QUILL:
AN INTERPRETER FOR CREATING MUSIC-OBJECTS
WITHIN THE DMIX ENVIRONMENT

Daniel V. Oppenheim
Center for Computer Research in Music and Acoustics (CCRMA)
Department of Music, Stanford University
Stanford, CA 94305
Dan@CCRMA.Stanford.edu

ABSTRACT: Quill is an interactive system for creating music-objects within Dmix—an object-oriented environment for composing and performing music. Quill makes full use of many compositional tools already found in Dmex and provides a general, flexible and extensible working environment that is fast and easy to use. Several simple data-entry formats may be freely combined with complex algorithmic descriptions. Functional programming is also supported, Quill works well in conjunction with the real-time, graphic and text editors found in Dmex: music created or modified by one can be passed on to another; several tools can sequentially modify a single object and provide multiple-representations.

Brief Description
Quill is an interactive system for creating music-objects within Dmex (Oppenheim 1989)—a highly level environment for creating, editing and performing music, implemented in Smalltalk-80. Quill is text-oriented and can work side by side with other tools, such as highly interactive graphic and real-time editors. It offers fast, convenient, general, flexible and extendable means for entering music via any combination of data-lists and algorithmic specifications.

The input to Quill is an ASCII stream, typically from Quill’s text-editor or a file. The editor highlights each segment of input as Quill evaluates it; if an error is encountered then a standard Smalltalk debugger opens.

Interactive Editing
As the input stream is evaluated, Quill creates its world of objects: music-events and utility-objects such as motives, functions, filters and loops. This world remains intact after compilation so that Quill and the objects it creates may be examined and modified directly from the editor. Any portion of the text may be selected and evaluated, the object that is returned can be played-back, inspected or edited instantly. This allows the user to edit interactively small segments of his input without having to compile and play-back the entire work.

Data Entry Formats
Quill provides several formats for data-entry that may be freely interchanged. The parameterGroup format is a convenient way for entering simple passages and is somewhat similar in concept to noteLists (see no. 1. Numbers refer to the graphic example on the last page). Unspecified parameters get default values or stick to their last value. Several mechanisms for enumeration are included (as in no. 4). Parameters may be set over time using functions (see no. 5). The Dmex noteList format is also supported to allow noteList generated by other objects to be pasted into Quill’s editor (no. 2). Finally, a voiceLoop format allows the specification of algorithms (see no. 6) and is similar to the PLA voice construct (Schottstaedt 1984). The voiceLoop is a macro that provides the full power and flexibility of programming in Smalltalk directly, yet it hides many
tedious programming requirements, manages time, keeps track of local variables and allows the user to access any object within Quill or the entire Dmix environment.

Implementation

By taking full advantage of Dmix's object-oriented environment and relying heavily on the existing Smalltalk language, Quill was implemented by introducing a very limited language-syntact. Every segment of input begins with a method-selector (pitch, tempo, etc.) and is followed by a string that will be referred to as a sourceString. When Quill reads a message-selector it simply sends itself the corresponding message with the following sourceString as the argument. Quill's functionality can easily be extended by writing new methods that will automatically become part of Quill's vocabulary.

Two methods are implemented for handling the sourceString. The code: method passes the sourceString on to Smalltalk's compiler for evaluation and returns the object that gets created. The askParameters: method parses the sourceString and returns a collection of parameters that are instances of the abstract super-class MusicMagnitude (frequency, midiKeyNumber, millisecond, beatDuration, etc.).

Using Smalltalk's compiler for implementing code: allows any legal Smalltalk code to be included in the input-stream and is one of Quill's most powerful features. This allows the user to create and use any object and to access any part of the Dmix environment, including Quill and it's instance-variables (Quill provides the compiler with it's own context for reference). For example the following statement

    code: 'voice play'  // see no. 7)

sends the message play to the object voice, resulting in immediate playback (voice is the last voice Quill created and is the value of an instance-variable in Quill). Note that playback is handled by the voice object and that Quill does not implement any method for doing so.

Many methods in Quill expect their sourceString arguments to be legal Smalltalk code and use the code: method as part of their implementation. For example, the method createFunction: creates a function and adds it to Quill's utilityObjects. It would be invoked in the input-stream by the following line (see no. 3):

    createFunction: '<sourceString — any Smalltalk code that returns a function'.

Quill implements this as follows: utilityObjects add: (self code: sourceString).

This arrangement not only provides maximum flexibility for collecting functions—by creating new ones, reading them in from disk, or grabbing existing functions from within the Dmix environment—but also eliminates the need to introduce a new syntax for doing so.

Conclusion

Quill allows music input via any combination of data-lists and algorithmic descriptions. It can be used interactively and it provides an easy way to use the various objects Dmix provides for composing. Only a minimal syntax is added above Smalltalk and a user can learn to use it in progressively more sophisticated ways as he gets to 'learn the Dmix and Smalltalk environments.

References


ICMC GLASGOW 1990 PROCEEDINGS 227