SAOL: THE MPEG-4 STRUCTURED AUDIO ORCHESTRA LANGUAGE

Eric D. Scheirer
Machine Listening Group
E15-401D MIT Media Laboratory, Cambridge MA 02139-4307

Abstract: The MPEG-4 standard, which will be published in October 1998, contains extensive provisions for sound synthesis as well as traditional methods of audio compression. At the heart of MPEG-4 Structured Audio, the sound-synthesis framework, is a new music-synthesis language called SAOL. This language, based on the Music-N model, is specified and defined fully in the MPEG-4 International Standard; a real-time implementation must be embedded in any device conforming to the full MPEG-4 Audio standard. In this paper, the structure and capabilities of SAOL are discussed, especially in comparison with other music languages. A discussion of the role of international standardization in the future development of computer-music tools is also presented.

1 Introduction

Since the beginning of the computer-music era, tools have been created which allow the description of music and other organized sound as concise networks of interacting oscillators and envelope functions. Originated by Max Mathews with his series of “Music-N” languages, this unit generator paradigm for the creation of music has proven highly effective for the creative description of sound and widely useful for musicians. Languages such as Csound, Nyquist, CLM, and SuperCollider are in wide usage in academic and production studios today.

As well as being an effective tool for marshalling a composer’s creative resources, these languages represent an unusual form of digital-audio compression (Vercoe, Gardner, and Scheirer 1998). A composition in such a language is much more succinct than the sequence of digital audio samples which it creates, and therefore this method can allow for more dramatic compression than traditional audio coding. The idea of transmitting sound by sending a description in a high-level music language and then performing real-time synthesis at the receiving end, which Vercoe et al (1998) term Structured Audio, was suggested as early as 1991 (Smith 1991). A project at the MIT Media Laboratory called “NetSound” constructed a working system based on this concept, using Csound as the synthesis engine (Casey and Smaragdis 1996), and allowing low bitrate transmission on the Internet. If it were possible to create a broad base of mutually-compatible installed systems and musical compositions designed to be transmitted in this manner, such a system would have broad utility for music distribution.

The Motion Pictures Experts Group (MPEG) will publish the MPEG-4 standard, formally ISO 14496, in October 1998; MPEG-4 will be designated as International Standard in December 1998. The workplan and technology of MPEG-4 represent a departure from the previous MPEG-1 (ISO 11172) and MPEG-2 (ISO 13818) standards. While MPEG-4 does contain capabilities similar to MPEG-1 and MPEG-2 for the coding and compression of audiovisual data, it also specifies methods for the compressed transmission of synthetic sound and computer graphics, and for the juxtaposition of synthetic and “natural” (compressed audio/video) material.

Within the MPEG-4 standard, there is a set of tools of particular interest to computer musicians called Structured Audio (Scheirer 1998; Scheirer in press). MPEG-4 Structured Audio allows synthetic music to be transmitted as a set of instructions in a unit-generator-based language, and then synthesized at the receiving terminal. The music language used in MPEG-4 for this purpose is a newly devised one called
SAOL, for Structured Audio Orchestra Language (pronounced “sail”). By integrating a music-synthesis language into a respected international standard, the required broad base of systems can be established, and industrial support for these powerful capabilities can be accelerated. The sound-synthesis capabilities in MPEG-4 have status equivalent to the rest of the coding tools; a compliant implementation of the full MPEG-4 audio system must include support for real-time synthesis from SAOL code.

In this paper, the structure and capabilities of SAOL will be described. Particular focus will be given to the comparison of SAOL with other modern music languages, focusing on strengths and weaknesses of the new language. A discussion of the role of the MPEG-4 International Standard in the development of future computer music tools concludes the paper.

2 SAOL: Structure and Capabilities

SAOL is a declarative unit-generator-based language. In this respect, it is more like Csound (Vercoe 1995; Boulanger 1998) than it is like SuperCollider (McCartney 1996a,b; Pope 1998) or Nyquist (Dannenberg 1997a), both of which have heavy elements of functional programming in their design; SuperCollider additionally employs an object-oriented model. SAOL extends the syntax of Csound to make it more understandable and concise, and adds a number of new features to the Music-N model which will be discussed below.

It is not our contention that SAOL is a superior language to the others we cite and compare here. In fact, our belief is somewhat the opposite: the differences between general-purpose software-synthesis languages are generally cosmetic, and features of the languages’ implementations are much more crucial to their utility for composers. For the MPEG-4 project, we developed SAOL anew because it has no history or intellectual-property encumbrances which could impede the acceptance of the standard.

There were several major design goals in the creation of SAOL. These were: to design a music language which is highly readable (so that it is easy to understand and to modify instrument code), highly modular (so that general-purpose processing algorithms can be constructed and reused without modification in many orchestras), highly expressive (so that musicians can do complex things easily), and highly functional (so that anything that can be done with digital audio can be expressed in SAOL). Additionally, SAOL as a language should lend itself to efficient implementations in either hardware or software.

As well as the new features of SAOL that will be described below, many well-established features of Music-N languages are retained. These include: the sample-rate/control-rate distinction, which increases efficiency by reducing sample-by-sample calculation and allowing block-based processing; the orchestra/score distinction, in which the parametric signal processing instructions in the orchestra are controlled externally, by a separate event list called the score; the use of instrument variables to encapsulate intermediate state within instruments and global variables to share values between instruments; and a heavy dependency on stored-function tables or wavetables to allow efficient processing of periodic signals, envelopes, and other functions. These historical aspects of SAOL will not be discussed further here, but excellent summaries on the evolution and syntactic construction of music languages may be found in other references (Roads 1996; Dannenberg 1997a; Boulanger 1998 among others).

Among advanced features of other music languages not included in SAOL are the following. Extended Csound has a very sophisticated set of MIDI-control functions which allow General MIDI synthesizer architectures to be rapidly prototyped in Csound. While SAOL allows MIDI control, it has a simpler model. Nyquist embeds the score in the orchestra language, allowing composers a subtle mixture between “scoring” and “orchestration” and steps in-between; SAOL preserves the classical distinction. Both Nyquist and SuperCollider allow the use of higher-order functions, in which functions can be passed as parameters and otherwise manipulated as data types. There are no such facilities in SAOL. Finally, SuperCollider has a robust object-oriented model; while some aspects of SAOL move in that direction, the model is not formalized as thoroughly or attractively as it is in SuperCollider.

2.1 Readability

Where Csound is “macro-assembly-like”, Nyquist is “Lisp-like”, and SuperCollider is “Smalltalk-like”, SAOL is a “C-like” language. In terms of making the language broadly readable, this is a good step, because C is the most widely-used of these languages. The syntactic framework of SAOL is familiar to anyone who programs in C, although the fundamental elements of the language are still signal variables,
unit generators, instruments, and so forth, as in other music languages. The following program shows a simple SAOL instrument that creates a simple beep by applying an envelope to the output of a single sinusoidal oscillator:

```saol
instr beep(pitch, amp) { // make a beep
  table wave(harm, 2048, 1); // sinusoidal wave function
  asig sound; // 'asig' denotes audio signal
  ksig env; // 'ksig' denotes control signal
  env = kline(0, 0.1, 1, dur-0.1, 0); // make envelope
  sound = oscil(wave, pitch) * amp * env; // create sound by enveloping an oscillator
  output(sound); // play that sound
}
```

A number of features are immediately apparent in this instrument. The instrument name (beep), p-fields (pitch and amp), stored-function table (wave), and table generator (harm) all have names rather than numbers. All of the signal variables (sound and env) are explicitly declared with their rates (asig for audio-rate and ksig for control-rate), rather than being automatically assigned rates based on their names. The standard name dur is a variable automatically declared in every instrument which gives the duration of the note which invoked the instrument; there is a set of about 20 standard names defined in the language.

There is a fully-recursive expression grammar, so that unit generators like kline and oscil may be freely combined with arithmetic operators. The stored-function tables may be encapsulated in instruments or in the orchestra when this is desirable; they may also be provided in the score, in the manner of Music V (Csound also allows both options). The unit generators kline and oscil are built into the language; so is the wavetable generator harm.

### 2.2 Modularity

There is a highly capable set of unit generators built into the SAOL specification (100 in all). This set is fixed in the standard, and all implementations of SAOL must provide them. However, SAOL may be dynamically extended with new unit generators within the language model. Where other languages require rebuilding the language system itself to add new unit generators, this capability is a fundamental part of SAOL:

```saol
instr beep2(pitch, amp) { // another beep
  table wave(harm, 2048, 1);
  asig sound;
  ksig env;
  env = kline(0, 0.1, 1, dur-0.1, 0);
  sound = voscil(wave, pitch, 0.05, 5) * env; // 'voscil' is not a built-in ugen...
  output(sound * amp);
}
```

```saol
aopcode voscil(table wave, ivar cps, ksig depth, ksig rate) { // ... so we declare it here.
  // voscil: 'oscil' with vibrato:
  // 'wave' is the waveshape, 'cps' the carrier freq,
  // 'depth' the vibrato depth as a fraction,
  // 'rate' the vibrato rate
  ksig vib, newfreq;
  asig sound;
  table vibshape(harm, 128, 1); // waveshape for vibrato
  vib = koscil(vibshape, rate); // sinusoidal vibrato
  newfreq = cps * (1 - vib * depth); // FM banking
  sound = oscil(wave, newfreq); // new output
  return(sound); // return 'sound' to caller
}
```

The beep2 instrument makes use of a unit generator, voscil, which is not part of the standard set. The user-defined opcode below it implements this unit generator from a certain set of parameters. The aopcode tag indicates that the new unit generator produces an a-rate signal. Each opcode parameter (wave, cps, depth, and rate) is similarly defined with a rate type (table, ivar, ksig, and ksig respectively) which indicates the
maximum rate-of-change of each parameter. Using the same core functionality as instruments, user-defined opcodes perform a certain calculation and then return their results. In this case, the voscil user-defined opcode makes use of the koscil and oscil core opcodes to calculate its result. Any orchestra containing this user-defined opcode may now make use of the voscil unit generator.

A similar feature is provided for wavetable (function) generators: about 20 built-in wavetable generators are provided, but composers may also write opcodes which act as generators for new functions.

2.3 Expressivity and control

SAOL instruments may be controlled either through MIDI files, real-time MIDI events, or through a new score language called SASL (pronounced "sazzle"). For the cases when MIDI is used, a set of standard names pertaining to MIDI allows access to the standard MIDI control, pitch-bend, and aftertouch parameters, and channel and preset mappings are supported in SAOL. In SASL, more advanced control is possible:

```saol
instr beep3(pitch) {
  imports ksig amp, off;    // controllers
  ksig vol, init;
  table wave(harm,2048,1);
  asig sound;
  if (!init) {               // first time we're called
    amp = 0.5; init = 1;
  }
  if (off) { turnoff; }      // we got the 'off' control
  vol = port(amp,0.2);       // smooth the volume signal
  sound = oscil(wave,pitch);
  output(sound * vol);
}
```

This SAOL program can be controlled with a score written in SASL:

```
s1: 0.0  beep3 1 440    // first note
     0.5  beep3 1 480    // second note
n2: 1.0  beep3 -1 220   // third note - unbounded initial duration
n2: 1.0  beep3 -1 440   // fourth note
n3: 1.0  beep3 -1 660   // fifth note
  2.0  control n2 amp   // applies to third and fourth notes
  2.5  control n2 amp 0.5
  3.0  control n3 amp 0.2   // applies to fifth note
  3.0  control n2 amp 0.2
  4.0  control n2 off 1
  4.0  control n3 off 1
```

In the orchestra, the control signals amp and off are declared with the tag imports, which indicates that they may be updated by the score. The amp signal allows continuous control of the amplitude of the instrument output, and the off signal allows the instrument to be instructed to turn itself off. Notice that, unlike with MIDI, the meanings of these control signals are not fixed in the standard; the composer is free to specify as many controllers as needed, with whatever meanings are musically useful. When the off control is received, the instrument uses the built-in turnoff command to turn itself off; the built-in port unit generator is used to convert the discrete changes in the amp control signal into a continuous amplitude envelope.

The if (!init) clause is used to control the behavior of the first pass through each instance of the instrument. All variables in SAOL are like static variables in C; they are initialized to 0 automatically and preserve their values between iterations. Thus, the assignments to init and amp are preserved in the next iteration.

In the SASL score, n1, n2, and n3 are labels that control the mapping of control information to note events. Two types of score lines are shown; each has an optional label and a timestamp, which indicates the time at which the event is dispatched. The instrument lines specify the instrument which is to be used to create a note (beep in each case), the duration of the note (-1 indicates that the duration is indefinite), and any other p-fields to be passed to the note, as defined in the orchestra. The control lines begin with the tag control
and then specify a label, a variable name, and a new value. The variable name given will be set to the new value in every note which was instantiated from an instrument line with the given label. In this way, score-based control is more general and flexible than in MIDI or Csound.

More advanced control mechanisms are also possible in SAOL and SASL. The built-in `instr` and `extend` commands allow instruments to spawn other instruments (for easy layering and synthetic-performance techniques) and dynamically change the durations of notes. The `cpuload` standard name allows dynamic voice-stealing algorithms to be included in an orchestra.

2.4 Advanced functionality

There are many advanced features of SAOL which were added during the evaluation stage to enable easier construction of complex instrument models. For example, arrays of signals and of unit generators may be created and easily manipulated, to create parallel algorithms such as filterbanks and additive synthesizers concisely. Multiple “opcode states” may be maintained and directly accessed; this makes the construction of certain complex algorithms (and a weak object-oriented model) simpler. Also, built-in spectral manipulation unit-generators (`fft` and `ifft`) are provided, to allow spectral-domain techniques based on overlap-add windowing.

Other advanced features of SAOL include a sophisticated fractional and multistep delay-line unit-generator (`fracdelay`); built-in support for granular synthesis (`grain`), Karplus-Strong plucked-string synthesis (`pluck`), and waveshaping synthesis (`waveshape`); and others. A template mechanism is provided for the concise description of multiple instruments which differ only in a simple way. Interested readers are referred to (Grill et al., 1998), which contains the definition of the language, for more details on these and other features.

3 Standardization in Computer Music

This section discusses the role that the international standardization process has played in the development of SAOL, and the advantage this process can serve for the computer music community. SAOL did not exist as a language prior to the MPEG-4 standard; the standards work was not undertaken in order to place an ISO “seal of approval” on previously existing technology. Rather, a concerted effort was made to design the best music language possible, with the broadest application, so that MPEG-4 Structured Audio can serve the best purpose of standards: to unify a sometimes fragmented field and marketplace and prevent duplication of effort. If the MPEG-4 Structured Audio standard becomes broadly accepted, all computer musicians will benefit from the resulting explosion of hardware accelerators and composition tools around it.

The ISO in general, and the MPEG working group in particular, represent an ideal of the open-standards process in the development of new technology. Any researcher or company may make suggestions as to the future development of MPEG-4 and its Structured Audio tools; according to MPEG’s rules of procedure, these contributions must be formally evaluated, and if they are judged to be an important and positive direction, they will be incorporated into the standard. Anyone can read and evaluate the capabilities of the standard, which is publically available, for themselves; in addition, MPEG maintains a software implementation of the entire MPEG-4 standard, which is available in source code form to anyone interested in developing their own tools compliant with MPEG-4.

The MIT Media Lab wrote and maintains the source code for the SAOL tools, and has released this source code into the public domain for free use by the community. The Media Lab maintains no intellectual property rights or proprietary control over the direction of the standard, and will not gain materially from its acceptance in any way.

3.1 Languages and implementations

When evaluating music languages as languages, it is important to clearly keep in mind the distinction between language and implementation. For a particular music system, the language is the set of syntactic rules that describe what sequences of characters make up legal programs, and the semantic rules that describe how to make sound from a program. An implementation, by contrast, is a particular computer program which can perform the required mapping from sequence-of-characters to sound; depending on the music system, other information such as scores, MIDI files, control sequences, or real-time interaction may affect this process.
It has traditionally been the case in the computer-music field that languages have been defined in regard to implementations. That is, the Csound “language” is not defined anywhere except as a computer program which accepts or rejects particular sequences of characters; as this implementation evolves and changes over time, the set of “legal” Csound orchestras changes as well (generally by expansion, in a backward-compatible manner). This is not the case in the computer-language world at large; languages such as C, Fortran, Lisp, Ada, C++, and Java have clearly written specifications which describe exactly what a legal implementation must and must not do.

3.2 Advantages and disadvantages of standardization

The primary advantage of the specified-language model is that it promotes compatibility between implementations. For a large set of non-pathological programs, a programmer is guaranteed that a C program will execute the same way under any legal C compiler; the ANSI C standard tells the developer of a new C compiler exactly what the compiler must do in response to a particular program. This interoperability promotes the development of an efficient marketplace for compilers, debuggers, editors, and other programming tools; if a company develops a new compiler which is much faster than others available, the marketplace can shift quickly because existing code is still useful, and so there is a competitive advantage to a company to develop such powerful compilers. If languages are not compatible between implementations, programmers (musicians) become “locked into” a certain implementation (because it is the only one which will run their instruments) and there is not a competitive reason to develop a better one.

The disadvantages of specified languages are that standards have a certain stifling effect on innovation (C and C++ have had a strong hold on the development of production code for many years), and they may not utilize the resources of a particular computing environment with maximal efficiency. Even after years of compiler development, hand-coded assembly, especially for high-performance architectures such as DSPs or vector processors, still may have performance advantages over code compiled from high-level languages. But since assemblers are generally incompatible from architecture to architecture, the development tools for assembly-language coding are not as advanced as those for high-level languages. With sufficient effort and programming skill, advanced development environments and high performance signal-processing capabilities can be merged in a single tool.

McCartney’s language and software-system SuperCollider (McCartney 1996a, b) is an excellent example of this: it provides highly efficient execution, a sophisticated front-end, and a music language with more advanced capabilities than SAOL, but only in a proprietary framework on a single platform. Note, though, that the quality of SuperCollider’s front-end and the efficiency of its execution are not a property of SuperCollider as a synthesis language, but rather a testament to McCartney’s skill in creating an implementation of the language. Similarly, the restriction to a single platform is not intrinsic to the language; the language could, once clearly specified, be implemented on other platforms with other front-ends.

As computers and signal-processing units get ever-faster, eventually the need for high-quality, compatible development tools for music languages will be more pressing than the need to squeeze every cycle out of existing processors. This is the world which the design of SAOL targets: one in which musicians are willing to accept 75% performance on state-of-the-art hardware in exchange for inexpensive yet powerful implementations, sophisticated interfaces, cross-platform compatibility, and a broad and competitive marketplace for tools, music, and hardware.

4 Conclusion

SAOL, the MPEG-4 Structured Audio Orchestra Language, is a powerful and flexible new music language. Further, as part of the MPEG-4 International Standard, there will soon be many real-time implementations, composer’s desktops, and other tools available for using it. A move to acceptance of powerful open standards in the computer-music field will create an explosion of opportunity for musicians and technologists with the creativity and skill these tools demand, and will create a “rising tide” supporting other real-time software synthesis tools.

The MPEG-4 Structured Audio tools have the potential to change the world of computer music. The capabilities provided will be taken seriously by large companies, for MPEG standards are an important touchstone in the computer industry. Through the inclusion of general-purpose software synthesis tools in
the MPEG-4 standard, the needs and inventions of the computer-music community have been exposed to this industry for a very broad hearing. We hope that other computer musicians will help in supporting the new standard, developing conformant implementations, and creating music and tools which can help it to prosper.

For readers interested in using or developing tools based on SAOL, the “SAOL home page” on the WWW may be found at <http://sound.media.mit.edu/mpeg4>. This site contains current information on the progress of the standard, up-to-date software implementations, example compositions, a library of user-defined unit generators, complete documentation on the SAOL language, and mailing lists which support the SAOL community.

References


