Siren: Software for Music Composition and Performance in Squeak

Stephen Travis Pope
Center for Research in Electronic Art Technology (CREATE)
Music Building, University of California, Santa Barbara, California, 93106 USA
stp@create.ucsb.edu http://www.create.ucsb.edu/~stp/

Abstract

Squeak is a new implementation of the Smalltalk programming environment. It was developed at Apple Labs, and has been ported to a variety of computers. Compared to other Smalltalk systems, Squeak has four important features: (1) portability (to the Macintosh, Windows PCs, and many flavors of UNIX); (2) speed (it uses native C for compute-intensive code); (3) price (free, including all source code!); and (4) sophistication (full Smalltalk-80 language, libraries, and tools, with many useful extensions).

The Siren system is a new object-oriented (OO) software tool kit for music applications. Siren's design was derived from the author's 14-years of experience building Smalltalk-based music tools. The intention is to support music composition, digital sound synthesis and processing, and live performance within a free, portable, high-level software tool kit.

This paper will briefly introduce the Squeak system, and then discuss the design of Siren. An on-line demonstration of Siren running on a lap-top computer is planned for the presentation at ICMC. Both Squeak and Siren are available in source code free via Internet ftp from the site ftp://ftp.create.ucsb.edu/pub/Smalltalk/Squeak/.

1 Introduction

The Squeak implementation of Smalltalk was developed at Apple Laboratories by a group that consisted of several of the original designers of the Smalltalk system (Alan Kay, Dan Ingalls, Ted Kaehler, et al.). Squeak is a "full-scale" Smalltalk implementation, and includes all of the standard class libraries and development tools.

The Siren package in Squeak is a member of the DoubleTalk/HyperScore/MODE family of systems developed by the author since 1983. (The name Siren was suggested by Curtis Roads; it is not an acronym.) Siren includes the following components:

- the Smoke music representation language;
  - music magnitude models (time, pitch, loudness, etc.),
  - events and event lists,
  - event generators (procedural or stochastic stream-like composite events),
  - event modifiers (functions that can be applied to event lists), and
  - software to read/write Smoke from/to many other music formats,
- classes for real-time sound synthesis;
  - OO models for synthesis and processing,
- sound and MIDI I/O support;
  - real-time I/O voices for many interchange formats,
- note list I/O for non-real-time synthesis packages;
  - read/write cmusic, cmix, or csound scores,
- GUI-based tools for score/sound manipulation
  - pitch-time diagrams, hierarchies, DSP, etc.

Compared to earlier Smalltalk music kits, Siren has more sophisticated models of the basic music magnitudes, flexible eager or lazy function application, a new user interface paradigm, and a complete model of objects for modular sound synthesis a la Music V.

2 The MODE History

The Musical Object Development Environment (MODE) was written during the author's sojourn at the STEIM Institute in Amsterdam in 1990; it was derived from the earlier HyperScore ToolKit (see ICMC 1987). The intention was to provide a portable system that could perform real-time MIDI and sampled sound I/O on a number of platforms (Macintosh, Sun, and PC). The MODE included a simple object-oriented music representation language, drivers for MIDI and sound I/O, and a collection of graphical user interfaces for various musical applications.

Throughout its life, the MODE was plagued by difficulties supporting the I/O drivers on more than one platform. The Macintosh MIDI drivers were only partially functional, and Sun failed to support MIDI at all on its new Solaris operating system. The complexity of building interfaces to C code in the ParcPlace Systems, Inc. VisualWorks Smalltalk implementation made it even harder to perform digital sound synthesis or processing from within the MODE.

During 1991 and 1992, a group of language designers met at the CCRMA Center at Stanford University and the CNMAT Center at U. C., Berkeley to formalize and extend the MODE's representation language. This project led to the Smoke language, which was described in a paper by the author in the 1992 ICMC Proceedings.

3 Siren Extensions to the MODE

Over the past five years, a number of weaknesses in the Smoke language and the MODE's implementation of it have become apparent. Other problems surfaced in the MODE's built-in applications and interfaces. These will each be introduced below.

Smoke had no explicit notion of intervals. Pitches are represented using a flexible framework of objects, but intervals were only seen as side-effects of pitch arithmetic. Taking a hint from Francois Pachet's MusES system (described in recent ICMC Proceedings), the Siren version of Smoke has both reified intervals and more powerful tonal chord models.

Smoke's models of event modifiers (e.g., crescendo or accelerando) were typically "eager," that is, they applied themselves to an event list when they were declared. In Siren, event modifiers can be attached to an event list for "lazy" (i.e., performance-time) application.

The MODE included a large "voice" framework that separated (abstract) event properties from (concrete) parameters of synthesis methods (such as the settings of a MIDI device or the mappings needed by an out-board synthesis package such as cmix). In order to support real-time sound synthesis in Squeak, this architecture had to be revisited, leading to a new voice model.

Lastly, the graphical library used for Siren GUIs is different from the MODE's simple display list graphics classes. Squeak includes John Maloney's "Morphic" user interface framework.

4 Squeak Compilation Technology

The Squeak system has an integral Smalltalk-to-C translator; this is used to generate the Squeak virtual machine (VM), the source for which is actually written in Smalltalk and then translated to C and compiled using a standard C compiler. The translator supports a subset of the full Smalltalk language, excluding (a) the fancy Smalltalk control structures that are not expressible in C in a straightforward way, and (b) the Smalltalk class libraries beyond those that represent basic C types and data structures.

The translator can also be used, for example, to accelerate the inner loops of signal processing functions. Informal benchmarks demonstrate the ability to perform 10 voices of real-time FM or plucked string synthesis at a 44.1 kHz sampling rate on modest hardware (Apple PowerBook 1400cs).

A Smalltalk method that is to be translated into C may include C-style declarations of shared variables, and a Smalltalk method body. The translator generates C code that is compiled and linked to the Squeak virtual machine, whereafter the method body can be replaced by a Smalltalk primitive message-send.

5 Sound Synthesis in Siren

Part of the Squeak class library is a digital sound synthesis package developed by John Maloney using the Smalltalk-to-C translator described above. Each class that represents a synthesis technique implements a synthesis method called

```
mixSampleCount:n into:aSoundBuffer startingAt:startIndex
    pan:panValue.
```

This method sums the given number of samples (n) into the given sound output buffer using the appropriate synthesis technique. As an example, John's implementation of FM in Squeak uses the following method (slightly simplified, monophonic).

```
FMSound methodsFor: 'sound generation'
mixSampleCount: n into: aSoundBuffer startingAt: startIndex
    "A simple implementation of Chowning's frequency-modulation synthesis
technique. The center frequency is varied as the sound plays by changing the
increment by which to step through the wave table."

    "Variable declarations."
    | lastIndex i sample |

    "Declare variables to be shared between Squeak and C - output buffer and
wave table."
    self var: #aSoundBuffer declareC: 'short int *aSoundBuffer'.
    self var: #waveTable declareC: 'short int *waveTable'.

    "Set up loop counter."
    lastIndex := (startIndex + n) - 1.

    "Sample computation loop."
    startIndex to: lastIndex do:
        [i|
    "Get sample value; index is wave table look-up index."
        sample := (amplitude * (waveTable at: index)).

    "Write sample to output buffer."
        aSoundBuffer at: i put: ((aSoundBuffer at: i) + (sample * pan)).

    "Update table indices for next loop."
        index := index + increment + ((modulation * (waveTable at: offset index))).

        index > waveTableSize ifTrue: [index := index - waveTableSize].
        index < 1 ifTrue: [index := index + waveTableSize].

        offsetIndex := offsetIndex + offsetIncrement.
        offsetIndex > waveTableSize ifTrue:
            [offset index := offsetIndex - waveTableSize]].
    "End of sample loop."
    count := count - n.
```

The current class hierarchy of sounds is as follows (whereby the indentation implies subclassing and the names in parentheses are the class's instance variables).

```
Object()
    AbstractSound(samplesUntilNextControl)
        MixedSound('sounds' 'panSettings' 'soundDone')
        PluckedSound(initialCount 'count' 'amplitude' 'ring' 'ringSize' 'ringIndx')
        RestSound(initialCount 'count')
        SequentialSound(sounds 'currentIndex')
        WaveTableSound(waveTable 'waveTableSize' 'initialCount' 'count'
                initialAmplitude 'amplitude' 'decayRate' 'increment'
                index)
            FMSound(initialModulation 'modulation' 'modulationDecay'
                offsetIncrement 'offset index')
```

A SoundPlayer class supports buffer-oriented sound output to the native operating system's play routines. It has variable sample rate and block size; these can be set up using a configuration message of the following form.

```
SoundPlayer startPlayerProcessBufferSize: 2205 rate: 44100
    stereo: true.
```

This sound synthesis system has been merged with the Siren event/voice framework in that events can have voices that represent sound classes, and these are triggered when the events are performed.

6 The Morphic User Interface

Another important Squeak extension over "standard" Smalltalk implementations is the "Morphic" user interface framework, originally developed for the Self language by John Maloney. Driven by the desire for easy portability, the MODE's user interfaces have been based on a simple display list class library for some time, and it was relatively easy to move these onto Morphic. The larger transition was adapting the MODE's "tool-centric" GUI to the Morphic "object-centric" interaction paradigm.

7 Still To Do

As with previous MODE versions, Siren is still weak in terms of MIDI support, and in terms of true portability of the sound I/O interfaces. Between the time of this writing and the 1997 ICMC, we hope to have MIDI running on the Macintosh platform, and the sound I/O code ported to run on UNIX (using the NetAudio libraries for portability and device independence).

8 Conclusions

Squeak is an exciting development in the Smalltalk world because of its level of sophistication, performance, portability, and cost. Siren is a significant improvement over the previous Smalltalk music packages such as the HyperScore ToolKit and the MODE.