The Musical Object Development Environment: MODE (Ten Years of Music Software in Smalltalk)

Stephen Travis Pope
Computer Music Journal and CNMAT: Center for New Music and Audio Technologies
Department of Music, Univ. of California, Berkeley, P.O. Box 9496, Berkeley, California 94709 USA
Electronic mail: stp@CNMAT.Berkeley.edu

Abstract

The author has developed a family of software tool kits for composers with the Smalltalk-80 programming system over the last decade. The current MODE Version 2 system supports structured composition, flexible graphical editing of high- and low-level musical objects, real-time MIDI I/O, software sound synthesis and processing, and other tasks. This poster will introduce the MODE and SmOKe, its representation language, and survey the various end-user applications it includes. The discussion will evaluate the system's performance and requirements.

Keywords: Software, music languages, sound synthesis, composition tools, object-oriented programming

1 What is MODE?

The MODE Version 2 is a large hierarchy of object-oriented software components for music; it supports structured algorithmic composition, sampled sound recording and signal processing, real-time music performance using MIDI or sampled sounds, and graphical user interfaces. The MODE has 5 main component or modules: (1) a language for the representation of musical parameters, sounds, events and event lists; (2) objects for the description of "middle-level" musical structures; (3) real-time MIDI, sound I/O, and DSP scheduling classes; (4) a user interface framework with components for musical applications; and (5) several built-in end-user tools. There are facilities in the MODE for algorithmic composition, ranging from hierarchically-described event generators, to real-time MIDI performance mappers, to compositional expert-systems in several flavors.

The best overviews of the MODE with usage examples are the articles *Computer Music Journal* and *Leonardo Music Journal* articles (Pope 1992a) and (Pope 1992b). The underlying representation language (SmOKe) is described in (Pope 1992c), and two of the higher-level algorithmic composition packages are documented in (Pope 1989) and (Pope 1991).

The MODE is written in the ParcPlace Systems, Inc. Visualworks\Smalltalk-80 programming system and runs on the Sun SPARCstation, SGI Indigo, Apple Macintosh, Windows-compatible, and other platforms (because Smalltalk-80 itself is portable); its full source code and documentation are available by Internet ftp from the directory `st80` on the server `ccrma-ftp.stanford.edu`.

The MODE is the result of over ten years of development of software tools for music using Smalltalk-80. Earlier versions stressed real-time MIDI performance (The HyperScore ToolKit) or interactive algorithmic composition (DoubleTalk). The current MODE (version 2) aims at being relatively general-purpose, but places more emphasis on high-level hyper-media composition tools and support for DSP of sampled sounds than on live performance via synthesizers.

2 Why?

The motivation for the development of the MODE is the desire for a powerful, flexible, and portable computer-based composer's software tool kit that runs on both (expensive) workstations and (inexpensive) personal computers. It should support structured and unstructured composition, production, and live or tape performance equally well. The engineering goal is to exhibit good object-oriented design principles and modern software engineering practice in a software framework for building musical applications.

3 What is SmOKe?

The "kernel" of the MODE are the classes related to representing the basic musical magnitudes (such as pitch, loudness, duration, etc.), and for creating and manipulating musical event and event list objects. This music language is known as the Smallmusic Object Kernel (SmOKe—name contributed by Danny Oppenheim). SmOKe is a language-independent music representation, description language, and interchange format that was developed by a group of researchers during 1990 and 1991. The basic design requirements are that the representation provide the following features:

- abstract models of the basic musical quantities (scalar magnitudes such as pitch or loudness);
- instrument/note (voice/event) abstractions;
- sound functions, granular description, or other (non-note-oriented) description abstractions;
- flexible grain-size of "events" in terms of "notes," "grains," "elements," or "textures;"
- support event, control, and sampled sounds;
- nested event-tree structures for flexible description of "parts," "tracks," or other parallel or sequential organizations;
- separation of "data" from "interpretation" (what vs. how or having "performer" objects);
- abstractions for the description of "middle-level" musical structures (e.g., chords, clusters, or trills);
- hyper-media-like annotation and marking of event tree structures with musical, text, or graphical information; and
- description of sampled sound synthesis, processing, and mixing.

In SmOKe, music (i.e., a musical surface or structure), can be represented as a hierarchy of "events" (which generally last from tens of msec to tens of sec). Events are simply property lists or dictionaries; they can have named properties whose values are arbitrary. These properties may be music-specific objects (such as pitches or spatial positions), and models of many common musical magnitudes are provided. Voice objects and applications determine the interpretation of events' properties, and may use "standard" property names such as pitch, loudness, voice, duration, or position.

Events are grouped into event collections or *event lists* by their relative start times. Event lists are events themselves and can therefore be nested into trees (i.e., an event list can have another event list as one of its events, etc.); they can also map their properties onto their component events. This means that an event can be "shared" by being in more than one event list at different relative start times and with different properties mapped onto it.

Events and event lists are "performed" by the action of a scheduler passing them to an interpretation object or voice. Voices map event properties onto the specific parameters of I/O devices; there can be a rich hierarchy of them. A scheduler expands and/or maps event lists and sends their events to their voices. Sampled sounds are also describable in terms of sample arrays, synthesis "patches," or signal processing scripts involving a vocabulary of sound manipulation messages.

4 The MODE Frameworks

In addition to the SmOKe classes, the MODE has several object frameworks for building music software.

The classes for event and signal I/O are organized into *voices* that map properties to parameters, *devices* that format streamed I/O, and *ports* that coordinate multiple devices, so that a good deal of synthesis-medium-independence can be achieved. An example of this is managing MIDI scores that include microtonal pitches and mapping these onto the different microtonal note-on commands used by various synthesizers.

The *event generator* and *event modifier* packages are higher-level compositional objects; they allow composers to design their own languages for describing music during the process of realization of a piece. Examples here are the many levels of chord, scale, and trill event generator objects, or the rubato and crescendo event modifiers

A number of graphical user interface tools have also been built, including construction kits for pitch-time-oriented score editors, hierarchical score and sound browsers, signal processing tools, and others.

5 MODE Applications

Several of MODE's applications will be illustrated in the poster. The recent references also provide a sampler of current composition and production tools.

6 Evaluation

The MODE has several advantages and disadvantages that are inherent in its design. The use of Smalltalk-80 as implementation language and delivery system also has several interesting ramifications.

The high-level of portability of the "kernel" code among Smalltalk-80 platforms is a major plus. I regularly move between different hardware and operating system platforms (Sun, SGI, and Mac), and appreciate not having to re-compile. The operating system independence and GUI portability are also useful. The ease of development within the Visualworks programming environment is another major advantage that has kept me from using the various C- and Lisp-based systems that have become popular in recent years. No other development environment offers Smalltalk's level of integrated source code browsers, debuggers, and library management tools.

The size in RAM of the MODE run-time (its "memory footprint"), and MODE's speed on slow processors are the two most-often cited problems; they are also related to the choice of Smalltalk. MODE will just barely run on a Macintosh with 12 MB of RAM, and Smalltalk is not used for the inner loops of the schedulers and DSP programs, which are in C and are dynamically linked to Smalltalk (adding a lot of complication).

Using the MODE to develop tools for composition has been a joy for me, but a major frustration for several other users who were not clear that MODE is a *developers tool kit* rather than a *finished application*. The level of robustness and functionality of the built-in demonstration applications has disappointed some users who were not proficient Smalltalk programmers.

7 Conclusions

The poster will illustrate examples of the SmOKe language and several MODE applications. I hope to demonstrate the possibility of building a sophisticated, portable, integrated programming environment for musical applications. The MODE is the work of several people, and the author wishes to acknowledge the on-going input of Daniel Oppenheim, Craig Latta, Giovanni Müller, and Guy Garnett.

References

Pope, S. T. 1992a "The Interim DynaPiano: An Integrated Tool and Instrument for Composers." Computer Music Journal 16(3): 73-91

Pope, S. T. 1992b "Producing 'Kombination XI:' Using Modern Hardware and Software Systems for Composition." Leonardo Music Journal, 2(1): 23-28.

Pope, S. T. 1992c "The SmOKe Music Representation, Description Language, and Interchange Format." Proceedings of the 1992 ICMC.

Pope, S. T. 1991 "A Tool for Manipulating Expressive and Structural Hierarchies in Music." Proceedings of the 1991 International Computer Music Conference.

Pope, S. T. 1989 "Modeling Musical Structures as EventGenerators." Proceedings of the 1989 ICMC.