Recent Developments in Siren: Modeling, Control, and Interaction for Large-scale Distributed Music Software

Stephen Travis Pope and Chandrasekhar Ramakrishnan
Center for Research in Electronic Art Technology (CREATE)
University of California, Santa Barbara (UCSB)
email: {stp, sekhar}@create.ucsb.edu

Abstract

This paper describes recent advances in platform-independent object-oriented software for music and sound processing. The *Siren* system is the result of almost 20 years of continuous development in the Smalltalk programming language; it incorporates an abstract music representation language, interfaces for real-time I/O in several media, a user interface framework, and connections to object databases. To support ambitious compositional and performance applications, the system is integrated with a scalable real-time distributed processing framework.

Rather than presenting a system overview (Siren is exhaustively documented elsewhere), we discuss the new features of the system here, including its integration with new DSP frameworks, new I/O interfaces, and its use in several recent compositions.

1 Introduction

Siren is a general-purpose software framework for sound and music composition and production; it traces its roots back to 1984. Siren is a library of about 350 Smalltalk classes for building sound/music applications; it is platform-independent and runs on Macintosh, Windows, and UNIX-based computers. The core elements of Siren are:

- the Smoke music representation language (classes for music magnitudes, events, event lists, generators, functions, and sounds);
- voices, schedulers and I/O drivers (real-time and file-based I/O for sound, OSC, and MIDI);
- user interface components for musical applications (UI tools and music/sound widgets); and
- several built-in applications (various editors and browsers for Siren objects).

The best references for Siren are (Pope 1987, 1992, 2001). This paper will not present the basic framework (which has been quite stable since 1989), but will discuss the recent evolution and applications of the system, including the port to VisualWorks non-commercial Smalltalk, the new support for I/O using the OSC protocol, database back-ends, and the system's use in several compositions.

2 Music Representation in Smoke

When the Smoke representation was developed (1988), it was designed as a language-independent representation for music. Yet the syntax of Smoke (and to some extent, its semantics) is that of Smalltalk, the language used for the first implementation.

At the time, Smalltalk was a natural choice of language; there were not many object-oriented languages with substantial user bases, and of those languages, Smalltalk was by far the most mature. In the intervening 15 years, the language landscape has changed. The reality of today—much to our disappointment as Smalltalk fans— is that Smalltalk is a language that many have heard of, but few know. Today, mainstream object-orientation means C++, Java, C#, ObjectiveC, Ruby, or Python.

This implies a new set of choices when it comes to moving Siren forward. If the goal is to make Siren available to the greatest number of users, one of the more widely known languages would be a better choice than Smalltalk. If the goal is to experiment and see how new language features can be applied to representation of music, other options may or may not be indicated. Class-based object-orientation is no longer novel, and languages with different features, such as prototype-based OO languages like Self or CommonLISP, have potential to bear fruit in the search of a flexible and elegant representation of music. If the goal is a strict interpretation of the timing information specified in a Smoke event list, then still other languages become natural choices. Timely processing of event lists requires a language designed for use in a real-time setting, such as SuperCollider or Erlang. Of course, there remains another possibility of designing and developing our own language aimed at music representation, using Smoke as a starting point.

While we continue to use the Smalltalk implementation of Siren for the bulk of our development (for reasons detailed in the Evaluation section below), we have begun tackling the goals mentioned above. To that end, we have made experimental versions of the Siren core in Ruby, Self, and SuperCollider.

Of the languages we determined would give Siren the widest distribution, we eliminated C++, Java, and C# because they are (usually) statically compiled, making them unsuitable for interactive use. Of the remaining two, we chose Ruby over Python because it is more different from Smalltalk. Ruby offers features like "continuations" that could be profitable employed in music representation (cf. SuperCollider patterns). For the exploration of new language paradigms, we chose Self because it is exotic enough to contain interesting language features, but stable enough to have a simple installation process and offer a powerful development environment. For real-time performance, we chose SuperCollider because it is widely used in the computer music community. We would still, at some point, like to try an Erlang implementation of Siren.

Moving Siren away from its Smalltalk origins has forced a re-evaluation of some assumptions inherent in the original description of Smoke. Though the Smoke specification indicates that Smoke is both a class-library specification and a syntax, we have chosen to abandon the syntax and retain only the class-library specification. To retain the syntax would, first of all, require the implementation of Smalltalk virtual machine to handle the Smalltalk block-closures which are permitted in Smoke events, but, more to the point, retaining the syntax would force the users to learn Smalltalk, which defeats the goal of bringing Siren to a wider audience. So, we have chosen to keep only the class-library specification. Thus, Smoke event lists are written in the same language as the implementation which is being used.

This recent work is altering some aspects of Siren, but the basic character of flexible representation and processing of musical information remains. We will have more to report on this front in the future.

3 Siren on Squeak (1996-2001)

The original platform for main-stream development in Smalltalk was Xerox PARC Smalltalk-80 version 2 (1982), which became an expensive commercial product when ParcPlace Systems, Inc. spun off from Xerox in 1988. In 1996, the sound/music framework then called MODE was ported to Squeak (http://www.squeak.org), a then-new open-source Smalltalk implementation. I renamed MODE to Siren at that time. Squeak is a wonderful, open system, and added many features not found in other Smalltalks, including a novel (though slow and buggy) GUI framework called Morphic, a Smalltalk-to-C translator for making native methods, and toy versions of sound synthesis and graphical rendering frameworks.

The final version of Siren on Squeak (3.0, which was released on a CD-ROM in 2001 [Pope 2001]) incorporated a set of low-level MIDI functions, streaming sound I/O, GUI applications based on Morphic, and an interface to the MinneStore object database system.

Even after six years of development, however, Squeak still lacks the performance, solidity, and interoperability of commercial Smalltalk environments, and suffers from the lack of clear management that is common to many open-source projects. Since Squeak is based on Smalltalk -80 version 1, it is missing all of the enhancements made at PARC since 1981, including integrated exception handling, the unified I/O framework, the improved Smalltalk compiler, the parser compiler (akin to YACC), and the addition of namespaces to the language. These factors (especially the system's mediocre virtual machine performance) became increasingly frustrating as time went on.

4 Siren on VisualWorks

In 2000 or so, ParcPlace released a free non-commercial versions of their flagship VisualWorks/Smalltalk system (called VisualWorks non-commercial or VWNC), the true descendent of the PARC Smalltalk-80 lineage. The Siren system has now been ported from Squeak "back" to VWNC. The main differences are not in the core system classes (which are largely compatible between all Smalltalk implementations), but in the facilities for file and socket I/O, and especially in the GUI frameworks. The VisualWorks tool set is also a good deal more sophisticated than that of Squeak; it includes a much better (multi-threaded) debugger, and tools for configuration management, team programming, etc.

5 Siren I/O: MIDI, OSC, etc.

One of the persistent problems with making cross-platform music tools has been the lack of good portable libraries and APIs for sound and MIDI I/O. In recent years, this has been greatly helped by the emergence of the cross-platform *PortAudio* (http://www.portaudio.com), *PortMIDI* (http://www-2.cs.cmu.edu/~music/portmusic), and *LibSndFile* (http://www.zip.com.au/~erikd/libsndfile) packages.

VisualWorks Smalltalk includes a powerful facility for constructing interfaces between Smalltalk and C libraries called the "Dynamic Linked Library and C Connect" package, or DLLCC. The DLLCC tools can read C header files, parsing data type definitions, macros, and function prototypes, creating equivalent methods in special Smalltalk classes called "external interfaces." The DLLCC loader can then be directed to load a given shared object file whenever an

instance of an external interface class is created. DLLCC has now been used to integrate Siren with the LibSndFile, PortAudio, PortMIDI, and FFTW (http://www.fftw.org) packages.

For simpler network- and file-oriented I/O, new voice objects have been developed to allow Siren to communicate over the OSC network protocol and to generate note-list files for use by programs written in the SuperCollider programming language.

Because of Siren's separation of musical data from the interpretation of abstract musical properties into concrete parameters (this is the gist of the event/voice distinction in Siren), these new voice classes were immediately usable by existing compositions.

6 Siren and CSL

One of our recent projects has been to use Siren as a front-end to the CREATE Signal Library (CSL, pron. "sizzle") (see [Pope and Ramakrishnan 2003] elsewhere in these Proceedings). Specifically, Siren applications that play back stored scores, implement generative algorithms, or map in-coming gesture data can send OSC messages to one or many CSL-based server programs, and can dynamically create, start/stop, and monitor CSL server instances.

7 Siren and CRAM

In other projects on distributed software for real-time object-oriented applications, we have developed a software infrastructure we call the CREATE Real-time Application Manager (CRAM) (Pope et al. 2001). CRAM consists of a description language, a software framework, a tool-set, and a server infrastructure for large-scale distributed processing.

Siren and CSL are designed from the ground up to be used in distributed systems, with several CSL programs running as servers on a local-area network, all controlled by Common Object Request Broker Architecture (CORBA, http://www.omg.org) and OSC messages sent from Siren programs. These CSL DSP servers receive control commands via the network and send their output sample blocks to other servers over the network. (Control in CSL is transmitted via OSC network messages, and any "wire" between elements in a CSL DSP graph can be deferred over a network socket.) A typical multi-server CSL/Siren configuration is illustrated in Figure 1 below.

In this example, each of the round-edged rectangles is a separate server program. The top four servers are CSL programs (written in C++); the larger box in the middle is the CRAM system manager (written in Smalltalk), and the Siren server at the bottom sends OSC messages to the CSL servers.

The control links (shown as dotted lines) use the CORBA and OSC protocols, and the inter-program sample streams (drawn as arrows) use the CSL sample streaming protocol. CRAM manages the distributed CSL/Siren application; it starts up the individual CSL servers, and monitors them during run-time.

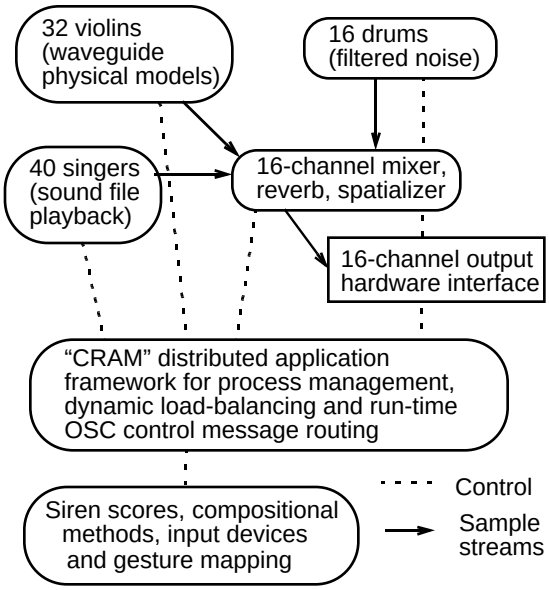

Figure 1: A distributed CSL/Siren configuration (each box is a separate server)

The heart of a large-scale synthesis system is a Smalltalk server running both Siren and the CRAM manager. Figure 2 on the next page shows the CRAM system monitor screen, in which one can see three main panes in the view:

- top: the table of available nodes with their characteristics (read in from the configuration database);
- middle: the list of running services on the selected machine (CRAM manages these); and
- bottom: the text of recent messages from the selected service (accessed via the node's Logging service).

8 Composition with Siren

The first author has used Siren (and its predecessors) in all of his compositions since 1984. Recently, new class libraries have been developed to support large speech databases with phoneme segmentation and detailed feature extraction for the works *Four Magic Sentences* (2000), *Sensing/Speaking Space* (2001/2), *Gates Still Open* (2002), *Eternal Dream* (2002/03), and *Leur Songe de la Paix* (2003).

The analysis core of the Siren speech database is the segmenter, which uses a combination of time-domain and spectral-domain features to break continuous speech into individual phonemes. Figure 3 below shows an example of the debugging screen of

the segmenter. In this case, the input was the sentence “Ice melts.” The lower part of the view shows the detailed spectrum and windowed RMS amplitude. The upper part of the view shows the octave-band spectrum and segment points (the vertical lines in the upper plot; this is what the segmenter generates).

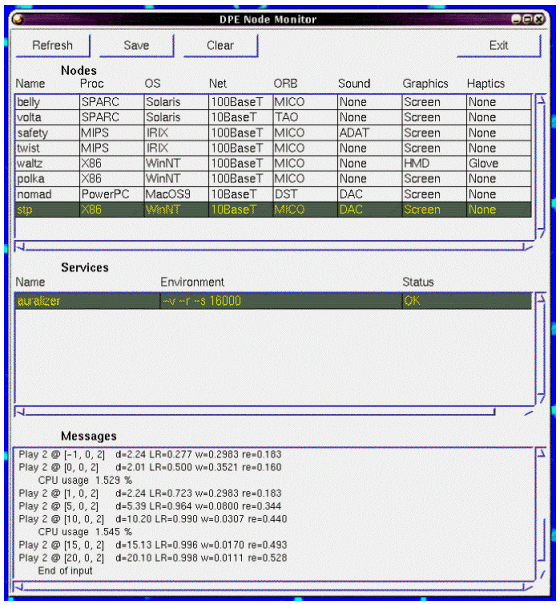

Figure 2: CRAM distributed application manager

The system then stores a set of properties for each phoneme: file name, start/stop samples, duration, max ampl., rms ampl., spectral centroid, spectral width, spectral bands, noise level, pitch estimate, pitch trajectory, and envelope class. A database has been created with over 20,000 phonemes.

In typical usage, the database is be queried (from Smalltalk or SuperCollider) by supplying a target phoneme and a distance metric function.

9 Database Interfaces

The Squeak-based version of Siren was integrated with the MinneStore (http://minnestore.sourceforge.net) object database system. Under heavy usage, however, the system proved not to have sufficient performance to support multimedia data being manipulated by interactive applications. With the port to VisualWorks, we have moved to the server-based Gemstone database (http://www.gemstone.com). The ramifications of this are both that the database is more invisible to the user (a very large set of objects is simply persistent between user sessions), provides much higher performance, and can be accessed from C++ programs (via the Gemstone C API). The goal (not yet fully realized) is that all of a user’s data—scores, sounds, edit scripts, etc.—be stored in a unified database, that it be available in any studio (or at home), and that it support modern database features such as replication-on-demand, versioning, query-by-example, roll-backs etc.

10 Future Siren Applications

There are several areas of active development of Siren by the authors. The new interfaces to CSL-based analysis/synthesis servers permit us to integrate Siren into ever more sophisticated compositional and signal processing applications.

One area of interest is developing new front-ends for extended granular synthesis in order to augment, and eventually replace, the CREATE PulsarGenerator program (http://www.create.ucsb.edu/PulsarGenerator).

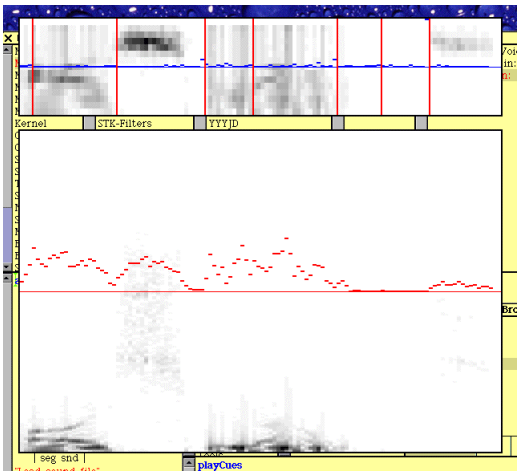

Figure 3: Output of the Siren speech segmenter

As Siren can now read and write OSC messages, and one of our graduate students (Garry Kling) has implemented an OSC interface to the Macromedia Director program (see http://www.mat.ucsb.edu/~g.kling/OSC/oscar.html), we look forward to controlling graphical animation programs from Siren in the near future.

Spatial sound is one of the prime R&D topics at CREATE, and we are now in a position to integrate Siren, a CSL-based many-channel panner, and our existing multi-modal input devices (e.g., Overholt 2002) to create interactive spatialization tools for composers.

An external interface is currently being built to access the linear prediction functions in the TSP signal processing package (http://www.tsp.ece.mcgill.ca/MMSP/Documents/Software/libtsp/libtsp.html).

11 Evaluation

So, given this very compelling sales pitch, would we recommend that all of you use Siren? Yes and no.

On the “yes” side, Smalltalk adherents have long

claimed that (1) it is the simplest imaginable programming language (all data is of the same type, and there is only one technique of method activation), (2) the class library is quite compact given its functionality, especially when compared to the competition: the total lack of useful C++ class libraries, and the complementary flood of mutually inconsistent Java APIs, (3) the development environment with code browsers, a rapid-turn-around compiler, an in-place debugger, and integrated code versioning and team programming tools allow for quite unequalled productivity, and (4) the language, libraries, and IDE have been mature and stable for a long time (you can still use the 20-year-old reference books for the core of today's system). In our opinion, these last two issues outweigh the first two.

Nevertheless, Smalltalk currently falls in the category of "semi-obscure" programming languages; it has consistently failed to gain market-share to match its mind-share.

The VisualWorks\Smalltalk implementation is a large, complicated, but also very sophisticated system. The base-line virtual image (akin to a Java .jar file) contains over 2000 classes; and Siren adds over 300 more. This is not a system that one learns over a weekend. The tools are also delivered in a cross-platform development environment, which is a valuable boon to the seasoned user, but increases the slope of the initial learning curve.

Like Java, Smalltalk programs are generally compiled to a virtual machine (VM) language, which might then be interpreted, translated, or cross-compiled at run-time. This provides for cross-platform portability of object code, but costs some level of run-time performance. Modern Smalltalk virtual machines use similar optimization techniques (e.g., polymorphic in-line caching and cross-method optimization) to best-of-breed Java VMs (and are actually much better than "average" Java VMs).

Both Smalltalk and Java also assume automatic storage reclamation (garbage collection), which makes development easier, but adds (often unpredictable) run-time overhead.

Lastly, the Siren package itself is large, complex, and implements a set of very specific design approaches to music representation, performance, sound and signal processing, and user interfaces. Many of the standard features of computer music software (e.g., simple MIDI sequencing or common-practise music notation editors) are still not present in Siren, due to lack of interest on the part of the authors. We choose to do with Siren that which we cannot do with some combination of SuperCollider, Peak, Finale, and ProTools.

12 Summary

Given my past schedule of "5-yearly" updates on Siren (1987, 1992, 1997), I'm a year late this time. This report introduces the current status of the Siren system, and the tools we've built at UCSB to interoperate with it.

We continue to develop and use Siren in Smalltalk, even as we experiment with implementations of systems based on the principles of Siren (and the Smoke music representation language) in other languages. The programming language situation (Smalltalk vs. other languages) reminds one of Winston Churchill's description of democracy: "it's the worst possible system, aside from every other one we've ever tried."

The demonstration at ICMC 2003 will illustrate many of the new features and applications of Siren.

13 References

(See the more complete reference list in the companion paper [Pope and Ramakrishnan, 2003] elsewhere in these proceedings.)

Overholt, Dan. 2002 "New Musical Mappings for the MATRIX Interface. "*Proc. 2002 ICMC*. (See also Dan's other projects described in http://www.create.ucsb.edu/~dano.)

Pope, S. T. 1987. "A Smalltalk-80-based Music Toolkit." *Proc. 1987 ICMC.*

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

Pope, S. T. 1997. "Siren: Software for Music Composition and Performance in Squeak." *Proc. 1997 ICMC.*

Pope, S. T. 2001. "Music and Sound Processing in Squeak Using Siren." in Guzdial, Mark and Kim Rose. *Squeak: Open Personal Computing and Multimedia.* (book and CD-ROM) Prentice-Hall.

Pope, S. T., A. Engberg, F. Holm, and A.Wolf. 2001. "The Distributed Processing Environment for High-Performance Distributed Multimedia Applications." *Proc. 2001 IEEE Multimedia Technology and Applications Conf.*, U. C. Irvine.

Pope, S. T. and C. Ramakrishnan, 2003. "The CREATE Signal Library ("Sizzle"): Design, Issues, and Applications." *Proc. 2003 ICMC.*