A TAXONOMY OF SEQUENCER USER-INTERFACES

Matthew Duignan, James Noble
School of Mathematics, Statistics & Computer Science
Victoria University of Wellington, New Zealand
Matthew.Duignan@mcs.vuw.ac.nz

Robert Biddle
Human Oriented Technology Laboratory
Carleton University, Ottawa, Canada
Robert_Biddle@carleton.ca

ABSTRACT

Sequencing tools play a central role in our ability to create computer music. Despite their importance, there has been little structured analysis of how the characteristics of sequencers impact our ability to use them effectively. This paper addresses this through a new taxonomy for classifying sequencing tools. This taxonomy can be used to help us better understand the strengths and weaknesses of current sequencer tools, and suggest novel possibilities for future sequencers.

1. INTRODUCTION

Using computers to sequence music has a long and rich history that has resulted in the development of a vast number of different sequencing tools. In this paper we use the term “sequencer” to refer to all computer tools used to sequence music, not just those conventionally described as “sequencers”. These tools have been developed in both academic and commercial contexts and each have different strengths and weaknesses. While much work has focused on conceiving new sequencing applications there is a need for a high level analysis examining the major characteristics of such applications and how these affect their use.

In this paper we present a new high-level taxonomy for sequencing software. This taxonomy consists of five axes which cleanly categorise each of the four major classes of sequencer in common use today. This taxonomy can be used to highlight and explain the relative strengths and weaknesses of various sequencer classes. Additionally, our taxonomy lays out a landscape of actual and possible sequencer types, only some of which have been explored. In this way, the taxonomy can suggest new forms of sequencers that could complement existing sequencer tools. This taxonomy can be used by sequencer designers who wish to gain insight into the impacts of high level interface decisions on the resulting system.

2. TAXONOMY

This section presents the taxonomy we have developed which categorises computer sequencing tools. The aim of this taxonomy is to present an analytical framework for understanding the characteristics of sequencer interfaces. This taxonomy can be used to classify and analyse any music tool with a sequencing component, including systems that have synthesis or effects processing features in addition to sequencing abilities. In these cases the classification is based solely on the sequencing characteristics.

2.1. Four common approaches to sequencing

As a starting point, our taxonomy must distinguish between the major groups of sequencing applications available today. The earliest sequencer applications were extensions and developments of programming systems built in the computer languages of the time. These evolved into special purpose textual based music programming languages such the MusicN languages, and SuperCollider, and also into extensions for existing programming languages such as Siren in the Smalltalk language. All of these tools can be described as *textual language music tools*.

At the same time, drum-machines and early analogue sequencers have continued to develop, and these interfaces have shaped sampling and loop playing tools, for both modern drum-machines, and hardware sequencers such as the MPC2000. This group includes all manner of pattern sequencers. We refer to this group as *sample and loop triggers*.

Many of the concepts and interaction patterns from textual language music tools soon found their way into the graphical domain in the guise of what we refer to as *music visual programming* tools. These tools such as MAX are often classic examples of the more general class of visual programming languages, but are designed to be particularly suited to musical applications.

The fourth common class of sequencer are what we will refer to as *linear sequencers*. These tools have become the high-profile face of sequencing, typically utilising a strong multi-track recorder user-interface metaphor [1]. These tools evolved from the recording studio which has increasingly become a location for composition and experimentation, in addition to the straight recording of musical performance. Modern linear sequencers such as Pro Tools and Logic allow the user to sequence both recorded, sampled and synthesised sound.

Now that we have identified the four types of common sequencers, we can now itemise each of the five axes of our taxonomy.

2.2. Textual vs. Graphical

The first major distinction in this taxonomy is between textual and graphical computer music systems. Textual computer music systems are essentially programming languages, often providing built in types to support computer music specific functionality. Graphical computer music systems utilise a combination of geometric shapes, colours and representations, along with some textual elements. Graphical systems are characterised by a far greater emphasis on non-textual representations.

Due to the expressive power of language, textual music programming tools tend to be highly flexible within their domain. In this environment music programmers are able to define their own structures, abstractions, and control flow logic. These environments offer a great deal of power to the user, but this comes at a significant cost in terms of learnability and usability to those who are unfamiliar with complex programming concepts.

A principle characteristic of graphical user-interfaces is the interaction style of *direct manipulation* described by Shneiderman [5]. Direct manipulation uses graphic approaches where users interact by simulated physical manipulation of interactive graphical objects. Shneiderman listed the benefits of direct manipulation systems which included faster learning and retention rates, rapid feedback, and the encouragement of exploration.

Graphical computer music tools provide a more direct and typically more learnable interface. Because interaction objects and interaction possibilities are made visible, users will often be able to intuit usage through direct manipulation techniques. These interactive interfaces are often markedly simpler to learn, but typically offer little of the flexibility and power of textual systems.

2.3. Predetermined vs. Custom Abstractions

Computer music sequencers can be categorised in the extent to which they provide predetermined or user-customisable abstractions. Abstraction is the general process of using classification, grouping, and hierarchy to hide and reuse details. All but the most primitive of sequencer user interfaces provide some form of basic predetermined abstraction whether it is *notes*, or *audio clips*. These basic abstractions allow users to avoid dealing with the high and low voltages flowing over a MIDI cable, and instead specify notes, durations and velocity. However, it is possible to support higher-level custom abstractions such reusable phrases, and custom relationships between entities.

The primary advantage of predetermined representations in user-interfaces is that they contain minimal conceptual overhead for the user to come to terms with. With fewer abstractions there is a reduced opportunity for hidden dependencies between abstracted components in the interface. More concrete sequencer interfaces can be easier to design and implement, because there is no requirement to carefully plan and build the conceptual model of the abstraction.

On the other hand, a well designed and implemented set of customisable higher-level abstractions can add considerably to the power of a music application [7].

2.4. Delayed vs. Eager linearisation

An important characteristic of sequencing interfaces is the point at which linearisation occurs. Music sequencing, and more generally the act of composition itself, can be seen as a task of creating an aesthetic linear ordering of musical material. Some sequencers can have a delayed approach to linearisation — where in the extreme case the final linear ordering can be held off until the point of performance. Alternatively, sequencer applications will often provide an environment where all musical material must be placed into an absolute position in the single canonical linear ordering of the piece.

Sequencers that employ eager linearisation have the advantage of a single linear representation that reduces interface complexity and cognitive overhead.

A system that delays linear ordering has the benefit that it may offset the problems of premature commitment [3]. A well designed delayed linearisation sequencer may be able to facilitate the rapid auditioning of various linearisations of musical material — a technique that is very familiar to composers who use traditional musical instruments as part of the composition process.

2.5. Data vs. Control-flow

Sequencers can utilise data- or control-flow paradigms. Control-flow systems can be identified as those that allow the user to specify the final sequencing in terms of order of events. This may include conditional tests or loops — which in a musical context maps to traditional scoring constructs such as first and second time repeat bars, and repeat sections and codas.

The alternative paradigm is data-flow. Data-flow systems are categorised by interfaces that require the user to determine the final sequence in terms of data flowing through a computational system. These systems allow the user to define computational elements which transform incoming data and conditionally send it to other computational elements. Such systems could be used to implement rule based systems for "improvising" with a musical score or other complex generative systems.

While many sequencer systems may utilise the data-flow paradigm for effect processing, their *sequencing* components are often predominantly control-flow based, and in those cases would be classified as such.

There is good evidence to suggest that data and control-flow systems are suited to dealing with different forms of information [2]. Research into visual data-flow environments has shown that they are particularly good at conveying the details of complex behaviour. In a visual programming context this was described as a suitability for communicating "abstract knowledge about what the program *does*" [2] . In contrast, control-flow systems were shown to place an emphasis on conveying "quite low-level sequential accounts" of programs.

2.6. Special vs. General purpose

The notion of a general vs. special purpose sequencing interface is largely a relative definition. Music applications can be seen to be general purpose for use with any music sequencing task, or special purpose in supporting particular types of music sequencing.

A special purpose music sequencing interface can be identified by evidence of a strong bias towards particular forms of musical sequencing. We define interfaces as special purpose where they are dominated by underlying assumptions and structure that favours one form of musical structuring over all others. Special purpose interfaces have the advantage that they can be optimised for a specific type of sequencing operation. This can win on simplicity and efficiency of operation if the interface is well designed.

In marked contrast to this, general purpose interfaces have the substantial advantage of being applicable to a wider scope of music sequencing tasks, removing the requirement for a separate specialised tool for every possible task.

2.7. Summary

This taxonomy classifies sequencing tools by distinguishing between the following characteristics:

Characteristic		
Medium	Textual	Graphical
Abstraction Level	Predetermined	Custom
Linearisation Stage	Eager	Delayed
Event Ordering	Control	Data
Applicability	Special	General

Every sequencing tool has an interface signature that is defined by how it is characterised in each of the five categories. With two possible values for each of the characteristics there are thirty-two possible signatures defined by this taxonomy, each of which would define a class of sequencer interfaces with its own set of strengths and weaknesses. However, only a small subset of these potential categories actually contain real sequencers in use today.

3. APPLYING THE TAXONOMY

This section presents the four major types of music sequencer applications that we introduced in section 2.1 and analyses and classifies their design characteristics in terms of the previously presented taxonomy.

3.1. Linear Sequencers

Graphical: Linear sequencers provide a graphical interface depicting a number of tracks. The graphical displays typically utilise direct manipulation techniques.

Predetermined Abstractions: For the large part these interfaces employ predetermined representations. In recording oriented interfaces users are presented with chunks of recorded audio which originate from recording “takes”. These can be split and merged, but typically cannot be manipulated in custom abstractions.

MIDI oriented conventional sequencer interfaces predominantly present musical events visually as individual objects placed on a two-dimensional spatial substrate. For the most part, the music is simply represented at just this event level, with little support for aggregation, collection or other abstractions. These sequencers use the multitrack metaphor which places an inflexible predetermined abstraction on the composer’s musical material.

Eager: Linear sequencing systems are highly biased towards eager linearisation. They are based on the underlying model of events placed on tracks. This means that every musical idea that exists in the piece the composer is working on must be placed in an absolute time location. The very act of creating a musical motif requires that it is given a location in the overall linear ordering of the piece.

Control: A control-flow paradigm is used in linear sequencers. The simplest sense of control-flow is at work here — as events are placed in absolute time locations, and playback typically proceeds from time zero through to the end of the piece. The use of markers and punch in / punch out recording functionality allows conditional beginning and end triggering.

General: Linear sequencers can be seen to be general purpose in that they do not limit the user to particular styles or arrangements of musical form due to the relatively low-level concrete interface.

3.2. Sample and Loop triggers

Graphical: These tools are highly graphical in nature. The contain graphical representations of the loop and sample objects — often including pictorial representations of wave forms. These can typically be moved or triggered through direct manipulation.

Predetermined Abstractions: Sample and loop triggers have a fixed, reasonably low level of abstraction. The user can deal with loops or samples, but typically cannot define their own complex ordering or nested structuring of abstractions.

Delayed: By allowing the user to trigger loops and one-shot samples in a real-time context, sample and loop triggering tools present a delayed linearisation interface. This means that users can rapidly experiment with different arrangements though interacting with the interface during playback.

Control: These tools utilise a control-flow paradigm. Musical material is placed in blocks that are triggered by temporal conditions, or though user interaction. Some of these tools also include data-flow effects processing systems, but the general sequencing model is highly control-flow driven.

Special: Sample and Loop triggers are special purpose music sequencing interfaces. They provide a specialised interface which facilitates the control of repeating loop based patterns.

3.3. Music Visual programming

Graphical: Being visual programming languages, these systems are graphical by definition. As users build their composition system, they can see the "objects", their parameters' values, and the links between them.
Custom Abstractions: Music visual programming environments provide a reasonable level of abstraction. In environments such as MAX and Reaktor users can define music event processing programs which can be embedded inside other programs. These user defined sub-programs are objects that can be used to abstract various music creation processing tasks and be reused in various programs. An integral part of these environments is also specifying the relationships between abstractions. By utilising the flexible nature of these systems it is possible for users to define their own relationships by controlling the interpretation of data flowing between objects.
Delayed: Because these systems only ever result in a linear musical representation on their execution, they are examples of highly delayed linearisation. The only point of linearisation is the linear stream of MIDI or audio data that are produced by these systems at runtime.
Data: Music visual programming systems utilise the dataflow paradigm. Musical event and audio data flows between the various computational components. This data is processed and results in various event and audio output.
General: These systems are highly flexible due to the general visual programming constructs they make available. Users can create various processing systems that can embody essentially any structuring of events and audio.

3.4. Textual language music tools

Textual: By definition, these forms of music programming languages are textual.
Custom Abstractions: Most music languages allow the user to define their own abstractions which they can use as required.
Delayed: Due to the use of abstractions it is possible for users to define the event structures before committing them to a final linear ordering. The use of abstraction also makes it convenient to rearrange the linear ordering as necessary.
Control: The event creation component of textual music languages are predominantly control-flow in nature. Score languages, as well as textual computer music tools embedded in procedural programming languages, are framed as instructions or the definition of events, and the control-flow logic that defines whether they are triggered.
General: Due to their flexibility, textual music languages are very general purpose. This is particularly true for textual music tools embedded in a fully fledged programming language such as Siren.

3.5. Summary

Both of the possible values for each of the five characteristics appear in one or more class of existing sequencing interface. This means that any interface we can conceive of can benefit from the wealth of knowledge that we have about these characteristics.

4. CONCLUSION

This paper has presented a taxonomy for music sequencers and applied it to the four most common types of sequencer interfaces. We have demonstrated how this can be used to analyse existing types of sequencer and gain insight into their strengths and weaknesses. More importantly, the taxonomy provides us with a principled understanding of the *causes* of these strengths and weaknesses which can be taken into account in designing future tools.

Additionally, this taxonomy can be used to discover new types of sequencer which combine different combinations of characteristics identified in the taxonomy. For example, we are currently designing a tool which will fit into a novel class identified through this taxonomy, characterised as a *graphical, custom, delayed, control-flow, general purpose* sequencing interface. There are twenty-eight novel types of sequencer that our taxonomy suggests outside of those identified here, many of which may be fertile grounds for investigation.

5. REFERENCES

[1] DUIGNAN, M., NOBLE, J., BARR, P., AND BIDDLE, R. Metaphors for electronic music production in Reason and Live. In *6th Asia-Pacific Conference on Computer-Human Interaction* (2004).

[2] GOOD, J. VPLs and novice program comprehension: How do different languages compare? In *Proceedings of the IEEE Symposium on Visual Languages* (1999), IEEE Computer Society, p. 262.

[3] GREEN, T. R. G. Instructions and descriptions: some cognitive aspects of programming and similar activities. In *Proceedings of the working conference on Advanced visual interfaces*. ACM Press, 2000, pp. 21–28.

[4] HOLMES, T. B. *Electronic and Experimental Music*. Charles Scribner's Sons, New York, 1985.

[5] SHNEIDERMAN, B. Direct manipulation for comprehensible, predictable and controllable user interfaces. In *IUI '97: Proceedings of the 2nd international conference on Intelligent user interfaces* (1997), ACM Press, pp. 33–39.

[6] THEBERGE, P. *Any Sound You Can Imagine: Makeing Music/Consuming Technology*. Wesleyan University Press, 1997.

[7] TRUAX, B. *The Language of Electroacoustic Music*. Palgrave, 1986, ch. Computer Music Language Design and the Composing Process.