobol source code for Random Number Generator

```
void GenerateKey(void)
{
RngSeed = RngSeed*214031+2531017;
RngSeed %= 999999937;
StEntFld1[st-i] = RngSeed;
}
```

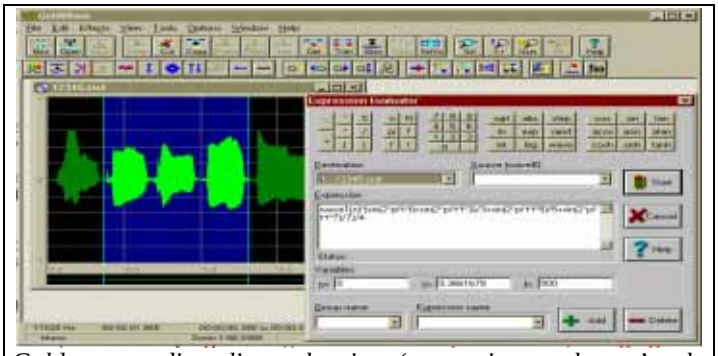
Equivalent C source code

Secondary notation provides a way for the user to add extra information that is not recognised as part of the formal structure, as with comment lines in a program. Almost all programming languages provide some sort of comment facility nowadays, but designers of other notational systems sometimes forget how useful this is.

Abstractions wrap up several symbols into one, or do other similar jobs. They are ways to change the fundamental notation. They can reduce diffuseness and increase clarity but they are potentially hard to understand. Moreover, choosing the appropriate abstractions can itself be a problem (premature commitment); changing them can difficult (viscosity); and changing one abstraction can lead to unforeseen changes to other ones (hidden dependencies).

This is a reasonably representative sample, but there are many more – 13 (or so), so we won't go through them here, let alone describe them in detail.

Subdevices: Abstractions are created, organised, named or maintained using some kind of abstraction manager. Sometimes this is a sophisticated sub-device such as a library system, sometimes it's rudimentary, sometimes it is handled in the same environment as the main notation; but however it is handled, it is a sub-device. Another kind of sub-device is a viewer, allowing the program structure to be displayed in a different form (e.g. a cross-referencer). Sub-devices have their set of values on the dimensions and are used for their own activities, so they must be evaluated separately.



Goldwave audio editor, showing 'expression evaluator' sub-device with entirely different notation.

Idealised Profiles: for a given activity, there will be an idealised set of requirements. For transcription, viscosity is not important, nor are hidden dependencies. For creative design, they matter much more.

Evaluation proceeds by relating the profile of a given system to the idealised profile for the typical user's activity. Not only the main system but also the sub-devices need to be evaluated. (E.g. Word is quite a good word processor but the manager for defining and maintaining style definitions is very hard to use.)

Remember that **no dimension is good or bad on its own**. It can only be evaluated when the user's activities are known. Furthermore there is no perfect system for all purposes – changing one dimension often has resulting trade-off effects on others. The dimensions describe the potential design options, but are not a panacea for usability.

Cognitive Dimensions and Music Systems

Common Practice Notation with pencil and paper has high *viscosity* (editing requires an eraser), a few *abstractions* (such as key signature, ornaments, and tablature), a few *hidden dependencies* (changing a key signature changes the notes), some *premature commitment* (a new instrument cannot be added in its logical position unless a blank stave was left), and is quite *diffuse* (no way to write "play broken chords in a simple sequence until the singer starts"). Tonic sol-fa reduces the *viscosity* in some ways (easy to change the key – which reduces the cost of the *premature commitment*).

Let us now consider a generic software sequencer. Sequencers offer several views, usually including a 'piano-roll', staff notation, and an event list; all of these can be edited.

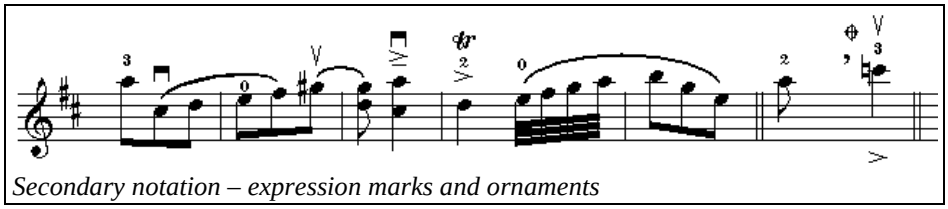
In general, *viscosity* is low – as there are multiple views of the same data, it can be changed easily in the most appropriate view. However, in some sequencers, the piano-roll notation may not show the note velocities. Even recent sequencers usually show them separately. Typically the interval between a note offset and the next onset cannot be displayed or edited directly.

There are few *hidden dependencies*. However, controller messages on one track may affect how other tracks play. Some sequencers allow a block of data to be repeated (e.g. using 'ghost parts' in Cubase), in which case editing the data affects all the copies of it.

Premature commitment is low, as the work can be built in any order, although where audio files are involved, we must typically first choose whether the work is destined for 44100 or 48000 Hz.

Usually *diffuseness* is low, as the MIDI data displayed is a fairly concise description. Arguably, though, a crescendo mark would be much briefer than a series of controller changes.

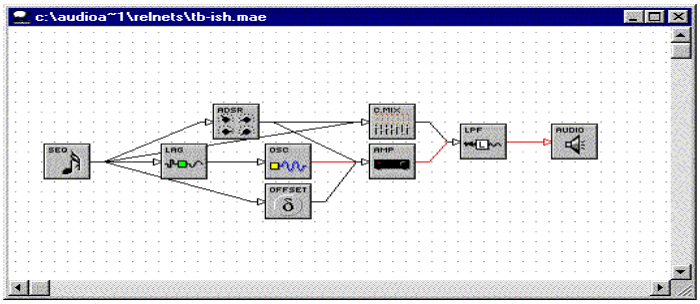
The printed score may include *secondary notation*, such as expression or ornament marks. An ideal music system might recognise the meaning of these marks, and modify the performance appropriately, but here they are like comments in a program: meaningful to the human reader, but the readers interpretation of the printed notation differs from the MIDI data to be played.



Secondary notation – expression marks and ornaments

A typical sequencer usually has many *sub-devices*, each with their own methods of control. These include dialogue boxes, software mixers, plug-in effects and synthesisers, 'in-line' MIDI processors, and so on.

The CD framework also applies to other programs. Let's compare two software synthesisers, Csound and Audio Architect. Csound, like other acoustic compilers, uses a very terse description. Audio Architect has a graphical interface that reveals some otherwise *hidden dependencies*, but relies on Windows dialogue boxes for data entry, thus reducing *visibility*.



Audio Architect

```
w 0 68
f 1 0 0 512 512 10 15 10 6 9 4 5 2 8 3 2 1 3
f 2 0 0 513 513 5 0.001 512 1
f 3 0 0 512 512 10 15 12 8 9 4 5 3 4
f 0 1 .882353
s
w 0 60
i 1   0    0    0.5  0.5  9.04 3400 0.02 0.03
i 2   0    0    0.5  0.5  9.01 3400 0.02 0.03
i 3   0    0    1    1    8.09 3400 0.02 0.03
i 4   0    0    0.5  0.5  8.01 3400 0.02 0.03
i 5   0    0    0.5  0.5  7.09 3400 0.02 0.03
i 7   0    0    1    1    7.09 3400 0.02 0.03
i 8   0    0    1    1    6.09 3800 0.02 0.03
(etc.)
```

```
        sr = 22050
        kr = 441
        ksmps = 50
        nchnls = 1
                        ;p4 amps here doubled
; guitar
        instr 1
kamp    linseg   0.0, 0.015, p4*2, p3-0.065, p4*2, 0.05, 0.0
asig    pluck    kamp, p5, p5, 0, 1
af1     reson    asig, 110, 80
af2     reson    asig, 220, 100
af3     reson    asig, 440, 80
aout    balance 0.6*af1+af2+0.6*af3+0.4*asig, asig
        out      aout
        endin
```

Csound score and orchestra files

Comparison of Programming Notations and Music Notation

In order to test our hypothesis that the usability of music notation systems can be assessed in the same way as the usability of programming languages, we constructed a questionnaire that allowed system users to consider the usability of their systems in the terms offered by the Cognitive Dimensions framework. The questionnaire approach has important advantages over methods such as watching users, analysing keystrokes, or analysing audio/video recordings of computer use, because it allows experienced users directly to convey their own experiences of the software.

We gave this questionnaire to eight programmers and eight music academics, and compared their responses to see whether their usability concerns were similar (Blackwell & Green 2000). The programmers described their experiences with the usability of a range of programming systems, while the music academics considered music typesetting packages. Both programmers and music academics were able to criticise the systems they use in response to a reasonably concise description of the cognitive dimensions, and the descriptions were mostly acceptably clear. In fact music academics were more productive than programmers in their ability to reflect on the potential modifications that could be made to their tools. This is very encouraging. The detailed results of that study are available in the paper referenced below, which can be found online. The same questionnaire will be distributed at this ICMC and results presented in a future paper.

Acknowledgements

Alan Blackwell's research is funded by the Engineering and Physical Sciences Research Council under EPSRC grant GR/M16924 "New paradigms for visual interaction".

References

Blackwell, A.F. & Green, T.R.G. (2000). A Cognitive Dimensions questionnaire optimised for users. In A.F. Blackwell & E. Bilotta (Eds.) Proceedings of the Twelth Annual Meeting of the Psychology of Programming Interest Group, 137-152

Blackwell, A.F. & Green, T.R.G. (1999). Investment of attention as an analytic approach to Cognitive Dimensions. In T.R. G. Green, R. Abdullah & P. Brna (Eds.) Collected Papers of the 11th Annual Workshop of the Psychology of Programming Interest Group (PPIG-11), pp. 24-35.

Blackwell, A.F. (1996). Metacognitive theories of visual programming: What do we think we are doing? Proceedings IEEE Symposium on Visual Languages. Los Alamitos, CA: IEEE Computer Society Press, pp. 240-246.

Green, T.R.G. & Blackwell, A. F. (1998). Cognitive Dimensions of information artefacts: a tutorial. Version 1.2, October 1998. (revision of “Cognitive Dimensions of notations and other Information Artefacts” at HCI’98) Available at http://www.ndirect.co.uk/~thomas.green/workStuff/Papers/

Green, T.R.G. & Petre, M. (1996). Usability analysis of visual programming environments: a 'cognitive dimensions' approach. Journal of Visual Languages and Computing, 7,131-174.

Green, T.R.G. (1989). Cognitive dimensions of notations. In A. Sutcliffe & L. Macaulay (Eds.), People and Computers V. Cambridge University Press.

Several of these publications are available online, from:
http://www.cl.cam.ac.uk/~afb21/publications/index.html