Audio Engineering Society

Convention Paper

Presented at the 116th Convention
2004 May 8–11 Berlin, Germany

Real-Time Internet MPEG-4 SA Player and the Streaming Engine

Alvin W.Y. Su[1], Yi-Song Xiao[2], Jia-Lin Yeh[3] and Jian-Lung Wu[4]

[1] SCREAM, Dept. Of CSIE., National Cheng-Kung University, Tainan, Taiwan, R.O.C.
alvinsu@mail.ncku.edu.tw

[2] SCREAM, Dept. Of CSIE., National Cheng-Kung University, Tainan, Taiwan, R.O.C.
al@cmslab.csie.ncku.edu.tw

[3]SCREAM, Dept. Of CSIE., National Cheng-Kung University, Tainan, Taiwan, R.O.C.
naturalized@msn.com

[4]SCREAM, Dept. Of CSIE., National Cheng-Kung University, Tainan, Taiwan, R.O.C.
f7488405@gmail.com

ABSTRACT

MPEG-4 Structure Audio is an algorithmic based coding standard designed for low bit-rate high quality audio. With this standard, the desired sound can be identical on both the encoder side and the decoder side by using Structured Audio Orchestra Language (SAOL) to generate sound samples. It requires a player and a streaming engine when real-time interactive internet presentations are necessary. In this paper, we present such a system implemented and applied over IBM PC based computers. The proposed streaming engine follows ISMA specification and its implementation is closely related to Apple's Darwin Server. After the streaming SA player receives the bitstream from the server, it converts SAOL data stream to JAVA codes and links to a proposed scheduler program generated from SASL data stream for direct execution such that one can hear the sound in real time. Unlike sfront, no intermediate C codes and C compilers are necessary. In order to improve the performance, optimized software modules such as the core opcodes and the core wavetable engine have been embedded. Significant speedup is achieved compared to the reference SAOLC decoder. Real-time demonstration of the system will be made during the presentation. Discussion of the possible future algorithmic coding method using JAVA is also given.

1. INTRODUCTION

MPEG-4 is the more recent multimedia standards for encoding, decoding, and transmission of multimedia data [1]. Both audio and video parts support object-based representation approaches for synthetic content to guarantee the reproduction quality at the decoder sides. Structured Audio (SA) was developed for synthetic audio objects and was rooted from quite a few computer languages for synthesized music reproduction [2]-[3]. SA contains several parts for different functions described as follows. The SAOL (Structured Audio Orchestra Language) defines the music instruments for sound generation and processing. Because SAOL is a C-like generic computer language and includes some useful mathematic functions, it can be used to adopt almost any synthesis algorithms, such as wavetable, FM, and physical modeling methods [4]-[6]. To execute the algorithms implemented with the SAOL is the most important part of the complete MPEG-4 SA system and it will take most of the computation power. The second part is the SASL (Structured Audio Score Language). The SASL maintains the music event lists which control the orchestra with an object-based description approach. The SASL performs the functions similar to the MIDI messages [7]. Because many past computer music applications use MIDI as their control, MPEG-4 SA adopts MIDI as a method to control the orchestra, too. To include the natural sound and wavetables, the MPEG-4 provides the SASBF (Structured Audio Sample Bank Format). When combining SASBF and SASL, it is similar to MIDI-DLS [8]. Based on the above tools, the MPEG-4 is divided into three profiles: the score-based profile, the wavetable profile and the full profile. The detail can be seen in [9]. Only the full profile contains the SAOL and the SASL tools.

Though MPEG-4 SA is mainly for synthetic sound reproduction, it can also be used to general audio coding methods such as MPEG-1 Layer I and LPC codec [10]-[11]. This is not surprising because the SAOL is general enough to perform such jobs. In other words, most sound applications can be implemented with the SAOL, natural or synthetic. Being such as a powerful tool, the MPEG-4 SA is still not popular though it has been proposed for years. In order to implement the full profile, the system must be able to execute the instructions written by the SAOL. However, the most popular tool for the SAOL is the sfront provided by [12]. The sfront tool converts the SAOL program to an ordinary C program. Then, one must use a C compiler to compile the C program such that it can be executed with a specific computer system. The sfornt is similar to a SAOL decoder but the decoded output cannot be executed in real time. If the MPEG-4 SA full profile messages are to form a dedicated track in a standard MPEG-4 bitstream for real-time internet streaming applications with other media decoders, this is simply not working. Therefore, it is necessary to develop a "decoder" that can decode the SA messages and generate the sound in real time. In this paper, we propose such a JAVA based decoder for SAOL message because JAVA can be executed at most computer platforms as long as JAVA virtual machines are installed and no C compiler is necessary. Because a system running on a JAVA based environment is not as efficient, we provide a few tools to reduce the computational load. First of all, a core wavetable engine and a default wavetable synthesizer are embedded. Second, a core opcode engine defined in the SAOL is also provided. It is noted that users can use his/her algorithms to generate the sounds without using the provided tools by writing the SAOL programs. Significant speedup is achieved compared to the reference decoder SAOLC [13] due to the addition of these two tools. Third, a program scheduler is provided to control the synthesis processing in real time based on the received SASL data stream. It is noted that the overall system is implemented with optimized C programs except that SAOL decoder engine because the functions for the SASL decoder, the core opcodes engine, and the core wavetable engine and the default wavetable synthesizer are all fixed functions. This can improve the processing speed significantly. In order for the real-time internet streaming applications, we provide a streaming tool that follows the ISMA (International Streaming Media Association) specification and is also closely related to the Apple Darwin Streaming Server® [14]. Because the Apple Quick-Time® player doesn't support the MPEG-4 SA data streams, a proprietary player is provided to demonstrate the proposed system. All of the works are developed over the Microsoft Window® based IBM PC® computers. In general, the proposed is still not perfect. However, this is the first tools to realize the promise made by the MPEG-4 SA standards.

The rest of the paper is organized as follows. The brief introduction of the SA system is given in section 2. In section 3, the implementation of the proposed player and data server is described. In section 4, the streaming engine and the data format are described. In section 5,

the performance comparison with sfront and SAOLC is provided. Conclusion and future works are in section 6.

2. STRUCTURE AUDIO BASED SYSTEM

The MPEG-4 Structure Audio is a software synthesis-description language. It was developed based on some past such languages such as Csound and Music V [15]. In addition to its basic synthesis functions, transmission of the SA bitstream is included in the MPEG-4 standard and this makes real-time internet applications possible. There are some basic components inside a SA based system, described as follows:

SAOL: A synthesis-description language used to synthesize sounds based on the algorithms implemented with SAOL. It is a digital signal processing language with some core opcodes that are designed for music synthesis.

SASL: A simple score or control language to work with SAOL messages about how and when to produce sounds. Very much like MIDI messages.

SASBF: It is used for storage or transmission of the audio samples that are required in the synthesis algorithms described by SAOL.

Scheduler: A run-time element of the SA based system in the decoding process. It converts the SASL or MIDI messages to real-time events to control the synthesizers implemented with SAOL.

Normative reference to MIDI standard: Since MIDI is used by many manufacturers, it is accepted to replace SASL though it is less powerful.

The most important kernel of the SA based system is the SAOL which is general enough to be used to "describe" almost any possible synthesis algorithms. As a matter of fact, it can be even used to implement some sound decoders such as MP-3 and AAC [16] though the performance might not be able to keep up with most dedicate decoders implemented with standard C programs or even low-level languages. Detail SA syntax can be found in [3].

3. THE PROPOSED MPEG-4 SA PLAYER

Figure 1 shows the basic jobs required in the proposed SA player - ScreamSA. When a MPEG-4 SA bitstream is received, it is divided into four parts: header, sample data (SASBF), score data (MIDI or SASL) and orchestra data (SAOL). They are stored in respective areas. The header and SAOL data are reconfigured such that it can be executed with the decoding computer. Usually, it requires a parser to convert the data to a program compatible with a high-level computer language such as C (This is what sfront does.). Then a corresponding compiler is used to compile the program and link the result for execution. Because the SAOL is object-based, synthesizers and other function modules can be generated such that they can be executed when they are called by the scheduler. The sample data is stored and will be used when any of the synthesizers is

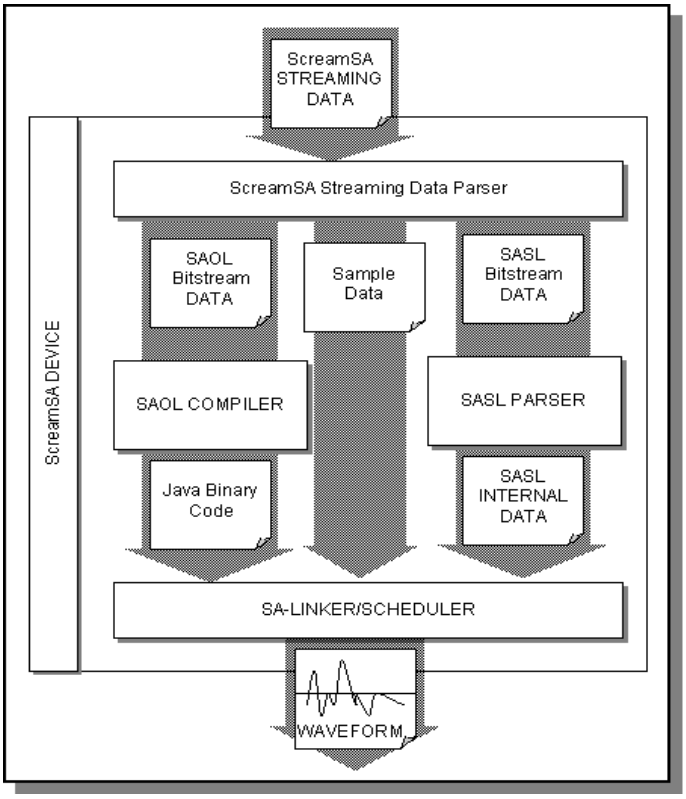

Figure 1 Structure Audio decoding process

asked to perform its work. The data can be audio samples, synthesis parameters, filter coefficients, and so on. Finally, the score data is used to construct the scheduler to control the sound generation process. Those who are familiar with MIDI processing should have no difficulties in understanding this part [7]. The score data usually contains two parts of information: the timing and the control. Based on the timing information, event lists will be generated in the scheduler. Then, the scheduler sends the commands to start the processing

units implemented with SAOL at the pre-determined time instants. A command usually also includes the control parameters of the synthesis algorithm. Appendix 1 and Appendix 2 show one SASL as the score data and a synthesis algorithm implemented with SAOL, respectively. It is noted that the normative reference to MIDI standard is currently not supported.

Because our system uses JAVA, JAVA binary codes are generated after the SAOL codes are converted. There are three types of information in the JAVA binary codes: global run-time environment parameters, instrument synthesis codes, and user-defined functions. A more detailed diagram of the SAOL compiler is shown in Figure 2. The ScreamSA supports all the core opcodes defined in SAOL. They are optimized coded and pre-compiled as a library to improve the speed. A wavetable synthesizer is also provided in case that one wants to use SASL only and rely on the basic synthesis engine of the ScreamSA. This is a very simple wavetable based synthesis engine and its sound quality is still not as good as those state-of-the-art wavetable engines. The SAOL bitstream is converted into JAVA text format data with the SAOL Parser. Then, a JAVA class compiler is used to generate the JAVA binary codes described above.

Figure 3 shows the ScreamSA Linker/Scheduler. It receives the SASL bitstream and generates Score Event Data (or Score Event List). Then it combines the Event Data with the JAVA binary codes and the sample data to perform the synthesis processing with the Instrument Machine Code Engine. Finally, audio samples are generated and sent to a soundcard for playing back.

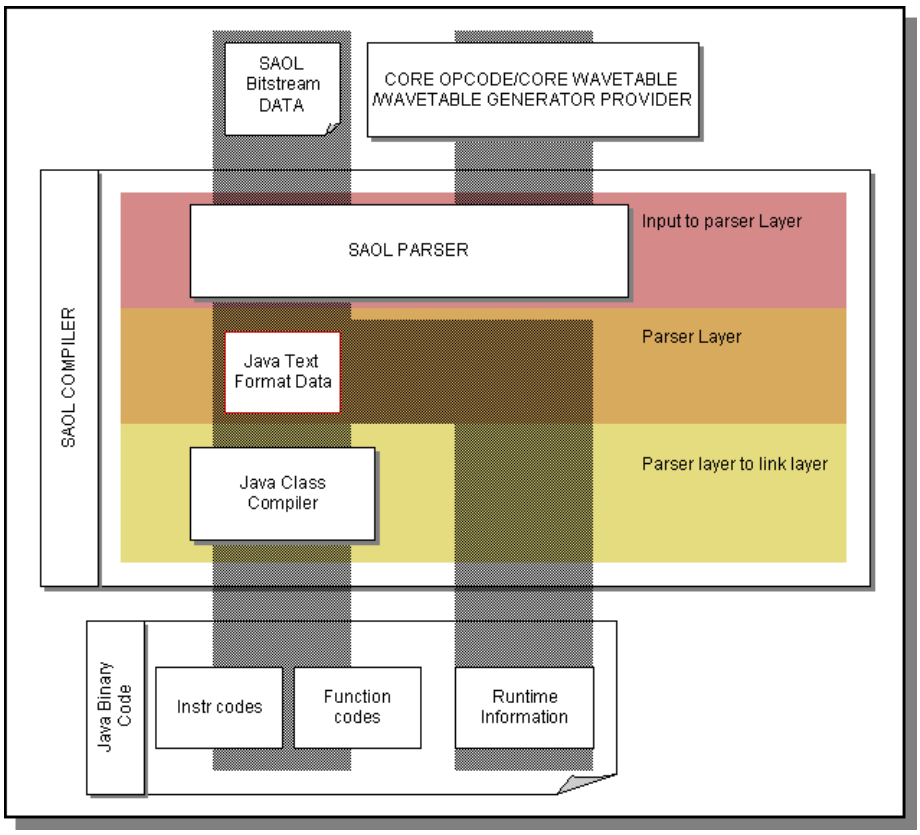

Figure 2 ScreamSA SAOL-JAVA Conversion Process

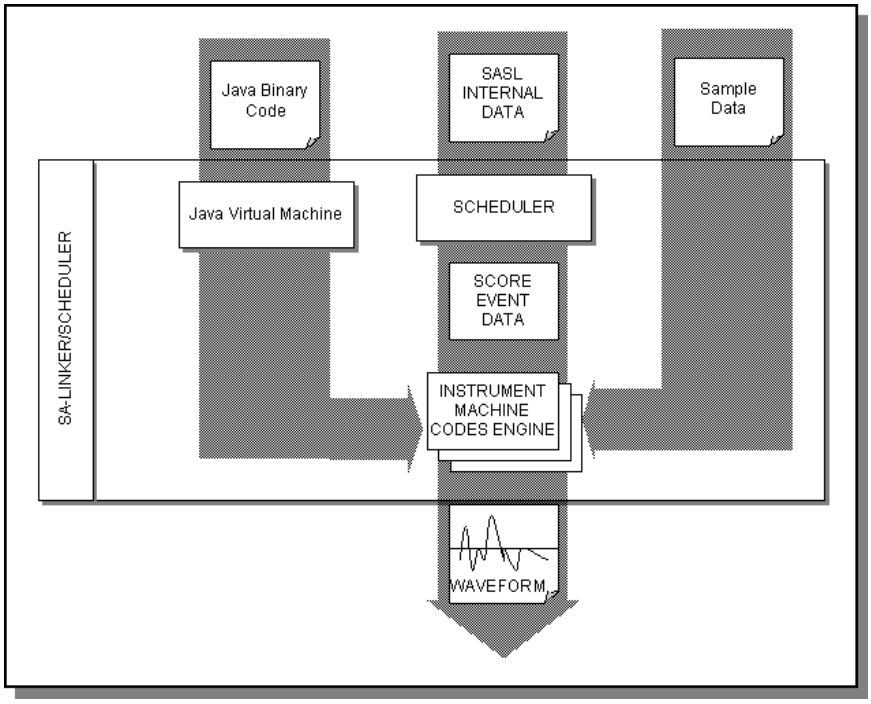

Figure 3 ScreamSA Linker/Scheduler

4. STREAMING APPLICATION

0 0	1	2	3	4	5	6	7	8	9	1 0	1	2	3	4	5	6	7	8	9	2 0	1	2	3	4	5	6	7	8	9	3 0	1
V		*P*	*X*	*CC*				*M*		*PT*						*Sequence number*															
Timestamp																															
synchronization source (SSRC) identifier																															

Table 1 RTP header format.

In order to apply the SA system to Internet applications, streaming technology is necessary. Because RTP is the official streaming protocol employed by ISMA (Internet Streaming Media Alliance) and MPEG-4, RTP will also be included in our system. Table 1 shows the RTP data header format. The definitions are as follows:

V: RTP version number.

P: 0 in our implementation.

X: Header extension

CC: CSRC count.

M: Indication of whether it is the last packet of the *sample* transmitted. The *sample* here is a MPEG-4 file format *sample* instead of audio samples.

PT: Payload type.

Sequence number: Generated randomly in initialization.

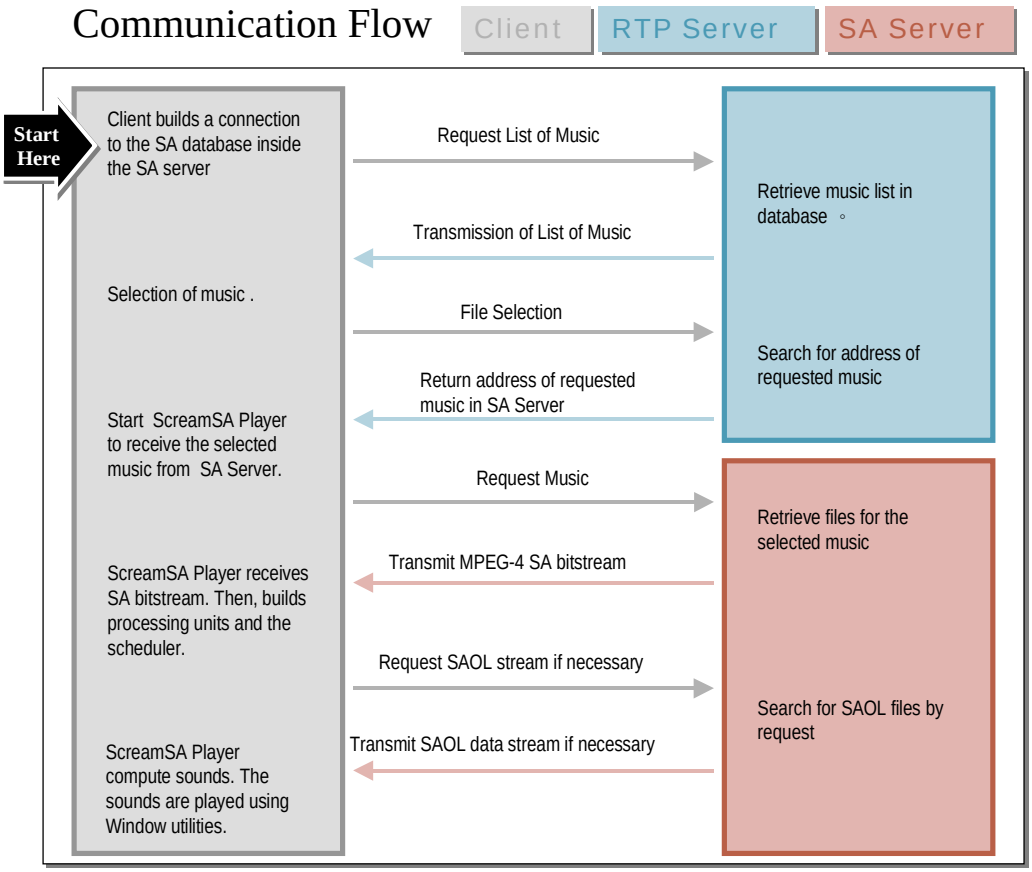

Figure 4 MPEG-4/ScreamSA session process.

Timestamp: We use tick as the basic unit that is identical to MIDI. The reason is that it is more precise.

SSRC: determined in the playback stage.

Figure 4 shows how the proposed client-server system processes a typical session. The SA media data uses the standard MPEG-4/QuickTime data format [17]. In order to combine with other media data such as video, appropriate timing information is necessary. In *Media header atom*, *timeScale* information is set to be the *krate* and *duration* has to be computed in advance and uses *krate* as its unit. In *MPEG-4 Audio atom, info* contains the important parameters such as *arate*, *track*, *outputBusWidth* and so on. In *Time-to-sample atom*, *SampleCount* is a 32-bit integer that specifies the number of consecutive samples that have the same duration and *Sample duration* is another 32-bit integer that specifies the duration of each sample. If the size of a sample is 2 second and krate is set to 10, *SampleDuration* is 20. If the length of a piece of music is 60 seconds, *SampleCount* is 30. The other arrangement of the data stream depends mostly on the original SA data itself and can be found in [19]. However, the above implementation has one major problem because *krate* and *duration* will be fixed. Therefore, the *settempo* opcode defined in SAOL is prohibited. Otherwise, synchronization and data communication of the system will have problems.

5. PERFORMANCE COMPARISON

For real-time Internet streaming applications, it is necessary to have a high performance player. Currently, the most popular tools have been the standard SAOLC [13] and sfront [12]. None of them provides convenient tools to construct a streaming server and the associate player. Nevertheless, the required computation power for a SAOL-implemented instrument is still a main issue. As mentioned in section 3, JAVA is used in the synthesis engine part. In fact, there are different ways of implementing the server-client system. First, a SAOL-JAVA converter can be used in the player side. Once the SAOL stream is received, it is converted to JAVA instructions that are executed through a JAVA virtual machine. This is what we have done in section 3. In order to reduce the workload of the player, one can also convert the SAOL stream in the server side. Then, the resultant JAVA instructions are converted into data stream and sent to the player side. Therefore, no SAOL-JAVA conversion is necessary in the player. The problem with the second approach is that it will not be standard-compliant and players have to have a JAVA virtual machine installed. The third implementation suggestion is to use JAVA or other similar high-level computer languages to replace the role played by SAOL. The reason is that it is very difficult to provide a SAOL-JAVA converter that is so efficient that the performance can be compared to that of the program originally coded with C or JAVA. Since JAVA can implement all the SAOL functions and has many advantages over conventional programming languages in the areas such as cross platform, memory management, include-library management and so on, a new standard may be formed to use such programming languages as the new orchestra language.

In order to test the above different implementations, we use a simple algorithm to test their performances. The Karplu-Strong plucked string model is used in this paper [18]. The SASL part and the SAOL part are shown in Appendix 1 and Appendix 2, respectively. Both SAOLC and sfront are used for comparison. These two tools can be obtained from [12]-[13]. The third program is generated from converting the SAOL program to a JAVA program automatically using the proposed tools. This is shown in Appendix 3. The last program is modified from the third program by adding a JAVA class called “Queue” which greatly improves the speed. This is shown in Appendix 4. Both the 3rd and the 4th programs are executed with Microsoft JAVA Virtual Machine. However, the case of the last program replies on a human programmer to write the codes instead of generating them automatically from the SAOL codes. For all the programs, *srate* is set to be 44100 and *krate* is set to be 100. The duration is 4 seconds. These programs are executed on a Pentium 3 1.2G/512MB personal computer running Window XP. The execution times of the four cases are shown in Table 2.

6. FUTURE WORKS

A complete Structure Audio based system called the ScreamSA consisting of a streaming server and a real-time player is presented in this paper. SAOL, the most computation heavy part, is converted into JAVA first. Then, it can be executed using JAVA virtual machines provided in many computer platforms. This increases the portability of the player. The streaming engine follows the suggestions of ISMA and the proposed system receives and decodes the bitstream in real time as long as the computation complexity is within the limit of the client platform. Though the currently existing system is slower than sfront, it nevertheless provides some convenient tools which realize the promise made by Structure Audio codec several years ago. The proposed codec system can be found and downloaded at [19].

This paper also raises a major problem about the performance limitation when SAOL is used. With SAOL, it is necessary to design a converter such that it can be executed, for example, sfront and our ScreamSA. Because some programming languages such as JAVA can fully replace the role played by SAOL with much better performance, one might want to use them in the future standards. Our testing example shows that over 4 times improvement can be achieved. Therefore, we will work on such a system in the near future. Of course, we will disclose all the implementation details and source codes, too. The normative reference to MIDI standard may be added and the quality of the default wavetable engine will be improved, too.

APPENDIX 1

SASL testing program

```
0 tempo 60
0 tone 1
1 tone 1
2 tone 0.5
2.5 tone 0.25
2.75 tone 0.25
3 tone 0.5
3.5 tone 0.5
4 end
```

APPENDIX 2

SAOL testing program

```
global {
  srate 44100;
  krate 100;
}
instr tone () {

  asig param[127],paramrun[127];
  asig fil, buf;
  asig counter, i;
  param[0]=7346.000000;
                  :
                  :
  param[126]=6582.000000;
  if (counter == 0) {
    i=0;
    while (i < 127)
    {
      paramrun[i]=param[i];
      i=i+1;
    }
  }
  if (counter < 600000) {
    buf=fil;
    fil=paramrun[126];
    i=126;
    while (i > 0) {
      paramrun[i]=paramrun[i-1];
      i=i-1;
    }
    paramrun[0]=0.5*fil+0.496*buf;
    output(0.00008*paramrun[0]);
  }
  counter = counter +1;
```

APPENDIX 3

JAVA program converted from SAOL program

```
import SSA.*;
public class KS extends SSA.Instr
```

	Execution Times (sec)
Saolc [13]	47.2
Sfront [12]	2. 5
ScreamSA	4.05
ScreamSA with Queue Java class	0.83

Table 2. Execution times of four SAOL decoders

```
{
   static float[]param;
   float[]paramrun;
   float fil, buf;
   int counter, i=0;
   public KS(){
   paramrun=(float[])param.clone();
   }
   public void arate(Bus inputBus,Bus outputBus)
   {
            int length=SAVM.activeSAVM.busLength;
```

```
            for(int x=0;x<length;x++)
            {
               if (counter < 600000)
               {
                  buf=fil;
                  fil=paramrun[126];
                  i=126;
                 while (i > 0)
                 {
                       paramrun[i]=paramrun[i-1];
                       i=i-1;
                 }
                 paramrun[0]=0.5f*fil+0.496f*buf;
                 outputBus.output(0.00008f*paramrun[0]);
               }
              counter=counter+1;
              outputBus.advanceCursor();
            }
    }
    static
    {
        param=new float[127];
        param[0]=7346.0f;
                        :
                        :
        param[126]=6582.0f;
    }
}
```

APPENDIX 4

Fast JAVA program

```
import SSA.*;
public class KSFast extends SSA.Instr
{
    static float[]param;
    float currentSample=0;
    Queue queue;
    public KSFast(){
            queue=new Queue();
            queue.setBuffer((float[])param.clone());
   }
    public void arate(Bus inputBus,Bus outputBus)
    {
        int length=SAVM.activeSAVM.busLength;
        for(int x=0;x<length;x++)
        {
            float last=currentSample;
            currentSample=queue.getItem(126);
            currentSample*=0.496f;
            currentSample+=0.5f*last;
            outputBus.output(0.00008f*currentSample);
            queue.push(currentSample);
            outputBus.advanceCursor();
        }
    }
    static
    {
            param=new float[127];
            param[0]=7346.0f;
                        :
                        :
            param[126]=6582.0f;
    }
}
final class Queue
{
            public int bufferLength;
            private float[]buffer;
            private int topIndex;
            public Queue()
            {
                        topIndex=0;
            }
            public Queue(int size)
            {
                        topIndex=0;
                        setBuffer(new float[size]);
            }
            public void setBuffer(float[]nb)
            {
                        buffer=nb;
                        bufferLength=nb.length;
            }
            public float getItem(int index)
            {
                        int i=index+topIndex+1;
                        if(i<0){
                                    i+=bufferLength;
                        }else if(i>=bufferLength){
                                    i-=bufferLength;
                        }
                        return buffer[i];
            }
            public void push(float v)
            {
                        buffer[topIndex++]=v;
                        if(topIndex==bufferLength)topIndex=0;
            }
}
```

7. ACKNOWLEDGEMENTS

This work was supported by National Science Council, Taiwan, R.O.C. under contract No. 93-2213-E-006-025.

8. REFERENCES

[1] Rob Koenen “Overview of the MPEG-4 Standard” ISO/IEC JTC1/SC29/WG11 N4668 March 2002

[2] Eric D. Scheirer, “The MPEG-4 Structured Audio stan-dard,” in Proc. IEEE International Conference on Acoustics,Speech and Signal Processing (ICASSP-98), 1998

[3] Scheirer, E. D. & Vercoe, B. L. (1999). SAOL: The MPEG-4 Structured Audio Orchestra Language. ComputerMusic Journal 23(2), 31-51

[4] R. Bristow-Johnson. “Wavetable synthesis 101: afundamental perspective”, Proc AES 101st, 1996 (AESReprint #4400).

[5] Scheirer, E. D. & Ray, L. (1998). “Algorithmic and wavetable synthesis in the MPEG-4 multimedia standard.” InProceedings of the 1998 105th Convention of the Audio Engineering Society (reprint #4811). San Francisco.

[6] Julius O. Smith, III, “Physical modeling synthesis update,”Computer Music Journal, vol. 20, no. 2, pp. 44–56, 1996

[7] The Association of Musical Electronics Industry, ”MIDI Media Adaptation Layer for IEEE-1394” November 30, 2000.

[8] MIDI Manufacturers Association, “MIDI Downloadable Sounds Level 1 Specification “ version 1.1(January 1999).

[9] Scheirer, E. D. (1999). “Structured audio and effects processing in the MPEG-4 multimedia standard” MultimediaSystems 7(1), 11-22

[10] S. Shlien, "Guide to MPEG-1 Audio Standard," IEEE Transactions on Broadcasting, Vol. 40, pp. 206--218, Dec. 1994.

[11] J.D.Markel and A.H. Gray,Jr.,Linear Prediction of Speech,Springer-Verlag,1976

[12] http://www.cs.berkeley.edu/~lazzaro/sa/sfman/

[13] http://web.media.mit.edu/~eds/mpeg4-old/

[14] "QuickTime Streaming Server Modules" and "QuickTime Streaming Server Administrator's Guide" documentations available on http://developer.apple.com/darwin/projects/streaming/

[15] B. L. Vercoe, Csound: A Manual for the Audio-Processing System (rev.1996). Cambridge, MA: MIT Media Lab., 1985.

[16] ISO/IEC JTC1/SC29/WG11 MPEG, InternationalStandard ISO/IEC 13818-7 "Generic Coding ofMoving Pictures and Associated Audio: AdvancedAudio Coding", 1997

[17] Apple Computer ,Inc. “QuickTime File Format” 2000

[18] Karplus,K., and Strong ,A.(1989)”Digital synthesis of plucked-string considered as an electrical transmission line,” J.Acoust. Soc. Am. 8,227-233.

[19] http://scream.csie.ncku.edu.tw/ScreamSA/index.htm